

Распространенные приложения контроля:

Руководящие принципы, шаблоны
проектирования

и примеры применения с IEC 61499

ПРОМЫШЛЕННЫЙ РЯД ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Редактор серии
RICHARD ZURAWSKI

**Распространенные приложения контроля:
Руководящие принципы, шаблоны проектирования
и примеры применения с IEC 61499**
Под редакцией Alois Zoitl и Thomas Strasser

**Руководство по эксплуатации промышленной
технологии связи, издание второе**
Под редакцией Richard Zurawski

Формальные методы в производстве
Под редакцией Javier Campos, Carla Seatzu
и Xiaolan Xie

**Справочник по встроенным системам, издание
второе**
Под редакцией Richard Zurawski

**Справочник по автомобильным встроенным
системам**
Под редакцией Nicolas Navet и Francoise Simonot-Lion

**Интегрированные технологии для промышленных
автоматизированных систем**
Под редакцией Richard Zurawski

**Автоматизация электронного проектирования для
управления интегральными схемами**
Под редакцией Luciano Lavagno, Grant Martin
и Lou Scheffer

Приложения распределенного управления:

Руководящие принципы, шаблоны
проектирования и примеры применения
с IEC 61499

Под редакцией

Alois Zoitl • Thomas Strasser



CRC Press

Taylor & Francis Group
Воса Raton Лондон Нью-Йорк

УДК 005
ББК 65.050
Ж (каз)

MATLAB® является торговой маркой MathWorks, Inc. и используется по разрешению. MathWorks не гарантирует точность текста или упражнений в этой книге. Использование этой книги или обсуждение программных продуктов MATLAB® или сопутствующих товаров не означает одобрения или поддержки MathWorks, конкретного педагогического подхода или использования конкретного программного обеспечения MATLAB®.

CRC Press
Taylor & Francis Group
NW Broken Sound Parkway, 6000, 300 экземпляров
Бока-Ратон, Флорида 33487-2742
Предыдущее издание было выполнено в мягкой обложке в 2017 г.
© 2016 Taylor & Francis Group, LLC

CRC Press не является собственностью Taylor & Francis Group, компании Informa
Нет претензий к деятельности правительства США
ISBN-13: 978-1-4822-5905-6 (hbk)
ISBN-13: 978-1-138-89295-8 (pbk)

Эта книга включает информацию, полученную из подлинных и высоконадежных источников. Нами были приложены все необходимые усилия для публикации надежных данных и информации, однако автор и издатель не могут принять на себя ответственность за обоснованность всех материалов или последствия их использования. Авторы и издатели попытались идентифицировать владельцев авторских прав всего материала, воспроизведенного в этой публикации, и приносят извинения владельцам авторских прав, если разрешение опубликовать материалы в этой книге не было получено. Если авторское право на какой-либо материал не было подтверждено, напишите и сообщите нам, и мы внесем исправления в будущие издания.

За исключением того, что разрешено в соответствии с американским Законом об авторском праве, никакая часть этой книги не может быть переиздана, воспроизведена, передана или использована в любой форме любыми электронными, механическими или другими средствами, существующими сейчас или изобретенными после издания книги, включая фотокопирование, микрофильмирование и запись в любой системе хранения информации или поисковой системе, без письменного разрешения от издателей.

Для получения разрешения фотокопировать или использовать материал этой работы в электронном виде, обратитесь на сайт www.copy-right.com (<http://www.copyright.com/>) или свяжитесь с Copyright Clearance Center, Inc. (CCC), 222 Диска Палисандра, Дэнверз, Массачусетс 01923, 978-750-8400. CCC является некоммерческой организацией, которая предоставляет лицензии и регистрацию для множества пользователей. Для организаций, которым были выданы лицензии на фотокопии CCC, существует отдельная система оплаты.

Товарный знак: Названия продуктов или названия компаний могут быть торговыми марками или зарегистрированными торговыми марками и используются только для их идентификации и демонстрации без намерения нарушить их права.

Посетите веб-сайт Taylor & Francis
<http://www.taylorandfrancis.com>
и веб-сайт CRC <http://www.crcpress.com>

ISBN 978-601-333-655-8 (каз)

Содержание

Предисловие	17
Редакторы	19
Список благодарностей	21
Участники	23
I. Основы IEC 61499	25
1. Проблемы и требования распределенной автоматизации в промышленной среде	27
<i>Thomas Strasser u Alois Zoitl</i>	
1.1 Тенденции в автоматизации производства	27
1.2 Требования к будущей архитектуре автоматизации	29
1.3 Перспективы	31
2. Основные принципы эталонной модели IEC 61499	35
<i>Thomas Strasser u Alois Zoitl</i>	
2.1 Введение	36
2.2 Эталонная модель IEC 61499	37
2.2.1 История	37
2.2.2 Обзор стандарта	38
2.2.3 Модель выполнения на основе событий	39
2.2.4 Функциональные блоки как основные элементы моделирования	40
2.2.5 Прикладная и системная модель. Модели устройства и ресурса	42
2.2.6 Модель распределения	43
2.2.7 Модель управления	44
2.2.8 Модель рабочего состояния	45
2.3 Основные отличия между первой и второй моделями	46
II. Руководство по проектированию и разработке приложений	49
3. Шаблоны проектирования, цели и методологии	51
<i>James H. Christensen</i>	
3.1 Введение, мотивация и обзор	51

3.2 Методология распределенного приложения	53
3.2.1 Цель	53
3.2.2 Методология	53
3.2.3 Поддержка инструмента	54
3.3 Шаблон «ргоху»	55
3.3.1 Цель	55
3.3.2 Методология	55
3.3.3 Пример	55
3.4 Многоуровневая Модель / Представление / Контроллер / Диагностика (MVCD)	56
3.4.1 Цель	56
3.4.2 Платформа	56
3.4.3 Методология	56
3.5 Локальный многоадресный шаблон	58
3.6 Теговый шаблон данных	59
3.7 Матричная платформа	59
3.8 Выводы	60

4. Применение парадигм дизайна IEC 61499: объектно-ориентированное программирование, компонентно- ориентированный дизайн и архитектура для обслуживания широкого круга запросов

61

Wenbin Dai, Valeriy Vyatkin и James H. Christensen

4.1 Введение	62
4.2 Существенные элементы в IEC 61499 проектов функционального блока	63
4.3 Применение парадигмы объектно-ориентированного программирования в IEC 61499 функциональных блоков	65
4.3.1 Предпосылки объектно-ориентированного программирования	65
4.3.2 Применение понятий объектно-ориентированного программирования в стандарте IEC 61499	65
4.3.3 Тематическое исследование парадигмы объектно- ориентированного программирования	68
4.4 Принятие компонентно-ориентированной парадигмы дизайна для функциональных блоков IEC 61499	75
4.4.1 Предпосылки компонентно-ориентированного дизайна	75
4.4.2 Компонентно-ориентированные принципы разработки для функциональных блоков IEC 61499	75
4.4.3 Тематическое исследование компонентно-ориентированной парадигмы дизайна	80
4.5 Представление новой парадигмы архитектуры для обслуживания широкого круга запросов для функциональных блоков IEC 61499	81

4.5.1 Предпосылки архитектуры для обслуживания широкого круга запросов IEC 61499	81
4.5.2 Как применить понятия архитектуры для обслуживания широкого круга запросов в разработке системы IEC 61499	82
4.5.3 Тематическое исследование парадигмы архитектуры для обслуживания широкого круга запросов	86
4.6 Краткое содержание парадигм проектирования IEC 61499	87
4.7 Выводы	90

5. Новые шаблоны проектирования для прогнозируемого во времени исполнения функциональных блоков **93**

Matthew M. Y. Kuo u Partha S. Roop

5.1 Введение	93
5.2 Кодирование инструкций для прогнозирования во времени	96
5.2.1 Циклы	96
5.2.2 Вещественные числа	97
5.2.3 Выделение памяти	99
5.2.4 Ответвления	99
5.2.5 Прерывания	99
5.3 Шаблоны проектирования	100
5.4 Требуемая квалификация	102
5.5 Упорядоченный шаблон синхронного проектирования	104
5.6 Задержка синхронного шаблона проектирования	110
5.7 Временной анализ	114
5.7.1 Статический временной анализ	115
5.8 Выводы	117

6. Автоматический реинжиниринг управляющих приложений на основе IEC 61131 в IEC 61499 **105**

Monika Wenger, Alois Zoitl u Georg Schitter

6.1 Введение	124
6.2 Сравнение IEC 61131 и IEC 61499	124
6.3 Исследования, связанные с реинжинирингом	126
6.3.1 Существующие инструменты реинжиниринга	126
6.3.2 Миграционные подходы IEC 61131 – IEC 61499	127
6.4 Разработанный процесс реинжиниринга	129
6.4.1 Обратный инженерный процесс	131
6.4.2 Соответствие библиотеки	132
6.4.3 Миграция языка	133

6.4.3.1 Общие правила преобразования	134
6.4.3.2 Правила преобразования ROU	135
6.4.3.3 Запросы модели	137
6.5 Доказательство концепции	137
6.5.1 Структура сортировочной станции	137
6.5.2 Результат реинжиниринга	139
6.5.3 Сравнение исходного и целевого поведения	140
6.6 Выводы	145

7. Модульное тестирование для IEC 61499 функциональных блоков **151**

Reinhard Hametner, Ingo Hegny u Alois Zoitl

7.1 Введение	151
7.2 Родственная работа	152
7.2.1 Обзор тестового процесса	152
7.2.2 Поблочное тестирование	153
7.2.3 Методы тестовой спецификации	153
7.2.4 Тестирование и проверки	155
7.3 Требования к модульным испытаниям IEC 61499	156
7.3.1 Программные блоки в IEC 61499	156
7.3.2 Наблюдаемое поведение функциональных блоков	157
7.3.3 Спецификация модульного теста для функциональных блоков	158
7.4 Модульные испытания в соответствии с IEC 61499	159
7.4.1 Разделенный тестовый компонент	159
7.4.2 Тестовая спецификация, основанная на данной модели	160
7.4.3 Использование диаграмм последовательности обслуживания для спецификации теста	161
7.4.4 Оценка подходов к спецификации тестов	162
7.5 Структура тестирования на основе результатов	164
7.6 Примеры применения	166
7.6.1 Тестирование функционального блока без сохранения состояния: E_PERMIT	167
7.6.2 Тестирование функционального блока с сохранением информации: E_STU	167
7.7 Выводы и продолжение работы	169

8. Проверка приложений IEC 61499 **175**

Petr Kadera u Pavel Vrba

8.1 Введение	175

8.2 Общая проверка программного обеспечения	177
8.2.1 Динамическая проверка	177
8.2.2 Статическая проверка	178
8.2.2.1 Проверка модели	178
8.2.2.2 Доказательство теорем	179
8.3 Проверка IEC 61131-3	180
8.4 Динамическая проверка IEC 61499	181
8.4.1 Контроль и отладка	181
8.4.2 Блочные тесты	183
8.4.3 Тестирование по ключевым словам	184
8.5 Статическая проверка IEC 61499	185
8.6 Выводы	187

9. Отказоустойчивые приложения IEC 61499 **191**

Mario de Sousa

9.1 Введение	191
9.2 Предпосылки	193
9.2.1 Системная надежность	193
9.2.2 Отказы, ошибки и сбои	195
9.2.3 Достижение надежности	195
9.3 Репликация в приложениях IEC 61499	197
9.3.1 Распространение сообщения	199
9.3.2 Семантика исполнения платформы	204
9.3.3 Функциональные блоки голосования	205
9.3.4 Функциональные блоки связи	207
9.4 Платформа репликации на FORTE	208
9.4.1 Семантика выполнения среды FORTE	208
9.4.2 Функциональные блоки связи	209
9.4.3 Функциональные блоки выбора	210
9.5 Пример реплицированного применения IEC 61499	211
9.5.1 Пример приложения	211
9.5.2 Добавление репликации	212
9.5.3 Обработка выбора	214
9.6 Количественная оценка надежности системы	214
9.7 Выводы	216

10. Разработка функциональных блоков интерфейса связи IEC 61499 в приложениях распределенного управления и автоматизации **219**

Georgios Sfiris и George Hassapis

10.1 Введение	220
10.2 Программирование IEC 61499 и коммуникационная семантика	223
10.2.1 Общая семантика	223
10.2.2 Функциональный блок интерфейса службы связи	223
10.2.2.1 Инициализация CSIFB	225
10.2.2.2 Однонаправленный обмен данными	225
10.2.2.3 Двухнаправленный обмен данными	226
10.2.2.4 Деинициализация CSIFB	227
10.2.3 Профили соответствия	228
10.3 Пример распределенного приложения	229
10.4 UDP/IP и протоколы связи TCP/IP в IEC 61499	231
10.4.1 Протокол пользовательских дейтаграмм / протокол интернета	231
10.4.2 Протокол управления передачей / протокол интернета	232
10.4.3 Пример приложения UDP/IP и реализации TCP/IP	232
10.5 Пример системы SCADA	235
10.5.1 Пример применения протокола UDP/IP в системе SCADA	238
10.6 Протокол связи Modbus в IEC 61499	238
10.6.1 Реализация протокола Modbus в IEC 61499	240
10.6.2 Конфигурирование CSIFB для использования протокола Modbus	240
10.6.3 Пример приложения реализации Modbus	243
10.7 Реализации других протоколов связи в IEC 61499	245
10.7.1 Коммерческие инструменты IEC 61499	246
10.7.1.1 ISaGRAF	246
10.7.1.2 nxtSTUDIO	246
10.7.2 Открытые инструменты IEC 61499	247
10.8 Пример программирования	247

11. Адаптированная методология проектирования на IEC 61499 для распределенного управления коммуникационными устройствами

255

Carlos Catalan, Alfonso Blesa, Felix Serna и Jose Manuel Colom

11.1 Краткая мотивация	256
11.2 Введение	257
11.3 Программное обеспечение системы управления для AMS: стандарт IEC 61499	258
11.4 Соединение устройств с IEC 61499: сфера применения платформы COSME	260
11.5 Цели проектирования платформы COSME	261
11.5.1 Как облегчить проектирование приложений	261
11.5.2 Соответствие принципам распределенных систем	262

11.5.3 Объединение функций с добавленной стоимостью	265
11.6 Модель COSME FB	265
11.7 Архитектура платформы COSME	267
11.8 Процесс проектирования COSME	270
11.9 Проблемы реализации	272
11.10 Выводы	273

III. Примеры промышленной реализации 277

12. Гибкое и допускающее повторное использование приложение для управления производственным процессом 279

Gemot Kollegger и Arnold Kopitar

12.1 Введение	280
12.2 Ожидания решений по автоматизации на основе IEC 61499	281
12.3 Требования к приложениям на основе IEC 61499	283
12.3.1 Требования к приложению автоматизации	283
12.3.2 Реальные компоненты как компоненты программного обеспечения	283
12.4 Мощность атрибутов	284
12.5 САТ: составной тип автоматизации	285
12.5.1 Вложенные компоненты программного обеспечения	285
12.5.2 Подключение аппаратных приложений к I/O	286
12.6 Приложение для управления процессом	288
12.7 Понятие и компоненты библиотек управления процессом	288
12.7.1 Основные важные компоненты	289
12.7.2 Общие режимы работы и режим САТ	289
12.7.3 Усовершенствованные компоненты	290
12.7.4 Внутренняя структура компонента	291
12.8 Применение конвейерных лент	298
12.8.1 Определение и использование адаптера IBeltDrive	299
12.8.2 Экземпляры САТ транспортной линии	300
12.8.3 Отображение приложения	300
12.9 Распределенный подход управления последовательностью	301
12.9.1 Функциональные блоки DSC	302
12.9.2 Отображение приложения DSC	306
12.9.3 Визуализация приложений DSC	306
12.9.4 Приложения DSC в данном проекте	306
12.10 Дозирование и реакторное приложение	307
12.11 Приложение для конфигурации и мониторинга оборудования	307
12.12 Выводы	311

13. Разработка системы автоматизации **313**

Gernot Kollegger u Arnold Kopitar

13.1 Введение	313
13.2 Требования к приложениям управления зданием	314
13.2.1 Как скрыть сложность с интерфейсами адаптера	315
13.2.2 Масштабируемые приложения	316
13.3 Система управления	317
13.3.1 Описание работы	317
13.3.2 Создание приложения для управления и приложения для мониторинга	317
13.3.2.1 Приложения для управления помещением	318
13.3.2.2 Масштабируемость сегментов помещения	322
13.3.3 Приложения HVAC и управления поставками	322
13.3.3.1 Архивация исторических данных и аварий	325
13.3.4 Аппаратное подключение в рамках проекта	325
13.4 Выводы	327

14. Программное обеспечение для управления станком для резки стекла, созданным с использованием платформы COSME: тематическое исследование **329**

Felix Serna, Carlos Catalan, Alfonso Blesa, Jose Manuel Colom u Josep Maria Rams

14.1 Введение	330
14.2 Проектирование на основе IEC 61499 в зависимости от предметной области	331
14.3 Моделирование процесса обработки стекла	335
14.3.1 Определенная функциональность	337
14.3.2 Универсальная функциональность	338
14.3.3 FB-дизайн и проектировочные шаблоны	339
14.3.4 Детальное проектирование FBN: сеть EtherCAT	343
14.4 Реализация	344
14.4.1 Платформа COSME	344
14.4.2 IDE COSME	346
14.4.3 Связь с внешними приложениями	349
14.5 Практические проблемы и выводы	352

15. Распределенное интеллектуальное зондирование и управление для автоматизации производства **357**

15.1 Введение	357
15.2 Похожие проблемы	359
15.2.1 Реконфигурирование распределенных систем	359
15.2.2 Интеллектуальное промежуточное программное обеспечение	360
15.3 Архитектура DISCS	363
15.4 Реализация функционального блока	366
15.5 Пример: мобильное объектное отслеживание	367
15.6 Дальнейшая работа	369

16. Управляемая моделями конструкция кардиостимулятора с использованием функциональных блоков IEC 61499	375

Yu Zhao и Partha S. Roop

16.1 Введение	375
16.2 Коротко о системе стимуляции	376
16.2.1 Основы электрофизиологии и ее моделирования	377
16.2.2 Требования к времени и устройству кардиостимулятора	379
16.2.3 Связанная с темой работа	381
16.3 Обзор предложенного подхода	382
16.4 Моделирование с использованием функциональных блоков IEC 61499	384
16.4.1 Модель логики стимуляции в режиме DDD	384
16.4.2 Иерархическая структура модели контроллера кардиостимулятора	390
16.5 Создание высококачественной модели	392
16.5.1 Временной анализ синхронных функциональных блоков	393
16.5.2 Высокочастотные модели контроллера кардиостимулятора	394
16.5.3 Создание модели среды для системы замкнутого цикла	398
16.6 Анализ времени отклика с использованием высокочастотных моделей	399
16.7 Выводы	402

17. Применение Smart Grid через экономическую диспетчеризацию с использованием IEC 61499	405

Srikrishnan Jagannathan и Peter Idowu

17.1 Введение	406
17.1.1 Функции интеллектуальной сети	407
17.1.2 Мотивация для исследования	408

17.1.3 Вклад	409
17.1.4 Улучшения, достигнутые благодаря внедрению IEC 61499	409
17.2 Ключевые понятия	410
17.2.1 Экономическая часть	410
17.2.2 Прогнозирование нагрузки	411
17.2.3 Ранжируемая стоимость энергии	411
17.2.3.1 Факторы, влияющие на LCOE	412
17.2.3.2 Вычисления LCOE	414
17.3 Программные инструменты	415
17.4 Разработка приложений	415
17.4.1 Описание сети энергосистемы	415
17.4.2 Этап 1 – нормальный режим работы модулятора	417
17.4.3 Этап 2 – экономическая часть единого микромодулятора	417
17.4.3.1 Список функциональных блоков, используемых на этапе 2	418
17.4.4 Этап 3 – Межмикрогридная экономическая диспетчеризация	419
17.4.4.1 Математическая формулировка	420
17.4.4.2 Список функциональных блоков, используемых на этапе 3	423
17.5 Совместное моделирование MATLAB® и FBDK	424
17.6 Результаты моделирования, выводы и дальнейшая работа	427
17.6.1 Результаты симуляции для выходных мощностей	428
17.6.2 Ежедневная стоимость генерации для различных этапов	332
17.6.3 Выводы	434
17.6.4 Дальнейшая работа	437

IV. Примеры лабораторной автоматизации **439**

18. Роботы для создания общего рабочего пространства: применение IEC 61499 **441**

*Matthias Plasch, Gerhard Ebenhofer, Michael Hofmann, Martijn Rooker,
Sharath Chandra Akkaladevi u Andreas Pichler*

18.1 Введение	442
18.2 Аналогичные работы	443
18.2.1 Платформы интеграции	443
18.2.2 Интегрированные технологии	445
18.3 Описание автоматизированной системы	446
18.3.1 Расположение оборудования	446
18.3.2 Эксперименты и демонстрационный вариант использования	447
18.3.3 Необходимые функциональные компоненты системы	448
18.3.3.1 Система 3D Vision	448
18.3.3.2 Планировщик манипуляций: Планирование путей без столкновений	449

18.3.3.3 Модуль для управления роботом	449
18.4 Подход к разработке	449
18.4.1 Интеграция неоднородных компонентов программного обеспечения	450
18.4.2 Поточковые типы функционального блока	451
18.4.2.1 Распределение сервисов	452
18.4.2.2 Сервисный контроллер с ленивым шаблоном подключения	454
18.4.3 Шаблоны проектирования приложений	455
18.4.3.1 Гибкий сервисный шаблон параметризации	455
18.4.3.2 Сценарии LUA FB для быстрого прототипирования алгоритма и координатии FB	456
18.5 Полученная архитектура системы	461
18.6 Сводка и выводы	465

19. Иерархически структурированное управляющее приложение для станции сбора и размещения **469**

Monika Wenger, Milan Vathoopan, Alois Zoitl u Herbert Prahofe

19.1 Введение	470
19.2 Принципы для иерархических структурированных управляющих приложений	470
19.2.1 Многоуровневая структура	471
19.2.2 Разделение на уровни с IEC 61499	472
19.3 Структура станции сбора и размещения	473
19.4 Приложение иерархического управления	475
19.4.1 Уровень 0: атомарные компоненты	475
19.4.2 Уровень 1: элементарные компоненты	476
19.4.2.1 Компоненты, связанные с манипулятором	477
19.4.2.2 Компоненты, связанные с конвейером	479
19.4.2.3 Компонент индикатора состояния	481
19.4.3 Уровень 2: координирование компонентов	481
19.4.3.1 Слайд	481
19.4.3.2 Манипулятор	482
19.4.3.3 Конвейер	482
19.4.3.4 Управление состоянием	484
19.4.4 Уровень 3: компонент управления процессом	485
19.5 Выводы	486

20. На пути к пакетной обработке с IEC 61499 **489**

Wilfried Lepuschitz u Alois Zoitl

20.1 Введение	489

20.2 Пакетное управление ANSI/ISA-88	490
20.2.1 Структурные модели	491
20.2.2 Конечный автомат	492
20.2.3 Поправки и область интересов	493
20.3 Подход 1: иерархическая структура на основе автоматизации компонентов	494
20.3.1 Концепция подхода	494
20.3.2 Реализация с IEC 61499	496
20.4 Подход 2: реализация конечного автомата S88	499
20.4.1 Концепция подхода	499
20.4.2 Реализация с IEC 61499	500
20.5 Подход 3: общая структура фаз S88 и соединение с промышленной системой для управления партиями	502
20.5.1 Концепция подхода	503
20.5.2 Реализация с IEC 61499	505
20.6 Выводы	506

21. Лабораторный подход к автоматизации интеллектуальной сети с использованием IEC 61499	509

*Filip Andren, Georg Lauss, Roland Brundlinger, Philipp Svec,
Christian Seidl u Thomas Strasser*

21.1 Введение и мотивация	510
21.2 Лаборатории интеллектуальной сети: потребности и требования	511
21.3 Краткий обзор лабораторной среды SmartEST	512
21.3.1 Цель лаборатории и ее использование	512
21.3.2 Варианты и сценарии лабораторного использования	514
21.3.3 Требования для лабораторной системы автоматизации	515
21.4 Лабораторная система автоматизации на базе IEC 61499	516
21.4.1 Архитектура и концепция	516
21.4.2 Прикладная модель автоматизации на основе IEC 61499	518
21.4.2.1 Конфигурация системы	518
21.4.2.2 Система управления	519
21.4.3 Среда автоматизации с открытым исходным кодом	525
21.5 Сводка и выводы	526

Предисловие

Автоматизация производства и системы управления на производстве, в энергетике и энергосистемах, а также в секторе логистики становятся все более сложной. Обычно в таких системах находится огромное количество приводов и датчиков, которые, взаимодействуя, образуют сеть различных типов устройств. Последние научно-исследовательские тенденции показывают, что эти устройства становятся все более интеллектуальными и сетевыми. Это означает, что они могут выполнять задачи автономно и непосредственно взаимодействовать друг с другом.

Для понимания сложности таких весьма взаимосвязанных и аналогичных устройств нужны передовые методы и понятия, которые относятся к целому жизненному циклу сетевого устройства (планирование, разработка, функционирование, обслуживание), потому что данные услуги по распределенному управлению должны обеспечить усовершенствованные функции, которые включают реконfigurирование компонентов программного обеспечения, выполнение операций в реальном времени (инициирование временем, основание на событии), диагностику и другие сервисы. Эталонная архитектура IEC 61499 была разработана для улучшения разработки и обслуживания таких систем автоматизации, а также для лучшего выполнения вышеуказанных требований.

Основная цель этой книги состоит в том, чтобы рассмотреть использование эталонной архитектуры IEC 61499 для распределенного и реконfigurируемого управления и ее принятия промышленностью. Это предоставляет шаблоны проектирования, руководящие принципы работы приложений и служит руководством для разработки приложений распределенного управления на основе эталонной модели IEC 61499. Кроме того, представлены и подробно рассмотрены выбранные примеры, полученные от различных промышленных доменов, а также от лабораторных сред. Имея такое содержание, эта книга дает хороший обзор доменов приложения IEC 61499 и помогает применить модели IEC 61 499 для реализации текущих решений по автоматизации.

MATLAB® является зарегистрированной торговой маркой MathWorks, Inc.

Для получения информации о продукте свяжитесь с MathWorks, Inc.

3 Apple Hill Drive

Натик, Массачусетс, 01760-2098 США

Тел.: 508-647-7000

Факс: 508-647-7001

Электронная почта: info@mathworks.com

Сайт: www.mathworks.com

Редакторы

Alois Zoitl получил степень доктора философии в области электротехники по направлению динамического реконфигурирования ограниченных приложений управления в реальном времени и степень магистра в области электротехники по направлению распределенных систем автоматизации производства в Венском технологическом институте. В настоящее время он возглавляет Исследовательскую группу по промышленной автоматизации в Мюнхене. Ранее он возглавлял Распределенную интеллектуальную группу по автоматизации (Лаборатория Odo Struger) в Институте автоматизации и управления Венского технического университета. Он также работает лектором в Техническом университете Мюнхена.

Доктор Zoitl является соавтором более 100 публикаций (книги, книжные главы, труды конференций и статьи в журналах) и сообладателем 4 патентов в его областях исследования. В ходе его исследований он руководит несколькими финансируемыми промышленностью проектами, координирует и участвует во многих публично финансируемых австрийских и европейских проектах. Он член-учредитель 4DIAC и инициативы открытого исходного кода OpENeg, член IEEE и пользовательской организации PLCopen, консультант CAN по автоматизации и член IEC SC65B/WG15 по стандарту автоматизации IEC 61499. В мае 2015 года он выступил как организатор собраний группы.

Thomas Strasser получил степень доктора философии в области машиностроения по теории автоматизации и управления и степень магистра в области промышленной инженерии по направлению систем производства и автоматизации в Венском техническом университете.

В течение нескольких лет он был старшим научным сотрудником в Министерстве энергетики австрийского Технологического института (AIT). Его основные обязанности включают стратегическую разработку научно-исследовательских работ интеллектуальной сети и воспитание квалифицированных младших научных сотрудников и докторов философии. Перед переходом в AIT доктор Strasser работал более 6 лет в качестве старшего научного сотрудника над усовершенствованием и реконфигурированием автоматизации и системы управления в PROFACTOR. Он также является лектором в Венском техническом университете и преподавателем в Университете прикладных наук в Зальцбурге. Доктор Strasser написал в соавторстве более 120 научных работ (передовые статьи, книжные главы, труды конференций и статьи в журналах) и является держателем 2 патентов в его областях исследования. Он активный участник конференций по IEEE, работает младшим редактором журналов IEEE и Springer. Также он старший член IEEE и член-учредитель 4DIAC, инициативы с открытым исходным кодом, и разработчик IEC SC65B/WG15, IEC TC65/WG17 и IEC SyC Smart Energy/WG6.

Список благодарностей

Мы хотели бы поблагодарить всех поучаствовавших авторов, особенно авторов от промышленных консорциумов, чья повседневная работа не связана с написанием глав в книгах.

Мы также хотим благодарить Nora Konopka, издателя этой книги; Kyrä Lindholm, Michele Smith, Jessica Vakili, Florence Kizza и Karen Simon из CRC Press Taylor & Francis.

Кроме того, мы хотели бы выразить нашу особую благодарность редактору серии, доктору Richard Zurawski, за приглашение создать эту книгу и его поддержку.

Участники

Sharath Chandra Akkaladevi
PROFACTOR GmbH, Штайр,
Австрия

Filip Andren
AIT GmbH, Вена, Австрия

Alfonso Blesa
Университет Сарагосы, Сарагоса,
Испания

Robert W. Brennan
Университет Калгари, Калгари,
Канада

Roland Brundlinger
AIT GmbH, Вена, Австрия

Carlos Catalan
Университет Сарагосы, Сарагоса,
Испания

James H. Christensen
Holobloc Inc.
Cleveland-Heights, Огайо

Jose Manuel Colom
Университет Сарагосы, Сарагоса,
Испания

Wenbin Dai
Шанхайский университет Цзяотун,
Шанхай, Китай

Gerhard Ebenhofer
PROFACTOR GmbH, Штайр,
Австрия

Reinhard Hametner
Thales Austria GmbH Вена, Австрия

George Hassapis

Университет Аристотеля в
Салониках, Салоники, Греция

Ingo Hegny
Венский технический университет,
Вена, Австрия

Michael Hofmann
PROFACTOR GmbH, Штайр,
Австрия

Peter Idowu
Университет штата Пенсильвания,
Мидлтаун, Пенсильвания

Srikrishnan Jagannathan
Университет штата Пенсильвания,
Мидлтаун, Пенсильвания

Petr Kadera
Чешский технический университет
в Праге, Прага, Чешская
Республика

Gernot Kollegger
nxtControl GmbH, Леоберсдорф,
Австрия

Arnold Kopitar
nxtControl GmbH, Леоберсдорф,
Австрия

Matthew M. Y. Kuo
Оклендский университет, Окленд,
Новая Зеландия

Georg Lauss
AIT GmbH, Вена, Австрия

Wilfried Lepuschitz

Практический институт
робототехники Австрии, Вена,
Австрия

Andreas Pichler
PROFACTOR GmbH, Штайр,
Австрия

Matthias Plasch
PROFACTOR GmbH, Штайр,
Австрия

Herbert Prahofner
Линцкий университет имени
Иоганна Кеплера, Линц, Австрия

Josep Maria Rams
Группа TUROMAS, Теруэль,
Испания

Martijn Rooker
PROFACTOR GmbH Штайр,
Австрия

Partha S. Roop
Оклендский университет, Окленд,
Новая Зеландия

Georg Schitter
Венский технический университет,
Вена, Австрия

Christian Seidl
AIT GmbH, Вена, Австрия

Felix Serna
Университет Сарагосы, Сарагоса,
Испания

Georgios Sfiris
Университет Аристотеля в
Салониках, Салоники, Греция

Mario de Sousa

Университет Порту, Порту,
Португалия

Thomas Strasser
AIT GmbH, Вена, Австрия

Philipp Svec
AIT GmbH, Вена, Австрия

Milan Vathoopan
Munchen fortiss GmbH, Мюнхен,
Германия

Pavel Vrba
Чешский технический университет
в Праге, Прага, Чешская
Республика

Valeriy Vyatkin
Технологический университет
Лулео, Лулео; Университет Аалто
Швеции, Хельсинки, Финляндия

Monika Wenger
Munchen fortiss GmbH, Мюнхен,
Германия

Yu Zhao
Оклендский университет, Окленд,
Новая Зеландия

Alois Zoitl
Munchen fortiss GmbH, Мюнхен,
Германия

Часть I

Основы ИЕС 61499

Проблемы и требования для распределенной автоматизации в промышленной среде

Thomas Strasser
AIT GmbH

Alois Zoitl
fortiss GmbH

СОДЕРЖАНИЕ

1.1 Тенденции в автоматизации производства	27
1.2 Требования к будущей архитектуре автоматизации	29
1.3 Перспективы	31

1.1. Тенденции в автоматизации производства

Домен автоматизации производства является ключевым фактором и опорой для других отраслей промышленности. Благодаря своим инновациям его можно рассматривать как основу для многих отраслей промышленности, таких как производство, технологические процессы, энергетика и энергетические системы. Так как производственные процессы все чаще выполняются автоматизированными машинами, фабриками и заводами, увеличенный уровень автоматизации замечен уже сегодня. Главной тенденцией и ведущим фактором для автоматизации производства в последние годы является возрастающая потребность для специализированных и индивидуализированных продуктов и товаров. Существует потребность развивать соответствующие решения по автоматизации в ответ на быстро изменяющиеся требования из-за новых производственных процессов и технологий. Также недавние успехи в аппаратных и программных решениях влияют на эту тенденцию. Это требует, чтобы поточные линии, машины и компоненты были созданы и быстро адаптированы к новым продуктам и производственным процессам [3, 10].

Таковыми высоко автоматизированными системами главным образом управляет обширный набор встроенных аппаратных и программных компонентов, которые неоднородны по своей природе. Как правило,

соответствующие спецификации и архитектура приводят к огромному увеличению сложности проектирования. В результате этой тенденции в ближайшие 15 лет можно ожидать увеличения расходов на разработку программного обеспечения примерно с 50% до 80% от общих расходов [11]. Однако более поздние исследования также показывают, что инженерные усилия могут быть сокращены до 70% за счет оптимизации всего инженерного процесса с использованием надлежащих методов, архитектур и соответствующих инструментов [4]. Такой результат оптимизации на самом деле не может быть достигнут для разработки программного обеспечения для управления, что является все еще ключевой проблемой. Еще одна особенность производства, а также области энергетических и энергетических систем заключается в том, что каждая установка или система может рассматриваться как прототип. Следовательно, технические затраты являются все еще ключевыми факторами стоимости для вышеупомянутых процессов и систем.

Прикладные методы разработки и подходы сильно зависят от определенной области применения. Хорошо известны следующие подходы к автоматизированным системам управления производством (АСУП) компьютеризированное числовое управление (ЧПУ), управление роботами, программируемая логическая интегральная схема (ПЛИС), распределенная система управления (PCU) и диспетчерское управление и сбор данных (SCADA). Также существуют другие подходы, такие как программируемая пользователем вентильная матрица (FPGA). Эти базирующиеся контроллеры становятся популярными. Такие подходы часто используются вместе для выполнения задач автоматизации и контроля конкретного приложения [1]. Проектирование и разработка таких неоднородных систем является сложной задачей, которая требует знаний из разных областей на всех этапах разработки. Для разработки решений промышленной автоматизации и управления наиболее важными факторами являются сложность системы, зависимость от домена или платформы, а также время проектирования. Чтобы держать сложность системы под контролем, должны быть удовлетворены следующие ключевые требования и потребности [9]:

- Спецификации системы должны быть подготовлены на легко понятном уровне. Средства спецификации должны поддерживать проектировку систем промышленной автоматизации и управления на каждом уровне архитектуры предприятия, чтобы определять желаемую функциональность и избегать ее реализации на уровне кода.

- Различные проблемно-ориентированные методы и подходы должны быть легко объединяемы, с преодолением потенциальных ограничений каждой процедуры. Это позволяет моделировать, анализировать и реализовывать сложные, неоднородные системы, способствующие автоматизации производства и формированию системы управления.

- Методы и инструменты для определения системной архитектуры не должны зависеть от общих языков, используемых для проектирования встроенной системы.

Для решения проблем и соблюдения требований автоматизации производства и систем управления ключевая тенденция проектирования должна поместить компоненты — блоки, сделанные из аппаратных средств, программного обеспечения и интеллектуальной собственности (как алгоритмы и структуры данных) — вместе [15, 16] в один допускающий повторное использование компонент. Это требует общих технических артефактов, независимых от языка моделей для представления, сохранения и многократного использования таких компонентов. Сегодня современные подходы, методы и соответствующие инструменты для разработки технических систем автоматизации производства не полностью способны к предоставлению пригодных решений для вышеупомянутых проблем [1, 15, 16]. Решения, разработанные для других областей, таких как автомобилестроение или аэрокосмическая промышленность, ориентированы на конкретные потребности и требования их собственных секторов. Они отличаются от требований промышленной автоматизации. Более того, большинство из этих решений требуют глубоких знаний в области разработки программного обеспечения. Однако в области промышленной автоматизации инженеры в области электроники и управляющие обычно проектируют системы управления. Данные эксперты могут иметь только базовое образование в области разработки программного обеспечения. Поскольку разработка программного обеспечения становится все более и более важной для промышленной автоматизации, поэтому необходимо разрабатывать методы и подходы, которые позволяют также и экспертам, не занимающимся разработкой программного обеспечения, эффективно и результативно разрабатывать программное обеспечение системы управления.

1.2. Требования для будущей архитектуры систем автоматизации

Вышеупомянутые конкретные тенденции демонстрируют потребность в более гибкой и адаптируемой будущей архитектуре системы автоматизации и в соответствующих инструментах, выполняющих требования различных проектов и клиентов. Как определено в [14], распределенные архитектуры и динамическая реконфигурация являются ключевыми факторами, поддерживающими адаптивность сред автоматизации. Чтобы сделать эту концепцию применимой, будущая архитектура автоматизации должна удовлетворять следующим ключевым требованиям.

Распределенный интеллект имеет потенциал для того, чтобы обеспечить возможность достижения управляемости и надежности больших и сложных промышленных процессов и машин.

Удобство использования является важным требованием для архитектуры автоматизации. Она должна быть простой в использовании, понятной и обслуживаемой, как современные системы на основе PLC. В то же время

дизайн должен позволять более выразительным образом разрабатывать и поддерживать их.

Проблемно-ориентированные языки моделирования являются полезными инструментами, разрешающими специалистам в интересующей их области, имея небольшие знания разработки программного обеспечения, описать задачи автоматизации на более высоком уровне абстракции. Это гарантирует возможность многократного использования, снижает трудозатраты и повышает качество программного обеспечения.

Простая конфигурация связи необходима для определения взаимодействия, а также обмена данными большого количества распределенных (встроенных) устройств управления.

Предсказуемость распределенных приложений (с точки зрения поведения синхронизации и потока выполнения) должна быть такой же простой для понимания, как и хорошо известное поведение циклического выполнения, обычно реализуемое в системах PLC.

Гетерогенное выполнение требуется, чтобы удовлетворить различные потребности прикладных задач. Выполнение по времени с джиттером ниже определенной границы (особенно для выполнения алгоритмов управления с обратной связью) и выполнение на основе событий в указанном временном промежутке должно поддерживаться в одном и том же контексте выполнения или в одной среде. Следовательно, функции не должны мешать друг другу, и им, возможно, придется взаимодействовать определенным образом.

Выполнение в реальном времени с гарантиями необходимо, поскольку системы промышленной автоматизации и управления взаимодействуют с реальными процессами, машинами и компонентами. В случае динамической реконфигурации временные ограничения также могут быть выполнены детерминистически.

Реконфигурирование в реальном времени с гарантиями позволяет изменять функциональность управляющих приложений и систем при полной работе. Это требуется, чтобы процесс реконфигурации не приводил к наблюдаемым помехам приложения автоматизации. Требуется переключение или перемещение приложений полного контроля наряду с тонкой детальной адаптацией частей приложения. Среда должна поддерживать реконфигурацию определенных событий или состояний системы промышленной автоматизации, а также переключение в фиксированные моменты времени. Должны быть предоставлены гарантии того, что процессы реконфигурации и переключения не нарушают ограничения выполнения запущенного приложения. Как правило, эти процессы также должны выполняться в пределах доступных ресурсов (например, памяти, вычислительной мощности, пропускной способности сети).

Долговечность систем промышленной автоматизации и управления должна соответствовать сроку службы системы, исчисляемому в десятилетиях. Даже если производимая продукция является менее долговечной, производственная система также может использоваться для широкого спектра другой продукции. Это требует долгосрочной восходящей и нисходящей совместимости для замены компонентов и устройств.

Соответствие стандартам требуется для обеспечения функциональной совместимости компонентов от разных поставщиков, это позволит совмещать любые инструменты, соответствующие стандартам, и обеспечит переносимость на уровне источника для повторного использования программного обеспечения. Тем не менее переносимость на двоичном уровне для миграции программных компонентов между устройствами управления может быть необходимой в качестве источника.

Форм-фактор, цена и надежность распределенных устройств промышленной автоматизации должны быть такими, чтобы устройства можно было устанавливать непосредственно на электромеханические компоненты.

1.3. Перспективы

В настоящее время область промышленной автоматизации и управления подвержена серьезным технологическим изменениям. Одним из ключевых факторов, лежащих в основе этой тенденции, являются киберфизические системы (CPS) [12]. CPS являются расширениями физического объекта встроенных систем. Это приводит к ключевому аспекту CPS: они могут быть связаны и взаимодействовать для задач, для которых они не предназначены.

Например, в Германии концепция CPS была основой для Инициативы «Индустрия 4.0» [2]. Эта инициатива заключается в том, что использование CPS в производственных системах приведет к Четвертой промышленной революции. В индустрии 4.0 предусмотрены следующие цели и ожидания.

- Способность произвести отдельные продукты согласно потребительскому спросу.
- Увеличенная гибкость производственной системы.
- Оптимизированное принятие решений.
- Большая эффективность ресурса и производительность.
- Расширенные бизнес-модели на основе новых сервисов, включающих полную стоимостную цепочку (то есть от поставщиков клиентам).
- Улучшение баланса между работой и личной жизнью и создание более безопасных рабочих мест с учетом демографических изменений рабочей силы, что позволяет работникам дольше оставаться трудоспособными.
- Переиндустриализация Европы.

В США, Китае, Корее и Японии подобные попытки уже предприняты, и их результат обозначается термином *умное производство*. Следующей технологией для эволюции автоматизации производства и систем управления является Интернет вещей (IoT) [7]. Цели IoT подключают все – от машин до продуктов и материалов – к Интернету и при необходимости также к облаку.

Подобные изменения также заметны в домене энергосистем. Из-за интеграции распределенных, возобновляемых источников, улучшений энергоэффективности и отмены госконтроля рынка изменения в планировании и работе энергосистем необходимы для гарантии устойчивой системы. Эта тенденция влияет также на стратегии автоматизации и управления,

используемые в области энергетики. Как и в Industry 4.0, с использованием технологий CPS и IoT становится возможным переход к так называемым интеллектуальным сетям, позволяющим лучше контролировать сложные электрические сети [5, 6, 8].

Резюмируя, эти тенденции показывают, что будущие системы промышленной автоматизации и управления все чаще состоят из интеллектуальных, взаимосвязанных устройств, использующих различные сетевые подходы. Такие будущие среды автоматизации также характеризуются большими объемами использования программного обеспечения. Как указано выше, разработка программного обеспечения управления уже является одним из ключевых факторов стоимости для разработки и развертывания автоматизации производства и систем управления. Для сокращения таких затрат, ведётся поиск новых подходов и соответствующих инженерных инструментов. IEC 61499 как эталонная модель для высоко распределенной, сетевой автоматизации производства и систем управления обеспечивает основание для реализации будущей распределенной архитектуры [13]. У этого есть потенциал для удовлетворения большинства вышеупомянутых потребностей.

В оставшейся части этой книги использование эталонной архитектуры IEC 61499 обсуждается путем введения соответствующих шаблонов проектирования, руководящих принципов и правил применения и реализованных примеров из различных промышленных областей и лабораторных сред.

Библиография

- [1] Roadmap on real-time techniques in control system implementation – control for embedded systems cluster, EU/IST FP6 Artist2 NoE, www.artist-embedded.org, 2006.
- [2] Arbeitskreis Industrie 4.0. Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0. Forschungsunion im Stifterverband für die Deutsche Wirtschaft e.V., Berlin, Oct. 2012.
- [3] Artemis Strategic Research Agenda, 2005.
- [4] M. Buchwitz. Neue Wege in der Software Entwicklung. *SPS Magazin*, 2012.
- [5] EC. European SmartGrids technology platform: vision and strategy for Europe's electricity networks of the future. Technical report, European Commission (EC): Directorate-General for Research-Sustainable Energy Systems, 2006.
- [6] H. Farhangi. The path of the smart grid. *IEEE Power and Energy Magazine*, Vol. 8, No. 1, 18–28, 2010.
- [7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, Vol. 29, No. 7, 1645–1660, 2013.
- [8] IEA. Smart grid insights. Technical report, International Energy Agency, 2011.

- [9] M. Jersak, K. Richter, and R. Ernst. Combining complex event models and timing constraints. In Proceedings of Sixth IEEE International High-Level Design Validation and Test Workshop, pages 89–94, 2001.
- [10] Y. Koren. The Global Manufacturing Revolution. 2010.
- [11] S. Kuppinger. Die Schlüssel zur Effizienz - Integration von Motion und Logic. IEE (Industrie elektrik+elektronik), Vol. 51, No. 1, 2006.
- [12] E. A. Lee and S. A. Seshia. Introduction to Embedded Systems, A Cyber-Physical Systems Approach. <http://LeeSeshia.org>, second edition edition, 2015.
- [13] TC 65/SC 65B. IEC 61499: Function blocks. International Electrotechnical Commission, Geneva, 2nd ed., 2012.
- [14] I. Terzic, A. Zoitl, B. Favre-Bulle, and T. Strasser. A survey of distributed intelligence in automation in european industry, research and market. In Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation, pages 221–228, Sept. 2008.
- [15] T. Tommila, J. Hirvonen, L. Jaakkola, J. Peltoniemi, J. Peltola, S. Sierla, and K. Koskinen. Next generation of industrial automation – Concepts and architecture of a component-based control system. VTT Research Notes 2303, 2005.
- [16] D.Woll. Setting the Stage for the Next Generation of Automation Control System Software: IEC 61499. ARC Advisory Group on Industry Trends, 2007.

2

Основные принципы международного стандарта IEC 61499

Thomas Strasser
AIT GmbH

Alois Zoitl
Fortiss GmbH

СОДЕРЖАНИЕ

2.1 Введение	36
2.2 Эталонная модель IEC 61499	37
2.2.1 История	37
2.2.2 Обзор стандарта	38
2.2.3 Модель выполнения на основе событий	39
2.2.4 Функциональные блоки как основные элементы моделирования	40
2.2.5 Прикладная и системная модель. Модели устройства и ресурса	42
2.2.6 Модель распределения	43
2.2.7 Модель управления	44
2.2.8 Модель рабочего состояния	45
2.3 Основные отличия между первой и второй моделями	46

2.1. Введение

За последние десятилетия технология автоматизации прошла через несколько эволюционных этапов. Централизованное управление на основе IEC (Международная электротехническая комиссия, Женева, Швейцария) 61131 было разработано для программируемых логических контроллеров (PLC) с одним или более тесно связанными процессорами [5]. Третья часть IEC 61131 предоставляет пять известных языков программирования (текстовый и графический) для PLC [3]. Однако этот стандарт не был разработан для распределенных систем автоматизации и управления, хотя в части 5 IEC 61131 приведены определения функциональных блоков связи (FB) для обеспечения взаимодействия между различными PLC. Спецификации части 5 недостаточны для обеспечения совместимости PLC разных поставщиков. Кроме того, IEC 61131 недостаточно поддерживает распространение кода управления для распределенных устройств, и разработка в большей степени ориентирована на устройства, чем на приложения. Однако подход IEC 61131, особенно языки программирования, определенные в части 3, широко принят и широко используется в промышленности для определения управляющего кода и алгоритмов.

Благодаря внедрению технологии fieldbus в начале 1980-х годов, тенденция распределенного интеллекта эволюционировала от децентрализованного подхода к автоматизации. Децентрализованное управление значительно уменьшило затраты на проводку систем автоматизации, но никаких существенных изменений в общей архитектуре управляющих программ и PLC разработано не было. Распределенный интеллект в автоматизации идет еще дальше, оснащая "неинтеллектуальные" сетевые периферийные устройства управляющим программным обеспечением [6]. Это помогает определить новые архитектуры без центральных блоков, поскольку все задачи управления выполняются в интеллектуальных блоках.

Программные конструкции современных подходов к PLC, основанные на IEC 61131-3, слишком монолитны и закрыты для распределенных приложений. Основным ограничением существующих систем является отсутствие поддержки для построения управляющего программного обеспечения из независимых компонентов, которые можно добавлять и удалять во время выполнения операций. В связи с этим необходимо разработать новые способы программирования и моделирования систем управления. Поэтому был разработан стандарт автоматизации IEC 61499 «Функциональные блоки» [4], введенный Техническим комитетом IEC (TC) 65. Он предоставляет собой эталонную модель для распределенных и реконфигурируемых систем автоматизации. Моделирование открытых и распределенных промышленных процессов, систем измерения и управления (IPMCS) также находится в центре внимания этого стандарта. Цель состоит в том, чтобы получить независимое от поставщика приложение для автоматизации и описание конфигурации оборудования для управления

усложнением систем автоматизации, в числе систем следующего поколения. Цели наивысшего уровня этого стандарта можно суммировать как переносимость проектов автоматизации, а также конфигурируемость и совместимость интеллектуальных полевых устройств в приложениях IPMCS. Это служит эталонной архитектурой для распределенных, модульных и гибких систем управления, отвечающих фундаментальным требованиям открытых распределенных систем [7, 8]. Следующие основные параметры кратко характеризуют этот распределенный стандарт автоматизации [8].

- Компонентные или объектно-ориентированные базовые строительные блоки, называемые FB.

- Графический интуитивно понятный способ моделирования алгоритмов управления через соединение входов и выходов FB.

- Модель исполнения на основе событий.

- Прямая поддержка для распределения.

- Точность для взаимодействия между устройствами разных производителей.

- Основная поддержка реконфигурирования

- Правила определения и переносимости приложений автоматизации между программными инструментами

- Основанность на существующих доменных стандартах (IEC 61131 и т. д.)

Основная цель этой главы – дать краткий обзор основных элементов стандарта IEC 61499, которые описаны в разделе 2.2. В разделе 2.3 обсуждаются основные различия между первым и вторым изданиями стандарта, которые необходимо учитывать при реализации приложений автоматизации и управления.

2.2. Международный стандарт IEC 61499

2.2.1. История

Изначально IEC 61499 был опубликован в 2000–2002 годах как серия общедоступных спецификаций (PASs) для пробного использования вместе с техническим отчетом (TR), содержащим учебную информацию. Большинство тех же экспертов от промышленности, консалтинга и академии, которые создали IEC 61131-3, также разработали эталонную модель IEC 61499. С тех пор модель подвергалась непрерывному совершенствованию в результате обширного тестирования в научных и промышленных лабораториях. В результате этих разработок в 2005 году был опубликован IEC 61499 как международный стандарт в трех основных частях. Впоследствии работа по обновлению продолжилась, что привело к улучшению трех частей стандарта и способствовало его популяризации. В 2012 году был опубликован второй выпуск IEC 61499.

2.2.2. Обзор стандарта

Стандарт автоматизации IEC 61499 охватывает широкий спектр видов деятельности, включая спецификацию различных элементов моделирования для IPMCS, их анализ и валидацию. Конфигурация устройств и их обслуживание также учтены. IEC 61499 состоит из следующих трех частей [4, 8]:

Часть 1 — Архитектура определяет основные элементы эталонной модели, обращаясь к следующим, важным моментам.

- Общие требования, область применения, определения, элементы моделирования.
- Правила объявления типов FB и правила выполнения режима экземпляров FB.
- Правила для использования FB для моделирования приложений управления распределенного IPMCS
- Правила моделирования конфигураций IPMCS (то есть устройства управления и системы связи) и распространения приложений на моделируемые устройства управления.
- Правила использования FB, удовлетворяющие коммуникационным требованиям распределенной IPMCS.

• Правила использования FB для управления распределенными приложениями, ресурсами и устройствами, используемыми в распределенной IPMCS.

Часть 2 — Требования к программным инструментам определяют соответствующие требования, поддерживающие следующие задачи системного проектирования IPMCS.

- Спецификация типов FB.
- Функциональная спецификация типов ресурсов и типов устройств.
- Спецификация, анализ и проверка распределенного IPMCS.
- Конфигурирование, реализация, оперирование и обслуживание распределенного IPMCS (то есть управление жизненным циклом).
- Обмен информацией и элементами между различными инструментами разработки программного обеспечения, использующими XML.

Часть 3 — Правила для профилей соответствия определяют требования и руководящие указания для спецификации, а также разработку профилей соответствия, используемых в различных областях и приложениях, в которых рассматриваются следующие атрибуты систем, устройств и программных средств IEC 61499.

- Переносимость программных средств для принятия и правильного восприятия элементов библиотеки, созданных другими программными средствами.
- Конфигурируемость устройств и их программных компонентов (выбранных, назначенных, взаимосвязанных и параметризованных) с помощью нескольких программных инструментов.
- Совместимость устройств для совместной работы в целях выполнения функций, определенных одним или несколькими распределенными приложениями.

ЕС / TR 61499-3, содержащий учебную информацию, был отозван в 2007 году как устаревший из-за широкого распространения учебных и учебно-методических материалов и книг. В настоящее время ведутся работы по повторной активации и обновлению части 3 и созданию новой части 5, вводящей новые элементы (поведение синхронизации в последовательностях обслуживания, развертывание двоичного XML, типы составных адаптеров, пространства имен, гибридные платформы IEC 61499 и IEC 61131 и т. д.) в эталонную модель IEC 61499. В следующих разделах описаны основные концепции и элементы моделирования, которые в основном рассматриваются в первой части стандарта.

2.2.3. Модель выполнения на основе событий

Модель исполнения IEC 61499 определяет FB, которое характеризуется двумя различными типами (I/O): события и данные. Выполнение инициируется событиями; необходимые данные должны быть действительными на входах FB до прибытия события (обозначено как конструкция WITH в IEC 61499, то есть вертикальные линии между входом или выходом события и данных, как показано на рисунке 2.1). Только если событие происходит на входе FB, среда выполнения должна обрабатывать выполнение этого FB. Следовательно, вполне очевидно, является ли приложение активным и какая именно часть в данный момент обрабатывается. По сравнению с подходами на основе PLC, такими как IEC 61131-3, подход IEC 61499 не определяет при исполнении какого-либо конкретного временного или циклического неявного поведения. Он поддерживает асинхронное выполнение FB и сетей FB за счет использования специальных событий FB и делает возможным синхронное выполнение FB.

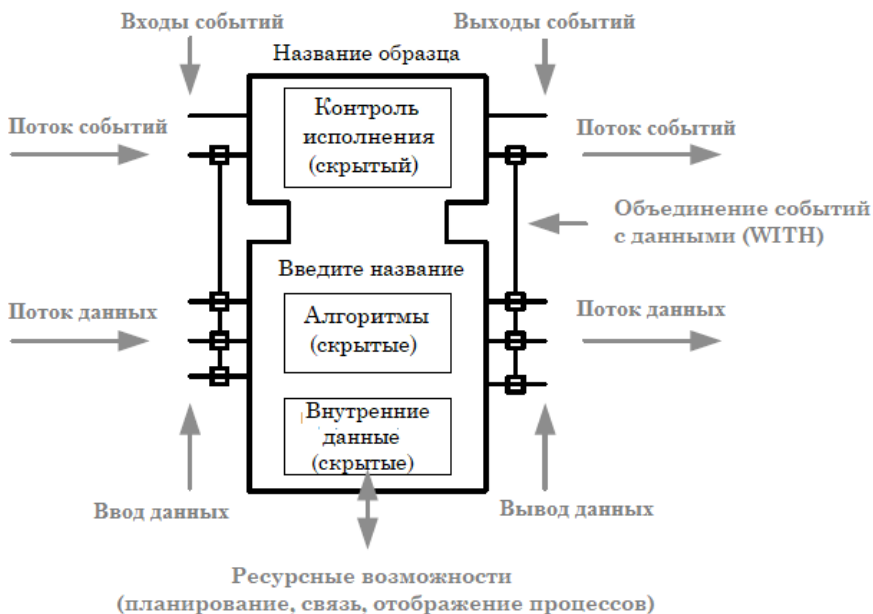


Рисунок 2.1. Характеристики IEC 61499 FB [4]

Стандарт определяет относительно простую модель исполнения для FB, которая показана на рисунке 2.2 (а). Выполнение FB запускается событием после того, как входные данные становятся доступными. На следующем этапе оценивается функция управления выполнением, а основная функция планирования (предоставляемая ресурсом) отвечает за планирование соответствующего алгоритма. Когда выполнение завершено, выходные данные обновляются и генерируется вывод события. Обычно это запускает выполнение другого подключенного FB. Результирующее временное поведение FB IEC 61499 приведено на рисунке 2.2 (b).

2.2.4. Функциональные блоки как основные элементы моделирования

Как сказано выше, основные элементы моделирования, определенные в IEC 61499, являются FB. Они устанавливаются для промышленного применения с целью определения устойчивых и допускающих повторное использование компонентов программного обеспечения. FB преимущественно хранят программные продукты и алгоритмы для различных проблем. Определение FB IEC 61499 основано на спецификациях IEC 61131-3, но оно расширено интерфейсом событий для управления выполнением, как уже описано выше.

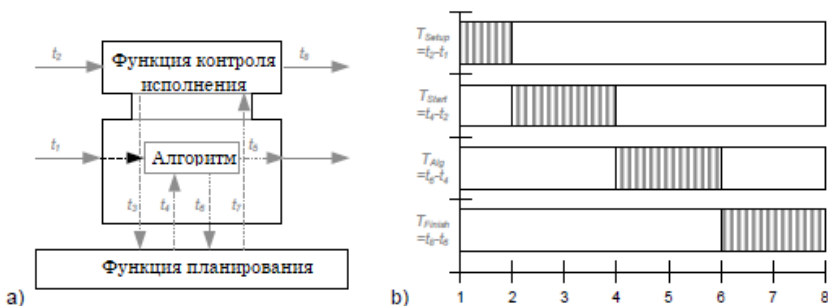


Рисунок 2.2. Характеристики исполнения ИЕС 61499: (а) модель и (б) временные характеристики [4]

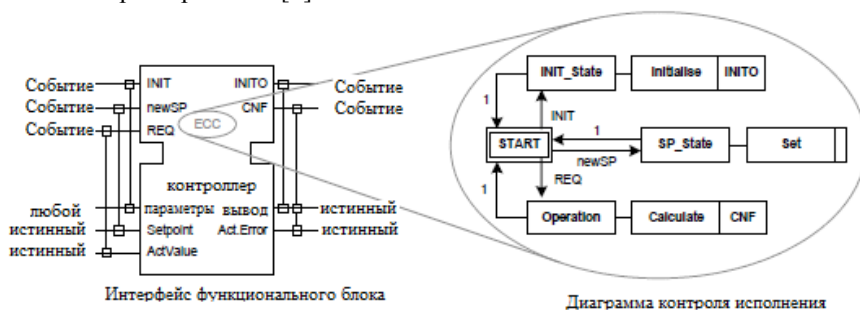


Рисунок 2.3. Пример базового FB [4]

Стандарт ИЕС 61499 определяет три основных типа FB: (1) базовый FB для инкапсуляции алгоритма; (2) функциональный блок интерфейса службы (SIFB), служащий в качестве интерфейса к элементам, не совместимым с ИЕС 61499 (сеть связи, процесс / машина и т. д.); (3) составной FB для функциональной агрегации. В дополнение к этим типам FB так называемый тип интерфейса адаптера позволяет определять интерфейсы взаимодействия компонентов. Для структурирования приложений введен так называемый тип подприложения.

Как следует из названия, базовый тип FB – это «атом», из которого могут быть построены «молекулы» более высокого уровня. Инженеры могут инкапсулировать алгоритмы, написанные на одном из языков программирования ИЕС 61131-3 или других языках программирования более высокого уровня (например, Java, C++, C#). Выполнение этих алгоритмов инициируется так называемыми диаграммами контроля выполнения (ЕСС), которые являются конечными автоматами, управляемыми событиями, как показано на рисунке 2.3.

Другие «атомарные» FB – это типы SIFB. Они используются для моделирования интерфейса с низкоуровневыми службами, предоставляемыми операционной системой или оборудованием контроллера. Например, SIFB используются в сетях связи, поскольку конкретные протоколы напрямую не

охватываются эталонной моделью. Только конкретные интерфейсы для обмена данными и информацией между устройствами и ресурсами представлены в виде SIFB для сферы FB. В стандарте предлагаются два различных высокоуровневых шаблона связи: (1) публикация / подписка и (2) модели клиент / сервер. Первый в основном используется для односторонней связи в соответствии с концепцией производителя и потребителя, тогда как второй предназначен для двусторонней связи, как показано на рисунке 2.4.

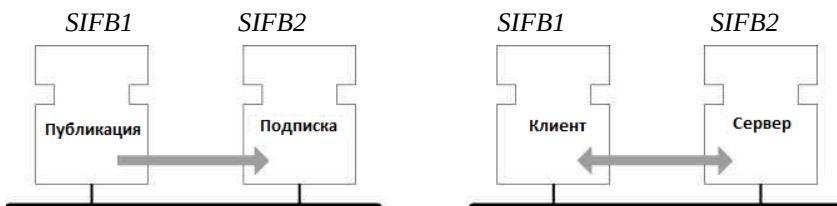
Инженеры также могут создавать «молекулы» FB более высокого уровня, называемые составными типами FB, из существующих FB (то есть компонентных FB). Это выполняется путем указания интерфейсов событий и данных составного типа с последующим моделированием инкапсулированной сети FB из внутренних компонентных FB. В этом типе FB выполнение алгоритмов в компонентных FB контролируется потоком событий от одного компонента к другому.

2.2.5 Прикладная и системная модель. Модели устройства и ресурса

Чтобы определить и смоделировать распределенную IPMCS с несколькими устройствами контроллера, используя вышеописанные FB, нужно прибегнуть к следующим дополнительным элементам, которые определены в IEC 61499 (см. рисунок 2.5) [5]:

- *Прикладная модель.* В архитектурной модели приложения создаются путем соединения экземпляров повторно используемых типов FB с соответствующими событиями и соединениями данных. Это сопоставимо с проектированием плат с интегральными схемами. Ключевой аспект IEC 61499 заключается в том, что приложения разрабатываются независимо от физической инфраструктуры управления (то есть модели системы).

- *Системная модель.* IPMCS моделируется как совокупность устройств, соединенных между собой и связывающихся друг с другом посредством сетевой связи, состоящей из сегментов и линий связи. Устройства подключены к сегментам сети через ссылки.



- *Модель устройства.* Описывает структуру управляющего оборудования, способного создавать сети FB согласно IEC 61499. Здесь содержится по крайней мере один интерфейс, либо процесс и/или интерфейс связи. В качестве исполнительных объектов устройства имеют ресурсы (см. модель

ресурсов ниже). Интерфейс процесса обеспечивает отображение между физическим процессом или машиной (аналоговые измерения, дискретный ввод-вывод и т. д.) и ресурсами. Информация, которой обмениваются в ходе физического процесса, представляется ресурсу как данные или события, или и то, и другое. Интерфейс связи обеспечивает отображение между ресурсами и информацией, которой обмениваются с другими устройствами через сеть связи.

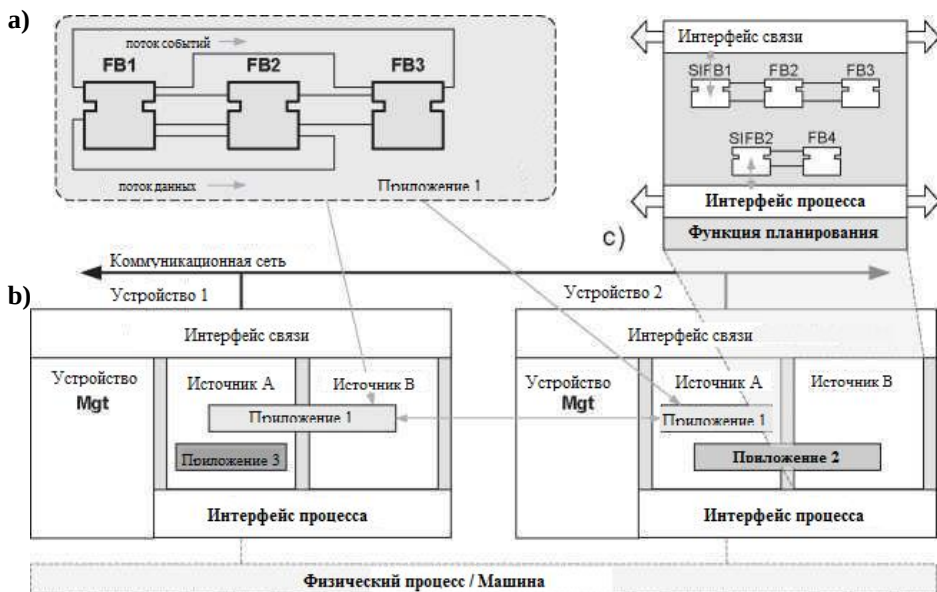


Рисунок 2.5. Концепции и модели IEC 61499: (а) применение, (б)

конфигурация системы и устройства и (с) ресурс [4]

- *Модель ресурса.* Ресурс можно рассматривать как функциональную единицу. Он заключен в устройстве и имеет независимый контроль над его работой. Ресурсом можно управлять (создавать, настраивать, параметризовать, запускать, удалять и т. д.), не затрагивая при этом другие ресурсы в пределах того же устройства.

2.2.6. Модель распределения

Одним из основных целей IEC 61499 является обеспечение распределенных приложений, используемых в IPMCS. Модель системы состоит из устройств, соединенных сетью связи, и приложений. Эти приложения могут быть распределены между устройствами, а также внутри

устройств (см. рисунок 2.5). Стандарт IEC 61499 позволяет разделять устройства на независимые ресурсы, которые являются контейнерами выполнения для сетей FB, как уже говорилось выше. Эта концепция позволяет локально распределять прикладную часть конкретного устройства на несколько ресурсов. Процесс проектирования начинается с функциональности верхнего уровня, которая должна быть реализована без привязки к конкретной аппаратной структуре. Используя библиотеки программных компонентов, пользователь моделирует необходимые приложения. Последний шаг в цикле разработки – это сопоставление приложений с конкретными аппаратными компонентами, независимо от того, выполняется ли приложение одним устройством или распределено по нескольким устройствам.

2.2.7. Модель управления

Конфигурирование распределенной IPMCS с использованием эталонной модели IEC 61499 может быть выполнено с использованием специальных функций управления. Они обычно обеспечиваются контрольными устройствами. Для этой цели стандарт определяет управление устройством (см. рисунок 2.5b) вместе с соответствующим приложением управления, представленным управляющим SIFB. Общий интерфейс изображен на рисунке 2.6. При использовании этого FB в сочетании с удаленным приложением возможен взаимный доступ между различными устройствами, соответствующими IEC 61499. Следующие стандартизированные функции управления определены в стандарте и могут использоваться для взаимодействия и настройки устройства.

- Инициирование выполнения элементов с помощью START, RESET.
- Остановка выполнения элементов с помощью STOP, KILL.
- Создание экземпляров FB, а также подключений к событиям и данным с использованием CREATE.
- Удаление экземпляров FB и соединений с помощью DELETE.
- Параметризация элементов с использованием WRITE, READ.
- Предоставление данных о состоянии устройств, ресурсов и FB с использованием QUERY.

Чтобы получить доступ к этим командам, стандарт предоставляет вышеупомянутый SIFB управления устройствами (см. рисунок 2.6), но оставляет протокол для доступа к нему открытым. В «Профиле соответствия IEC 61499 для технико-экономических обоснований» предложена спецификация на основе XML [1]. На рисунке 2.7 приведен пример для этого протокола XML.

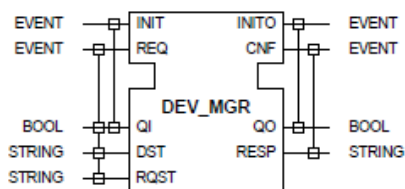


Рисунок 2.6. Устройство управления FB [4]

<Идентификатор запроса = "3" Действие =

"СОЗДАЮТ"> <Имя FB = Тип "DIAG" =
"SUBL_2"/> </Запрос>

<Идентификатор запроса = "4" Действие = "СОЗДАЮТ">

<Имя FB = "РЕГИСТРИРУЮТ" Тип =
"DIAG_LOG"/> </Запрос>

<Идентификатор запроса = "7" Действие = "СОЗДАЮТ">

<Источник соединения = Место назначения "DIAG.IND" =

"LOG.REQ"/> </Запрос>

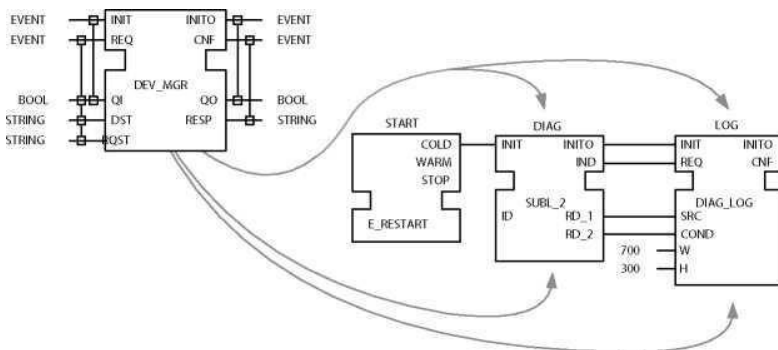


Рисунок 2.7. Пример конфигурации с использованием концепции диспетчера устройств [1]

2.2.8. Модель рабочего состояния

В дополнение к вышеупомянутым моделям стандарт IEC 61499 определяет модель рабочего состояния. Он используется для моделирования жизненного цикла системы, что означает, что любая конфигурация системы, соответствующая IEC 61499, должна быть (1) задана, (2) спроектирована и смоделирована, (3) внедрена, (4) введена в эксплуатацию, (5) эксплуатирована, и (6) поддерживаема. Как правило, система состоит из функциональных блоков (то есть устройств, ресурсов, приложений, FB), имеющих свои собственные жизненные циклы. Для поддержки этих функциональных блоков на каждом этапе жизненного цикла необходимо выполнять различные действия. Поэтому в стандарте предлагаются рабочие состояния, чтобы определить, какие действия могут быть выполнены. Примерами могут быть «Оперативный», «Настраиваемый», «Загруженный», «Остановленный» и т. д. Каждое рабочее состояние указывает, какие действия разрешены, а также ожидаемое поведение. Кроме того, может быть организована система, в которой определенные функциональные блоки могут иметь или приобретать право изменять состояния других функциональных блоков.

2.3. Основные отличия между первой и второй моделями

Со времени выпуска первой редакции IEC 61499 было сделано несколько реализаций и приложений в научных и промышленных приложениях. Некоторые усовершенствования были необходимы для дальнейшей разработки данной эталонной модели для распределенной IPMCS. В результате в 2012 году было опубликовано второе издание IEC 61499, включающее несколько изменений, сделанных в ответ на примерно 120 редакционных и 40 технических комментариев, полученных от национальных комитетов, с дополнительными редакционными изменениями в соответствии с требованиями IEC. Наиболее важные и значимые технические изменения по сравнению с первым изданием перечислены ниже [2, 4].

- *Контроль выполнения:* модель выполнения FB, особенно ECC, была одной из наиболее обсуждаемых проблем спецификации IEC 61499 в промышленности и в научных кругах с момента ее публикации. Чтобы прояснить эту проблему, во втором издании была представлена более понятная и точная спецификация. Была решена проблема с параллельностью и согласованностью данных и добавлено уточнение семантики события.

- *Временные переменные:* аналогично IEC 61131-3, временные переменные в алгоритмах, инкапсулированных в базовые FB, теперь возможны. Они видны только в тех алгоритмах, в которых используются. Это значительно улучшает читаемость базовых FB, поскольку переменные, необходимые только внутри алгоритма, теперь не нужно объявлять как внутренние

переменные FB. Типичным примером применения является переменная, используемая в качестве счетчика цикла.

- *Типы сетей и сегментов*: в исходной спецификации IEC 61499 использование различных типов сегментов в модели системы было невозможно. Поскольку существуют различные решения для полевой шины и промышленного Ethernet, их использование в модели системы является предпочтительным. Эта проблема была решена в новой редакции, когда появилась возможность указывать свойства типа сети и соответствующих ей протоколов. Это позволяет четко документировать общую структуру системы распределенной IPMCS.

- *Взаимодействие с PLC*: поскольку IEC 61131-3 широко используется в IPMCS, гармонизация с IEC 61499 имеет смысл [9]. Следовательно, второе издание IEC 61499 определяет использование специальных SIFB, которые могут выступать в качестве клиентов услуг связи *PLC*, определенных в IEC 61131-5. Теперь возможны READ FB для синхронного чтения данных *PLC*, UREAD FB для асинхронного чтения и WRITE FB для синхронной записи данных *PLC*. Удаленный запуск задач *PLC*, как определено в IEC 61131-3, теперь становится возможным с использованием нового TASK FB.

- *Упрощенные команды READ и WRITE*: в первом издании IEC 61499 была применена концепция пути доступа из IEC 61131-3. С помощью команд управления READ и WRITE был возможен доступ к внутренним переменным базисных FB. Это противоречит объектно-ориентированному подходу IEC 61499. Как следствие, эта опция была удалена в новой редакции IEC 61499. С этого момента командам управления READ и WRITE разрешен только доступ к интерфейсу FB, устройствам и ресурсам, а все внутренние устройства от них скрыты.

- *Дальнейшие изменения*: кроме того, во второе издание также был внесен ряд небольших исправлений и изменений. Например в опорную модель, была добавлена забытая команда управления RESET. Другое изменение было связано с последовательностями услуг: первоначально определено для описания поведения SIFB и теперь разрешено для всех типов FB. Кроме того, теперь можно использовать интерфейсы адаптера для всех типов FB, а не только для составных FB.

Библиография

[1] J. H. Christensen. HOLOBLOC.com. Function Block-Based, Holonic Systems Technology, Access Date: June 2015.

[2] J. H. Christensen, T. Strasser, A. Valentini, V. Vyatkin, and A. Zoitl. The IEC 61499 Function Block Standard: Overview of the Second Edition. In Proceedings of ISA Automation Week, September 24-27, Orlando, Florida, 2012.

[3] R. W. Lewis. Programming Industrial Control Systems Using IEC 1131-3 (IEE Control Engineering Series). Institution of Electrical Engineers, Stevenage, UK, 1998.

[4] TC 65/SC 65B. IEC 61499: Function blocks. International Electrotechnical Commission, Geneva, 2nd ed., 2012.

[5] TC 65/SC 65B. IEC 61131: Programmable controllers. International Electrotechnical Commission, Geneva, 3rd ed., 2012.

[6] I. Terzic, A. Zoitl, B. Favre-Bulle, and T. Strasser. A survey of distributed intelligence in automation in European industry, research and market. In Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation, pages 221–228, Sept. 2008.

[7] V. Vyatkin. IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review. IEEE Transactions on Industrial Informatics, Vol. 7, No. 4, 768–781, 2011.

[8] A. Zoitl and R. Lewis. Modelling Control Systems Using IEC 61499. Control, Robotics & Sensors. Institution of Engineering and Technology, 2nd edition, 2014.

[9] A. Zoitl, T. Strasser, C. Sünder, and T. Baier. Is IEC 61499 in harmony with IEC 61131-3? IEEE Industrial Electronics Magazine, Vol. 3, No. 4, 49–55, Dec. 2009.

Часть II

Руководство по проектированию и разработке приложений

Шаблоны проектирования, цели и методология

James H. Christensen

Holobloc Inc.

СОДЕРЖАНИЕ

3.1 Введение, мотивация и обзор	51
3.2 Методология распределенного приложения	53
3.2.1 Цель	53
3.2.2 Методология	53
3.2.3 Поддержка инструмента	54
3.3 Шаблон «проху»	55
3.3.1 Цель	55
3.3.2 Методология	55
3.3.3 Пример	55
3.4 Многоуровневая Модель / Представление / Контроллер / Диагностика (MVCD)	56
3.4.1 Цель	56
3.4.2 Платформа	56
3.4.3 Методология	56
3.5 Локальный многоадресный шаблон	58
3.6 Теговый шаблон данных	59
3.7 Матричная платформа	59
3.8 Выводы	60

3.1. Введение, мотивация и обзор

Шаблон проектирования определяется как «формализация подхода к общей проблеме в контексте» [4, с. 219]. В нашем случае контекст задается архитектурой IEC 61499 для функциональной инкапсуляции, повторного использования и развертывания программного обеспечения в распределенных системах автоматизации и управления [2]. Адаптация привычных шаблонов проектирования в этом контексте может уменьшить время, необходимое для обучения и реализации приложений в данной архитектуре.

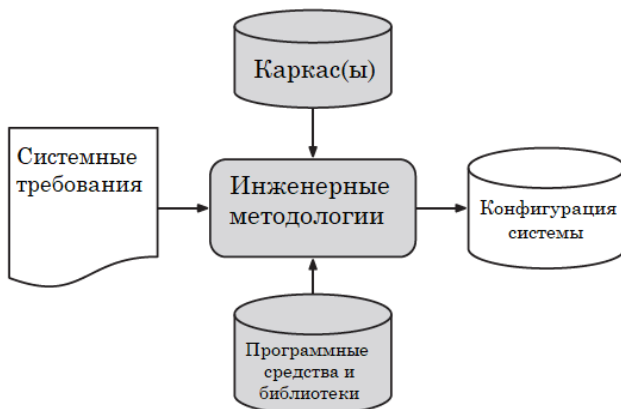


Рисунок 3.1. Применение элементов проектирования шаблона (закрашено).

Как показано на рисунке 3.1, для каждого шаблона проектирования адаптация состоит из трех основных этапов.

1. Определение *проблемы*, которая должна быть решена.
2. Определение подходящей *основы* для решения проблемы. В объектно-ориентированном дизайне каркас считается скелетной структурой, которую необходимо конкретизировать для создания законченного приложения. В архитектуре IEC 61499 [6, 7] структура будет состоять из набора связанных типов, включая, среди прочего:

- (a) типы функциональных блоков;
- (b) типы данных;
- (c) типы ресурсов;
- (d) типы устройств.

3. Определение *методологии разработки*, включающей процедуры использования различных типов в структуре для:

- (a) создания соответствующих *устройств и ресурсов*;
- (b) заполнения устройств и ресурсов соответствующими *функциональными блоками*;
- (c) установления соответствующего *события и соединения данных* между функциональными блоками;
- (d) правильной *настройки* устройства, ресурсов и функциональных блоков;
- (e) *установки, ввода в эксплуатацию и управления* разработанными конфигурациями в физических устройствах.

Может потребоваться указание требований к *программным средствам* для поддержки шаблона проектирования.

В этой главе приводится краткое описание ряда типовых шаблонов проектирования и структур для архитектуры IEC 61499, включая:

- методологию распределенных приложений;

- шаблон прокси;
- многоуровневую модель / представление / контроллер / диагностику (MVCD);

- шаблон локальной многоадресной рассылки;
- помеченный шаблон данных;
- матрица рамки.

В следующих разделах эти важные схемы проектирования, относящиеся к стандарту IEC 61499, рассматриваются более подробно.

Подробная информация и рабочие примеры доступны в Интернете по адресу: [1]. Кроме того, исходные и исполняемые коды для примеров, приведенных в этой главе, включены в комплект разработки функциональных блоков (FBDK) [5]. Дополнительные и альтернативные шаблоны проектирования со связанными с ними структурами, поддержкой инструментов и методологиями разработки предоставляются другими программными инструментами и средами выполнения для архитектуры IEC 61499, такими как 4DIAC-IDE с открытым исходным кодом и связанная с ним среда выполнения FORTE, файлы и описание которой также доступны онлайн [8].

3.2. Методология распределенного приложения

3.2.1. Цель

Этот шаблон помогает инженерам в настройке распределенных приложений в архитектуре IEC 61499.

3.2.2. Методология

Чтобы применить этот шаблон, необходимо выполнить следующие шаги.

1. Разработайте и/или приобретите библиотеки типов функциональных блоков, доступных для создания приложений, которые будут распространяться наряду с типами устройствами и типами ресурсов, необходимых для реализации системы.

2. Определите *приложение (приложения)*, которое будет использоваться для реализации требуемых системных функций, в качестве сетей соответствующих взаимосвязанных и параметризованных *экземпляров* функциональных блоков.

3. Выполните *сопоставление* экземпляров функциональных блоков и связанных с ними соединений из приложений в соответствующих ресурсах. Рисунок 3.2 иллюстрирует этот шаг для отображений, определенных текстовыми декларациями IEC 61499.

MAPPING

```

SORT.SENSOR ON SENSOR.R1.SENSOR;
SORT.GATE ON SENSOR.R1.LOGIC;
SORT.ACTUATOR ON ACTUATOR.R1.VALVE;
END_MAPPIN

```

4. Настройте устройство и экземпляры ресурса в системе.

5. Сконфигурируйте и соедините функциональные блоки интерфейса службы связи (SIFB), чтобы реализовать соединения событий и соединения данных приложения через границы ресурсов и устройств.

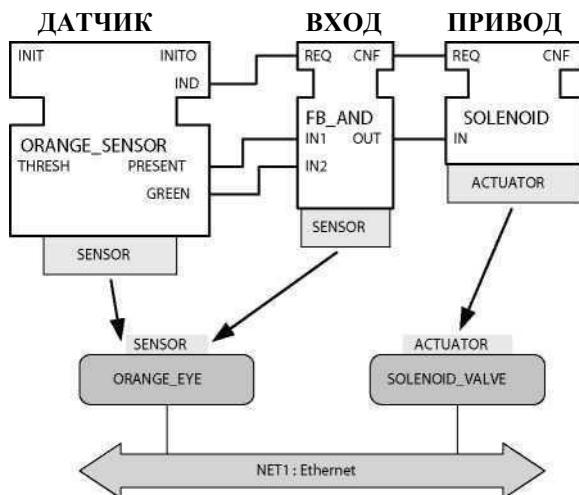


Рисунок 3.2. Сопоставление функциональных блоков приложения с устройствами (пример: сортировка приложения)

3.2.3. Поддержка инструмента

На рисунке 3.2 представлен пример поддержки инструмента для графического отображения сопоставления приложения SORT и устройств SENSOR и ACTUATOR через метки, прикрепленные к нижней части отображенных функциональных блоков. Поскольку эта поддержка инструмента не определена в стандарте IEC 61499, различные инструменты, например, FBDK [5] и 4DIAC [8], могут предоставлять различные виды поддержки.

3.3. Шаблон «проху»

3.3.1. Цель

Этот шаблон «отделяет клиентов от их серверов, создавая локальный прокси или резерв для менее доступного сервера. Когда клиенту необходимо запросить службу у сервера, например, получить значение, он запрашивает свой локальный прокси-сервер. Затем прокси-сервер может направить запрос на исходный сервер...» [4, с. 224].

3.3.2. Методология

Чтобы применить этот шаблон, необходимо выполнить следующие шаги:

- 1) документировать локальный интерфейс и наблюдаемое поведение прокси-сервера в качестве интерфейса и службы (то есть набора последовательностей услуг) типа SIFB IEC 61499;
- 2) реализовать желаемое поведение, как для базового или составного типа функционального блока (последнее предпочтительней для повторного использования типов SIFB связи для стандартизированной связи с удаленным сервером).

3.3.3. Пример

На рисунке 3.3 показано использование прокси-SIFB в двух отдельных устройствах, каждое из которых обменивается данными через собственную проприетарную линию связи с датчиками и исполнительными механизмами соответствующего устройства. Локальное управление каждой машиной осуществляется приложением в его собственном устройстве управления, а работа машин координируется посредством связи между двумя устройствами уП

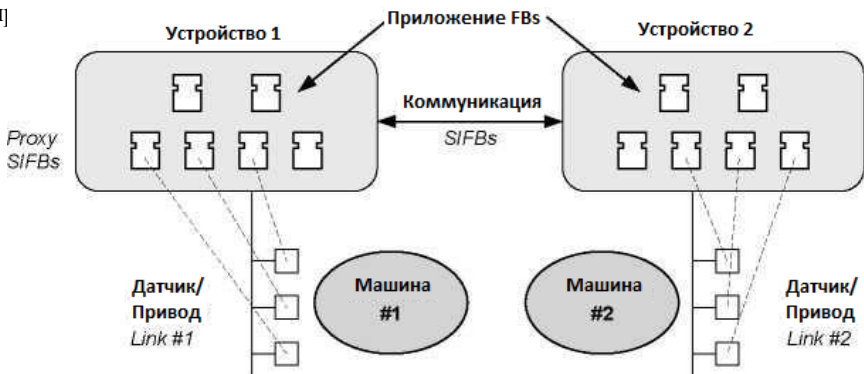


Рисунок 3.3. Пример шаблона «проху» [3]

3.4. Многоуровневая Модель / Представление / Контроллер / Диагностика (MVCD)

3.4.1. Цель

Этот шаблон расширяет традиционный объектно-ориентированный шаблон MVC, чтобы обеспечить интеграцию моделирования, визуализации, диагностики и устранения неисправностей контролируемой машины или процесса в процесс проектирования систем управления и автоматизации с использованием архитектуры IEC 61499 [4, с. 262ff].

3.4.2. Платформа

Как показано на рисунке 3.4, инфраструктура MVCD включает в себя типы *устройств*, *типы ресурсов* и *типы FB*, необходимые для реализации следующих функциональных уровней.

Уровень Model реализует зависящее от времени логическое поведение управляемой системы или устройства. Этот уровень обычно заменяется интерфейсами с реальной физической системой после проверки соответствующего уровня управления.

Уровень View обеспечивает графическое отображение и, возможно, пользовательский ввод, связанный с элементами слоя Model.

Уровень Control реализует функции управления, которые должны выполняться по отношению к элементам уровня модели, включая соответствующие интерфейсы событий и данных для интеграции блоков контроллера.

Уровень Diagnostic выполняет обнаружение и диагностику и по возможности устранение неисправностей оборудования или устройств.

Уровень HMI содержит элементы, необходимые для интерфейса человек–машина посредством установки и считывания параметров и данных с контроллера и диагностических элементов, присутствующих на уровне управления.

3.4.3. Методология

Сокращенная методология, описанная ниже, использует преимущества визуальной природы инфраструктуры MVCD для создания и поддержания визуального представления работы управляемого процесса или машины на протяжении всего процесса системного проектирования.

1. Из исходного эскиза и словесного описания желаемого протекания процесса или поведения машины, которой нужно управлять, разработайте содержимое слоя View вместе с подходящим слоем HMI, чтобы получать пользовательские входные данные и отправлять значения данных, которые

будут отображаться, на просмотр. Используйте эту конфигурацию для разработки последовательности снимков, представляющих желаемую последовательность состояний системы.

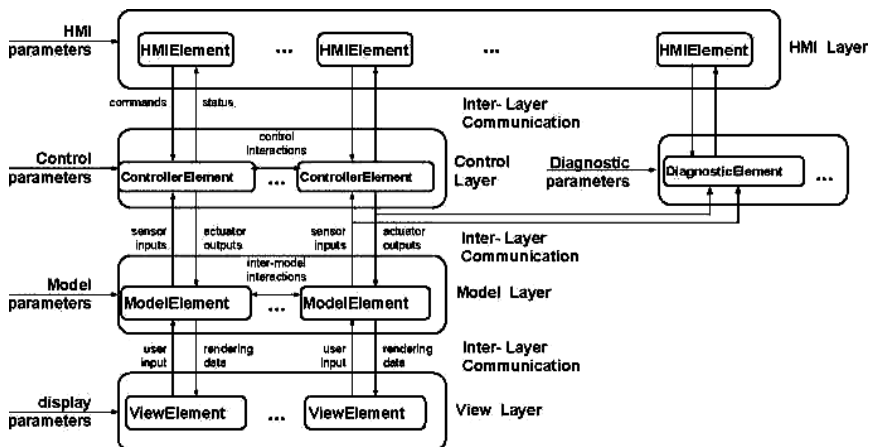


Рисунок 3.4. Модель / вид / контроллер / рамки диагностики

2. Разработайте слой Model для моделирования динамического поведения подключенной машины или технологического оборудования. Интерфейс модель / вид должен заменить предыдущие интерфейсы HMI / вид, а также должен быть разработан новый уровень HMI для приема имитированных входных сигналов от датчиков и представления имитированных выходов исполнительного механизма на уровне модели. Комбинированная конфигурация HMI / модель / представление может затем использоваться для наблюдения за откликами уровня модели на входах моделируемого привода и при необходимости уточнения содержимого уровня модели.

3. Заполните уровень управления взаимосвязанными экземплярами типов контроллеров FB, которые инкапсулируют необходимую логику управления для достижения требуемого поведения системы. Замените соединения датчика и привода от предыдущих интерфейсов HMI / модель соответствующими соединениями на новом интерфейсе контроллер / модели. Реструктурируйте уровень HMI, чтобы он содержал те элементы, которые необходимы для выдачи команд и получения уведомлений о состоянии от уровня Controller, и используйте этот новый HMI для проверки того, что уровень Controller вызывает требуемое поведение системы в ответ на команды от HMI.

4. Добавьте диагностические элементы на уровень управления, чтобы отслеживать выходы исполнительного механизма и отклики датчиков на уровне модели и определить, правильно ли ведет себя система. Если

наблюдается недопустимое поведение, диагностические элементы должны сообщать об этом на уровень HMI через сообщения о состоянии ошибок и реализовывать процедуры восстановления после сбоев либо автоматически, либо по команде от HMI (в качестве альтернативы восстановление исключений может быть встроено в элементы контроллера).

Интерфейсный элемент

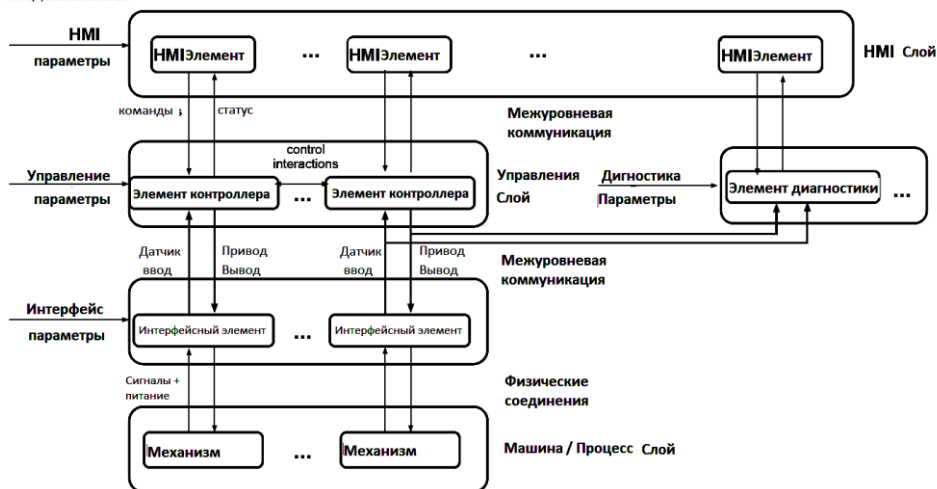


Рисунок 3.5. Физическое проектирование

5. Как только правильная работа системы будет достигнута за счет того, что обнаружение и устранение неисправностей будет выполнено с помощью моделируемой системы на уровне модели, можно реализовывать физическое проектирование, заменив уровни модели и представления уровнем интерфейса между уровнем управления и фактическим физическим процессом или механизмом, как показано на рисунке 3.5 (вероятно, можно повторно использовать некоторые элементы уровня View в качестве части HMI, например, для визуального представления текущего состояния процесса).

6. В зависимости от ситуации дополнительные этапы могут включать в себя распределение блоков из соответствующих уровней для распределенных устройств, используя распределенный шаблон приложения и выделение контроллера и диагностических элементов для встроенных контроллеров, тесно связанных с локальными механизмами, то есть мехатронное проектирование.

3.5. Локальный многоадресный шаблон

Локальный многоадресный шаблон, структура и методология доступны для обеспечения эффективной внутрипроцессной многоадресной передачи данных без необходимости прибегать к более дорогим методам межпроцессного взаимодействия, таким как многоадресная передача UDP (User Datagram

Protocol — протокол пользовательских датаграмм). Как показано на рисунке 3.6, эта эффективность достигается за счет замены кодирования и декодирования данных для последовательной многоадресной рассылки простым копированием данных в локальной многоадресной рассылке.

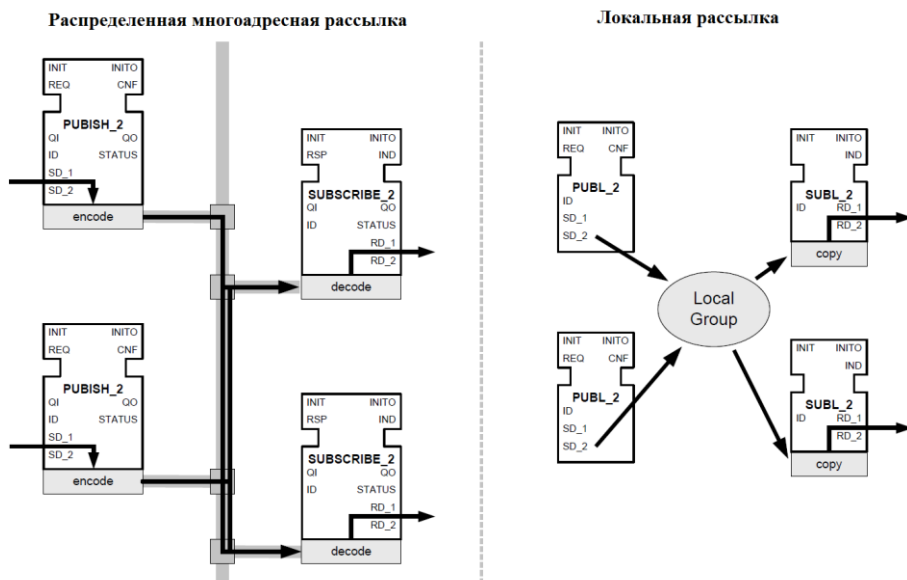


Рисунок 3.6. Локальная или распределенная многоадресная рассылка [3].

3.6. Теговый шаблон данных

Теговый шаблон проектирования данных, структура, методология и поддержка инструментов могут использоваться для обеспечения того, чтобы:

- данные, используемые в локальном многоадресном канале, соответствовали данным, используемым в соответствующем распределенном многоадресном канале
- данные, подписанные с локального или распределенного многоадресного канала, соответствовали данным, опубликованным на канале.

3.7. Матричная платформа

Матричная платформа доступна для поддержки операций над двумерными матрицами значений с плавающей точкой. Эта структура включает в себя:

- тип данных MATRIX, который может быть реализован внутри, используя язык высокого уровня, такой как Java или C;
- типы функциональных блоков для выполнения общих операций с матрицами, таких как сложение, вычитание, умножение и инверсия;
- тип SIFB для инициализации и выполнения перечисленных операций над инкапсулированной матрицей;
- типы SIFB для таблично-ориентированного матричного ввода и вывода.

3.8. Выводы

Для эффективного использования архитектуры, определенной в стандарте IEC 61499-1, для инкапсуляции, повторного использования и развертывания интеллектуальной собственности (IP) в распределенных системах управления и автоматизации требуются шаблоны проектирования и связанные с ними структуры, инженерные методологии и поддержка программных средств. К счастью, требования к программному инструменту определены в IEC 61499-2.

Требования к профилю соответствия, определенные в IEC 61499-4, делают разработку таких платформ и программных инструментов систематической, как показали примеры в этой главе.

Библиография

- [1] J. Christensen. IEC 61499 Design Patterns.
<http://www.holobloc.com/doc/despats>, access date July 2015.
- [2] J. Christensen and A. Zoitl. IEC 61499: A Standard for Software Reuse in Embedded, Distributed Control Systems.
<http://www.holobloc.com/papers/iec61499/overview.htm>, access date July 2015.
- [3] J. H. Christensen. Automation modeling, design and programming with the iec 61499 function block standard. Helsinki Technical University, June 2004.
- [4] B. Douglass. Real-Time UML. Addison Wesley Longman, 1998.
- [5] Holobloc Inc. FBDK – Function Block Development Kit.
<http://www.holobloc.com/doc/fbdk/>, access date July 2015.
- [6] TC 65/SC 65B. IEC 61499: Function blocks. International Electrotechnical Commission, Geneva, 2nd ed., 2012.
- [7] A. Zoitl and R. Lewis. Modelling Control Systems Using IEC 61499. Control, Robotics & Sensors. Institution of Engineering and Technology, 2014.
- [8] A. Zoitl, T. Strasser, and A. Valentini. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study. In Proceedings of IEEE International Symposium on Industrial Electronics, pages 3817–3819, 2010.

Применение парадигм дизайна IEC 61499: объектно-ориентированное программирование, компонентно- ориентированный дизайн и архитектура для обслуживания широкого круга запросов

Wenbin Dai

Шанхайский университет Цзяотун

Valeriy Vyatkin

Технологический университет Лулео и Университет Аалто

James H. Christensen

Holobloc Inc.

СОДЕРЖАНИЕ

4.1 Введение	62
4.2 Существенные элементы в IEC 61499 проектов функционального блока	63
4.3 Применение парадигмы объектно-ориентированного программирования в IEC 61499 функциональных блоков	65
4.3.1 Предпосылки объектно-ориентированного программирования	65
4.3.2 Применение понятий объектно-ориентированного программирования в стандарте IEC 61499	65
4.3.3 Тематическое исследование парадигмы объектно- ориентированного программирования	68
4.4 Принятие компонентно-ориентированной парадигмы дизайна для функциональных блоков IEC 61499	75
4.4.1 Предпосылки компонентно-ориентированного дизайна	75
4.4.2 Компонентно-ориентированные принципы разработки для функциональных блоков IEC 61499	75
4.4.3 Тематическое исследование компонентно-ориентированной парадигмы дизайна	80
4.5 Представление новой парадигмы архитектуры для обслуживания широкого круга запросов для функциональных блоков IEC 61499	81
4.5.1 Предпосылки архитектуры для обслуживания широкого круга запросов IEC 61499	81

4.5.2 Как применить понятия архитектуры для обслуживания широкого круга запросов в разработке системы IEC 61499	82
4.5.3 Тематическое исследование парадигмы архитектуры для обслуживания широкого круга запросов	86
4.6 Краткое содержание парадигм проектирования IEC 61499	87
4.7 Выводы	90

4.1. Введение

С ростом требований к интеграции и взаимодействию с машинами, роботами и другими системами сложность систем промышленной автоматизации в течение последнего десятилетия постоянно увеличивалась. Из-за ограничений производительности и проблем избыточности современные системы промышленной автоматизации часто управляются несколькими программируемыми логическими контроллерами (PLC), соединенными через промышленные сети на основе Ethernet. Традиционные системы промышленной автоматизации контролируются PLC IEC 61131-3 [10]. В связи с ограничением программной модели IEC 61131-3 традиционные программы управления PLC разрабатываются на основе каждого контроллера. Взаимодействия между несколькими PLC должны обрабатываться неявно. В результате возможности многократного использования и гибкость существующих программ PLC IEC 61131-3 в значительной степени снижаются.

С другой стороны, еще один международный стандарт – стандарт IEC 61499 [9] – предназначен для обеспечения методологии проектирования на уровне системы для систем распределенной автоматизации. Системы могут быть сконфигурированы, даже если информация о развертывании оборудования в стандарте IEC 61499 отсутствует. Блочный дизайн обеспечивает многократное использование логики управления за счет инкапсуляции, скрывает сложности, обеспечивая иерархическую структуру программного обеспечения, и отображает визуальный образ управления программой. Однако без применения определенных методик проектирования средне- и крупномасштабные функциональные блок-схемы сети могут быть запутанными и значительно снижать эффективность проектирования программного обеспечения. Чтобы не тратить массу времени на разработку, для функциональных блоков IEC 61499 необходимы зрелые парадигмы проектирования. В этой главе будет рассмотрено несколько парадигм разработки программного обеспечения для стандарта IEC 61499, а также будут обсуждаться следующие вопросы: как и где применять эти парадигмы. Остальная часть этой главы организована следующим образом: в разделе 4.2 представлены основные элементы конструкции функциональных блоков. После этого в разделах 4.3, 4.4 и 4.5 соответственно представлены три основные парадигмы проектирования: парадигма объектно-ориентированного программирования, парадигма компонентно-ориентированного проектирования и парадигма сервис-ориентированной архитектуры. Эти

парадигмы иллюстрируются на примере системы обработки багажа в аэропорту. Сравнение же и обсуждение этих парадигм вы найдете в разделе 4.6.

4.2. Существенные элементы в ИЕС 61499 проектов функционального блока

С точки зрения разработки программного обеспечения, парадигмы являются моделями для реализации группы приложений с общими свойствами [20]. Основной целью парадигм проектирования программного обеспечения является повышение эффективности проектирования, внедрения и развертывания программного обеспечения за счет увеличения возможности повторного использования и переносимости существующих управляющих программ. Специально разработанный для систем распределенной автоматизации стандарт ИЕС 61499 [9] направлен на устранение известных недостатков в отношении распределенного управления в стандарте ИЕС 61131-3 [10]. Основной особенностью стандарта ИЕС 61499 является функциональный блок, инициируемый событиями, в котором алгоритмы управления должны быть инкапсулированы в функциональные блоки и могут выполняться только с помощью специальных механизмов запроса, управляемых событиями [18]. Чтобы применять парадигмы проектирования, необходимо обобщить некоторые общие элементы, используемые в программной модели ИЕС 61499. В проекте ИЕС 61499 элементы сгруппированы в двух основных компонентах, а именно в модульности и связи, как показано на рисунке 4.1. Модульность является фундаментальным элементом для проектирования программ распределенной автоматизации. Программа, основанная на модульных блоках, обеспечивает возможность многократного использования этих компонентов с минимальными усилиями для настройки [7]. Методология модульного проектирования направлена на повышение эффективности разработки программного обеспечения и повышение качества получаемого программного обеспечения, что достигается за счет повторного использования проверенных модулей. Связь обеспечивает механизмы обмена данными между модулями, такими как вызовы функции, события и обмен сообщениями.

Связь предназначена для улучшения совместимости и взаимодействия между модулями. Модульность в сочетании с коммуникацией формирует распределенную парадигму проектирования программного обеспечения. Парадигма проектирования программного обеспечения – это модель для решения класса задач, которые имеют общий набор характеристик [8]. Парадигма разработки программного обеспечения должна указывать, как создавать отдельные модули и как объединять эти модули в законченные программы.



Рисунок 4.1. Основные элементы распределенных программ автоматизации

В области компьютерных наук уже разработано много парадигм программирования, таких как структурированные, императивные, декларативные, функциональные, логические и объектно-ориентированные парадигмы программирования [16]. Эти парадигмы программирования обычно хорошо подходят для одной или нескольких конкретных областей. Программы могут разрабатываться в мультипарадигмальных стилях, а язык программирования может поддерживать множество парадигм. Некоторые парадигмы уже оказали значительное влияние на область промышленной автоматизации. Например, объектно-ориентированная парадигма официально поддерживается третьим изданием стандарта IEC 61131-3 [21]. Объектно-ориентированная парадигма также применяется к стандарту IEC 61499 в нескольких приложениях, таких как система обработки багажа в аэропортах [2], система управления процессами [1], интеллектуальные сети [22] и системы автоматизации зданий [19]. Что еще важнее, общее руководство по проектированию, как ожидается, будет обобщено исходя из существующих подходов, чтобы обеспечить всеобъемлющие рекомендации по применению различных парадигм проектирования в программировании распределенной автоматизации. В следующих нескольких разделах будут проиллюстрированы три основные парадигмы. Прежде всего, будет рассмотрена наиболее распространенная парадигма проектирования – объектно-ориентированное программирование (ООР). Во вторую очередь будет рассматриваться парадигма проектирования на основе компонентов (CBD), поскольку функциональные блоки IEC 61499 являются естественными компонентами. Наконец, для промышленной автоматизации вводится парадигма сервис-ориентированной архитектуры (SOA). Пока какой-либо работы по внедрению SOA не ведется ни на уровне разработки, ни на уровне исполнения. Все парадигмы иллюстрируются примером использования системы обработки багажа в аэропорту.

4.3. Применение парадигмы объектно-ориентированного программирования в ИЕС 61499 функциональных блоков

4.3.1. Предпосылки объектно-ориентированного программирования

Объектно-ориентированная парадигма широко применяется в таких языках программирования, как C++, Java, C# и Python. Парадигма ООР изначально была предложена для улучшения возможности повторного использования и повышения эффективности языков программирования.

Ключевым понятием объектно-ориентированной парадигмы является объект [11]. Объект представляет собой повторно используемый модуль, который состоит из двух частей – кода и данных. Часть кода включает в себя функциональное поведение объекта, а часть данных содержит текущее состояние объекта. Обычно для каждого типа объекта определяется класс, который содержит единственную копию части кода и определение части данных. Новый экземпляр, то есть копия части данных и ссылка на общую часть кода, инициализируется для каждого конкретного объекта. Ключевой особенностью ООР является инкапсуляция, с помощью которой системы делятся на модульные объекты, данные которых физически изолированы. Методы, как интерфейсы между объектами, являются функциями, которые используются для доступа к данным, инкапсулированным в экземпляр объекта. Наследование – это возможность расширять классы объектов в подклассы с помощью новых методов и данных.

Эти функции обеспечивают возможность повторного использования определенной части существующего кода, но есть некоторые исключения. Объект может иметь числовые определения, так как требования варьируются для каждого проекта. Становится чрезвычайно сложно управлять всеми вариациями объектов в долгосрочной перспективе с использованием парадигмы ООР. Также любое изменение интерфейса в объектном методе потребует значительного времени на реконструкцию. Трудно определить, действительно ли можно сэкономить время, применяя парадигмы ООР.

4.3.2. Применение понятий объектно-ориентированного программирования в стандарте ИЕС 61499

Применение объектно-ориентированной парадигмы в стандарте ИЕС 61499 основано на рассмотрении функциональных блоков ИЕС 61499 в качестве объектов (как продемонстрировано экспериментально [2, 19]). Прежде всего, каждый тип функционального блока ИЕС 61499 сопоставляется с классом. В ИЕС 61499 определены три типа функциональных блоков: базовый функциональный блок (BFB), функциональный блок интерфейса службы (SIFB) и составной функциональный блок (CFB). И BFB, и SIFB являются атомарными функциональными блоками, но предназначены для разных целей. BFB обычно используются для представления алгоритма управления на основе

конечного автомата, поведение которого определяется диаграммами управления выполнением (ECC). SIFB используются в качестве «черных ящиков», которые управляются последовательностями действий. Как BFB, так и SIFB могут использоваться для определения классов, как показано на рисунке 4.2.

Поскольку все типы функциональных блоков имеют идентичные интерфейсы, каждый вход события любого типа функционального блока сопоставляется с методом. Входные данные, связанные с этим входом события, используются в качестве входных параметров метода. Методы вызываются соединениями событий в подходе IEC 61499. В BFB методы реализуются алгоритмами ЕС, которые программируются на языках программирования IEC 61131-3. В SIFB методы могут быть реализованы на любом языке программирования, управляемом сервисными последовательностями. Для возврата параметров из методов должен быть создан выход события, а выходные данные, связанные с этим выходом события, рассматриваются как возвращаемые параметры. При создании нового экземпляра FB данные сохраняются как внутренние переменные в BFB и SIFB. CFB предоставляют средства для достижения наследования по частям в приложениях IEC 61499. Как показано на рисунке 4.3, исходный BFB или SIFB инкапсулируется новым CFB, а исходный интерфейс напрямую отображается в новом интерфейсе CFB. Новые методы могут быть реализованы в отдельном BFB или SIFB как часть внутренней сети FB CFB. Однако есть одна небольшая проблема: как обрабатывать данные между базовыми и расширенными FB. Поскольку в CFB нет внутренней переменной, переменные данных должны оставаться в базовом классе (BFB или SIFB). Для доступа к этим внутренним переменным данных вводятся дополнительные каналы PUBLISH и SUBSCRIBE между исходным и расширенным FB, как показано на рисунке 4.3. Когда некоторые внутренние переменные требуются расширенным FB, значения извлекаются из базового FB, а обновленные переменные возвращаются в базовый FB с использованием выделенной пары PUBLISH и SUBSCRIBE SIFB. Внутренние переменные, введенные новыми методами, будут помещены в расширенные FB. Если требуется дополнительное расширение, вводятся дополнительные расширенные функциональные блоки, а также пара канала связи – PUBLISH / SUBSCRIBE.

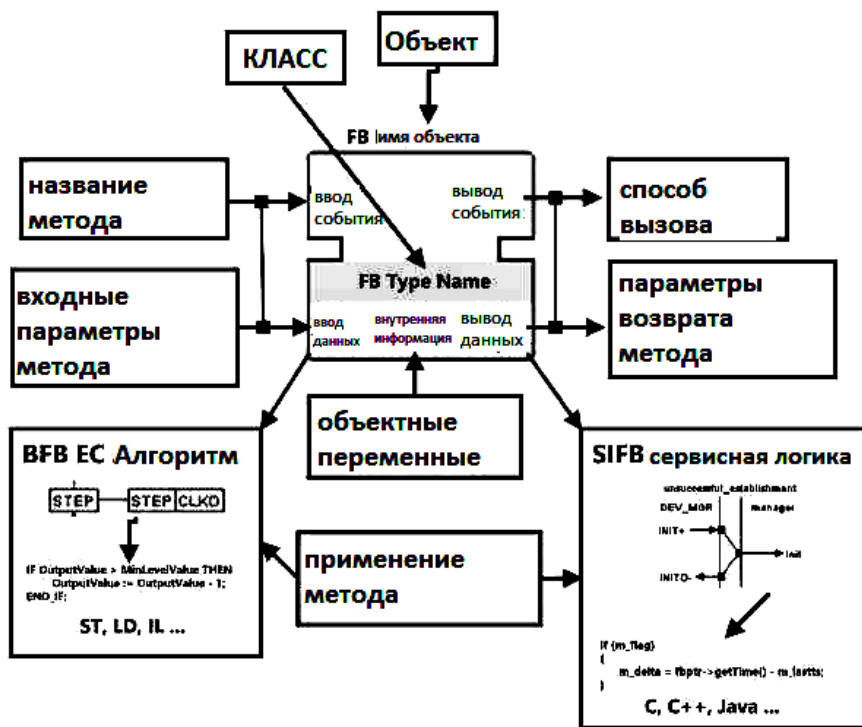


Рисунок 4.2. Отображение FB IEC 61499 для функций OOP

В качестве альтернативы все внутренние переменные могут быть помещены в отдельные данные FB. Как показано на рисунке 4.4, независимо от того, какие переменные запрашиваются из базового FB или расширенного FB, значения будут передаваться в этот конкретный FB через каналы связи PUBLISH / SUBSCRIBE. Переменные данных будут записаны обратно в FB данных, когда их значения будут изменены во время процессов. Этот шаблон проектирования предоставляет общее решение для всех базовых и унаследованных объектов, однако тут есть некоторые недостатки. Во-первых, больше времени будет потрачено на проектирование и разработку, так как все переменные данных должны быть настроены вручную с помощью PUBLISH / SUBSCRIBE SIFB. Во-вторых, разделенные данные и код создадут дополнительные коммуникационные издержки, которые влияют на производительность системы.

Для обоих подходов базовый функциональный блок является обязательным для добавления расширенных функций в целях обеспечения общего наследования. Первый подход создает меньше накладных расходов на связь, но ожидается, что больше действий, выполненных вручную, потребуется для индивидуальной связи между базой и расширенным функциональным блоком. Второй подход подходит для всех сценариев, но скорость выполнения будет

зависеть от большого объема обмена данными в типе унаследованных функциональных блоков.

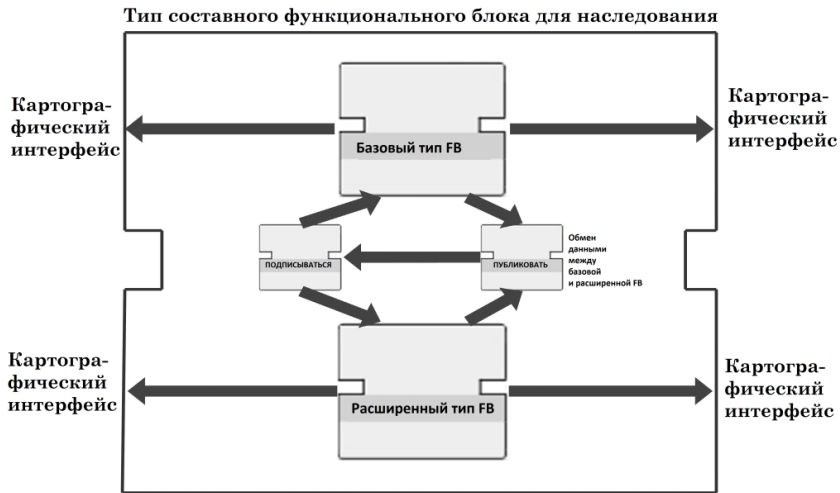


Рисунок 4.3. Отображение CFB для наследования ООР

Общий же подход состоит в том, чтобы рассматривать объект в функциональных блоках IEC 61499 как физическое устройство на предприятии. Поскольку объекты определяются как пассивные функции, для организации этих объектов в логическом порядке необходимы блоки управления. Дополнительные функциональные блоки введены для выполнения функций планирования для производственных процессов. Эти функциональные блоки планирования отвечают за соединение моделей устройств с физическими операциями, вызывая методы из объектов. Функциональные блоки планирования могут быть развернуты на нескольких PLC для обеспечения распределенного управления.

4.3.3. Тематическое исследование парадигмы объектно-ориентированного программирования

Объектно-ориентированная парадигма будет проиллюстрирована при тематическом исследовании системы обработки багажа в аэропорту (BHS). Схема въезда в аэропорт BHS приведена на рисунке 4.5. Мешки, выгруженные из самолета, будут переданы на входящую BHS с конвейера IB101. Мешки перемещаются от конвейера IB101 к IB105 через защитную дверь IB1D1 и, наконец, достигают ленты выдачи багажа IB1. В этой входящей BHS определены две зоны: воздушная сторона от IB101 до конвейера IB103 с защитной дверью IB1D1 и открытая сторона от IB104 до IB1. Для работы в аварийных ситуациях установлены три кнопки аварийной остановки. При

нажатию кнопки аварийной остановки все устройства в этой зоне будут немедленно остановлены. На общественной стороне установлены две кнопки аварийной остановки (ESTOP1 и ESTOP2) и одна на противоположной стороне (ESTOP3).

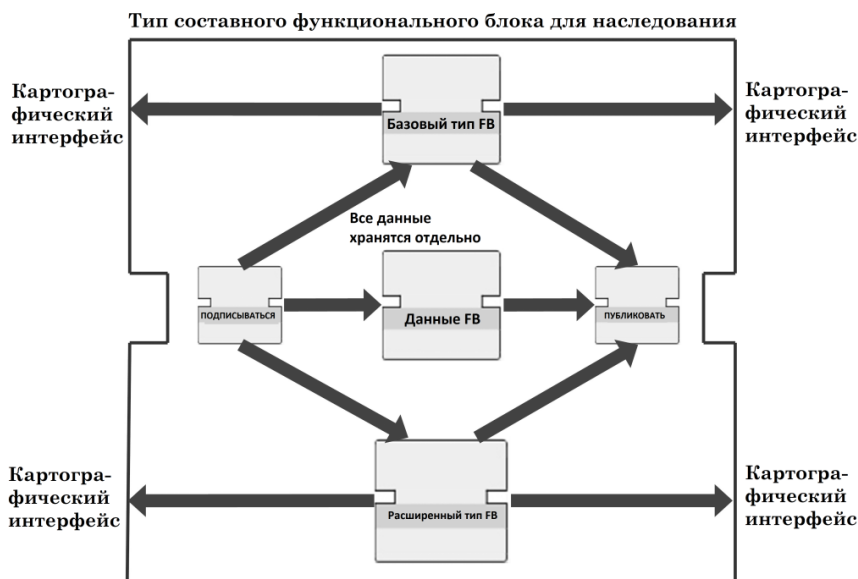


Рисунок 4.4. Альтернативное решение для обработки данных для наследования ООР

Проект входящего ООР BHS приведен на рисунке 4.6. В начале цепочки событий A SIFB FB_BHS_INPUTS запрашивает входные значения из полевых шин за фиксированный период. К таким входам относятся датчики фотоглаза и сигналы обратной связи при запуске двигателя с конвейеров, состояние кнопок аварийной остановки, датчики приближения и сигналы обратной связи при запуске двигателя от дверей и кнопок управления системой (запуск, остановка и сброс). В конце цепочки событий другой SIFB FB_BHS_OUTPUTS записывает выходные значения обратно в полевые шины, которые содержат сигналы запуска двигателя для конвейеров и сигналы запуска двигателя «вперед» и «назад» для дверей. Три подхода (типы функциональных блоков) определены в подходе ООР: аварийная остановка, конвейер и дверь. Тип функционального блока аварийной остановки (а именно FB_EStop_OOP, как показано на рисунке 4.6) принимает входные данные кнопки E-Stop и обновляет данные кнопки аварийной остановки, которые хранятся внутри. Конструкция функционального блока конвейера типа FB_Conveyor_OOP показана на рисунке 4.7. В качестве входных данных используются данные фотоэлемента датчика и сигнал обратной связи при работе двигателя, а также

генерируется выходной сигнал двигателя и текущее состояние конвейера. Согласно определению парадигмы OOP IEC 61499, в интерфейсе перечислены четыре входа (метода) события: INIT, RUN, STOP и UPDATE. Каждый вход события будет переключаться в отдельное состояние ЕС с выделенным алгоритмом ЕС и возвращаться в состояние START сразу после обработки этого события. Состояния RUN, STOP и UPDATE совместно используют один выходной сигнал CNF для обновления выходного сигнала двигателя и состояния конвейера. Состояния RUN и STOP отвечают за запуск и остановку работающего конвейера. Когда входные значения поступают из полевых шин, событие UPDATE будет инициировано для обновления входных значений, хранящихся в функциональном блоке.

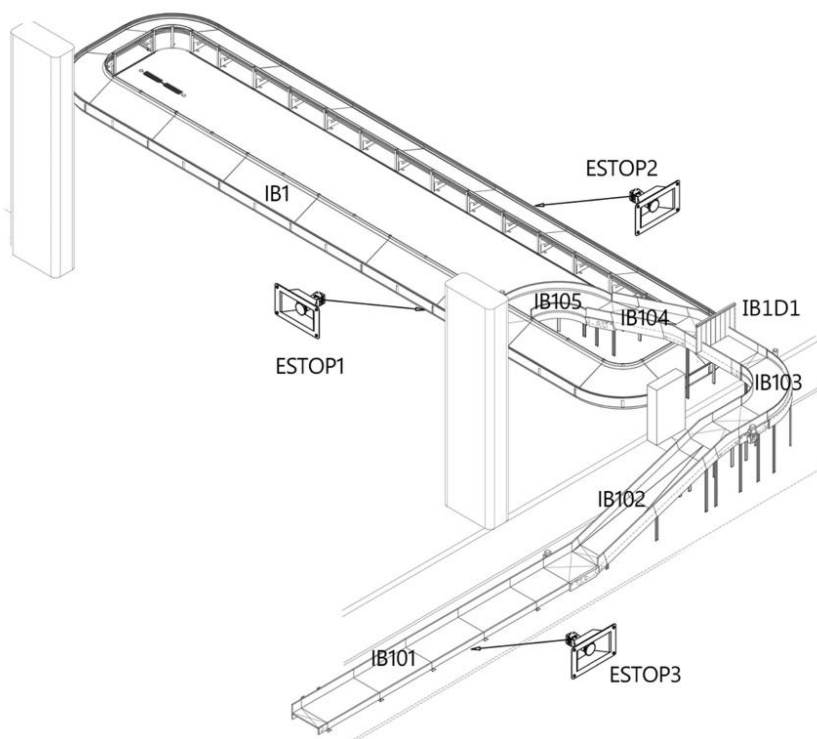
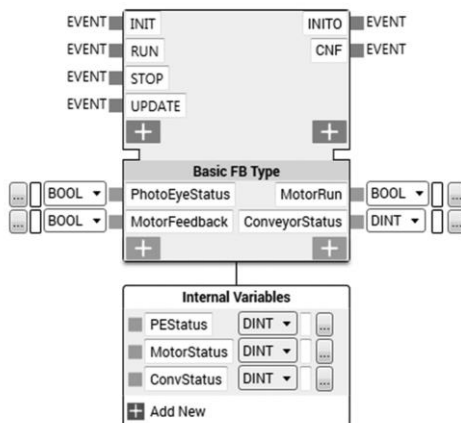
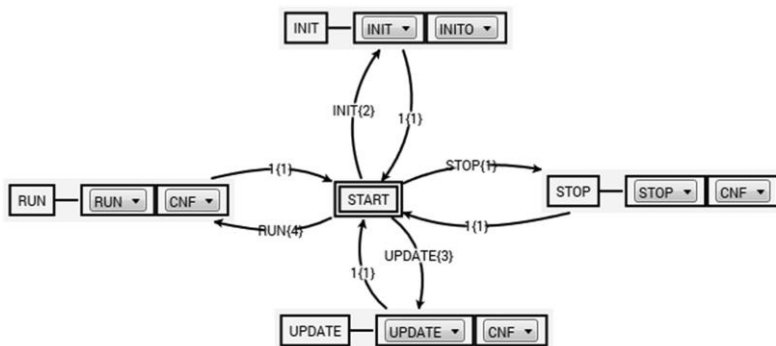


Рисунок 4.5. Пример схемы системы обработки багажа в аэропорту

Дверь класса FB_Door_OOP имеет аналогичную конструкцию в сравнении с конструкцией конвейера FB. Еще одна особенность, которая требуется для защитных дверей, заключается в том, что дверь должна двигаться в прямом (открытом) и обратном (закрытом) направлениях. Как показано на рисунке 4.8, вместо использования одного состояния RUN в конструкции ECC есть два состояния ЕС – RUNF и RUNR, служащие для управления движением дверей в заданном направлении.



(a) Interface Design



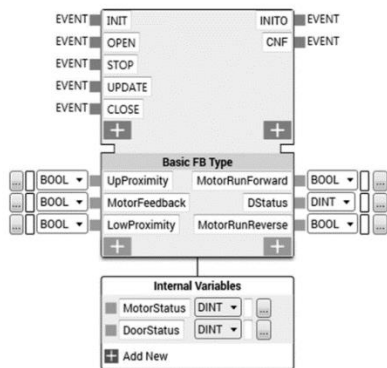
(b) ECC Design

Рисунок 4.7. Дизайн входящего конвейера BHS OOP FB

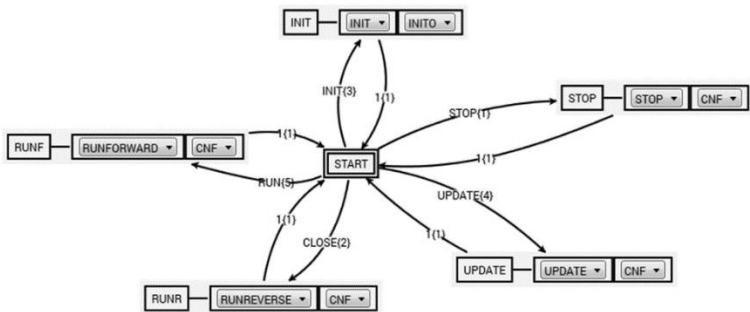
Теперь все объекты созданы, но еще должен быть планировщик, чтобы обеспечить функциональные возможности системного уровня путем связывания объектов. Планировщик FB FB_SEQ_OOP представлен, как показано на рисунке 4.9. Он использует текущее состояние всех развернутых конвейеров и сигналы управления системой в качестве входных данных и генерирует сигналы управления для всех устройств, включая конвейеры и двери. При каждом цикле сканирования I/O событие UPDATE активируется SIFB FB_BHS_INPUTS, и текущее состояние полевых устройств обновляется. Если нажата какая-либо кнопка аварийной остановки, ECC перейдет в состояние ESTOP, и будет выдан выходной сигнал события ESTOPALL, чтобы

все устройства немедленно прекратили работу. В качестве альтернативы, если не возникнет аварийной ситуации, ЕСС вернется в состояние START.

В ЕСС определены две последовательности процессов: последовательность запуска и последовательность остановки. Для последовательности запуска конвейеры запускаются в обратном порядке, причем сначала запускается самый нижний конвейер или каскадный запуск. Каскадный запуск предназначен для предотвращения застревания сумок, вызванного их врезанием в остановленные нижестоящие конвейеры. После нажатия кнопки запуска системы ЕСС откроет защитную дверь IB1D1 и запустит все конвейеры с IB1 до IB101. Для последовательности остановок конвейеры останавливаются в прямом порядке, так что самый верхний конвейер останавливается первым. Эта функция предназначена для удаления всех сумок с конвейера до его остановки. Как только сигнал для остановки системы получен, конвейеры останавливаются один за другим от IB101 до IB1. Защитная дверь IB1D1 закрывается после остановки всех конвейеров. После этого BHS будет перезапущена в обратном порядке, чтобы возобновить нормальную работу.

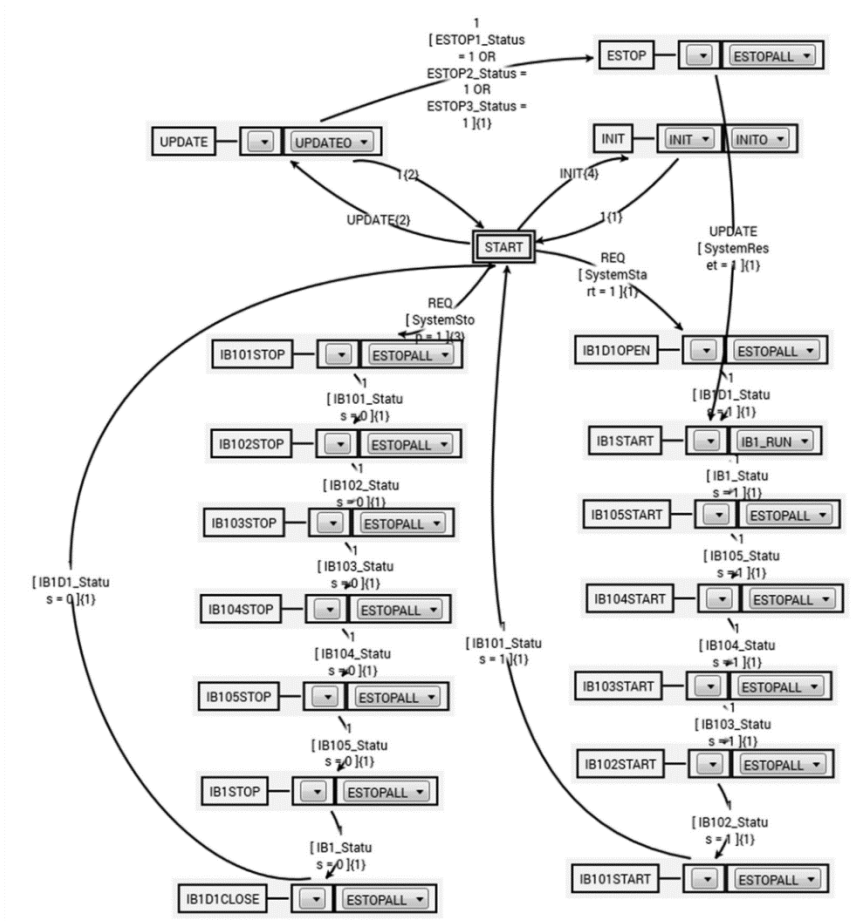


(a) Interface Design



(b) ECC Design

Рисунок 4.8. Дизайн входной двери BHS OOP FB



ECC Design

Рисунок 4.9. Дизайн управления входящей последовательностью OOP BNS FB

Таким образом, парадигма OOP начинается с разработки индивидуальных классов объектов. Для каждого типа функционального блока объекта все входные значения должны эшироваться внутри этого функционального блока, а статус при этом должен храниться как локальные переменные. Поскольку функциональные блоки пассивны, вводится планировщик для генерации событий для запуска объектов.

4.4. Принятие компонентно-ориентированной парадигмы дизайна для функциональных блоков IEC 61499

4.4.1. Предпосылки компонентно-ориентированного дизайна

Компонентное проектирование направлено на решение различных задач одного и того же класса в объектно-ориентированном программировании. Основанная на компонентах парадигма предназначена для разделения функциональных возможностей, а отдельный программный компонент содержит набор семантически связанных функций и данных. Компоненты связываются друг с другом через предопределенные интерфейсы [5]. Интерфейсы обеспечивают использование компонентов в «черном ящике» без указания деталей реализации.

Компонент может быть легко заменен другими компонентами во время разработки и выполнения, поскольку его интерфейс остается неизменным. Библиотека программного обеспечения может быть создана из набора компонентов, что обеспечивает высокую степень повторного использования и надежность.

Стандарт IEC 61499 следует принципу парадигмы проектирования на основе компонентов. Типы функциональных блоков IEC 61499 имеют четко определенные интерфейсы: входы событий, выходы событий, входы данных и выходы данных [5]. Любой функциональный блок может быть заменен другим функциональным блоком с идентичными интерфейсами, посылая команды управления ресурсам IEC 61499, даже совершенно разные [3, 14, 17] алогормы, инкапсулированные в этих функциональных блоках. Никакая модификация алгоритмов внутри функциональных блоков не повлияет на конфигурацию системы. Это обеспечивает большую гибкость, поскольку разработчики программного обеспечения могут разрабатывать управляющие программы на основе интерфейсов функциональных блоков, даже если реализации этих программных компонентов не готовы.

4.4.2. Компонентно-ориентированные принципы разработки для функциональных блоков IEC 61499

Принципы конструирования компонентной парадигмы для приложений IEC 61499 можно резюмировать, уделив внимание двум важным моментам. Во-первых, основанная на компонентах парадигма направлена на группирование семантически связанных данных и функций для минимизации обмена данными между модулями. В отличие от объектно-ориентированной парадигмы, компонентная парадигма ориентирована на расщепление систем по функциональным возможностям, а не по физическим объектам. Поскольку большинство данных, необходимых для функционального блока IEC 61499 в парадигме CBD, хранятся внутри, общее количество данных между функциональными блоками значительно сокращается. Иерархическая

программная архитектура обычно используется для минимизации соединений между функциональными блоками.

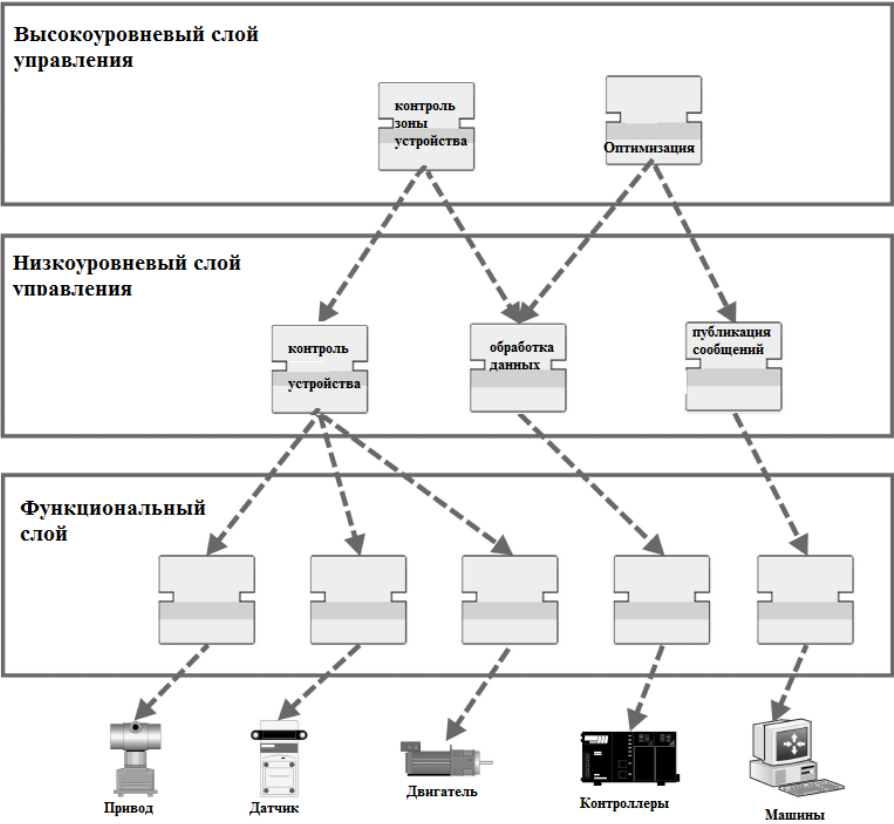


Рисунок 4.10. Многослойная парадигма CBD для IEC 61499

Как показано на рисунке 4.10, предложена многоуровневая программная архитектура для парадигмы проектирования на основе компонентов IEC 61499. Для применения CBD IEC 61499 три уровня являются минимальными требованиями: функциональный уровень, уровень управления низкого уровня (LLC) и уровень управления высокого уровня (HLC). Функциональный уровень содержит функциональные блоки для доступа к датчикам, исполнительным механизмам, двигателям и приводам через полевые шины, обслуживающие внешние коммуникационные шлюзы и библиотеку функций многократного использования для обработки строк и сложных математических вычислений.

Все FB, используемые для прямого управления физическим оборудованием, размещаются на функциональном уровне. Средний уровень – это уровень управления низкого уровня. Уровень LLC отвечает за обеспечение основных функций управления для фабрики. Функциональные блоки на уровне LLC обычно представляют физическое устройство в цехе. Этап процесса может потребовать взаимодействия между несколькими функциональными блоками как функционального уровня, так и уровня LLC, например, обеспечивая управление конвейером для BHS аэропорта. Эти шаги обычно ориентированы на отдельные устройства, поэтому взаимодействие между различными устройствами не требуется.

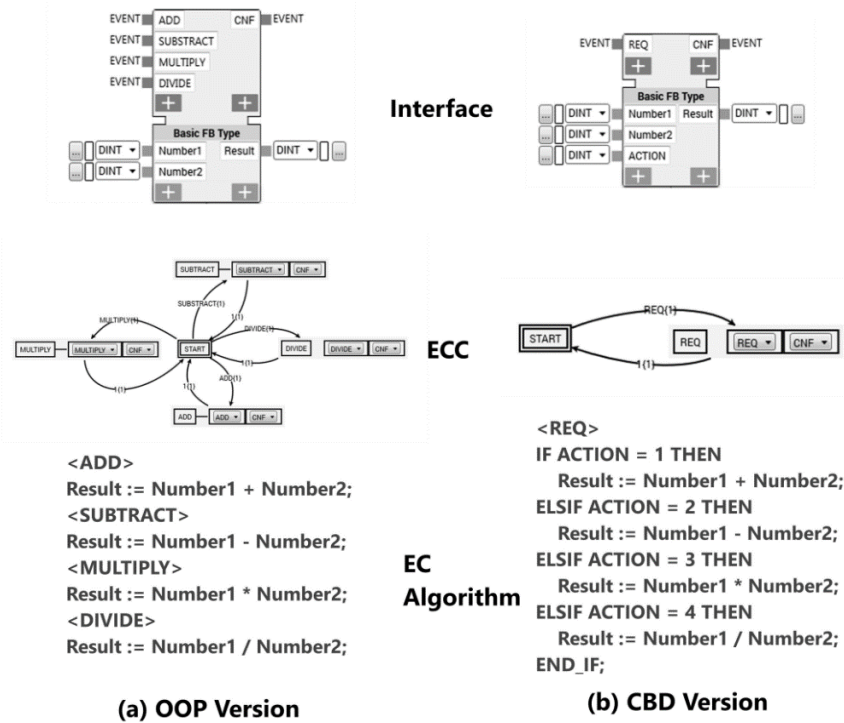


Рисунок 4.11. Пример конструкции OOP и CBD функционального блока

Верхний уровень – это уровень управления высокого класса, отвечающий за выполнение сложных функций путем координации нескольких устройств, например, распределения нагрузки в BHS аэропорта. Уровень HLC координирует компоненты для выполнения функций системного уровня. Благодаря применению этой многоуровневой архитектуры для CBD в функциональных блоках IEC 61499 соединения между функциональными блоками значительно сокращаются.

Во-вторых, интерфейсы между функциональными блоками должны быть абстрактными и общими. Вместо использования выделенного события и алгоритма для каждого вызова метода в ООР, несколько функций совместно используют один алгоритм с одним общим вводом события в CBD. Например, функциональный блок для выполнения простого вычисления между двумя входными числами и возврата номера результата показан на рисунке 4.11а. В подходе ООР создается отдельный вход события для сложения, вычитания, умножения и деления функций. В версии CBD (рисунок 4.11b) эти входы событий объединяются в одно событие REQ, связанное с переменной ACTION ввода данных, чтобы указать выбор действий (1-сложение, 2-вычитание, 3-умножение, 4-деление). Все алгоритмы размещены в одном алгоритме ЕС и различаются путем интерпретации значения переменной ACTION.

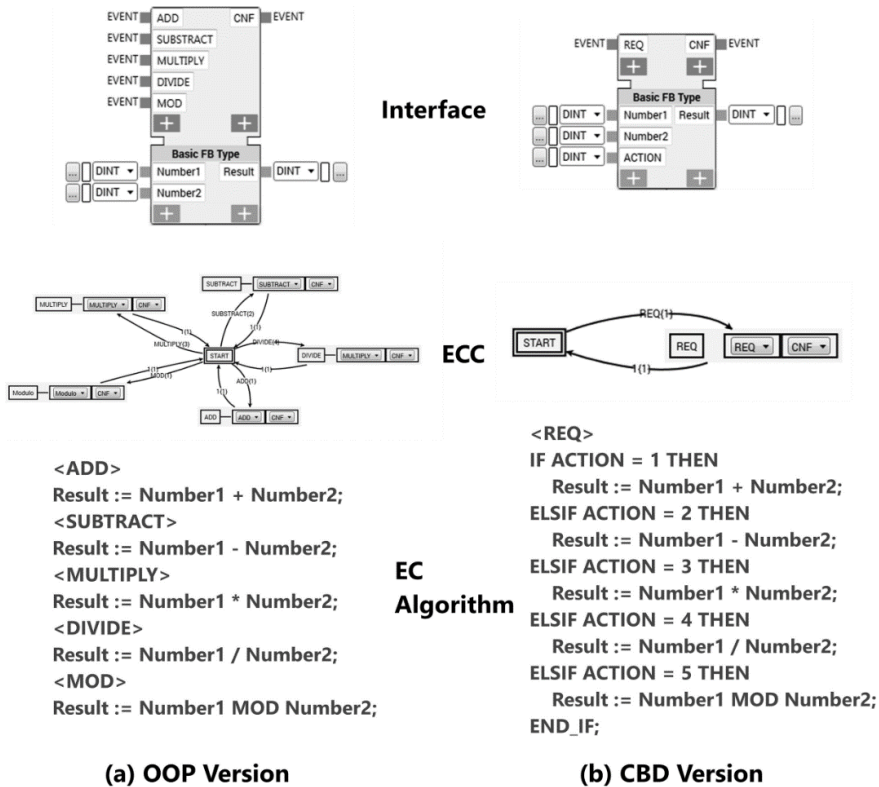


Рисунок 4.12. ООР в сравнении с конструкцией функционального блока CBD: вставка нового примера

Несмотря на то, что сложности отдельных алгоритмов возрастают, функциональные блоки ИЕС 61499 по-прежнему могут быть легко расширены без изменения каких-либо соединений в конфигурациях системы. Например,

если необходимо вычислить модуль для FB, представленного на рисунке 4.12, в подходе OOP вводится новый входной MOD-сигнал, и ECC модифицируется новым введенным ЕС-состоянием MODULO. В системе CBD не требуется никаких изменений для интерфейса FB, и вводится новое предопределенное значение (например, 5) для ввода данных ACTION. В алгоритме ЕС операция по модулю вставляется путем проверки значения ACTION (в данном случае равного 5). Поскольку для интерфейса не требуется никаких изменений, добавляются новые функциональные возможности без каких-либо изменений в конструкции сети функциональных блоков.

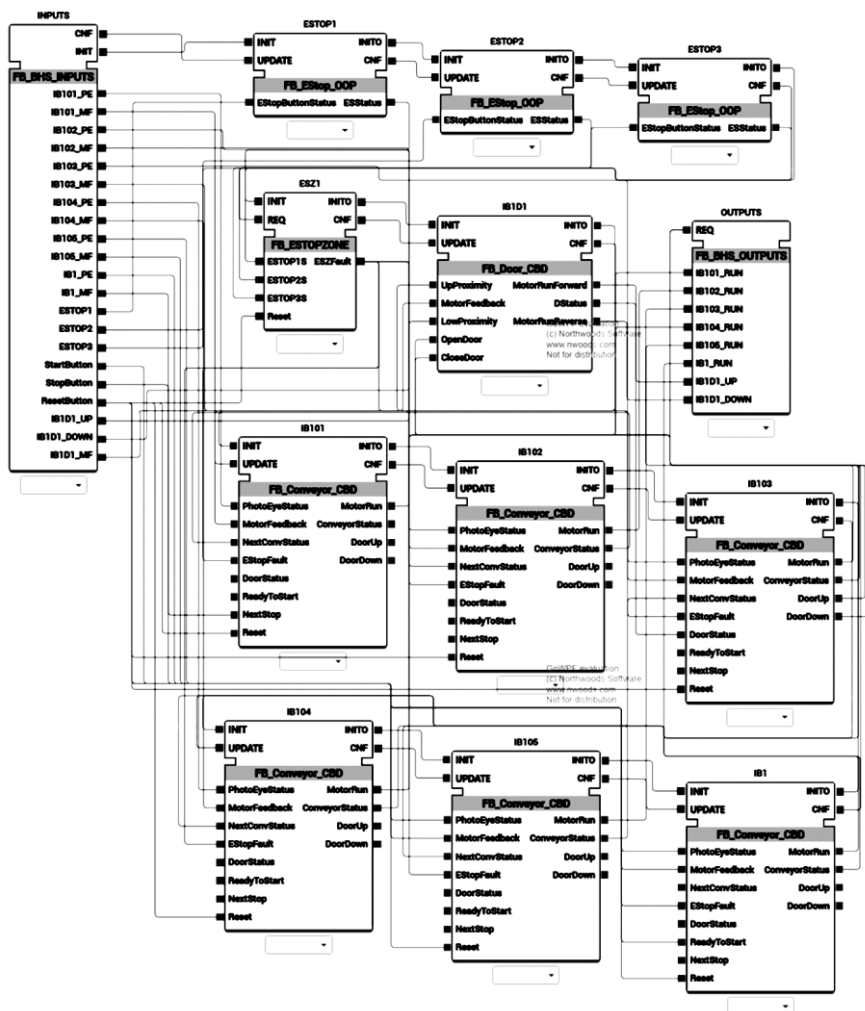
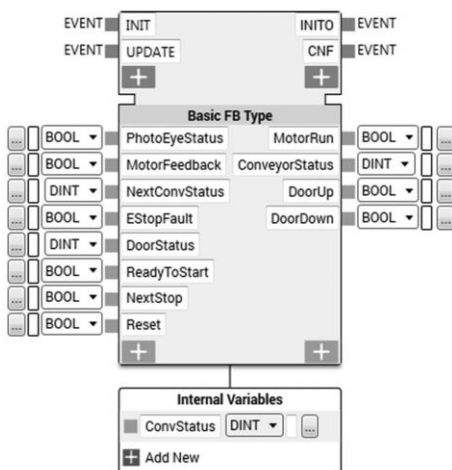


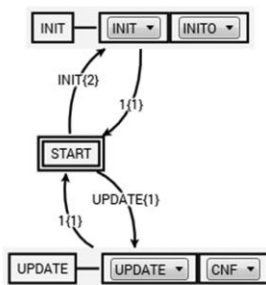
Рисунок 4.13. Пример дизайна входящего BHS CBD

4.4.3 Тематическое исследование компонентно-ориентированной парадигмы дизайна

Чтобы объяснить, как применяется предложенная архитектура CBD, в предыдущем разделе был дан пример BHS аэропорта (рисунок 4.13). В подходе CBD отсутствует функциональный блок планирования для координации систем управления. Вместо этого координация распространяется на несколько распределенных программных компонентов. Добавлен новый компонент FB FB_ESTOPZONE, чтобы обеспечить управление зоной аварийной остановки для информирования BHS при нажатии любой кнопки ESTOP. Система возобновит работу после высвобождения кнопки ESTOP и нажатия кнопки сброса системы.



(a) Interface Design



(b) ECC Design

Рисунок 4.14. Дизайн входящего конвейера BHS CBD FB: (a) Дизайн интерфейса, (b) Дизайн ECC

Версия CBD типа функционального блока управления конвейером приведена на рисунке 4.14. Одно очевидное отличие версии CBD от версии ООР состоит в том, что все управляющие сигналы используют переменные данных вместо событий. Входы и выходы событий и ECC упрощаются путем применения унифицированных интерфейсов. В ECC перечислены только два состояния: INIT, которое отвечает за назначение начальных значений, и UPDATE, в котором реализована вся логика управления. Вся информация, требуемая конвейерами, указывается в качестве входных данных в интерфейсе FB, например, состояние нижестоящего конвейера, состояние аварийной

остановки, связанное состояние защитной двери, а также сигналы запуска, остановки и сброса системы. Как видно из рисунка 4.13, некоторые устройства могут не требовать ввода и вывода некоторых данных. Например, защитная дверь IB1D1 прикреплена к конвейеру IB103, другие конвейеры не имеют никакой защитной двери. Также сигнал запуска системы подается только на конвейер IB1 для каскадного запуска, а сигнал остановки системы подается только на конвейер IB101 для очистки всех оставшихся мешков в системе.

Подводя итог вышесказанному о парадигме CBD, можно констатировать, что возможность многократного использования улучшена путем применения унифицированных интерфейсов для функциональных блоков. Сложности скрыты от уровня приложения, поскольку вся логика управления заключена в алгоритмы ЕС. Недостатком является то, что для всех экземпляров требуются не все входы и выходы, но они все равно должны быть частью интерфейса FB.

4.5. Представление новой парадигмы архитектуры для обслуживания широкого круга запросов для функциональных блоков IEC 61 499

4.5.1. Предпосылки архитектуры для обслуживания широкого круга запросов IEC 61499

В последние годы с принятием технологий связи на основе Ethernet на уровне цеха сервис-ориентированная архитектура (SOA) становится привлекательной в качестве основы для программных систем распределенной автоматизации [12]. Парадигма SOA состоит из набора программных компонентов, описания интерфейсов которых могут быть опубликованы и обнаружены [6]. Поставщики услуг и потребители слабо связаны между собой для того, чтобы обеспечить минимальную зависимость между услугами. Характер взаимодействия между службами определяется в договоре об обслуживании, который может быть опубликован в архиве услуг. Одни службы можно комбинировать с другими службами (называемыми оркестровкой и хореографией служб) и оставлять в таком состоянии, когда сообщение отсутствует. Такие гибкие парадигмы также могут быть представлены с использованием артефактов IEC 61499. Мы рассматриваем каждый функциональный блок IEC 61499 как сервис, и его интерфейс определен в контракте на сервис. Так как среда обнаружения и сервисное хранилище должны поддерживаться средой выполнения, то в этой главе не будет обсуждаться, как публиковать и обнаруживать сервисные контракты. Благодаря службам, которые используют только механизмы обмена сообщениями для обмена данными, подобно подходу OOP, каждый вход события определяется как тип сообщения, и все входы данных, связанные с этим событием, считаются содержимым сообщения. Поскольку соединения в стандарте IEC 61499 являются однонаправленными, для подключения адаптера IEC 61499 требуется отдельная пара соединений событий и данных для возврата ответных сообщений.

4.5.2. Как применить понятия архитектуры для обслуживания широкого круга запросов в разработке системы IEC 61499

Как показано на рисунке 4.15, парадигма SOA для IEC 61499 также предлагается в качестве многоуровневой архитектуры. Хотя доступ к сервисам должен осуществляться из любого места, общая гибкая конструкция системы может привести к снижению производительности. Четыре уровня – это минимум, необходимый для применения парадигмы SOA IEC 61499. Базовый уровень – это место, где требуется обработка данных. Поскольку сервисы не сохраняют состояние, когда они не активированы, уровень данных отвечает за хранение сервисов. На уровне данных допускаются только BFB и SIFB, поскольку в CFB нет внутренней переменной. Поверх базового уровня находится слой компонентов, где реализованы часто используемые функции. Такие компоненты включают в себя функции интерпретации сообщений, функции ввода-вывода и обработки данных и другую многократно используемую логику для сервисов. Следующий уровень – сервисный уровень, на котором расположена управляющая логика для систем автоматизации. Функциональные блоки на сервисном уровне могут вызывать другие функциональные блоки на уровне данных и компонентном уровне.

Наконец, верхний слой – это технологический слой, где описаны процессы. На уровне процессов CFB используются для инкапсуляции функциональных блоков на уровне обслуживания для выполнения полных физических процессов, известных как оркестровка и хореография услуг. В парадигме SOA в конфигурациях системы IEC 61499 показаны только физические процессы предприятия, а сложности логики управления скрыты.

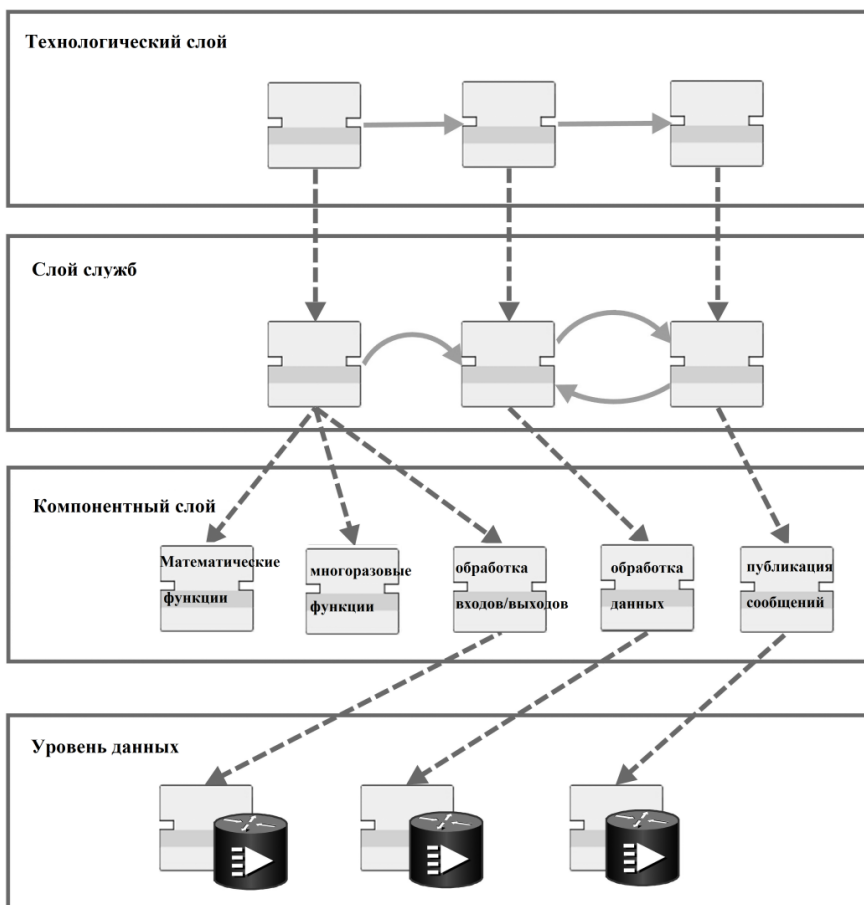


Рисунок 4.15. Архитектура парадигмы SOA для IEC 61499

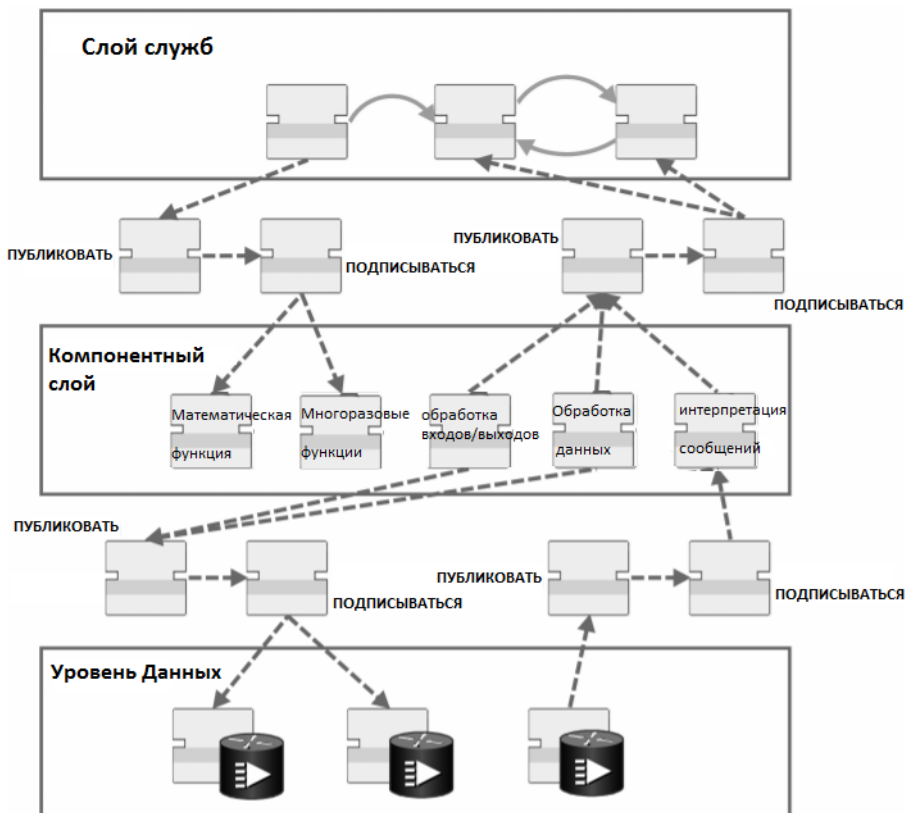
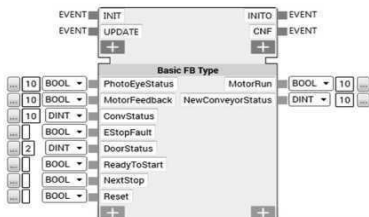


Рисунок 4.16. Обработка данных в парадигме IEC 61499 SOA

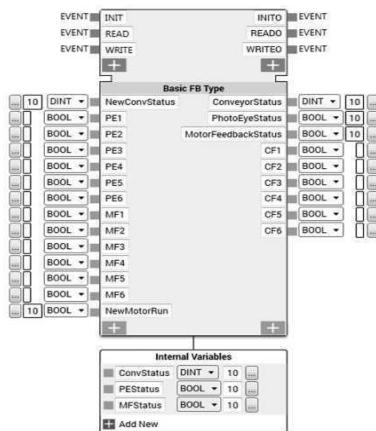
Поскольку все данные хранятся отдельно от управляющей логики, обработка данных между функциональными блоками на уровне данных, уровне компонентов и уровне обслуживания представляет собой проблему. Как показано на рисунке 4.16, если функциональный блок вызывается более чем одним функциональным блоком, должен применяться механизм PUBLISH / SUBSCRIBE, уже описанные в этой главе. Выходные данные из исходных функциональных блоков передаются PUBLISH SIFB с одинаковыми идентификаторами в один и тот же SUBSCRIBE SIFB с уникальными идентификаторами для ввода данных на другой стороне. Это связано с тем, что для каждого ввода данных любого функционального блока IEC 61499 разрешено только одно соединение для согласованности данных и семантической устойчивости.

4.5.3. Тематическое исследование парадигмы архитектуры для обслуживания широкого круга запросов

После описания парадигмы SOA для IEC 61499 пример тематического исследования BHS в аэропорту снова изменяется, как показано на рисунке 4.17, для соответствия парадигме SOA. Согласно парадигме SOA, каждый тип устройства делится на функциональные блоки для логики и данных. Функциональный блок данных хранит как входные и выходные значения, так и текущее состояние устройств в качестве локальных переменных. В отличие от парадигмы CBD, массивы используются для группировки данных с идентичным типом устройства. Количество экземпляров функциональных блоков в приложениях IEC 61499 сокращается путем применения парадигмы SOA. Дизайн интерфейса FB логики версии SOA имеет большое сходство с версией CBD. Есть только два небольших изменения: во-первых, переменные данных, такие как состояние устройства, объявляются как массивы; во-вторых, все переменные, обрабатываемые извне, должны возвращаться в функциональные блоки данных.



(a) Logic Part Interface Design



(b) Data Part Interface

Рисунок 4.18. Дизайн входящего FB конвейера BHS SOA

Служба управления конвейером показана на рисунке 4.18. Конструкция ЕСС функционального блока логической обработки для управления конвейером идентична его версии CBD, и интерфейс остается таким же, за исключением того, что введены массивы для статуса датчика положения глаз, состояния обратной связи двигателя, состояния конвейера и состояния защитной двери. Также добавлен новый вывод данных для возврата обновленного состояния конвейера обратно в функциональный блок обработки данных. Конструкция функционального блока обработки данных приведена на рисунке 4.18b. В ЕСС определены три состояния ЕС, и для процессов чтения и записи реализованы две задачи. Процесс считывания принимает последние входные значения фотоэлементов датчика, обратной связи при работе двигателя и обновляет состояние фотоглаза и состояние двигателя. В процессе записи заменяются существующие состояния конвейера и сигналы работы двигателя новыми значениями из функциональных блоков логической обработки.

От парадигмы CBD до парадигмы SOA для каждого типа устройства обязателен только один экземпляр функционального блока. Переменные состояния одного и того же типа устройства группируются в виде массива и обновляются функциональными блоками логической обработки. Переменные состояния должны быть возвращены в функциональные блоки обработки данных, поскольку функциональные блоки логической обработки не хранят никаких данных. Несмотря на разделение логики и данных, модификация одной части не повлияет на другую часть. Кроме того, количество экземпляров функциональных блоков и соединений в значительной степени уменьшается благодаря применению парадигмы SOA.

4.6. Краткое содержание парадигм проектирования IEC 61499

В предыдущих разделах проиллюстрированы три общие схемы проектирования для IEC 61499 и их применение в системах обработки багажа в аэропортах. В этом разделе обобщены некоторые события и приведены рекомендации касательно проектирования. Во-первых, парадигма объектно-ориентированного программирования идеальна для представления физических схем систем автоматизации. Устройства взаимно-однозначно сопоставляются с функциональными блоками в приложениях IEC 61499, и их физическое местоположение может отражаться сетями функциональных блоков. Парадигма ООР обеспечивает интуитивное представление о дизайне системы и применима для многих промышленных областей, где необходимо быстро определить местоположение устройств, например, для систем обработки материалов, систем автоматизации зданий и управления процессами. Возможность многократного использования значительно улучшена, если рассматривать объекты как функциональные блоки по сравнению с монолитным кодом. Однако хороший объектно-ориентированный дизайн не

обязательно создает хороший компонентный дизайн в области информатики и промышленной автоматизации [15]. При переходе от парадигмы ООР к парадигме CBD [13] широко применяются методы отображения класса на компоненте и метода на событии. Основные проблемы для парадигмы ООР состоят в том, что объекты слишком специфичны и содержат много ненужных деталей, для каждого нового проекта с немного отличающимися требованиями классы должны быть скорректированы, прежде чем их можно будет повторно использовать. Кроме того, сложно управлять вариациями одного и того же объекта. При применении парадигмы CBD зависимость между компонентами в значительной степени уменьшается, так что компоненты могут использоваться повторно, не мешая друг другу. Затраты на разработку снижаются за счет сборки систем из стандартных библиотек компонентов. Поскольку компоненты хорошо зарекомендовали себя, независимые модули и системы, построенные из таких компонентов, более надежны, чем системы, требующие каждый раз индивидуальные настройки. Другая проблема парадигмы ООР состоит в том, что некоторые функции, возможно, не должны быть реализованы как объекты или не могут быть отражены в физической компоновке. Например, простые математические вычисления – это просто монолитные коды, а обработчики связи – это виртуальные объекты, которые не существуют физически.

Парадигма CBD теоретически может быть применена к любым системам промышленной автоматизации, но необходимо соблюдать несколько принципов. Во-первых, специфичные для приложения методы должны быть удалены, чтобы сделать компоненты универсальными. Например, дополнительные требования могут быть заключены в отдельные функциональные блоки, даже если их функциональные возможности принадлежат другому функциональному блоку. Эти компоненты могут быть интегрированы путем инкапсуляции в CFB, чтобы уменьшить зависимость.

Во-вторых, имена событий должны быть общими, чтобы избежать частой модификации интерфейсов. В парадигме ООР события отображаются как методы. Но в парадигме CBD для поддержки интерфейса события заменяются входными данными для выбора подходящих методов. При переходе от парадигмы ООР к парадигме CBD, возможность повторного использования и гибкость улучшаются, как указано в таблице 4.1.

Таблица 4.1. Сравнение парадигм

Функции	ООР	CBD	SOA
Программный блок	Классы (объекты)	Программное обеспечение	Программное обеспечение
Функции	Наследование, полиморфизм	Инкапсуляция	Слабая связь
Интерфейс	Вызов методов	Предопределенность	Сервис, контракт
Допускающий	Библиотека	Компонент,	Сервис,

повторное использование	объектов	библиотека	репозиторий
Система, иерархия	Вложенные классы	Вложенные компоненты	Сервис, оркестровка, состав
Метамодель	Диаграмма классов (UML)	Сеть компонентов	Бизнес, процесс, выполнение, язык (BPEL), сервис, последовательность
Гибкость	Низко	Носитель	Высоко

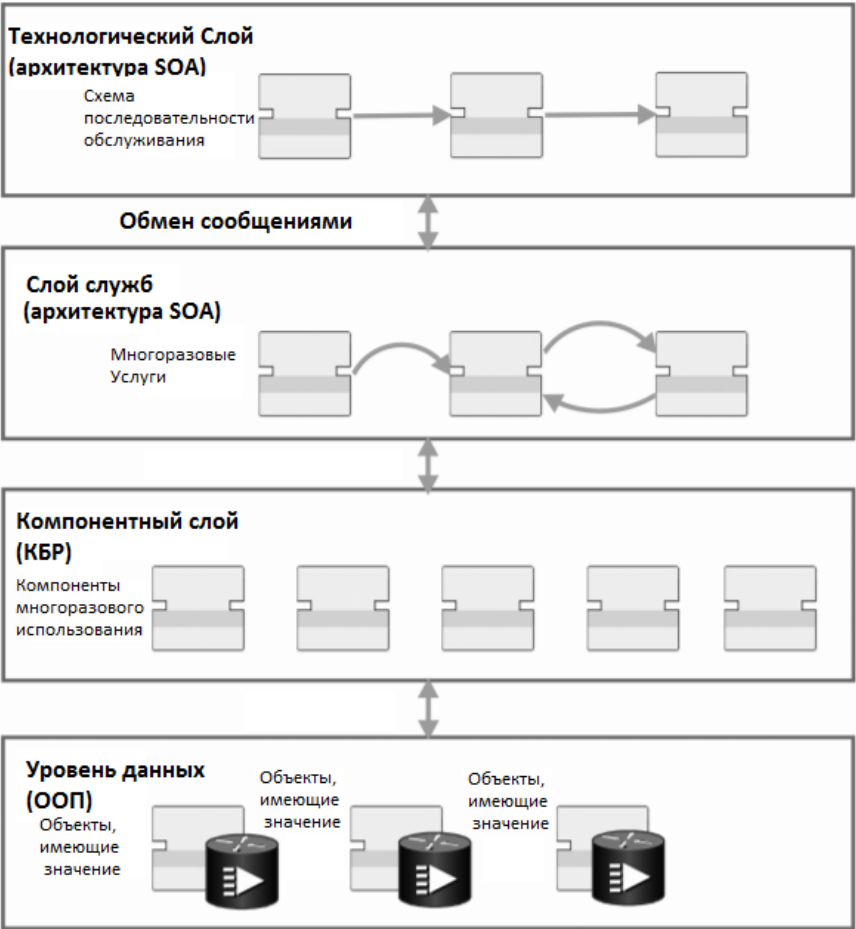


Рисунок 4.19. Общий сервис-ориентированный дизайн для IEC 61499

CBD и парадигма SOA предназначены для предоставления слабосвязанных и совместимых программных модулей. Хотя обе парадигмы имеют сходство,

парадигма SOA обеспечивает повышенную гибкость и функциональную совместимость [4]. Один экземпляр компонента в CBD управляет только одним ресурсом, а служба управляет набором ресурсов, которые в основном не имеют состояния. Служба работает как менеджер, который отвечает за создание, изменение и удаление набора экземпляров одного и того же типа или даже нескольких типов. Парадигму SOA можно считать развитием парадигм OOP и CBD с гибкой архитектурой и настраиваемыми интерфейсами. Как показано на рисунке 4.19, общая ориентированная на услуги конструкция для IEC 61499 представляет собой комбинацию парадигм OOP, CBD и SOA. На уровне данных парадигмы SOA функциональные блоки реализуются с использованием парадигмы OOP, где данные группируются по типам объектов. На уровне компонентов функциональные блоки действуют как шлюзы для соединения объектов и сервисов, предоставляя интерфейсы. Функциональные блоки реализуются в соответствии с парадигмой CBD с фиксированными интерфейсами. На сервисном уровне функциональные блоки выбирают, обрабатывают и возвращают связанные данные через интерфейсные шлюзы.

С точки зрения возможности повторного использования все три парадигмы основаны на концепции модульного дизайна, но обеспечивают разные уровни гибкости. Парадигма CBD повышает эффективность проектирования, определяя интерфейсы между объектами, предлагаемыми парадигмой OOP. При условии, что интерфейсы остаются неизменными, модификации отдельных функциональных блоков не влияют на дизайн системы. Парадигма SOA дополнительно расширяет гибкость, обрабатывая несколько экземпляров в одном сервисном блоке. Чтобы достичь этого, разделение между логикой и данными является обязательным. Однако производительность выполнения парадигмы SOA снижается по сравнению с парадигмами OOP и CBD, поскольку данные должны передаваться по сетям функциональных блоков. Необходимо учитывать баланс между эффективностью проектирования и производительностью: цель состоит в том, чтобы разделить функциональные возможности на модули для повышения эффективности при минимизации соединений между функциональными блоками для повышения производительности.

4.7. Выводы

В этой главе для стандарта IEC 61499 представлены три общепринятые парадигмы проектирования в разработке программного обеспечения: объектно-ориентированное программирование, проектирование на основе компонентов и сервис-ориентированная архитектура. Для каждой парадигмы дизайна приведены определения и примеры. Парадигма OOP связывает физическую структуру с ее кибернетическим представлением путем сопоставления устройств с программными объектами. Возможность

повторного использования и гибкость улучшаются за счет введения предварительно определенных интерфейсов между объектами в парадигме CBD. Парадигма SOA обеспечивает лучшую гибкость благодаря использованию многоуровневой архитектуры, хотя производительность снижается из-за интенсивной обработки данных в сетях функциональных блоков. Нельзя сделать вывод о том, что одна парадигма значительно лучше, чем другие. Исходя из конкретных требований различных систем промышленной автоматизации, должна быть выбрана правильная парадигма проектирования программного обеспечения путем оценки возможности повторного использования, гибкости и производительности парадигм.

Библиография

- [1] F. Basile, P. Chiacchio, and D. Gerbasio. On the Implementation of Industrial Automation Systems Based on PLC. *IEEE Transactions on Automation Science and Engineering*, Vol. 10, No. 4, 990–1003, 2013.
- [2] G. Black and V. Vyatkin. Intelligent Component-Based Automation of Baggage Handling Systems with IEC 61499. *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 2, 337–351, 2010.
- [3] R. Brennan, M. Fletcher, and D. Norrie. An Agent-Based Approach to Reconfiguration of Real-Time Distributed Control Systems. *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 4, 444–451, 2002.
- [4] A. Brown, S. Johnston, and K. Kelly. Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications, Rational Software Corporation. 2012.
- [5] L. De Alfraro and T. A. Henzinger. *Interface theories for componentbased design*. Springer, Heideberg, 1992.
- [6] T. Erl. *Service-oriented architecture: concepts, technology and design*. Prentice Hall, 2005.
- [7] R. Hametner, A. Zoitl, and M. Semo. Automation Component Architecture for the Efficient Development of Industrial Automation Systems. In *IEEE Conference on Automation Science and Engineering*, pages 156–161, 2010.
- [8] D. Helic and N. Scherbakov. Software Paradigms: Introduction and Procedural Programming Paradigm, 2013. 707.023 Lecture Notes, Graz, Austria.
- [9] International Electrotechnical Commission. IEC 61499-1: Function blocks, part 1: Architecture, 2012. Geneva, Switzerland.
- [10] International Electrotechnical Commission. IEC 61131-3 programmable controllers, part 3: Programming languages, 2013. International Electrotechnical Commission, Third Edition.
- [11] I. Jacobsen, C. Magnus, J. Patrik, and O. Gunnar. *Object Oriented Software Engineering*. Addison-Wesley, 1992.

- [12] F. Jammes and H. Smit. Service-Oriented Paradigms in Industrial Automation. *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 1, 62–70, 2005.
- [13] M. Koskela, M. Rahikainen, and T. Wan. Software development methods: SOA vs CBD, OO and AOP, 2007. Technical Report, Aalto University, Finland.
- [14] W. Lepuschitz, A. Zoitl, M. Vallee, and M. Merdan. Toward Self-Reconfiguration of Manufacturing Systems Using Automaton Agents. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, Vol. 41, No. 1, 52–69, 2011.
- [15] D. Lorenz and J. Vlissides. Designing Components versus Objects: A Transformational Approach. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 253–263, 2001.
- [16] P. V. Roy. *Programming Paradigms for Dummies: What Every Programmer Should Know*. Number 104. New Computational Paradigms for Computer Music, 2009.
- [17] T. Strasser and R. Froschauer. Autonomous Application Recovery in Distributed Intelligent Automaton and Control Systems. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, Vol. 42, No. 6, 1054–1070, 2012.
- [18] C. Sünder, A. Zoitl, J. Christensen, H. Steininger, and J. Rritsche. Considering IEC 61131-3 and IEC 61499 in the Context of Component Frameworks. In *6th IEEE International Conference on Industrial Informatics*, pages 277–282, 2008.
- [19] V. Vyatkin, C. Pang, Y. Deng, M. Sorouri, and H. Mayer. System-Level Architecture for Building Automation Systems: Object-Oriented Design and Simulation. In *39th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pages 5334–5339, 2013.
- [20] B. Warboys. The software paradigm. *ICL Technical Journal*, Vol. 10, 71–79, 1995.
- [21] B. Werner. Object-Oriented Extensions for IEC 61131-3. *IEEE Industrial Electronics Magazine*, Vol. 3, No. 4, 36–39, 2009.
- [22] G. Zhabelova and V. Vyatkin. Multiagent Smart Grid Automation Architecture Based on IEC 61850/61499 Intelligent Logical Nodes. *IEEE Transactions on Industrial Electronics*, Vol. 59, No. 5, 2351–2362, 2012.

Новые шаблоны проектирования для прогнозируемого во времени исполнения функциональных блоков

Matthew M. Y. Kuo

Оклендский университет

Partha S. Roop

Оклендский университет

СОДЕРЖАНИЕ

5.1 Введение	93
5.2 Кодирование инструкций для прогнозирования во времени	96
5.2.1 Циклы	96
5.2.2 Вещественные числа	97
5.2.3 Выделение памяти	99
5.2.4 Ответвления	99
5.2.5 Прерывания	99
5.3 Шаблоны проектирования	100
5.4 Требуемая квалификация	102
5.5 Упорядоченный шаблон синхронного проектирования	104
5.6 Задержка синхронного шаблона проектирования	110
5.7 Временной анализ	114
5.7.1 Статический временной анализ	115
5.8 Выводы	117

5.1. Введение

Функциональные блоки ИЕС 61499 используются для проектирования многих критических с точки зрения безопасности и прогнозируемых во времени систем из области интеллектуальных сетей [31], робототехники [9] и медицинского оборудования [32]. Корректность систем реального времени связана с их функциональностью, а также с их поведением во времени [7, 29]. В качестве примера рассмотрим манипулятор, представленный на рисунке 5.1.

Робот-манипулятор отвечает за перемещение объектов с приемного конвейера на конвейер выдачи. Для безопасности он использует датчик безопасности световой завесы, чтобы не нанести вред операторам. В таблице 5.1 представлено краткое описание каждого функционального блока в роботизированной руке.

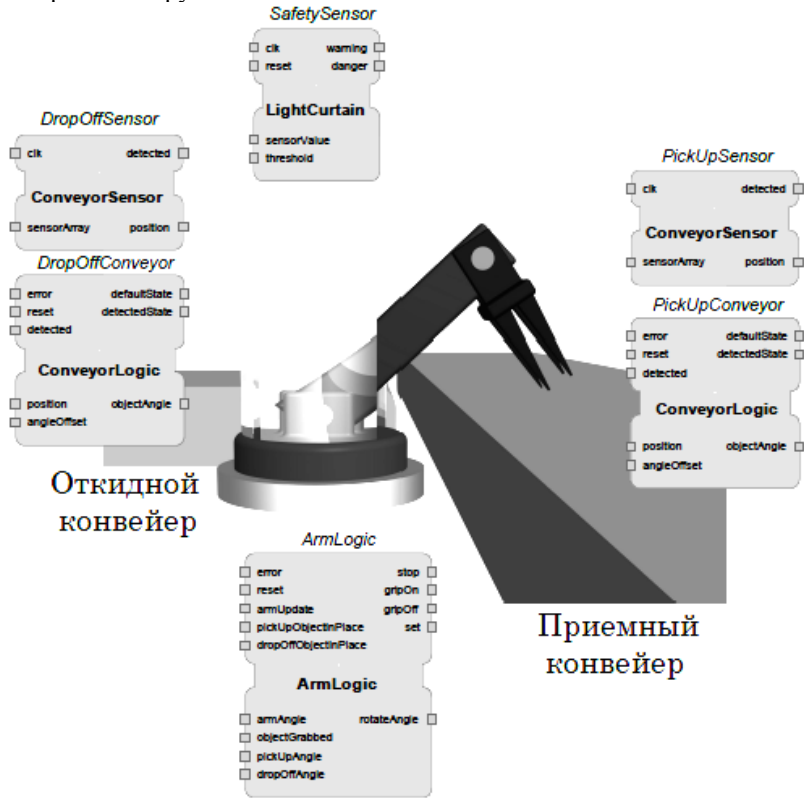


Рисунок 5.1. Пример роботизированной руки

Правильная работа системы роботизированного манипулятора не ограничивается ее функциональными возможностями, то есть роботизированная рука должна быть перемещена в нужное время в правильное положение. Требования в реальном времени подразделяются на две основные категории: жесткие в реальном времени и мягкие в реальном времени [17]. Жесткие системы реального времени не могут допустить какого-либо замедленного реагирования, поскольку это может привести к катастрофическим последствиям, таким как гибель людей, финансовая ответственность или значительный ущерб активам. Например, если для датчика безопасности световой завесы требуется слишком много времени, чтобы отреагировать на перегруженного оператора, роботизированная рука не

сможет вовремя остановиться и может нанести вред или серьезно травмировать оператора. С другой стороны, мягкие системы реального времени, могут переносить некоторые пропущенные сроки. Несоблюдение сроков в таких системах обычно приводит к отсутствию оперативности или снижению качества обслуживания без серьезных последствий. Например, рассмотрим конвейер системы манипуляторов. Медленная обработка может привести к незначительным потерям времени и повреждениям из-за столкновений, вызванных пропущенными событиями, но не приведет к катастрофическим последствиям.

Таблица 5.1. Описание функциональных блоков в системе роботизированной руки

Экземпляр функционального блока	Тип функционального блока	Функциональность
Датчик безопасности	Световой занавес	Обнаруживает вторжение световой завесы
Датчик приема	Датчик конвейера	Определяет, готов ли объект к приему
Датчик выдачи	Датчик конвейера	Определяет, готов ли объект к выдаче
Конвейер приема	Конвейерная логика	Управляет движением приемного конвейера
Конвейер выдачи	Конвейерная логика	Управляет движением конвейера выдачи

Существуют ограниченные подходы [9, 33], которые выступают в качестве руководящих принципов для систематического проектирования критических с точки зрения безопасности и прогнозируемых во времени систем с использованием ИЕС 61499. Для проектирования прогнозируемой во времени системы все аппаратные и программные компоненты должны быть прогнозируемыми во времени (это показано на рисунке 5.2). В этой главе основное внимание будет уделено прогнозируемой во времени разработке приложений и реализации ИЕС 61499. Дизайн и философию точного синхронизированного оборудования можно найти в [11].

Выразительность стандарта ИЕС 61499 позволяет конечным пользователям в своих приложениях ИЕС 61499 создавать и реализовывать свои собственные алгоритмы (в рамках ЕСС) и специализированный пользовательский функциональный блок, называемый функциональным блоком интерфейса службы. Это означает, что пользователь может написать и выполнить любой произвольный код в модели функционального блока. Чтобы достичь предсказуемости во времени, важно проявлять особую осторожность при разработке таких алгоритмов. Таким образом, эта глава начнется с

представления руководящих принципов кодирования, чтобы помочь конечному пользователю в разработке приложений с предсказуемым во времени результатом работы.



Рисунок 5.2. Предсказуемая во времени система

Однако поведение приложений зависит от кода, а также от базовой реализации IEC 61499. Чтобы решить эту проблему, в разделе 5.3 представлены новые шаблоны дизайна для проектирования в соответствии с прогнозируемыми во времени результатами IEC 61499.

5.2. Кодирование инструкций для предсказуемости во времени

В этом разделе будут представлены рекомендации по кодированию для предупреждения распространенных ошибок, возникающих при разработке прогнозируемых во времени приложений IEC 61499. Наши рекомендации основаны на хорошо известных принципах, сформулированных в стандарте кодирования MISRA-C [2], стандарте кодирования Лаборатории реактивного движения (JPL) [1], «Power of 10» [15], а также в статье о структуре программного обеспечения и предсказуемости WCET [13].

5.2.1. Циклы

Циклы используются для итерационных вычислений, а также для ожидания занятости на входных событиях или коммуникационных пакетах. Для достижения предсказуемости во времени важно статически определить максимальное количество итераций цикла, известного как ограничение цикла. Для приложения с циклами с неизвестными границами не может быть предоставлено никаких гарантий относительно его наибольшего времени выполнения. Программист должен максимально упростить структуру цикла, чтобы обеспечить возможность определения границы цикла. Переменные, используемые для счетчика итераций цикла, не должны изменяться в теле цикла. Операторы Continue следует заменить на операторы if и else [13].

Некоторые языки программирования, такие как PRET-C [3], позволяют программистам определять границы цикла непосредственно в исходном коде. Код генерируется для принудительного завершения циклов в указанных пределах. На рисунке 5.3 представлены две функции, которые вычисляют наибольший общий делитель (GCD), используя евклидов алгоритм [16]. Функция слева представляет собой оригинальный евклидов алгоритм, в котором число итераций зависит от входных данных, и, следовательно, его трудно определить во время компиляции. Поэтому никакие гарантии относительно времени не могут быть предоставлены. Функция справа реализует четкую границу. Если результат не определен в пределах установленной границы, функция вернет -1, что укажет на ошибку. Ошибка должна быть надлежащим образом обработана программистом.

5.2.2. Вещественные числа

Вещественные числа могут быть представлены в виде чисел с плавающей или с фиксированной запятой. Числа с плавающей запятой, как правило, более популярны, поскольку они являются примитивными типами данных в языках программирования. Однако при использовании чисел с плавающей запятой в системах, предсказуемых по времени, требуется особая осторожность. Программист должен сначала определить точность арифметического устройства с плавающей запятой (одинарного или двойного) целевой платформы. Если целевая платформа не включает арифметическое устройство с плавающей запятой или программист должен использовать точность, превышающую аппаратную поддержку, компиляторы будут использовать библиотеки программного обеспечения для выполнения вычислений с плавающей запятой. Как правило, эти программные библиотеки разработаны с учетом выполнения в типичном случае [13] и содержат циклы, которые трудно связать статически. Поэтому такие библиотеки не подходят для систем, предсказуемых по времени. Вместо этого следует использовать арифметику с фиксированной запятой, когда аппаратная поддержка вычислений с плавающей запятой ограничена или недоступна. Арифметика с фиксированной точкой – это манипулирование целочисленными значениями. Они изначально поддерживаются большинством процессорных архитектур.

```

int gcd(int a, int b) {
    while (a != b) {
        if (a > b) {
            a = a - b;
        }else{
            b = b - a;
        }
    }
    return a;
}

int gcd(int a, int b) {
    int i = 0; int ret = -1;
    for (i = 0; i < [bound]; i++) {
        if (a != b) {
            if (a > b) {
                a = a - b;
            }else{
                b = b - a;
            }
        }else{
            ret = a;
        }
    }
    return ret;
}

```

Рисунок 5.3. Перевод неограниченного цикла (слева) в ограниченный цикл (справа)

Вещественные числа могут быть представлены в виде чисел с плавающей запятой или с фиксированной запятой. Числа с плавающей запятой, как правило, более популярны, потому что они являются примитивными типами данных на языках программирования. Однако дополнительный уход требуется при использовании плавающих точек в предсказуемых временем системах. Программист должен сначала определить точность арифметической единицы с плавающей точкой (единственный или двойной) целевой платформы. Если целевая платформа не будет включать арифметическую единицу с плавающей точкой, или программист обязан использовать точность вне поддержки оборудования, компиляторы используют библиотеки программного обеспечения для выполнения вычислений с плавающей точкой. Обычно эти библиотеки программного обеспечения разработаны со средним выполнением случая в памяти [13] и содержат циклы, которые тверды к связанному статически. Поэтому такие библиотеки являются неподходящими для предсказуемых временем систем. Вместо этого вычисления с фиксированной точкой должны использоваться, когда поддержка оборудования для вычисления с плавающей точкой ограничена или не доступна. Вычисления с фиксированной точкой являются по существу управлением целочисленными значениями. Они исходно поддерживаются большинством архитектур процессора.

Упражнение

Если оба значения *a* и *b* на рисунке 5.3 являются 8-битными числами, то каково будет наименьшее значение для границы при условии, что верный результат всегда будет возвращен?

Подсказка: значение от 1 до 256.

5.2.3. Выделение памяти

Выделение области памяти может быть достигнуто статически или динамически. Динамическое распределение памяти позволяет приложениям назначать необходимый объем памяти во время выполнения операций. Напротив, статическое выделение памяти предварительно назначает области памяти во время компиляции для конкретных задач. В целом, динамическое распределение памяти более эффективно с точки зрения использования памяти, чем статическое распределение. Однако для систем, предсказуемых во времени, динамического выделения памяти следует избегать, если не используются специализированные алгоритмы с предсказуемым во времени распределением [14, 22]. Для этого есть три основных причины. Во-первых, динамическое распределение памяти требует, чтобы система искала доступное пространство во время выполнения. Время, необходимое для выделения памяти, сильно варьируется. Следовательно, чтобы обеспечить гарантию времени выполнения, время распределения в худшем случае должно приниматься для каждого выделения памяти. В худшем случае это предположение будет сильно переоценивать время выполнения системы, что приведет к непрактичным значениям синхронизации. Во-вторых, для систем с иерархией памяти трудно определить точную границу доступа к памяти. Следовательно, результатом может быть переоценка времени выполнения. В-третьих, динамическое распределение памяти может привести к ошибкам во время выполнения, когда может быть выделено недостаточно доступной памяти (то есть из-за фрагментации или нехватки памяти).

5.2.4. Ответвления

Существуют две категории операторов ветвления: статические ветви и вычисляемые ветви. Статические ветви – это операторы, такие как вызовы функций или операторы `if` и `else`. Эти операторы ветвления подходят для систем, предсказуемых во времени. Напротив, вычисляемые ветви представляют собой такие операторы, как указатели на функции и вычисленные операторы `goto`. Для систем, предсказуемых во времени, следует избегать этих операторов ветвления, поскольку они создают трудности для статического анализа [13].

5.2.5. Прерывания

Прерывания позволяют приложениям переключать контекст для обслуживания запроса из среды (переключение контекста). В общем случае прерывания могут возникать в любое время в течение срока службы приложения, и количество вхождений обычно не ограничено. Если нет ограничений частот, с которыми могут происходить прерывания, они не подходят для систем, предсказуемых во времени. Альтернативами прерываниям являются опросы и реактивные функциональные единицы. Опрос использует программное обеспечение для периодического такта запросов из

среды. Скорость, с которой приложение должно опрашивать (пробовать) среду, должна быть выше, чем скорость, с которой среда может делать запросы. Опрос наиболее подходит для непрерывных сигналов, таких как мониторинг температуры окружающей среды. Дискретные сигналы, такие как нажатия кнопок, могут быть пропущены, если они возникают, когда программное обеспечение не проверяет входы. Реактивные функциональные блоки находятся в специальных процессорах, называемых реактивными процессорами [20, 24, 30], в которых входы фиксируются, когда среда делает запрос. Следовательно, при использовании реактивных процессоров дискретные входы не будут пропущены, если входы обслуживаются быстрее, чем время их поступления. Каждый раз, когда входы обслуживаются, входные защелки очищаются в заранее определенных точках приложения, что делает их идеальными для проектирования систем, предсказуемых во времени.

5.3. Шаблоны проектирования

Шаблоны проектирования – это многократно используемые решения, которые решают повторяющиеся проблемы проектирования с использованием четко определенной структуры [6, 12]. Когда перед разработчиками ставится новая задача, типичной практикой является повторное использование и адаптация проверенных решений. Существует множество категорий шаблонов: от контроля доступа до систем связи и интерактивных систем [6]. Шаблоны проектирования могут зависеть от языка программирования. Одним из наиболее популярных шаблонов проектирования, независимых от языка программирования, является шаблон управления представлением модели (MVC) [18]. Традиционно шаблон MVC был разработан для объектно-ориентированного программирования в целях реализации пользовательских интерфейсов с разделением программного обеспечения на три компонента. Модель реализует поведение приложения и не зависит от визуального представления. Представление реализует визуальное представление некоторых или всех данных модели. Контроллер управляет входными данными от пользователя или среды и отправляет их модели в виде команд. Шаблон MVC был адаптирован для IEC 61499 с некоторыми изменениями [8]. Модель представляет собой функциональный блок, который реализует поведение контролируемой системы (то есть установки). Представление – это функциональный блок, который реализует графический интерфейс, связанный с моделью (то есть визуализатором). Контроллер представляет собой функциональный блок, который реализует контроль над системой. Существуют также шаблоны проектирования, разработанные специально для IEC 61499. Например, Serna и соавторы предложили схему проектирования для управления сбоями функциональных блоков путем введения функциональных блоков зон отказов, функциональных блоков управляемых отказов и функциональных блоков супервизора отказов [25]. Функциональные блоки зон отказов отвечают за сообщения о сбоях и восстановление после сбоев с

помощью функциональных блоков, управляемых сбоями. Функциональные блоки супервизора отказов отвечают за выявление того, произошел ли сбой из-за совокупного влияния нескольких функциональных блоков. Когда супервизор сбоев обнаруживает сбой, он отправляет данные о событии сбоя в соответствующие зоны сбоя. Функциональные блоки зоны сбоя могут быть соединены в рамках иерархии, чтобы они соответствовали зависимостям сбоя в сети. События сбоя распространяются через иерархию, так что сеть функциональных блоков может предпринимать соответствующие действия.

Дубинин и Вяткин предложили семантически устойчивый шаблон проектирования для IEC 61499 путем введения специальных служебных блоков, которые должны быть размещены между соединениями функциональных блоков [10]. Служебные блоки управляют распространением событий. Поскольку функциональные блоки инициируются событиями, служебные блоки эффективно контролируют выполнение сети.

Несмотря на то, что для IEC 61499 существует ряд семантик исполнения [26], нет никаких рекомендаций для выбора семантики на основе требований приложения. Поскольку шаблоны проектирования предлагают эффективные руководящие принципы, помогающие проектированию, нам необходимо разработать шаблоны проектирования, чтобы устранить этот пробел. В частности, мы намереваемся разработать шаблоны проектирования, которые помогают проектировать системы, предсказуемые во времени. Наш подход может быть расширен в будущем, чтобы он соответствовал другим требованиям приложения к соответствующей семантике выполнения через шаблоны проектирования. В этой главе мы сначала определим требования к приложениям между моделями централизованных и распределенных функциональных блоков. После этого будут предложены два шаблона проектирования, чтобы помочь разработчику выбрать подходящую семантику выполнения при проектировании систем, предсказуемых во времени. Один из шаблонов проектирования предназначен для централизованных систем, а другой – для распределенных систем.

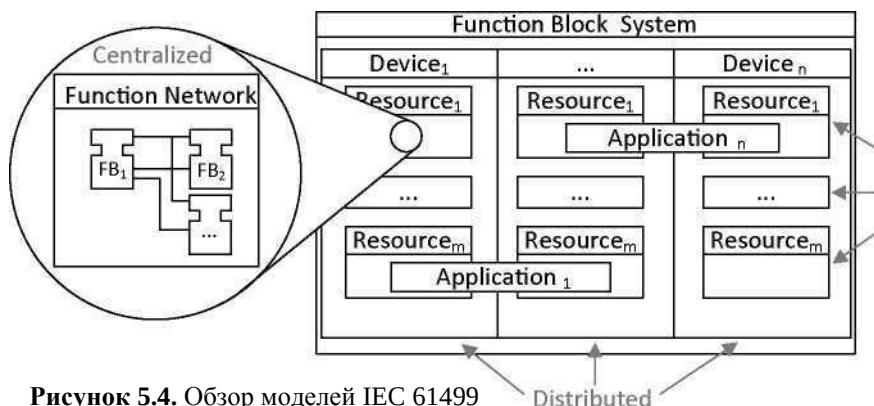


Рисунок 5.4. Обзор моделей IEC 61499

5.4. Требуемая квалификация

Рисунок 5.4 иллюстрирует модели функциональных блоков. Краткое описание этих моделей можно найти во вступительных главах и [29]. В этой главе мы сосредоточимся на создании функциональных блоков внутри ресурсов, называемых внутрисистемными функциональными блоками. Мы также рассмотрим создание функциональных блоков в нескольких ресурсах, которые называются межресурсными функциональными блоками. В таблице 5.2 приведены основные различия между функциональными блоками внутри ресурсов и между ними.

Таблица 5.2. Централизованные и распределенные системы

Централизованные	Распределенные
Меньше издержек при выполнении	Больше издержек при выполнении
Запланировано одновременно	Запланировано параллельно
Одно устройство	Несколько устройств
Быстрые встроенные шины или память	Медленная сетевая связь

Функциональные блоки внутри ресурсов – это централизованные системы, в которых функциональные блоки совместно используют одни и те же аппаратные ресурсы, такие как процессор и связанная память. Цель централизованных систем – эффективно выполнять операции с минимальными накладными расходами. Связь между внутрисистемными функциональными блоками обычно требует нескольких инструкций процессора, поскольку они совместно используют один и тот же ресурс. Межресурсные функциональные блоки представляют собой распределенные системы, в которых функциональные блоки развернуты на отдельных виртуальных / реальных устройствах. Для распределенных систем важна модульность, поскольку она дает возможность избежать сильной зависимости между ресурсами и устройствами. Зачастую модульная конструкция связана с дополнительными затратами. Связь между ресурсами и устройствами распределяется с использованием соответствующих сетевых или локальных программных гнезд. Ожидается, что задержка связи будет значительно больше по сравнению с функциональными блоками внутри ресурса.

Чтобы рассмотреть различные потребности внутрисистемных и межресурсных функциональных блоков, мы предлагаем два отдельных шаблона проектирования для достижения предсказуемого во времени исполнения систем IEC 61499. Оба шаблона проектирования основаны на парадигме синхронного программирования [4, 29]. Преимуществами синхронного подхода являются детерминизм и реактивность [21]. В этом контексте детерминизм относится к системам, которые вырабатывают одну и ту же выходную последовательность при стимуляции одной и той же входной последовательностью. С другой стороны, реакционная способность относится

к системам, которые всегда остаются чувствительными к воздействиям окружающей среды.

Синхронное программирование вдохновлено синхронными схемами. В синхронных цепях компоненты приводятся в действие с использованием фактического тактового сигнала. Входы и выходы схем изменяются мгновенно относительно границ тактовых импульсов. Аналогично синхронные программы выполняются в терминах тактов логических часов. Во время каждого такта приложение выбирает свои входные данные, вычисляет результаты и выдает соответствующие выходные данные (см. рисунок 5.5). Предполагается, что логические такты являются мгновенными и требуют нулевого времени. Однако на практике все вычисления требуют времени. Следовательно, для практического применения достаточное условие для получения отклика в реальном времени состоит в том, что наибольшая длительность среди всех возможных тактов должна быть равна времени, которое требуется среде системы для изменения входных данных, или быть больше него. Самая большая продолжительность среди всех возможных тактов обычно называется временем реакции наихудшего случая (WCRT) синхронной программы [3, 5]. Подобное условие также может наблюдаться в синхронных цепях, в которых все электрические сигналы должны стабилизироваться до следующей границы тактового сигнала.

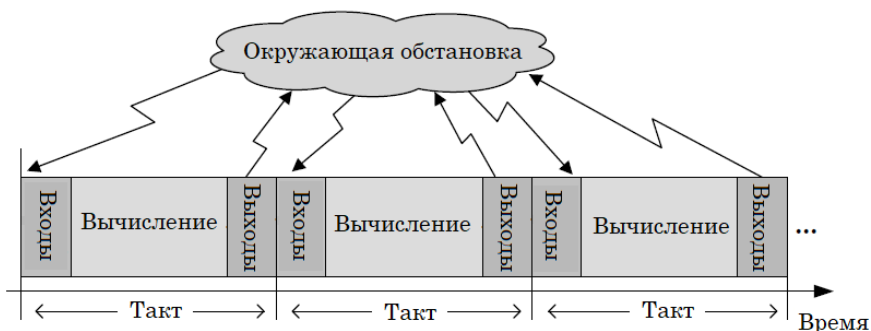


Рисунок 5.5. Синхронное выполнение

Шаблон проектирования, который мы предлагаем для прогнозируемого во времени выполнения внутриресурсных функциональных блоков, называется упорядоченным синхронным шаблоном проектирования. Шаблон же проектирования, который мы предлагаем для прогнозируемого во времени выполнения функциональных блоков между ресурсами, называется шаблоном синхронного проектирования с задержкой. Эти два шаблона проектирования описывают, как реализации функциональных блоков должны планироваться и выполняться для предсказуемости во времени. Упорядоченный шаблон синхронного проектирования основан на языке синхронного программирования PRETC [3]. Как следует из названия, этот шаблон обеспечивает последовательный порядок планирования функциональных

блоков в течение каждого такта. Связь между каждым функциональным блоком происходит мгновенно (то есть происходит в течение одного такта). Задержка синхронного шаблона проектирования инспирирована [28]. Как следует из названия, связь между ресурсами и устройствами не является мгновенной (то есть не происходит в пределах одного такта), но требует единичной задержки. Эта задержка позволяет межресурсному функциональному блоку выполняться параллельно и независимо от других ресурсов или устройств в течение каждого такта.

На рисунке 5.6 сопоставлены предложенные шаблоны проектирования с моделями функциональных блоков. Упорядоченный шаблон синхронного проектирования применяется к моделям, закрашенным темно-серым цветом (функциональные блоки внутри ресурса). Шаблон синхронного проектирования с задержкой применяется к моделям, закрашенным светло-серым цветом (межресурсные функциональные блоки).

5.5. Упорядоченный шаблон синхронного проектирования

Для внутриресурсных функциональных блоков мы предлагаем упорядоченный синхронный шаблон. Упорядоченное сообщение относится к тому факту, что фиксированный порядок планирования является синхронным шаблоном проектирования (см. счетчик пульсаций, представленный на рисунке 5.7). Счетчик пульсаций построен с использованием триггера, соединенного последовательно. Система управляется одним источником синхронизации. Счетчик пульсаций «выполняется» в строгом порядке от первого до последнего. Связь, выборка входов и эмиссия выходов происходят мгновенно (то есть происходят до следующей границы тактового сигнала). Любые задние границы воспринимаются только на следующей границе часа. Соединение от Q надчеркнутого до Q инвертирует значение, сохраняемое каждым триггером, когда обнаруживается нарастающий фронт. Для удобства ссылки порт Q первого триггера $D1$ будет называться Q_{D1} и так далее. Счетчик пульсаций работает, инвертируя выход Q_{D1} на каждой передней границе входного тактового сигнала и инвертируя выход Q_{D2} на каждой второй нарастающей границе и инвертируя выход Q_{D3} на каждой четвертой нарастающей границе.

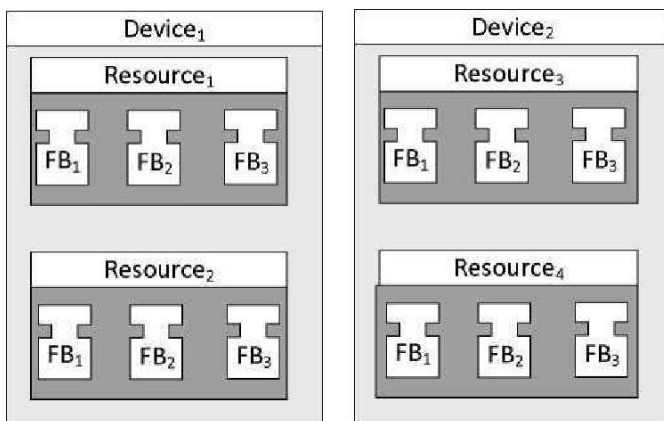


Рисунок 5.6. Сопоставление предлагаемых шаблонов проектирования и моделей функциональных блоков (темно-серый цвет – упорядоченный шаблон синхронного проектирования; светло-серый цвет – задержка синхронного шаблона проектирования)

По умолчанию: все три D-триггера имеют нулевые значения в качестве входных данных; следовательно, значение Q равно нулю, а значение Q надчеркнутое равно единице для всех трех триггеров. Десятичное значение для счетчика равно нулю (000_b).

Первая нарастающая тактовая граница: D1 считывает значение единица из D_{D1} (предыдущее значение Q_{D1} надчеркнутое) и устанавливает Q_{D1} в единицу, а Q_{D1} надчеркнутое в ноль. И D2, и D3 не изменяют значений, потому что нарастающая граница не обнаружена, поскольку Q_{D1} надчеркнутое равно нулю. Десятичное значение для счетчика теперь равно единице (001_b).

Второй восходящий фронт тактовой частоты: D1 считывает нулевое значение из D_{D1} и устанавливает Q_{D1} в ноль, а Q_{D1} надчеркнутое в единицу. D2 обнаруживает нарастающую границу, установленный D1, и считывает значение 1 из D_{D2} (предыдущее значение Q_{D2} надчеркнутое) и устанавливает Q_{D2} в единицу и Q_{D2} надчеркнутое в ноль. D3 не изменяется в значении, потому что передняя граница не обнаружена. Десятичное значение счетчика теперь равно двум (010_b).

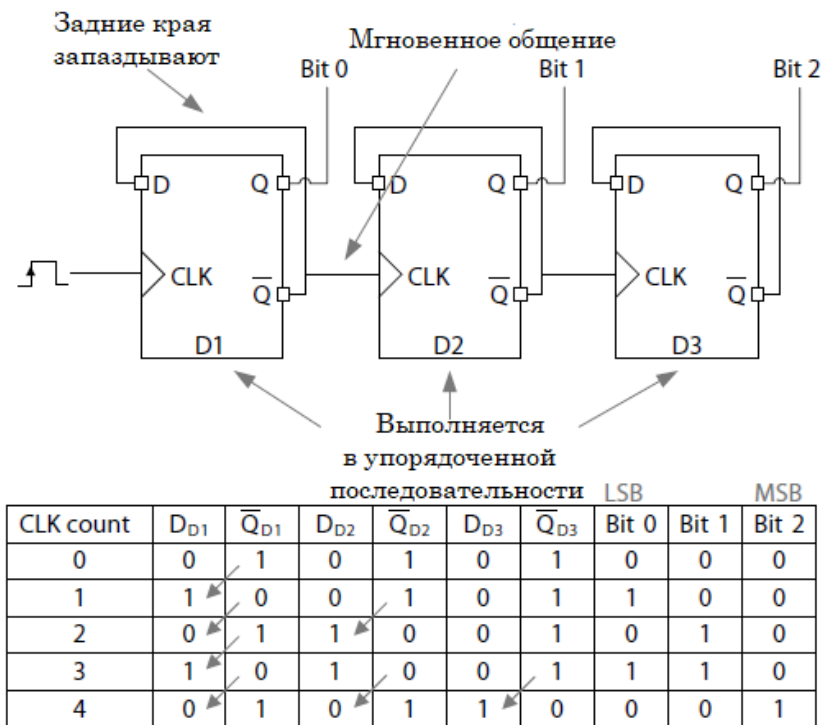


Рисунок 5.7. Аналогия упорядоченного синхронного шаблона проектирования с использованием счетчика пульсаций

Здесь D-триггеры аналогичны функциональным блокам внутри ресурса при упорядоченной композиции. Взаимосвязь между триггерами представляет собой сеть функциональных блоков, в которой связь между функциональными блоками происходит мгновенно, а связь на задней границе задерживается до следующего такта. Тактовые импульсы аналогичны логическим тактам синхронного программирования.

Мы будем использовать символ $\times$ для представления композиции под упорядоченным синхронным шаблоном. Например, $D1 \times D2 \times D3$ указывает, что функциональный блок D1 одновременно составлен с функциональными блоками D2 и D3, и они выполняются с использованием упорядоченного синхронного шаблона (упорядоченной композиции). Фиксированный порядок планирования идет слева направо от композиции, то есть D1, затем D2 и затем D3.

Давайте рассмотрим конкретную сеть функциональных блоков в данном ресурсе. На рисунке 5.8 представлен централизованный роботизированный манипулятор. Пунктирные соединения в 1 и 2 будут использоваться позже, чтобы проиллюстрировать временные свойства упорядоченного шаблона синхронного проектирования.

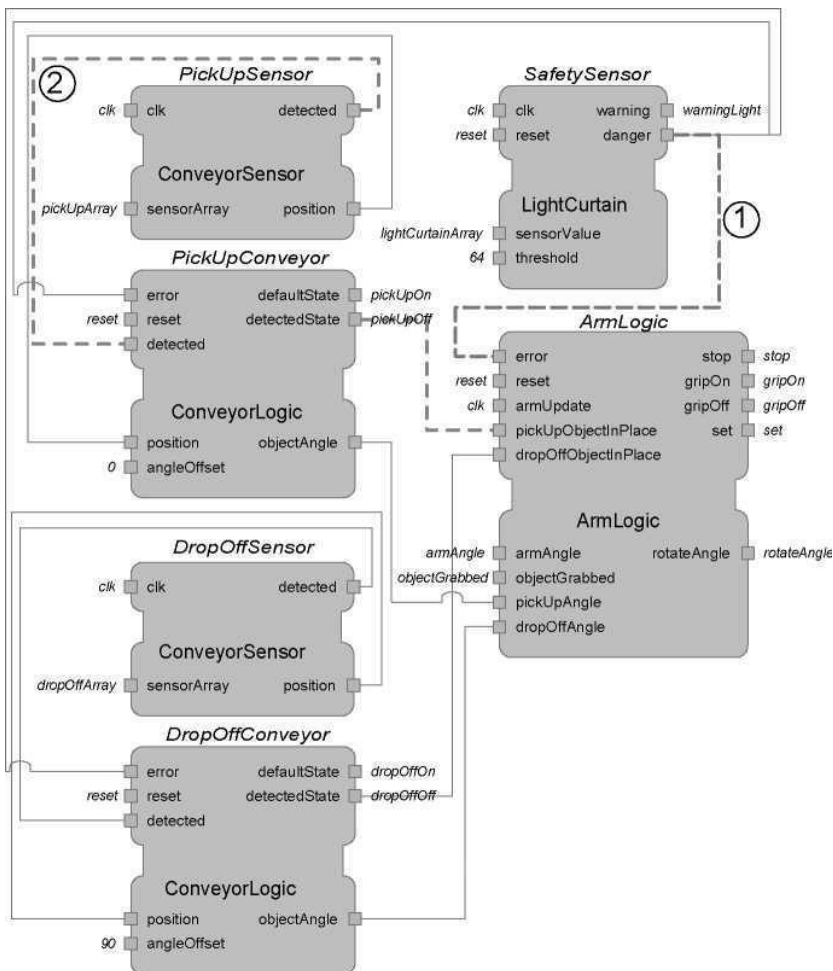


Рисунок 5.8. Функциональный блок сети робота-манипулятора

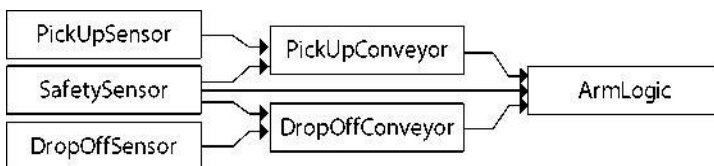


Рисунок 5.9. Входные и выходные зависимости сети манипулятора

Чтобы применить этот шаблон проектирования в сети функциональных блоков, необходимо явное упорядочение функциональных блоков, подобное порядку слева направо в примере счетчика импульсов. В сети функциональных

блоков такой общий порядок не является явным, поэтому такой порядок должен быть определен. Первым шагом является определение частичного порядка путем анализа входных и выходных зависимостей в сети функциональных блоков. Рисунок 5.9 иллюстрирует зависимости ввода и вывода для примера работы манипулятора.

FB1 > FB2: функциональный блок (FB1) вызывается перед другим функциональным блоком (FB2), если его выходной порт подключен к входному порту другого. Например, `PickUpConveyor > PickUpSensor`.

FB1 = FB2: Функциональный блок (FB1) можно вызвать до или после другого функционального блока (FB2), если между ними нет соединений. Например, `SafetySensor = PickUpSensor`.

Для сетей функциональных блоков с циклическими зависимостями разработчик системы решает, какой функциональный блок будет упорядочен, прежде чем другие прервут зависимость. Циклы похожи на задние края D-триггеров на рисунке 5.7. Частичный порядок для роботизированного примера: `SafetySensor = PickUpSensor = DropOffSensor > PickUpConveyor > DropOffConveyor > ArmLogic`.

Второй шаг – это создание графика выполнения путем преобразования частичного заказа (на первом шаге) в общий заказ. Функциональным блокам, которые равны в частичном порядке, произвольно назначается порядок между ними, например, `SafetySensor > PickUpSensor > DropOffSensor`. Порядок между этими функциональными блоками не имеет значения, потому что между ними нет никаких зависимостей. Результатом является общий порядок функциональных блоков, которые можно напрямую использовать для создания упорядоченной композиции. Заказанная композиция для манипулятора: `SafetySensor × PickUpSensor × DropOffSensor × PickUpConveyor × DropOffConveyor × ArmLogic`.

Как только упорядоченная композиция определена, принцип выполнения в каждом такте выглядит следующим образом. Во-первых, система производит выборку входов окружающей среды. Затем каждый функциональный блок в композиции создается в соответствии с определенным порядком. Наконец, в конце такта выводятся выходные данные.

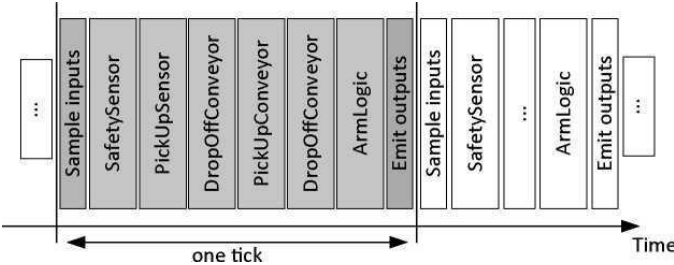


Рисунок 5.10. Заказанное синхронное выполнение действия роботизированной рукой

На рисунке 5.10 показано, как работает сеть роботизированных манипуляторов во время каждого такта. Сначала отбираются входные данные из окружающей среды. Затем функциональные блоки выполняются в порядке их составления, то есть SafetySensor, затем PickUpSensor и так далее. Наконец, выходные данные отправляются в среду в конце такта.

Ключевой особенностью упорядоченного синхронного шаблона является то, что события и данные распространяются мгновенно (в пределах одного такта) через функциональные блоки в статически определенном и предсказуемом порядке. Это означает, что время, которое требуется системе для ответа на входные данные среды и для выдачи соответствующего выходного сигнала (время отклика), всегда является единичным тактом для нециклических систем (то есть сетей функциональных блоков без циклов). Время отклика для циклических систем (то есть сетей функциональных блоков с циклами) составляет не более двух тактов.

Например, на рисунке 5.11 показано, как события и данные распространяются через сеть в примере с роботизированной рукой. Каждая горизонтальная линия означает такт. На рисунке 5.11a представлен сценарий, когда SafetySensor обнаруживает вторжение в рабочую область. Устройство SecuritySensor обнаруживает вторжение, и сообщение об ошибке передается в ArmLogic. ArmLogic останавливает роботизированную руку, чтобы предотвратить возможные поломки, создавая событие остановки. Сетевое соединение, способствующее этому сценарию, выделено пунктирным соединением 1 на рисунке 5.8.

На рисунке 5.11b представлен сценарий, в котором объект прибывает в пункт выдачи. PickUpSensor обнаруживает объект и уведомляет PickUpConveyor, отправляя обнаруженное событие. PickUpConveyor приостанавливает работу конвейера и уведомляет функциональный блок ArmLogic, чтобы забрать предмет. Затем ArmLogic генерирует событие set, чтобы установить роботизированную руку в нужное положение. Сетевые соединения, способствующие этому сценарию, выделены пунктирным соединением 2 на рисунке 5.8. Для обоих сценариев время отклика составляет один такт, хотя входное событие распространяется через разное количество функциональных блоков, то есть два для рисунка 5.11a и три для рисунка 5.11b.

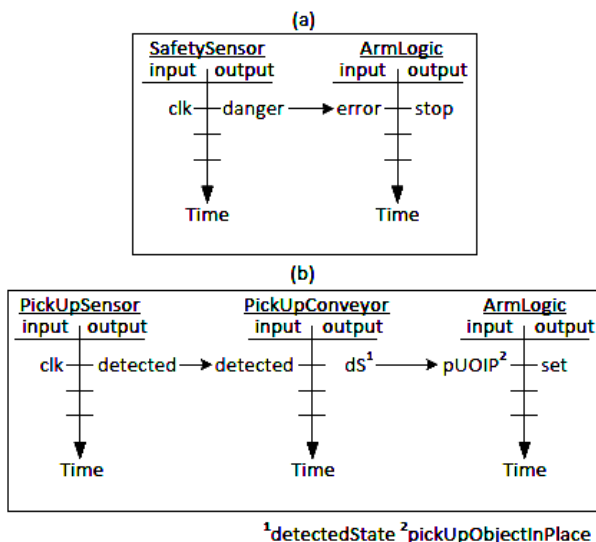
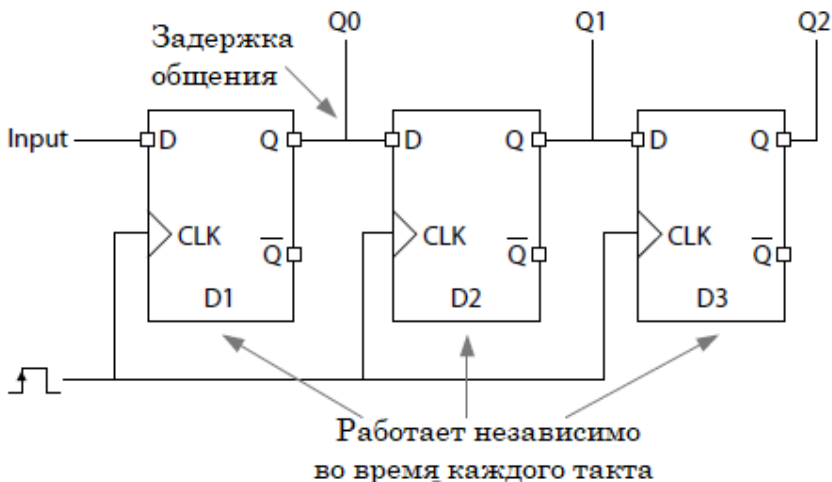


Рисунок 5.11. Время отклика для примера с роботизированной рукой с использованием упорядоченного синхронного шаблона

5.6. Задержка синхронного шаблона проектирования

Межресурсные функциональные блоки (то есть устройства и ресурсы) являются распределенными компонентами, которые могут выполняться параллельно. Следовательно, упорядоченный шаблон синхронного проектирования не подходит для функциональных блоков между ресурсами, поскольку строгий порядок выполнения является ограничительным для распределенного выполнения. Таким образом, для межресурсных функциональных блоков мы предлагаем шаблон синхронного проектирования с задержкой. Слово «задержанный» описывает тот факт, что задержка подразумевается для любого обмена данными (то есть обмена в прямом и обратном направлениях) между функциональными блоками через множество ресурсов. На рисунке 5.13 показан пример, где роботизированная рука распределяется на два ресурса: $\text{Resource}_{\text{conveyor}}$ и $\text{Resource}_{\text{arm}}$. $\text{Resource}_{\text{conveyor}}$ содержит связанные с конвейером функциональные блоки, а $\text{Resource}_{\text{arm}}$ содержит функциональные блоки ArmLogic и SafetySensor . Связь между этими двумя ресурсами достигается с помощью блоков издателя и блоков подписчика, например, PubDropOff и SubDropOff . За подробной информацией о функциональных блоках издателя и подписчика читатели могут обратиться к [29]. Мы используем традиционный символ с параллельными линиями $\parallel$ для представления композиции для отложенного синхронного шаблона (задержанная композиция). Например, состав для рисунка 5.13 выглядит как $\text{Resource}_{\text{conveyor}} \parallel \text{Resource}_{\text{arm}}$.



CLK count	Input	Q _{D1}	Q _{D2}	Q _{D3}
0	0	0	0	0
1	1	1	0	0
2	0	0	1	0
3	0	0	0	1

Рисунок 5.12. Аналог модели отложенного синхронного проектирования с использованием регистра сдвига

В отличие от упорядоченного синхронного шаблона порядок составления ресурсов или устройств не имеет значения. Ресурсы и устройства обмениваются данными только после того, как все ресурсы и устройства в системе достигают завершения каждого такта. Поэтому вся связь между ресурсами и устройствами задерживается на один такт. Например, событие вывода из `Resourceconveyor` доступно только для `Resourcearm` в следующем тике. Эта задержка разъединяет любые входные и выходные зависимости между ресурсами и устройствами. Поскольку ресурсы и устройства не взаимодействуют во время каждого такта, они могут работать независимо (во время каждого такта).

Для аналогии с шаблоном синхронного проектирования с задержкой рассмотрим сдвиговый регистр, представленный на рисунке 5.12. Сдвиговый регистр построен с использованием трех D-триггеров, каждый из которых подключен параллельно. Каждый D-триггер управляется своими собственными часами, причем сигналы поступают от одного и того же источника. Сдвиговый регистр выполняется параллельно, и каждый триггер выполняется независимо на каждой границе тактовой частоты.

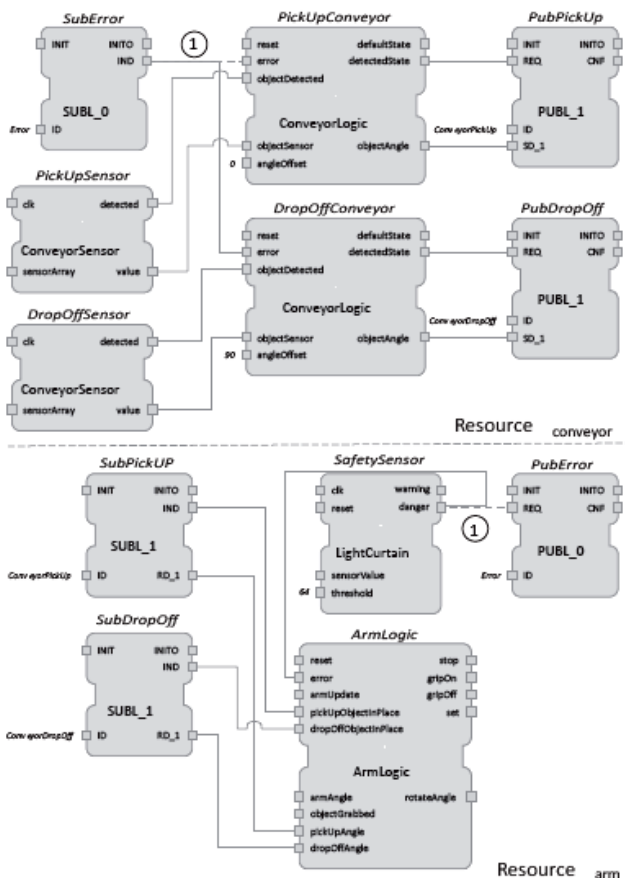


Рисунок 5.13. Распределенная роботизированная рука

Задержка связи означает, что задержка происходит на следующей границе тактового сигнала. Взаимосвязь между триггерами представляет сеть функциональных блоков, в которой связь между функциональными блоками задерживается до следующего такта. Тактовые импульсы аналогичны логическим тактам синхронного программирования. D-триггеры работают аналогично ресурсам функциональных блоков или устройствам с задержкой композиции ($D1 \parallel D2 \parallel D3$). Например, в начале каждого такта три триггера выбирают свои входы параллельно. Значения, которые они выбирают, взяты из предыдущего такта (см. Таблицу истинности на рисунке 5.12). Это так же, как отложенная композиция, где каждый ресурс и устройство работает независимо во время такта, а связь между ресурсами и устройствами задерживается до следующего такта.

Теперь давайте рассмотрим конкретный пример, иллюстрирующий поведение функциональных блоков с использованием этого шаблона

проектирования. На рисунке 5.14 показано, как выполняются функциональные блоки в примере с распределенным роботизированным манипулятором. Через ресурсы функциональные блоки выполняются с использованием синхронизированного шаблона с задержкой. Внутри каждого ресурса функциональный блок выполняется в соответствии с упорядоченным синхронным шаблоном, как представлено в предыдущем разделе. Следовательно, состав функциональных блоков для примера распределенного роботизированного манипулятора может быть представлен как:

$$\{ \text{SubPickUp} \times \text{SubDropOff} \times \text{SafetySensor} \times \text{ArmLogic} \times \text{PubError} \} \parallel \\ \{ \text{SubError} \times \text{PickUpSensor} \times \text{DropOffSensor} \times \text{PickUpConveyor} \times \\ \text{DropOffConveyor} \times \text{PubPickUp} \times \text{PubDropOff} \}.$$

Как и во всех синхронных системах, задержанный синхронный шаблон также выполняется в тактах. Во время каждого такта каждый ресурс выбирает входные данные из среды и выполняет функциональные блоки, которые он инкапсулирует независимо, а также синхронизирует и передает выходные данные в среду. Синхронизация необходима, потому что нет физических часов, управляющих началом каждого такта. Поэтому ресурсы должны согласовываться, когда начинается следующий логический такт. Синхронизация начинается после того, как все ресурсы выполнили функциональные блоки, которые они инкапсулируют. Если ресурс завершает выполнение досрочно, он должен ждать (останавливаться), пока все другие ресурсы не завершат выполнение. Во время синхронизации ресурсы обмениваются событиями и данными через блоки издателя и подписчика. Как только синхронизация завершается, ресурсы и устройства генерируют свои соответствующие выходные данные и начинают свой следующий такт.

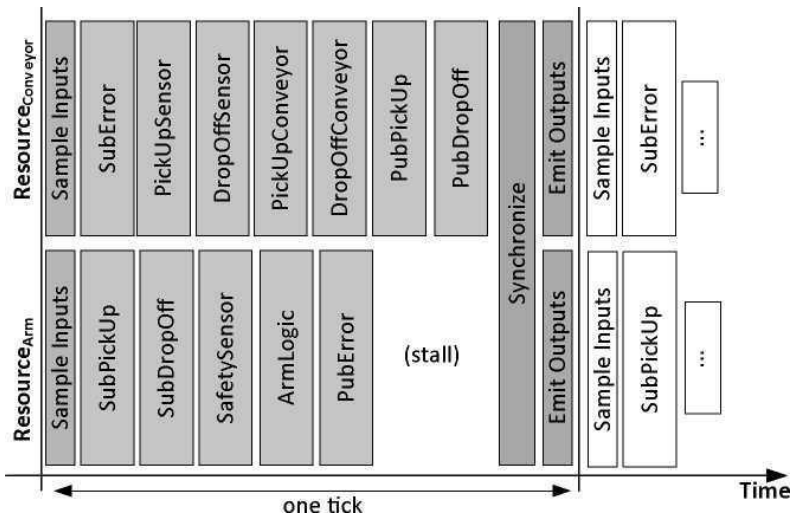


Рисунок 5.14. Распределенное исполнение роботизированной руки

Чтобы проиллюстрировать свойство синхронизации шаблона отложенного проектирования, на рисунке 5.15 показано, как события и данные распространяются между $\text{Resource}_{\text{conveyor}}$ и $\text{Resource}_{\text{arm}}$. Представленный сценарий такой же, как на рисунке 5.11a, где SafetySensor обнаруживает вторжение в рабочее пространство. На рисунке 5.13 цепочка событий выделена пунктирными соединениями 1. Во время первого такта SafetySensor обнаруживает вторжение и отправляет событие ошибки в PubError . Затем эти два ресурса синхронизируются, и данные отправляются из $\text{Resource}_{\text{conveyor}}$ в $\text{Resource}_{\text{arm}}$ через PubError и SubError . Во втором такте SubError отправляет сигнал, полученный от PubError , в PickUpConveyor , который отправляет сигнал об остановке конвейера. Из этого примера мы видим различия во времени между упорядоченным синхронным шаблоном и задержанным синхронным шаблоном. Упорядоченный синхронный шаблон требует только одного такта для ответа (см. рисунок 5.11a), тогда как отложенный синхронный шаблон требует двух тактов. В общем, упорядоченный шаблон всегда имеет время отклика от одного до двух тактов, а шаблон с задержкой имеет время отклика, равное количеству ресурсов в цепочке событий. Шаблон синхронного проектирования с задержкой идеально подходит для прогнозируемого по времени распределения двумя способами. Во-первых, самая длинная длина такта – это просто максимальная WCRT между всеми ресурсами и устройствами. Во-вторых, в отличие от семантики GALS для функциональных блоков [29], детерминизм сохраняется при параллельном выполнении. Ограничением шаблона синхронного проектирования с задержкой является необходимость синхронизации в конце каждого такта.

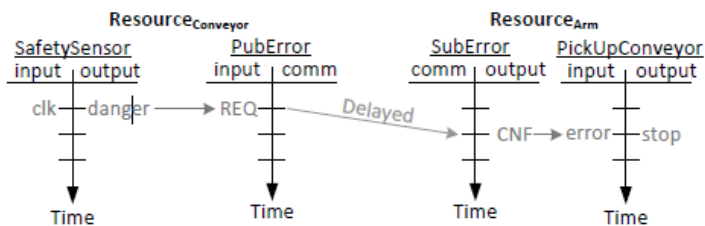


Рисунок 5.15. Время отклика для примера робота-манипулятора по синхронной схеме с задержкой

5.7. Временной анализ

В предыдущем разделе мы рассмотрели логическое время с точки зрения тактов. Для практического применения требуются значения в реальном времени. Чтобы преобразовать логическое время в реальное, для расчета WCRT системы используется анализ времени. Вычисленное значение WCRT – это длительность одного такта в реальном времени. WCRT – это время

выполнения в наихудшем случае (WCET) для всех возможных тактов в системе. Методы временного анализа можно разделить на две категории: основанный на измерениях временной анализ и статический временной анализ [27]. Методы, основанные на измерениях, выполняют данное приложение с большим набором входных векторов для вычисления WCET. Основным недостатком этого подхода является отсутствие гарантированной надежности. Измеренное значение WCET может быть заниженным, поскольку входные данные, необходимые для выявления наихудшего случая, могут не использоваться [23]. Методы статического анализа создают модели данного приложения, используя звуковые абстракции. WCET приложения определяется с использованием алгоритмов перечисления пути для вычисления самого длинного пути выполнения. Эти методы носят исчерпывающий характер и всегда обеспечивают надежное и даже чрезмерное приближение фактического наихудшего случая. На рисунке 5.16 показано различие между фактическим временем выполнения и измеренным временем выполнения типичной программы. Область, заштрихованная светло-серым цветом, показывает распределение фактического времени выполнения типичной программы. Область, заштрихованная темно-серым цветом, показывает распределение измеренных длительностей выполнения.



Рисунок 5.16. График, показывающий обобщение наблюдаемых значений времени по сравнению с фактическими значениями времени

Из рисунка видно, что анализ на основе измерений обычно отбирает подмножество всех возможных программных результатов. Следовательно, измеренный WCET обычно является недооценкой фактического WCET программы. Для сравнения, статический временной анализ обеспечивает надежную и безопасную переоценку WCET программы.

5.7.1. Статический временной анализ

Статический анализ синхронизации – это процесс вычисления WCET системы с целью обеспечить выполнение всех требований к синхронизации. Рисунок 5.17 иллюстрирует обзор блок-схемы процесса статического анализа синхронизации. Для функциональных блоков IEC 61499 процесс начинается с

описания системы в формате XML. Когда система спроектирована и готова к развертыванию, целевая реализация компилирует XML в отдельный двоичный файл (машинный код) для каждого устройства в системе. Анализ времени обычно выполняется отдельно для каждого устройства, а затем для всей системы.



Рисунок 5.17. Обзор статического анализа времени синхронизации

- В процессе построения CFG анализируются инструкции в двоичном файле и создается CFG, отображающий возможные пути через программу. CFG состоит из узлов и переходов. Узлы представляют собой последовательные блоки команд, а переходы представляют собой ветвления между этими блоками команд.
- Временной анализ команд определяет стоимость выполнения каждой инструкции в двоичном виде из таблиц данных и/или моделей оборудования, то есть моделей кэша, моделей конвейеров и моделей сети.
- Анализ самого длинного пути использует объединенные данные из CFG и времени выполнения команд, чтобы определить путь, вносящий вклад в WCET устройства.

Как только WCET всех устройств вычислены, значение WCET системы может быть легко получено, если используется шаблон отложенного синхронного проектирования. WCET системы – это просто максимальный WCET всех устройств. Помимо WCET, еще одним свойством синхронизации, представляющим интерес для разработчиков, является сквозное время отклика системы. Оно определяется как время, необходимое для излучения набора выходов в ответ на конкретную комбинацию входов. Значение времени отклика вычисляется для каждой пары входных и выходных портов с использованием анализа времени отклика. Для шаблонов проектирования, предложенных в этой главе, сквозной ответ – это просто WCET, умноженный

на количество ресурсов в цепочке событий от входных событий до выходных ответов. Более подробная информация представлена в [29].

5.8. Выводы

Шаблоны проектирования – это многократно используемые решения, которые устраняют повторяющиеся проблемы проектирования с использованием четко определенной структуры [6, 12]. Когда возникает новая задача для разработчиков является обычной практикой повторное использование и адаптация проверенных решений. Хотя существует множество схем проектирования, предназначенных для содействия функциональному проектированию систем IEC 61499 [10, 18, 25], существуют ограниченные подходы [9, 33], которые действуют в качестве руководящих принципов для систематического проектирования критических с точки зрения безопасности систем с предсказуемым временем.

В этой главе мы рассмотрели новые шаблоны проектирования, чтобы помочь в разработке систем с предсказуемым временем. Предлагаемые шаблоны проектирования дают рекомендации по выбору подходящей семантики на основе требований приложения при проектировании системы с предсказуемым временем. Мы подразделяем модели функциональных блоков на две категории: внутрисервисные и межсервисные функциональные блоки. В таблице 5.3 представлено сравнение между упорядоченным шаблоном синхронного проектирования и шаблоном синхронного проектирования с задержкой. Обе модели проектирования предсказуемы по времени и обеспечивают детерминизм и реактивность. Формальная семантика и связанные с ней доказательства приведены в [19]. Внутрисервисные функциональные блоки, например, базовые и составные функциональные блоки, находятся внутри ресурсов. Это централизованные системы, где функция блокирует общие аппаратные ресурсы, такие как процессор и связанная память. Также мы рассмотрели упорядоченный шаблон синхронного проектирования для функциональных блоков внутри ресурсов. Разработка шаблона проектирования была вдохновлена языком синхронного программирования PRET-C [3]. Этот шаблон обеспечивает предсказуемое во времени параллельное выполнение функциональных блоков и идеально подходит для централизованных систем. Связь между функциональными блоками происходит мгновенно (в пределах одного такта). Время отклика составляет один такт для сетей с нециклическими функциональными блоками и два для циклических сетей, которые имеют возможность осуществлять обратную связь. Длительность такта – это WCRT системы. Параллельное выполнение для этого шаблона не является идеальным, поскольку строгий порядок выполнения является обязательным для упорядоченного синхронного шаблона проектирования.

Таблица 5.3. Сравнение упорядоченного шаблона синхронного проектирования и шаблона отложенного синхронного проектирования

Свойство	Шаблон упорядоченного синхронного проектирования	Шаблон отложенного синхронного проектирования
Предсказуемость во времени	Да	Да
Детерминизм	Да	Да
Наличие обратной связи	Да	Да
Подходит для...	внутриресурсных функциональных блоков	межресурсных функциональных блоков
Связь между функциональными блоками	В пределах такта	Задержка из-за такта
Время отклика	1 или 2 такта	p количество тактов, где p – количество ресурсов в цепочке событий
Параллельное исполнение	Нет	Да

Межресурсные функциональные блоки находятся между ресурсами и устройствами. Это распределенные системы, в которых функциональные блоки развернуты на отдельных виртуальных или реальных устройствах. Для межресурсных функциональных блоков мы предлагаем шаблон синхронного проектирования с задержкой, который обеспечивает предсказуемое во времени параллельное выполнение функциональных блоков и идеально подходит для распределенных систем. Этот шаблон дизайна вдохновлен [28]. Связь между функциональными блоками задерживается до следующего такта. Это разъединяет коммуникационные зависимости между ресурсами и устройствами. Время отклика равно количеству ресурсов, через которые должно пройти входное событие, чтобы вычислить соответствующее выходное событие. Параллельное выполнение возможно потому, что ресурсы и устройства не взаимодействуют в пределах одного такта.

Хотя мы разработали два разных шаблона, для выполнения распределенных функциональных блоков необходимо сочетание обоих, как показано на рисунке 5.14. Обе модели проектирования предсказуемы во времени, сохраняя детерминизм и реактивность. В будущем наш подход может быть расширен,

чтобы соответствовать другим требованиям приложения к соответствующей семантике выполнения с помощью новых шаблонов проектирования.

Библиография

- [1] JPL Institutional Coding Standard for the C Programming Language, Mar. 2009.
- [2] MISRA-C: 2012: Guidelines for the Use of the C Language in Critical Systems, 2013.
- [3] S. Andalam, P. S. Roop, A. Girault, and C. Traulsen. Predictable Framework for Safety-Critical Embedded Systems. *IEEE Transactions on Computers*, Vol. 99, No. 1, 100–114, Feb. 2013.
- [4] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The Synchronous Languages 12 Years Later. *Proceedings of the IEEE*, Vol. 91, No. 1, 64–83, January 2003.
- [5] M. Boldt, C. Traulsen, and R. von Hanxleden. Worst Case Reaction Time Analysis of Concurrent Reactive Programs. *Electronic Notes in Theoretical Computer Science*, Vol. 203, No. 4, 65–79, June 2008.
- [6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, New York, 1996.
- [7] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications Second Edition*. Springer, 2005.
- [8] J. H. Christensen. Design patterns for systems engineering in IEC 61499. In *Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung (VA2000)*, pages 63–71, Otto-von-Guericke-Universität Magdeburg, Germany, 2000.
- [9] G. Doukas and K. Thramboulidis. A Real-Time-Linux-Based Framework for Model-Driven Engineering in Control and Automation. *Industrial Electronics, IEEE Transactions on*, Vol. 58, No. 3, 914–924, March 2011.
- [10] V. N. Dubinin and V. Vyatkin. Semantics-Robust Design Patterns for IEC 61499. *Industrial Informatics, IEEE Transactions on*, Vol. 8, No. 2, 279–290, May 2012.
- [11] S. A. Edwards and E. A. Lee. The Case for the Precision Timed (PRET) Machine. In *Proceedings of the 44th Annual Design Automation Conference, DAC '07*, pages 264–265, New York, 2007. ACM.
- [12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Boston.
- [13] G. Gebhard, C. Cullmann, and R. Heckmann. Software Structure and WCET Predictability. In *Bringing Theory to Practice: Predictability and Performance in Embedded Systems*, volume 18 of *OpenAccess Series in Informatics (OASIs)*, pages 1–10, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

- [14] J. Herter, P. Backes, F. Hauptenthal, and J. Reineke. CAMA: A Predictable Cache-Aware Memory Allocator. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 23–32, Porto, Portugal, July 2011.
- [15] G. J. Holzmann. The Power of 10: Rules for Developing Safety-Critical Code. *IEEE Computer*, Vol. 39, No. 6, 95–97, 2006.
- [16] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley, Boston, 1997.
- [17] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, Dordrecht, 1997.
- [18] G. E. Krasner and S. T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, Vol. 1, No. 3, 26–49, August/September 1988.
- [19] M. M. Y. Kuo. *Precision Timed Industrial Automation*. PhD thesis, University of Auckland, New Zealand, 2015.
- [20] X. Li and R. von Hanxleden. Multithreaded Reactive Programming – the Kiel Esterel Processor. *Computers, IEEE Transactions on*, Vol. 61, No. 3, 337–349, March 2012.
- [21] F. Maraninchi and Y. Rémond. Argos: an Automaton-Based Synchronous Language. *Computer Languages*, pages 61–92, 2001.
- [22] M. Masmano, I. Ripoll, P. Balbastre, and A. Crespo. A Constant-Time Dynamic Storage Allocator for Real-Time Systems. *Real-Time Systems*, Vol. 40, No. 2, 149–179, 2008.
- [23] A. Roychoudhury. *Embedded Systems and Software Validation*. Morgan Kaufmann, Boston, 2009.
- [24] Z. A. Salcic, P. Roop, M. Biglari-Abhari, and A. Bigdeli. REFLIX: A Processor Core for Reactive Embedded Applications. In *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*, pages 945–945. Springer, 2002.
- [25] F. Serna, C. Catalan, A. Blesa, and J. M. Rams. Design Patterns for Failure Management in IEC 61499 Function Blocks. In *Emerging Technologies and Factory Automation, IEEE Conference on*, pages 1–7, Bilbao, Spain, 20 [26] V. Vyatkin. IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review. *Industrial Informatics, IEEE Transactions on*, Vol. 7, No. 4, 768–781, 2011.
- [27] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al. The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 7, No. 3, 36, 2008.
- [28] L. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic. A Synchronous Approach for IEC 61499 Function Block Implementation. *Computers, IEEE Transactions on*, Vol. 58, No. 12, 1599–1614, 2009.
- [29] L. H. Yoong, P. S. Roop, Z. E. Bhatti, and M. M. Y. Kuo. *Model-Driven Design Using IEC 61499 A Synchronous Approach for Embedded and Automation Systems*. Springer, 2015.

- [30] S. Yuan, S. Andalam, L. H. Yoong, P. S. Roop, and Z. Salcic. STARPro – A New Multithreaded Direct Execution Platform for Esterel. *Electronic Notes in Theoretical Computer Science*, Vol. 238, No. 1, 37–55, 2009.
- [31] G. Zhabelova and V. Vyatkin. Multiagent Smart Grid Automation Architecture Based on IEC 61850/61499 Intelligent Logical Nodes. *Industrial Electronics, IEEE Transactions on*, Vol. 59, No. 5, 2351–2362, May 2012.
- [32] Y. Zhao. A Model-Driven Approach for Designing Time-Critical Medical Devices . M.E. thesis, Electrical and Computer Engineering Department, University of Auckland, New Zealand, 2014.
- [33] A. Zoitl. *Real-Time Execution for IEC 61499*. International Society of Automation eBooks, 2009. 10.

Автоматический реинжиниринг управляющих приложений на основе IEC 61131 в IEC 61499

Monika Wenger
fortiss GmbH

Alois Zoitl
fortiss GmbH

Georg Schitter
Венский технический университет

СОДЕРЖАНИЕ

6.1 Введение	124
6.2 Сравнение IEC 61131 и IEC 61499	124
6.3 Исследования, связанные с реинжинирингом	126
6.3.1 Существующие инструменты реинжиниринга	126
6.3.2 Миграционные подходы IEC 61131 – IEC 61499	127
6.4 Разработанный процесс реинжиниринга	129
6.4.1 Обратный инженерный процесс	131
6.4.2 Соответствие библиотеки	132
6.4.3 Миграция языка	133
6.4.3.1 Общие правила преобразования	134
6.4.3.2 Правила преобразования ROU	135
6.4.3.3 Запросы модели	137
6.5 Доказательство концепции	137
6.5.1 Структура сортировочной станции	137
6.5.2 Результат реинжиниринга	139
6.5.3 Сравнение исходного и целевого поведения	140
6.6 Выводы	145

6.1. Введение

По мере перехода к индивидуальному производству и увеличения объема выпускаемой продукции уровень взаимосвязи [3] промышленных приложений, основанных на новых технологиях, значительно возрастет. Будет повышен спрос на работу распределенных систем [35] с взаимодействующими устройствами разных производителей, что связано с интеграцией большей функциональности в мощные встроенные микрокомпьютеры, подключенные к различным типам объектов [3]. Реальные производственные системы уже достигают границы программируемости [4, 5], вызванной неуклонно возрастающей сложностью систем управления [24]. Во многих операциях технического обслуживания усилия по интеграции новой функциональности и усилия по изменениям в существующих системах становятся неэкономичными [4]. Поэтому будущие системы требуют лучшей адаптации к постоянно меняющимся условиям окружающей среды, а также более легкой обработки растущей сложности и более высоких уровней взаимосвязи. Старение оборудования, дополнительная функциональность и необходимость большей гибкости вынуждают компании менять свои системы. Однако новые компоненты не всегда совместимы с существующими системами управления и поэтому увеличивают затраты на управление различными системами программного обеспечения. Для противодействия все возрастающему управлению различными системами программного обеспечения управления и повышения гибкости действующих решений Международной электротехнической комиссии (IEC) 61131, основанных на стандарте, программное обеспечение управления может быть заменено в целом. Большим недостатком столь масштабной замены является тот факт, что компании уже вложили много денег в разработку своих нынешних систем управления на основе стандарта IEC 61131 и поэтому обычно не хотят начинать с нуля. Для того чтобы действующие системы оставались конкурентоспособными, необходимо автоматизированное преобразование существующих систем управления в системы, основанные на новых технологиях, таких как распределенные системы управления [44]. В этой главе представлен метод автоматического реинжиниринга, который реализован для IEC 61499 в качестве целевой технологии и применяется с управляющим кодом сортировочной станции.

6.2. Сравнение IEC 61131 и IEC 61499

В настоящее время большинство приложений для управления основаны на стандарте IEC 61131. Стандарт был разработан для централизованных систем, что отражается в его языковой структуре, особенно в схеме циклического выполнения, в ее коммуникационной инфраструктуре и поддержке глобальных переменных. Конечно, можно представлять распределенные системы также с помощью методов IEC 61131 и использовать глобальные переменные для

синхронизации и связи, но за счет снижения конфигурируемости, гибкости и ремонтпригодности. Для гибких систем автоматизации требуется все более тонкая модульность, которой вряд ли можно управлять с помощью возможностей моделирования IEC 61131. Объектно-ориентированные (ОО) концепции, представленные в третьем издании IEC 61131, не затрагивали эти темы, но в основном вызвали сдвиг от предметно-ориентированного языка для домена автоматизации до универсального языка программирования (GPL). Вместо этого разработка более богатых и мощных языков программирования приводит только к механически и умственно неуправляемым языкам [12].

Прошлое показывает, что было бы лучше использовать действительно новые и улучшенные инструменты, которые, кстати, не обязательно должны противоречить всему устоявшемуся. С технологиями, которые действительно заслуживают того, чтобы их называли новыми технологиями, мы выиграем из-за более удобного и надежного программного обеспечения, которое проще в использовании и обслуживании» [6]. IEC 61499 [30] как усовершенствованный IEC 61131 обещает быть такой новой технологией. Вместо циклического контроля выполнения и монолитной конструкции, IEC 61499 предоставляет управляемый событиями контроль выполнения для проектирования распределенных приложений. IEC 61499 также обещает безопасный и многократно используемый код с двумя важными характеристиками. Во-первых, он не предоставляет глобальные переменные и поэтому избегает создания скрытых зависимостей между программными модулями, которые были определены как одни из самых сложных и дорогих программных дефектов [14]. Кроме того, он обеспечивает надежную инкапсуляцию данных с помощью алгоритмов, которые способны принудительно выполнять проверки диапазона определенной переменной [14]. Различные типы функциональных блоков (FB), предоставляемые стандартом IEC 61499, могут рассматриваться как отдельные программные компоненты [21], поскольку они соответствуют модели компонентов и стандартизированному интерфейсу без скрытых элементов интерфейса. Они также могут быть развернуты независимо и составлены без модификации и без побочных эффектов от композиции, поскольку компонентная модель способна явно описывать зависимости контекста. IEC 61499 FB могут быть повторно использованы на любой платформе и поэтому являются гибкими. Поскольку каждый компонент обрабатывает свои собственные неисправности, система позволяет легко локализовать их и, следовательно, снижает затраты на ввод в эксплуатацию и техническое обслуживание.

Кроме того, IEC 61499 также обеспечивает независимость от платформы, предлагая расширяемый язык разметки (XML) в качестве формата хранения для всех проектных артефактов и протоколов управления устройствами, что позволяет разрабатывать код программируемого логического контроллера (PLC), который не зависит от аппаратной платформы [30]. Благодаря введению функциональных блоков интерфейса службы (SIFB) он включает механизм защиты интеллектуальной собственности, поскольку вместо предоставления полного исходного кода для FB может передаваться представление XML,

содержащее только интерфейс и описание поведения [36]. Кроме того, IEC 61499 предлагает общее администрирование устройств разных производителей и управляющего программного обеспечения. Благодаря своей архитектуре и картографическим функциям, IEC 61499 позволяет разрабатывать и обслуживать программное обеспечение для управления всей установкой в рамках одного инструмента. Возможность распространения управляющего программного обеспечения на доступные устройства обеспечивает необходимую гибкость для будущих установок [17, 8]. Поскольку IEC 61499 поддерживает онлайн-реконфигурацию [45], данный стандарт также обеспечивает адаптивность, необходимую для будущих систем [3]. Эти характеристики стандарта позволяют ему работать с готовыми системами автоматизации Industry 4.0.

Для создания будущих заводов, IEC 61499 был выбран в качестве целевой технологии для разработки автоматического реинжиниринга приложений управления на основе IEC 61131. После анализа существующих инструментов и подходов к конвертации, начинается процесс реинжиниринга. В основном процесс реинжиниринга применим на любом целевом языке. Предложенный процесс реинжиниринга затем применяется к управляющему коду сортировочной станции, демонстрируя эквивалентность поведения при создании исходной и целевой систем посредством динамического тестирования программного обеспечения на предприятии.

6.3. Исследования, связанные с реинжинирингом

В конце XX века компании затрачивали около 80 % имеющихся у них ресурсов, выделенных на программирование, для поддержки и реинжиниринга существующего кода [9]. В начале XXI века во многих крупных компаниях [28] этот процент превысил 90 % из-за большей сложности программных проектов и быстрого развития новых технологий. Это привело к разработке инструментов реинжиниринга и концепций преобразования в области программного обеспечения и автоматизации.

6.3.1. Существующие инструменты реинжиниринга

Refine™ был коммерческим инструментом, который можно было настроить для желаемой задачи реинжиниринга в сотрудничестве с его поставщиком, Reasoning Software, Inc. (США). Его система языковой обработки поддерживала Refine, Ada, C, COBOL, IBM JCL, SQL и SDL для создания инструментов управления программным обеспечением и реинжиниринга. Refine использовался в качестве основы для некоторых коммерческих проектов – для автоматической миграции в «черный ящик» [2], например, прототипа для автоматического преобразования Natural 1.2 в Natural 2.0, который обрабатывал несколько ключевых несовместимостей между двумя языковыми версиями [9].

Akers и соавторы [2] представили систему поддержки проектирования (DMS) от Semantic Designs Inc., коммерческую систему анализа и преобразования программ. DMS использовалась для создания инструмента для модернизации крупной промышленной авиационной системы на C++. Heidenreich и соавторы [20] представили парсер и принтер Java Model (JaMoPP) для реинжиниринга Java-кода. JaMoPP позволяет представлять код Java в пространстве моделирования, где присутствуют все операторы и аргументы. Поэтому операции реинжиниринга, такие как анализ моделей и подходы к трансформации, могут выполняться и в сфере моделирования. В области автоматизации французская компания Itrix Automation предоставляет преобразователь PLC, способный преобразовать один проект из Schneider-Electric (SMC, Orphée, Série 7), Rockwell Automation (SLC500, PLC5) или Siemens (S5) в специальный PLC (Quantum, Premium, M340, M580) для Schneider Electric Unity (вступает в силу в 2014 году). Для неизвестных продуктов разработки используется база знаний, которая позволяет вручную сопоставлять доступные элементы. Большинство доступных коммерческих инструментов рекомендуют поддержку, поскольку существует «значительная кривая обучения в создании инструментов, основанных на преобразованиях» [2]. Обычно по своей природе эти инструменты не поддерживают специфичные для домена языка автоматизации или ограничиваются специальными типами PLC в качестве платформ ввода и вывода.

6.3.2. Миграционные подходы IEC 61131 – IEC 61499

Для замены целостных приложений PLC приложениями IEC 61499 Hussain и соавторы [22] представили ручное обновление приложений для управления PLC, написанных в сетях Петри с интерпретацией сигнала (SIPN), в IEC 61499. Описание приложения в SIPN используется для генерации программ IEC 61131 и должно быть расширено генератором IEC 61499 на основе их миграционных правил. Обычно управляющие приложения не доступны как SIPN, но уже существуют как приложения IEC 61131, для которых требуется предыдущий переход с IEC 61131 на SIPN.

Riedl и соавторы [29] обеспечили подход для представления последовательных функциональных схем (SFC) сетями FB IEC 61499. Чтобы перестроить приложения SFC как сети FB, авторы вводят два FB: FB_Step и FB_Split_2. FB_Step представляет переход с одним последующим шагом, тогда как FB_Split_2 представляет два альтернативных перехода, выполненных последовательно. Поэтому поток для управления SFC представляется потоком событий. Алгоритмы, выполняемые после активации определённого шага, не принимаются во внимание при их повторной реализации в среде исполнения (runtime environment RTE) распределённой объектной модели (DOME) на языке IEC 61499 или на языке C++.

Чтобы обеспечить первый шаг к независимости производства средних компаний, Gerber и соавторы [15] представили шесть правил преобразования FB IEC 61131 в IEC 61499. Правила сосредоточены на ручном преобразовании

некоторых FB IEC 61131, поскольку они являются кирпичиками распределенной системы управления. Правила преобразования, полученные из миграции этих FB, учитывают определение интерфейсов FB IEC 61499, а также разделение алгоритмов в основных функциональных блоках (BFB). Эти правила не охватывают преобразование всего проекта и не являются достаточно подробными, чтобы поддерживать автоматическое преобразование всего проекта.

Shaw и соавторы [32] представили полуавтоматический реинжиниринг программных организационных единиц (POU), реализованный в виде релейной схемы (LD) Rockwell Automation, которая позволяет производителям промышленной автоматизации оставаться конкурентоспособными и производить гибкие и перестраиваемые системы. Программные модули автоматически преобразуются в специфический для C код RTE, что приводит к нескольким заголовочным и исходным файлам, содержащим типы данных, объявления переменных и процедуры LD в качестве эквивалентов функций C. Сгенерированный C-код должен быть инкапсулирован вручную в соответствующий FB. Следовательно, POU с поведением конечного автомата должны быть вручную встроены в диаграмму управления выполнением (ECC) BFB.

Для более сложных, основанных на состоянии процедур LD они расширили определение ECC в IEC 61499 с помощью иерархических ECC, которые охватывают вызовы подпрограмм, и их также необходимо разрабатывать вручную. Составные функциональные блоки (CFB) используются для инкапсуляции связанного поведения. Интерфейсы этих FB, а также сети FB не генерируются, а выводятся вручную из понимания на уровне системы, полученного путем изучения документа спецификации. Глобальные переменные и переменные, относящиеся к конкретному FB, объединяются заголовочными файлами, указанными в теге CompilerInfo FB. По предположению Вяткина и соавторов [37], они также инкапсулируют непосредственно представленные переменные в SIFB, чтобы предложить простую замену аппаратных изменений. Основными недостатками являются его ограничение на конкретный RTE из-за генерации кода C, несоответствие стандарту IEC 61499 из-за использования иерархических ECC и т. д.

Дай и соавторы [11] представили концепцию миграции для ручной передачи централизованно управляемых систем обработки багажа на основе IEC 61131 в распределенные системы управления, в соответствии с IEC 61499, для преодоления недостаточной производительности PLC в системе обработки багажа в аэропорту при попытке организовать распределенное управление и обеспечить функции сортировки и обнаружения неисправностей для каждого PLC. Они предлагают подход объектно-ориентированного преобразования для управления обработкой. Каждое устройство представлено в виде FB, и, следовательно, исходный код PLC преобразуется в конечный автомат, который впоследствии представляется ECC BFB. Для обработки управляющего кода, когда машины состояний не могут быть легко восстановлены, а части управляющего кода сохраняют последовательную логику, они предлагают

подход объектно-ориентированного повторного использования, который вставляет поведение конечного автомата вместе с последовательной логикой в виде кода структурированного текста (ST) в один алгоритм BFB ECC. Для обработки логистических систем или систем с интенсивной обработкой данных они предлагают подход, ориентированный на класс, где BFB представляют собой единый сервис предприятия.

Ни один из этих подходов миграции не поддерживает автоматическую миграцию целых управляющих приложений IEC 61131 в IEC 61499. Они либо ограничены конкретным RTE, и учитывают только несколько или весьма специфических правил преобразования, или очень общих правил, достаточных для ручной миграции.

6.4. Разработанный процесс реинжиниринга

Считается, что предполагаемый процесс реинжиниринга создаст независимый от поставщика формат для распределенных систем в соответствии со стандартом IEC 61499. Независимый формат касается инженерных инструментов, а также различных типов IEC. Предполагается, что этот процесс объединит различные входные проекты IEC 61131 в рамках одного целевого проекта.

Чтобы автоматизировать языковую миграцию, преобразование программ часто является наиболее эффективной технологией, поскольку требуемые изменения носят структурный (а не текстовый) характер и должны осуществляться повсеместно в больших объемах исходного кода приложения [9]. Для преобразования приложение должно быть подготовлено так, чтобы структурная информация была доступна для механизма преобразования. Представление предметной системы в структурированной форме является основной задачей обратного инжиниринга.

Чтобы улучшить взаимодействие между поставщиками IEC, в 2005 году PLCopen предоставил формат обмена XML [34], который до сих пор не поддерживается всеми поставщиками IEC 61131-3 и поэтому не может использоваться для сокращения количества диалектов IEC 61131-3. Существует столько диалектов IEC 61131-3, сколько имеется структурных описаний. С внедрением третьего издания стандарта IEC 61131-3 в 2013 году пытались решить проблему возрастающей сложности систем автоматизации программирования по мере развития языка и ввели расширения ОО, такие как наследование, ссылки и классы. Поскольку все функции, определенные в стандарте IEC 61131-3, являются дополнительными [27], поставщики обычно поддерживают только подмножества стандартных определений, что, в свою очередь, увеличивает количество диалектов IEC 61131-3. Чтобы упростить программирование в соответствии со своими возможностями, поставщики также добавляют свои собственные определения или функции. Все эти специфичные для поставщика определения создают новые диалекты IEC

61131-3, которые делают мобильность приложений для управления еще более сложной [46].

Из-за большого количества различных диалектов IEC 61131-3 мы предлагаем промежуточный этап вместо прямой миграции с исходного языка на конкретный целевой язык. Это также позволит объединить несколько управляющих приложений в одном целевом проекте. Для окончательного переноса приложений необходимы анализ и фильтрация ценной информации для установления «необходимых структурных отношений» [20]. Райт [43] отметил, что библиотечные элементы, предоставляемые инструментами IEC 61131-3, не всегда присутствуют во всех IEC 61131 и не реализованы одинаково, что приводит к другому поведению. Это обстоятельство действует между инструментами IEC 61131-3 и также влияет на библиотеки IEC 61131 и любой желаемый целевой язык. Чтобы устранить различия в библиотеках, необходимо сопоставить элементы библиотеки как исходной, так и целевой платформы. Выше приведенные требования приводят к процессу реинжиниринга, показанному на рисунке 6.1. Процесс реинжиниринга состоит из трех основных этапов. Первый шаг содержит обратный инжиниринг, который разделен на шаг кода в модель (C2M), который предоставляет структурную информацию с точки зрения конкретной модели поставщика, и шаг модели в модель (M2M), который преобразует специфичную модель поставщика в промежуточную, независимую от поставщика модель. На втором этапе рассматривается соответствие требуемой библиотеки – сопоставление элементов библиотеки исходного проекта с элементами целевой библиотеки. Последний шаг касается языковой миграции, которая преобразует промежуточную модель в желаемую целевую модель и использует модель соответствия библиотек перед сериализацией целевой модели.

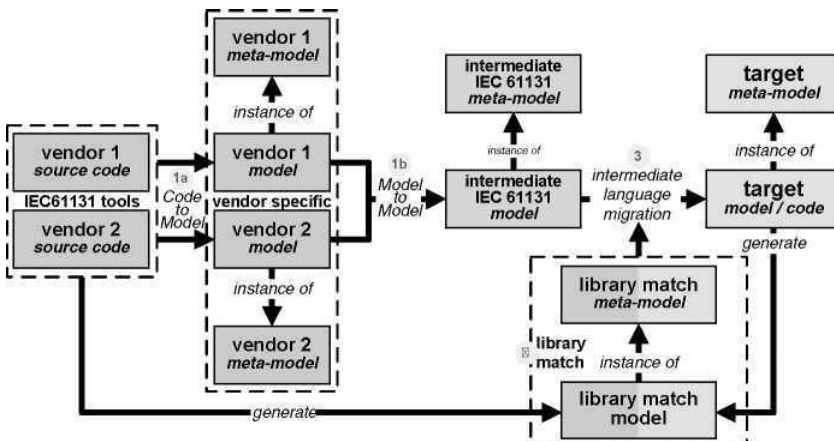


Рисунок 6.1. Процесс реинжиниринга для автоматической миграции устаревших систем управления на основе прямой миграции представлен в [42]

Этот модельно-ориентированный подход выбран, поскольку его технологии обеспечивают поддержку поиска элементов и их сбора без необходимости перебирать все элементы модели. Это упрощает реализацию и поддержку правил преобразования. Кроме того, он позволяет использовать платформы разработки языка, такие как Xtext [13] или EMFtext [19], которые обеспечивают полную языковую инфраструктуру от синтаксических анализаторов через компоновщик, компилятор или интерпретатор, а также классы доступа, сгенерированные из моделей. Второй шаг процесса обратного инжиниринга и миграции языка был реализован с помощью правил преобразования, реализованных в языке преобразования Xtend бывшего openArchitectureWare (oAW) [16]. Три шага реинжиниринга и соответствующие им модели описаны в следующем разделе.

6.4.1. Обратный инженерный процесс

Двухэтапный процесс обратного инжиниринга считывает устаревшие управляющие коды разных поставщиков, чтобы создать независимую от них промежуточную модель IEC 61131. Поскольку есть много поставщиков, использующих свои собственные диалекты языков IEC 61131, шаг C2M приводит к нескольким зависимым от производителя моделям IEC 61131. Из-за этой зависимости от поставщика необходимо выбрать поставщиков стандарта IEC 61131, которые предположительно будут решать проблемы в ближайшей перспективе. Среди различных поставщиков и их инструментов, соответствующих стандарту IEC 61131, Smart Software Solutions GmbH (3S), которая предоставляет CoDeSys V2.3 [1], была выбрана для интеграции в процесс реинжиниринга, поскольку она является распространенным инструментом в домене и предоставляет чистый текстовый источник. Для CoDeSys V2.3 была разработана и расширена собственная метамодель, основанная на структуре промежуточной модели, чтобы интегрировать аппаратную структуру, которая состоит из нескольких модулей и специфических параметров модуля. Аппаратная модель использовалась для восстановления необходимых аппаратных соединений между приложением и аппаратным доступом, разработанным для каждого модуля. Специфичные вендоры метамодели определяют структуру кода и используются в среде разработки языка для генерации синтаксических анализаторов и принтеров, а также для определения и обработки правил преобразования шага M2M.

Специфичные для поставщика правила преобразования обрабатываются для преобразования модели, специфичной для поставщика, в промежуточную модель. Поэтому семантика различных моделей определяется сочетанием метамодели с правилами преобразования. Поскольку существует несколько диалектов IEC 61131-3, существует и множество правил, специфичных для поставщика, таких как диалекты, которые должны быть включены в процесс. Все наборы правил должны компенсировать вариации элементов стандарта

IEC 61131. Например, в управляющих приложениях CoDeSys V2.3 обработка сетевых переменных представляет большой интерес.

Сетевые переменные – это специфичные для CoDeSys V2.3 переменные, которые используются для связи между двумя PLC. Они определены как глобальные переменные и объявляются в теле любого POU, который их использует. Стандарт IEC 61131-3 не предусматривает сетевые переменные, а предоставляет переменные доступа. Поэтому специфическое правило преобразования CoDeSys V2.3 – это отображение его сетевых переменных в виде стандартных переменных доступа IEC 61131-3. Результатом шага M2M является промежуточная модель IEC 61131, которая представляет определения языка стандарта IEC 61131 и должна обеспечивать независимый от производителя формат. Он основан на расширенной форме стандарта Бэкуса-Наура (EBNF) [23] стандарта IEC 61131, который обеспечивает базовую архитектуру IEC 61131 и содержит определения букв, цифр, идентификаторов и литералов, всех возможных типов данных и типов переменных, а также определения для конфигураций и POU. Описание EBNF также содержит три из пяти языков IEC 61131-3, а именно ST, список инструкций (IL) и SFC, однако в нем не представлены функциональная блок-схема графических языков (FBD) и LD. В рамках этой работы метамодель реализована для базовой архитектуры, а также для IL, SFC, ST и FBD, на основе результатов, представленных в [39, 40]. Полученная метамодель была расширена для возможности повторного использования ее элементов в модели соответствия библиотеки. Также были добавлены элементы FBD, отсутствующие в определениях EBNF стандарта IEC 61131. Эти элементы FBD основаны на формате обмена XML, предоставленном PLCopen, а также на определениях сети FB, предоставленных IEC 61499.

6.4.2 Соответствие библиотеки

Существующие управляющие приложения IEC 61131 часто используют стандартные библиотечные блоки или блоки ограниченного поставщика. Обычно нет никакого исходного кода, доступного для этих блоков в виде кода C / C ++ или на любом из языков IEC 61131. Такие ограниченные блоки, которые не могут быть далее разложены и где нет исходного кода, находятся в конце иерархии вызовов. Также непосредственно представленные переменные IEC 61131 относятся к этим типам блоков, так как они осуществляют аппаратный доступ путем отправки или приема сигналов, при этом они скрывают свою фактическую реализацию для аппаратного доступа. Чтобы назначить любой из этих блоков соответствующему блоку целевой системы, была разработана модель соответствия библиотеки на основе [41].

Модель соответствия библиотеки назначает каждый элемент библиотеки IEC 61131 соответствующему элементу IEC 61499, в то время как она также служит таблицей поиска различий имен соответствующих элементов библиотеки и различий интерфейса. В отличие от использования правил

преобразования, модель соответствия библиотек способна компенсировать не обобщаемые различия. Различия типов между исходным и целевым элементами библиотеки также не нужно вычислять путем поиска в истории вызовов, поскольку типы интерфейсов содержатся в модели соответствия библиотеки. Неопределенные ROU на целевой стороне также могут быть обнаружены в процессе миграции, в то время как модель соответствия библиотеки соответствует элементам библиотеки различных инструментов.

Элементы библиотеки каждого поставщика и элементы целевой платформы описаны в их собственных библиотечных файлах, тогда как эти элементы повторно используются в модели соответствия библиотеки в ключевых словах ссылок на блоки различных файлов библиотеки. Модель соответствия библиотеки в настоящее время создается вручную и поэтому требует определенных усилий по инициализации. Но эти усилия требуются только в том случае, если отсутствуют элементы библиотеки или предполагается, что новый поставщик со своими собственными элементами библиотеки будет интегрирован. В этом случае недостающие элементы библиотеки должны быть добавлены в библиотечную модель конкретного поставщика и указаны в модели соответствия библиотеки. Кроме того, они должны быть переопределены для желаемого целевого инструмента. Модель соответствия библиотеки в конечном итоге используется в качестве входных данных для языковой миграции.

6.4.3 Миграция языка

Языковая миграция является последним этапом процесса реинжиниринга. Она превращает промежуточную модель в желаемую целевую модель. Так как мы выбрали IEC 61499 для демонстрационных вопросов, его метамодель представляет определения языка, предоставленные стандартом IEC 61499. Стандарт обеспечивает EBNF и определение типа документа (DTD) для разработки метамодели. DTD может быть преобразован в соответствующее определение схемы XML (XSD), которое может непосредственно использоваться в качестве метамодели. Поэтому он обеспечивает более быструю интеграцию в предлагаемый подход к реинжинирингу, а также напрямую поддерживает постоянство XML, рекомендованное стандартом. Поэтому предоставленный XSD был использован для генерации метамодели Ecore [26] и соответствующих ей классов доступа к модели. По сути, не требуется ни преобразование XSD в Ecore, ни создание классов доступа к модели, также называемых статической реализацией, поскольку выбранный язык преобразования поддерживает модели XSD и способен работать с динамическими реализациями. Даже если статическая реализация не требуется и должна поддерживаться по мере развития модели, она предпочтительна по двум причинам. Во-первых, ей требуется меньше памяти и она обеспечивает более быстрый доступ к данным [25], а также упрощает определение правил преобразования, поддерживая завершение кода в редакторе Xtend и

предоставляя код модели в качестве справочного материала. Во-вторых, она является предпочтительной, поскольку целевая метамодель должна быть расширена для поддержки метамодели соответствия библиотеки и для реализации части метамодели, соответствующей стандарту IEC 61499, для пяти языков стандарта IEC 61131. Это связано с тем, что стандарт IEC 61499 относится только к этим языкам, но не содержит конкретных описаний всех необходимых элементов метамодели. С одной стороны, эти расширения требуют использования ссылок без содержания, которые поддерживаются моделями Ecoge, а не моделями XSD. С другой стороны, определение такого механизма связывания требует равных типов метамодели. Для выполнения языковой миграции необходимы общие правила и правила преобразования ROU. Кроме того, модельные запросы используются для поддержки процесса преобразования с точки зрения соответствия библиотеки, функциональности анализа и проектных решений.

6.4.3.1. Общие правила преобразования

Общие правила преобразования касаются преобразования базовой архитектуры IEC 61131 в элементы базовой архитектуры IEC 61499. Sünder и соавторы [33] представили два основных подхода к преобразованию: преобразование на основе имен и преобразование на основе выполнения, как показано в таблице 6.1. Использование управляемой исполнением или управляемой именем концепции зависит от элементов, предоставляемых поставщиком, а также от конкретной структуры управляющего приложения.

В то время как концепция, управляемая выполнением, согласована с исполняющими элементами, концепция управления именем согласована в стандарте IEC 61131 и стандарте IEC 61499. Поскольку управляемая исполнением концепция ориентирована на исполняемые элементы, ее основным недостатком касается обработки конфигураций, поскольку они суммируются в одной системе и, следовательно, исчезают во время преобразования. Каждая конфигурация представляет один PLC, в то время как в IEC 61499 PLC представлены устройствами. Поскольку все ресурсы 61131 преобразуются в отдельные устройства, это приводит к наличию нескольких устройств, содержащих равные IP-адреса, даже если устройство определено как независимый физический объект [30]. Как следствие, концепция, ориентированная на исполнение, подходит поставщикам, которые поддерживают только конфигурации с одним ресурсом, или для управляющих приложений, которые используют только один ресурс, но множество различных задач. Концепция, основанная на именах, ориентирована на согласованность имен. Ее основным недостатком является обработка задач, поскольку несколько задач могут быть сопоставлены в одном приложении, что может привести к конфликтам выполнения. Следовательно, концепция на основе имен лучше подходит для управляющих приложений, которые

содержат несколько конфигураций и ресурсов 61131 и при этом только одну или совсем мало задач на программу.

Таблица 6.1. Отображение элементов на основе управления именем и на основе управления исполнением [33]

IEC 61131-3	IEC 61499	
	Концепция управления исполнением	Концепция управления именем
Проекты	Система	–
Конфигурации	Устройства	Система
Ресурсы	Ресурсы	Устройства
Задачи	Приложения	Ресурсы
Программы	Приложения	
Функции, функциональные блоки	Функциональные блоки	

Помимо общей архитектуры, Sünder и соавторы [33] также представили правила преобразования для задач, переменных доступа, глобальных переменных и переменных конфигурации. Каждая задача представлена сетью FB, состоящей из FB E_CYCLE для периодического выполнения и E_SWITCH для запуска и остановки цикла, а также SIFB, реагирующим на передний фронт логической переменной для запуска по событию. Желаемая сеть Task FB запускает весь алгоритм управления, в начале каждого цикла любой аппаратный вход считывается, а в конце каждого цикла записывается каждый аппаратный выход. В рамках этой работы мы разделили сеть FB на три разные сети FB, по одной для каждого типа задач – циклических, периодических и запускаемых по событиям, чтобы уменьшить количество дополнительных FB в приложении.

Переменные доступа обеспечивают связь между PLC. В IEC 61499 расширен интерфейс каждого FB, который осуществляет внутренний доступ к такой переменной доступа. Элементы интерфейса, которые представляют прежнюю переменную доступа, соединяются через PUBLISH и SUBSCRIBE SIFB. Глобальные переменные и переменные конфигурации представлены в виде SIFB, тогда как прежние переменные конфигурации должны быть инициализированы во время фазы инициализации приложения, а прежние глобальные переменные должны обновляться при изменении их значения.

6.4.3.2. Правила преобразования POU

Интерфейс любого POU поддерживает входные переменные, выходные переменные, внутренние переменные, временные переменные, внешние переменные, переменные ввода-вывода и пути доступа. Поскольку в IEC 61499

используются типы данных IEC 61131, входные и выходные переменные могут передаваться каждому типу FB, тогда как внутренние переменные предоставляются только BFB без дополнительных усилий, а временные переменные должны обрабатываться как внутренние переменные и должны сбрасываться перед выполнением FB. Для внешних переменных, переменных ввода-вывода и путей доступа требуются элементы интерфейса ввода и вывода, которые позволяют им обновлять значения при каждом изменении.

POU могут программироваться на двух текстовых языках – ST и IL, а также на трех графических языках – FBD, LD и SFC. Предпочтительно, чтобы текстовые языки были интегрированы в алгоритмы BFB или, в случае внутренних вызовов FB, представляли комбинацию BFB и CFB, поскольку BFB и SIFB не способны создавать внутри экземпляры других типов FB.

Поскольку демонстратор, используемый в этой работе, реализован с использованием ST, суммируются только правила для этого языка. На основе структуры ST создается код BFB или CFB. Поэтому код ST добавляется к алгоритму ECC BFB. POU, которые внутренне вызывают другие POU или обновляют значение любого доступа или глобальной переменной, должны быть разделены на несколько алгоритмов, как показано на рисунке 6.2.

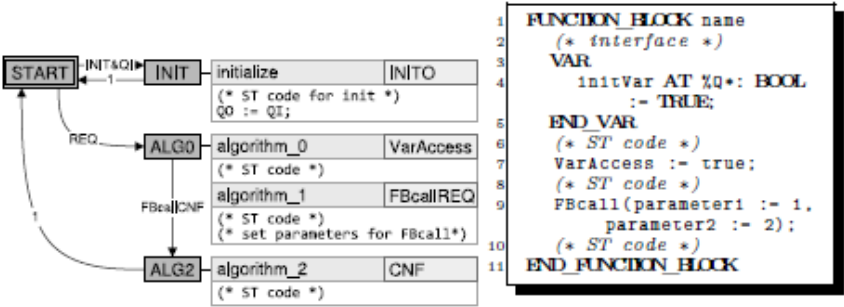


Рисунок 6.2. Схема управления выполнением (слева) для структурированного текстового кода с переменными доступа и вызовом функционального блока (справа)

Поэтому интерфейс BFB расширяется дополнительными элементами для обновления доступа или глобальных переменных и вызова других FB. Доступ к чтению и/или запись для любого доступа или глобальной переменной достигается с помощью дополнительной пары – ввода данных события и вывода данных события. Для каждого вызванного FB в интерфейс добавляется событие запроса и подтверждения, а также зеркальный интерфейс этого экземпляра POU. Зеркальный интерфейс позволяет нам устанавливать и получать параметры вызываемого FB. Для кода ST, который указывает на инициализацию, часть инициализации добавляется в ECC, как показано на рисунке 6.2. Интерфейс расширяется событиями инициализации, связанными с соответствующими входами и выходами данных.

6.4.3.3. Запросы модели

Для того чтобы собрать информацию для существования или генерации определенного шаблона, как описано в [40], или решить, должен ли генерироваться целевой шаблон, задаются модельные запросы. В этой работе различают два типа модельных запросов. Запросы сбора информации проверяют наличие определенного шаблона. Существуют внутренние и внешние запросы сбора информации, которые выполняют внутри сбор информации или вызывают другие запросы для сбора информации. Запросы модели принятия решений основаны на метриках, чтобы автоматически решать, как генерировать целевую модель. Таким образом, целевая модель может быть оптимизирована, например, с точки зрения меньших и более простых для чтения сетей FB. Эти запросы состоят из части сбора информации и части сравнения. Первая часть собирает информацию из исходной модели в соответствии с нужной метрикой по запросам сбора внешней или внутренней информации. Вторая часть сравнивает собранную информацию с заранее определенным и эмпирически определенным порогом соответствующей метрики. На основании результатов этого сравнения выбрано определенное правило преобразования модели. Запросы модели принятия решений используются для проверки того, требуется ли инициализация или ST должен быть представлен, например, BFB или CFB. Расчет для этого решения достигается путем сравнения строк кода (LOC) различных типов операторов. Чтобы провести различие между BFB и CFB, используется следующая метрика:

$$(LOC_{for} + LOC_{while} + LOC_{repeat} + LOC_{case} > 0) \vee \\ (LOC_{statement} - LOC_{CFB_{invocation}} > LOC_{CFB_{invocation}}).$$

6.5. Доказательство концепции

Чтобы показать применимость предложенного процесса реинжиниринга, используется сортировочная станция, как показано на рисунке 6.3 (слева). Машина сортирует три типа объектов, которые транспортируются на поддонах с помощью конвейерной ленты. Сортировочная станция состоит из станции идентификации и двух единиц обработки. Каждая станция оснащена датчиком, который обнаруживает присутствие поддона, и стопором, который необходимо отключить для освобождения поддона.

6.5.1 Структура сортировочной станции

Идентификационная станция сканирует транспортируемый предмет с помощью имитированного датчика цвета (ручной набор определенного цвета). После идентификации поддон освобождается. Если обе единицы обработки неактивны, предполагается, что обе последующие станции свободны. На

основе идентифицированного цвета деталь проходит через систему или сортируется в одном из двух узлов обработки. Каждый неактивный блок обработки отключает свой ограничитель, так что неподходящие части могут проходить через систему, тогда как активный блок обработки активирует свой ограничитель, пока совершается манипуляция. Манипуляции выполняются двумя пневматическими приводами Festo, которые допускают горизонтальные и вертикальные сквозные перемещения благодаря четырем датчикам положения и вакуумному захвату.

Каждая станция управляется одним PLC – Festo CPX-CEC, запрограммированным с помощью CoDeSys V2.3 в ST. PLC CPX-CEC используется, поскольку он поддерживает RTE IEC 61131, а также существует портирование RTE IEC 61499. CoDeSys V2.3 использовался в качестве инструмента разработки IEC 61131, поскольку он поддерживается PLC CPX-CEC. Поскольку каждый из этих проектов содержит программу PLC_PRG, они не требуют постановки задачи и все выполняется циклически. Все три проекта, станция идентификации и обе единицы обработки, проходят реинжиниринг по предлагаемому методу. Сортировочная станция использовалась в лаборатории Одо-Стругер при Институте автоматизации и управления (ACIN) Венского технического университета для демонстрации в рамках финансируемого проекта. В связи с преподавательской деятельностью в этой лаборатории компоненты сортировочной станции, такие как датчик цвета, были необходимы для других станций, и доступное пространство было ограничено. Поэтому для тестирования был доступен только первый блок обработки, как показано на рис. 6.3 (справа). Поэтому второй модуль обработки подвергается реинжинирингу, но не загружается в PLC для целей тестирования. Поскольку вторая единица обработки ведет себя как первая, за исключением сортировки другого цвета, можно предположить, что поведение результата реинжиниринга равно поведению источника, если поведение первой единицы обработки равно поведению источника.

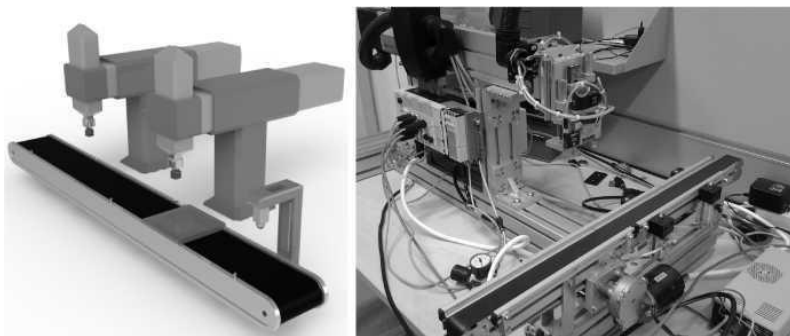


Рисунок 6.3. Структура сортировочного приложения на основе [18] (слева) и аппаратной реализации (справа)

6.5.2. Результат реинжиниринга

Проекты сортировочной станции были обработаны предложенным методом реинжиниринга. В таблице 6.2 перечислены элементы исходных проектов, связанные с соответствующими целевыми элементами, которые автоматически создаются или добавляются в процессе языковой миграции.

Целевые элементы создаются в соответствии с правилами преобразования имен, чтобы лучше сохранить структуру оборудования. Следовательно, целевая модель содержит одну систему с тремя устройствами, по одному для каждой конфигурации исходного проекта и дерева приложений, по одному для каждой программы. Каждое приложение сопоставлено с одним ресурсом 61499, который был добавлен к одному из устройств. Все приложения содержат сеть FB, которая представляет циклическое поведение выполнения задачи и считывает входные данные аппаратного обеспечения, указанные модулями PLC в таблице 6.2, и впоследствии выполняют требуемый алгоритм и записывают выходные данные аппаратного обеспечения, прежде чем снова будет инициировано чтение аппаратных входов. Следовательно, исходный порядок выполнения представлен соответствующей цепочкой событий. Определяемые пользователем программы и FB в исходных проектах определяются как наилучшие реализации в виде BFB посредством запросов модели. Поскольку все программы вызывают подпрограммы в терминах пользовательских FB, а также используют несколько переменных доступа, невозможно просто вставить код программы в один алгоритм, но можно разделить их на несколько алгоритмов, как описано в правилах преобразования POU. На рисунке 6.4 показано результирующее приложение IEC 61499 для одного из двух блоков обработки, состоящее из FB связи (SUBSCRIBE, PUBLISH), аппаратного доступа (DI8, DO4, MPA1S) и всего приложения для управления (PLC_PRG_0, HandlingUnit), запущенного в цикле для представления циклической задачи приложения IEC 61131. Структуры двух других станций аналогичны, поскольку они также содержат связи, аппаратный доступ и все приложения управления FB, выполненные в цикле.

Таблица 6.2. Результат реинжиниринга

Элементы IEC 61131-3	Элементы IEC 61499
Конфигурации: <i>не названы</i>	Устройства: <i>Configuration_0, Configuration_1, Configuration_2</i>
Ресурсы: <i>не названы</i>	Ресурсы: <i>Resource_0, Resource_1, Resource_2</i>
Циклическое задание	Сеть FB
Программы: <i>PLC_PRG</i>	Приложения: <i>PLC_PRG_0, PLC_PRG_1, PLC_PRG_2</i>
Переменные доступа: <i>SORT1_start, SORT2_start, SORT1_complete, SORT2_complete</i>	Переменные интерфейса, подключенные через PUBLISH / SUBSCRIBE

Пользовательские FB: <i>идентификация, единица обработки</i>	BFB: <i>Identification, HandlingUnit</i>
Библиотека FB: <i>AND, OR, TOF</i>	Библиотека FB: <i>F_AND, F_OR, FB_TOF</i>
Модули PLC: <i>DI8, DO4, MPA1S</i>	Библиотека SIFB: <i>DI8, DO4, MPA1S</i>

6.5.3. Сравнение исходного и целевого поведения

Из имеющегося в настоящее время программного обеспечения для промышленной автоматизации, соответствующего стандарту IEC 61499 [10], в качестве целевого инструмента была выбрана структура распределенной промышленной автоматизации и управления (4DIAC) [47], поскольку она была разработана компанией Fortiss совместно с Profactor, ACIN из TU Vienna and MT. Результат реинжиниринга был импортирован в инструмент IEC 61499, 4DIAC-IDE, и среда исполнения 4DIAC (FORTE) была запущена на обоих CPX-CEC Festo. Конфигурация_2 для станции идентификации была загружена в PLC 2, а Конфигурация_0 для первого блока обработки – в PLC 1. На основе функций сортировки было определено несколько тестовых случаев, аналогично подходу, представленному в [38].

```

1  PROGRAM PLC_PRG
2  VAR
3      active : BOOL;
4      gripper : HandlingUnit;
5  END_VAR
6  VAR
7      sensorPresence AT %I*: BOOL;
8      sensorStopper AT %I*: BOOL;
9      stopper AT %Q*: BOOL;
10 END_VAR
11 SORTi_start;
12 SORTi_compl;
13 IF SORTi_start-TRUE AND SORTi_compl-FALSE THEN
14     active := TRUE;
15     stopper := TRUE;
16 END_IF
17 IF active-TRUE AND sensorPresence-TRUE AND sensorStopper-FALSE THEN
18     gripper(active := active);
19     IF gripper.complete-TRUE THEN
20         SORTi_compl := TRUE;
21         active := FALSE;
22         stopper := FALSE;
23     END_IF
24 END_IF
25 IF SORTi_start-FALSE AND SORTi_compl-TRUE THEN
26     SORTi_compl := FALSE;
27 END_IF
28 END_PROGRAM

```

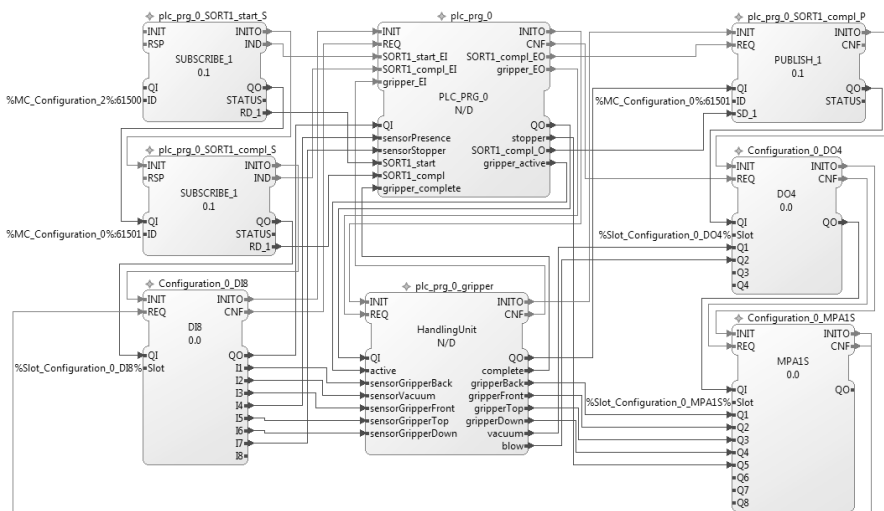



Рисунок 6.4. Исходная программа модуля обработки в структурированном тексте (вверху) и результат реинжиниринга IEC 61499 для модуля обработки (внизу), состоящий из связи (SUBSCRIBE, PUBLISH), доступа к аппаратному обеспечению (DI8, DO4, MPA1S) и всего приложения управления (PLC_PRG_0, Модуль обработки (HandlingUnit)) функциональные блоки.

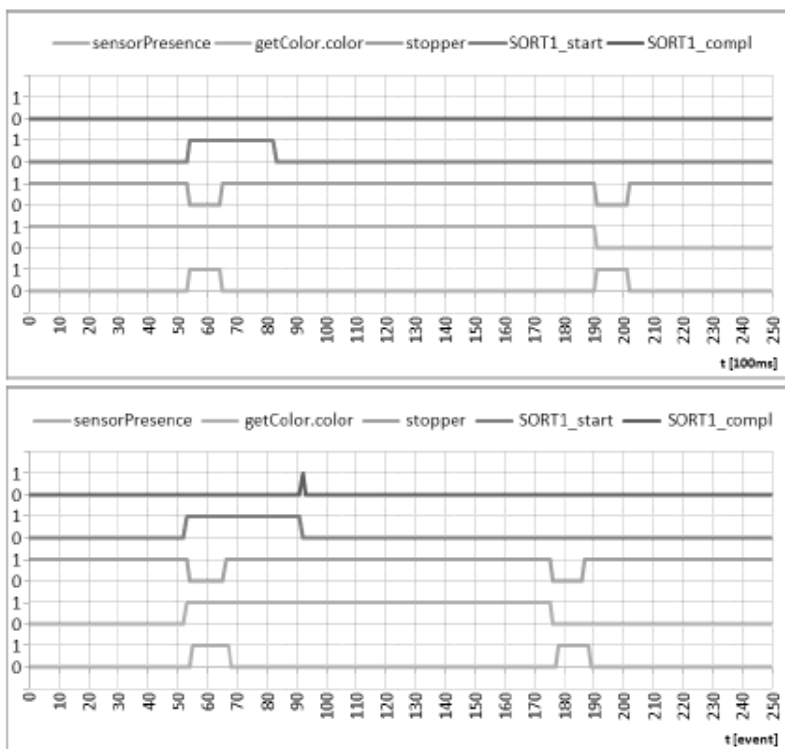
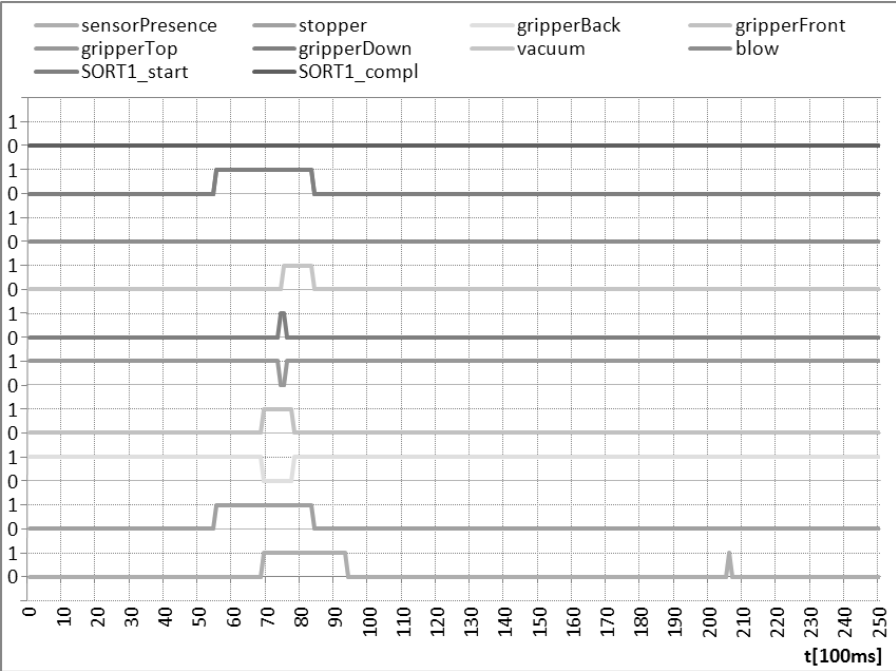


Рисунок 6.5. Сравнение изменений переменных во время выполнения тестового примера ID-станции для CoDeSys V2.3 (вверху) и 4DIAC (внизу).

Чтобы охватить все тестовые примеры, была выполнена следующая последовательность: первая часть снимается с поддона блоком обработки, где вручную устанавливается требуемый вход датчика имитируемого цвета, затем вторая часть проходит через систему, где ввод смоделированного датчика цвета сбрасывается вручную. Тестовые случаи были выполнены для версии сортировочной станции CoDeSys V2.3, а также для реинжиниринговой версии 4DIAC. Во время этих испытаний были измерены соответствующие входы и выходы станций ID и HU.

Реинжиниринговая версия была расширена FB CVS_WRITER, запускаемой основными FB станций (бывшими программами), для записи желаемых значений. Эти измерения показаны на рисунке 6.5 и рисунке 6.6. Обнаружение поддонов представлено двумя пиками. Первый из двух поддонов имел груз, который должен быть отсортирован из системы, тогда как второй прошел через систему вместе со своим грузом. Различные тестовые случаи могут быть идентифицированы по движению захватывающего устройства, проиллюстрированному значениями `gripperBack`, `gripperFront`, `gripperTop` и

gripperDown, а также по перетаскиванию через вакуумный выход и стопор поддона на выход стопора (рисунок 6.6). Временной интервал пиков отличается между реализациями CoDeSys V2.3 и 4DIAC, поскольку они записываются с частотой дискретизации 100 мс, а также записываются в соответствии с числом асинхронных событий CNF, отправленных основными FB. Это особенно отражается при втором обнаружении поддона через сенсор присутствия (рис. 6.6), где основной FB HU выполняется примерно 18 раз, пока поддон не пройдет датчик присутствия.



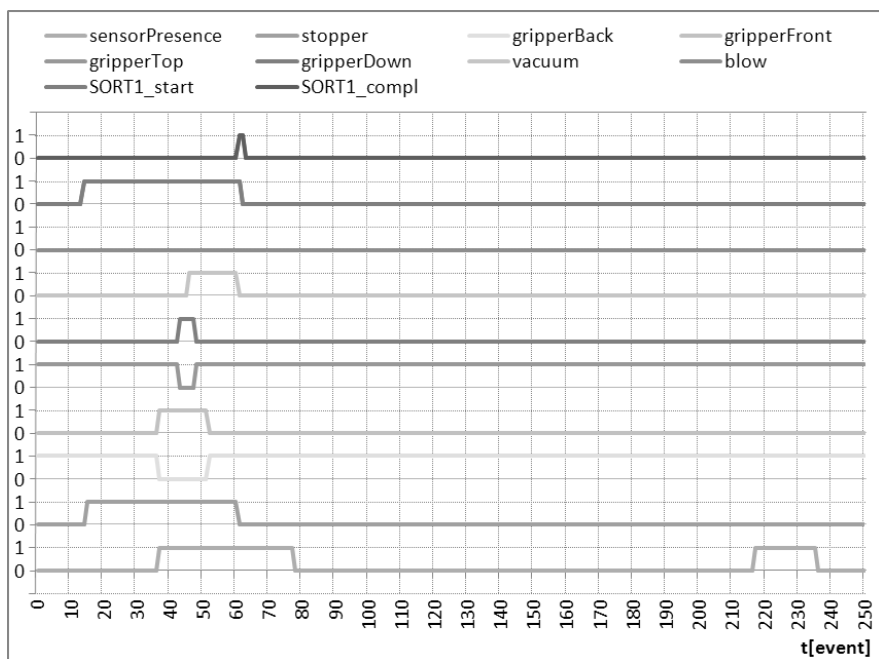


Рисунок 6.6. Сравнение изменений переменных во время выполнения тестового примера станции HU для CoDeSys V2.3 (вверху) и 4DIAC (внизу)

В CoDeSys V2.3 выходной поток (рис. 6.6) никогда не становится истинным, даже если он установлен и сброшен в приложении, что также верно для сетевой переменной SORT1_compl. Такое поведение проистекает из способности PLC выполнить одну операцию примерно за 5 мкс. В пределах 4DIAC не наблюдается никакого пика для выхода удара, поскольку между его установкой и сбросом событие CNF не отправляется, тогда как изменение переменной SORT1_compl может наблюдаться, поскольку изменение значения этой переменной связано с событием.

При сравнении результатов идентификации станции, может наблюдаться разница в значении getColor.color. Для исходного приложения значение getColor.color равно true до обнаружения поддона, тогда как для целевого приложения оно не имеет значения true до обнаружения поддона. Такое поведение исходного приложения происходит из-за смоделированного датчика цвета и, поскольку приложение инициализируется значениями аппаратных входов и выходов (IO) непосредственно после загрузки, независимо от того, запущено приложение или нет. Даже если это не меняет наблюдаемого поведения объекта, это показывает разницу между инициализацией исходного и целевого приложения.

В целевом приложении выходные значения FB, на которые влияет аппаратная инициализация, не обновляются на этапе инициализации. Чтобы

устранить данное различие, в алгоритм инициализации необходимо добавить соответствующие операторы присваивания. Это различие в поведении не вызвало видимых различий в поведении системы, и, поскольку оно было вызвано смоделированным датчиком цвета, его нельзя наблюдать для реального датчика цвета. Таким образом, предложенный подход реинжиниринга привел к успешному автоматизированному преобразованию сортировочной станции на основе IEC 61131-3 в реализацию IEC 61499 на основе 4DIAC, которая демонстрирует такое же поведение.

6.6. Выводы

В этой главе для CoDeSys V2.3 ST представлен процесс реинжиниринга на основе модели для автоматического преобразования приложений для управления, реализованных в IEC 61131, в системы управления, реализованное в IEC 61499. Код ST анализируется модельными запросами для генерации CFB с сетью FB или BFB с ECC в зависимости от его структуры. Результаты реинжиниринга оцениваются контрольным приложением сортировочной станции, которое выполняется в 4DIAC. Весь процесс выполняется автоматически, за исключением сопоставления библиотеки, которое происходит вручную и поэтому требует определенных усилий по инициализации. Для сортировочной станции не нужно было менять ни модель соответствия библиотеки, ни библиотеку целевого инструмента, поскольку ее можно было повторно использовать из предыдущего тестового приложения.

В настоящее время предложенный подход реинжиниринга поддерживает проекты CoDeSys V2.3 ST и проекты logi.CAD FBD. Дополнительные языки и поставщики могут быть интегрированы путем добавления соответствующего шага обратного инжиниринга, который является двух- или однофазным на основе исходного кода. Стоит провести дополнительные исследования для автоматизации начального сопоставления библиотеки и улучшения анализа модели, а также соответствующих запросов модели.

Еще один аспект касается масштабируемости. Примерно из 1200 элементов, сгенерированных из примерно 1275 LOC, включая информацию о конфигурации оборудования, пример демонстрации содержит относительно немного объектов, но большие программные модели могут содержать до 10^9 элементов [31]. Было показано, что EMF очень хорошо справляется с размерами моделей с $3,6 \times 10^6$ объектами для таких задач, как создание, изменение, перемещение, запрос и частичная загрузка [7]. Для больших размеров модели можно было бы рассмотреть возможность уменьшения элементов модели путем изменения метамодели и/или использования базы данных для загрузки или незагрузки отдельных объектов по требованию вместе с методами фрагментации для повышения эффективности использования времени [31].

Насколько нам известно, предлагаемый процесс реинжиниринга в настоящее время является единственным, способным преобразовать весь

проект IEC 61131 из его конфигурации с помощью реализации его POU в эквивалентный проект IEC 61499. Кроме того, он создает независимый от поставщика и платформенно-независимый формат инструмента, который можно использовать для любого PLC, поддерживающего выполнение RTE IEC 61499 с функционально адекватной реализацией специфичных для поставщика библиотечных элементов и обеспечивающего необходимое количество операций ввода-вывода, а также надлежащую инфраструктуру связи. Он даже может объединить несколько проектов IEC 61131. Поэтому получающееся в результате системы управления больше не состоит из нескольких отдельных проектов, а представляет собой только один проект IEC 61499 для всей системы, которая распределяет свои приложения на соответствующее количество PLC.

Библиография

- [1] 3s Smart Software Solutions GmbH. Codesys v2.3.
- [2] R. L. Akers, I. D. Baxter, M. Mehlich, B. J. Ellis, and K. R. Luecke. Re-engineering C++ component models via automatic program transformation. In *12th IEEE Working Conference on Reverse Engineering (WCRE)*, Pittsburgh, pages 7–11. IEEE Computer Society, 2005.
- [3] Arbeitskreis Industrie 4.0. *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*. Forschungsunion im Stifterverband für die Deutsche Wirtschaft e.V., Berlin, Oct. 2012.
- [4] Arbeitskreis 'TB Konzept Normung zu Industrie 4.0' des DKE-Fachbereichs 'Leittechnik' (FB 9). *Die Deutsche Normungs-Roadmap Industrie 4.0*. DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE, Frankfurt am Main, 1.0 edition, Dec. 2013.
- [5] F. Basile, P. Chiacchio, and D. Gerbasio. On the implementation of industrial automation systems based on PLC. *IEEE Transactions on Automation Science and Engineering*, Vol. 10, No. 4, 990–1003, 10 2013.
- [6] M. Bautsch. Cycles of software crises. *European Network and Information Security Agency (ENISA)*, Vol. 3, No. 4, 3–5, 2007.
- [7] A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, and D. Launay. Neo4EMF, a scalable persistence layer for EMF models. In J. Cabot and J. Rubin, editors, *European Conference on Modeling Foundations and Applications*, volume 8569, pages 230–241. Springer, 2014.
- [8] G. Black and V. Vyatkin. Intelligent component-based automation of baggage handling systems with IEC 61499. *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 2, 337–351, April 2010.
- [9] S. Burson, G. B. Kotik, and L. Z. Markosian. A program transformation approach to automating software reengineering. In *IEEE International Computer Software and Applications Conference*, pages 314–322, Chicago, 1990.

- [10] J. H. Christensen, T. Strasser, A. Valentini, V. Vyatkin, and A. Zoitl. The IEC 61499 function block standard: Software tools and runtime platforms, presented at ISA Automation Week, 2012.
- [11] W. Dai and V. Vyatkin. Redesign distributed PLC control systems using IEC 61499 function blocks. *IEEE Transactions on Automation Science and Engineering*, Vol. 9, No. 2, 390–401, 2012.
- [12] E. W. Dijkstra. The humble programmer. *Communications of the Association for Computing Machinery (ACM)*, Vol. 15, No. 10, 859–866, 10, 1972.
- [13] S. Efftinge and M. Völter. oAW xText: A framework for textual DSLs. In *Eclipsecon Summit Europe*, Esslingen, Germany, 2006.
- [14] K. A. Fischer. Concepts and applications of the IEC 61499 PLC programming standard for naval engineers. In *American Society Naval Engineers-Electric Machines Technology Symposium (EMTS)*, Villanova, PA, 2014.
- [15] C. Gerber, H. Hanisch, and S. Ebbinghaus. From IEC 61131 to IEC 61499 for distributed systems: A case study. *EURASIP Journal on Embedded Systems*, 2008.
- [16] A. Haase, M. Völter, S. Efftinge, and B. Kolb. Introduction to openArchitectureWare 4.1.2. In *MDD Tool Implementers Forum (Part of the TOOLS Conference)*, 2007.
- [17] V. Hajarnavis and K. Young. An assessment of PLC software structure suitability for the support of flexible manufacturing processes. *IEEE Transactions on Automation Science and Engineering*, Vol. 5, No. 4, 1545–5955, Oct 2008.
- [18] I. Hegny, M. Wenger, and A. Zoitl. IEC 61499-based simulation framework for model-driven production systems development. In *15th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1–8, Bilbao, 2010.
- [19] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. Modelbased language engineering with emfext. In R. Lämmel, J. a. Saraiva, and J. Visser, editors, *Generative and Transformational Techniques in Software Engineering IV*, volume 7680 of *Lecture Notes in Computer Science*, pages 322–345. Springer, Heidelberg, 2013.
- [20] F. Heidenreich, J. Johannes, M. Seifert, and C. Wende. Construct to reconstruct: Reverse engineering Java code with Java model parser and printer (jamopp). In *International Workshop on Reverse Engineering Models from Software Artifacts*, Lille, 2009.
- [21] G. T. Heineman and W. T. Councill, editors. *Component-based Software Engineering: Putting the Pieces Together*. Addison-Wesley, Boston, 2001.
- [22] T. Hussain and G. Frey. Migration of a PLC controller to an IEC 61499-compliant distributed control system: Hands-on experiences. In *IEEE International Conference on Robotics and Automation*, pages 3984–3989, Barcelona, 2005.
- [23] IEC SC65B. IEC 61131-3 Programmable Controllers. Part 3: Programming Languages, 2003. Geneva.
- [24] K. H. John and M. Tiegelkamp. *SPS-Programmierung Mit IEC 61131-3*, volume 4. Springer, Heidelberg, 2009.
- [25] T. LiB. Learning EMF. Online, 2013.

- [26] W. Moore, D. Dean, A. Gerber, G. Wagenknecht, and P. Vanderheyden. *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. 2004. IBM Redbooks.
- [27] H. Otto. Aktuelles zur IEC 61131 Teil 3. *ATLAS Themenreihe Steuerungstechnik*, pages 45–47, 2012.
- [28] Reasoning Systems, Inc. Reengineering and rewriting legacy software systems (component-based reengineering technology), 1 2002. Advanced Technology Program (ATP) funded project 94-06-0026, project duration 1.1.1995 to 31.12.1997.
- [29] M. Riedl, C. Diedrich, and F. Naumann. SFC inside IEC 61499. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 662–666, Prague, 2006.
- [30] I. SC65B. IEC 61499-1 Function Blocks for Industrial Process Measurement and Control Systems. Part 1: Architecture, 2005. Geneva.
- [31] M. Scheidgen, A. Zubow, J. Fischer, and T. H. Kolbe. Automated and transparent model fragmentation for persisting large models. In *15th International Conference on Model Driven Engineering Languages and Systems*, pages 102–118, 2012.
- [32] G. D. Shaw, P. S. Roop, and Z. Salcic. Reengineering of IEC 61131 into IEC 61499 function blocks. In *8th IEEE International Conference on Industrial Informatics*, pages 1148–1153, Osaka, 2010.
- [33] C. Stünder, M. Wenger, C. Hanni, I. Gosetti, H. Steininger, and J. Fritsche. Transformation of existing IEC 61131-3 automation projects into control logic according to IEC 61499. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 369–376, Hamburg, 2008.
- [34] Technical Committee 6. XML formats for IEC 61131-3. Technical report, PLCopen, 2009.
- [35] VDI/VDE-GMA-Fachausschuss. Industrie 4.0 auf dem Weg zu einem Referenzmodell, 2014.
- [36] V. Vyatkin. IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, Vol. 7, No. 4, 768–781, 11 2011.
- [37] V. Vyatkin, M. Hirsch, and H. Hanisch. Systematic design and implementation of distributed controllers in industrial automation. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 633–640, Prague, 2006.
- [38] M. Wenger, R. Hametner, and A. Zoitl. IEC 61131-3 control applications vs. control applications transformed in IEC 61499. In *IFAC Workshop on Intelligent Manufacturing Systems*, pages 30–35, Lisbon, 2010.
- [39] M. Wenger and A. Zoitl. IEC 61131-3 model for model-driven development. In *38th IEEE Conference on Industrial Electronics Society*, pages 3744–3749, Montreal, 2012.
- [40] M. Wenger and A. Zoitl. Re-use of IEC 61131-3 structured text for IEC 61499. In *IEEE International Conference on Industrial Technology*, pages 78–83, Athens, 2012.

- [41] M. Wenger, A. Zoitl, C. Sünder, and H. Steininger. Semantic correct transformation of IEC 61131-3 models into the IEC 61499 standard. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1–7, Mallorca, 2009.
- [42] M. Wenger, A. Zoitl, C. Sünder, and H. Steininger. Transformation of IEC 61131-3 to IEC 61499 based on a model driven development approach. In *IEEE International Conference on Industrial Informatics*, pages 715–720, Cardiff, 2009.
- [43] T. Wright and C. Powers. Re-engineering legacy controls systems using IEC 61131. In *IEE Colloquia on the Application of IEC 61131 in Industrial Control: Improve Your Bottom-Line Through High Value Industrial Control Systems*, pages 6/1–6/24. Institution of Electrical Engineers (IEE), 2000.
- [44] M. B. Younis. *Re-Engineering Approach For PLC Programs Based On Formal Methods*. PhD thesis, Technische Universität Kaiserslautern, 2006.
- [45] A. Zoitl. *Real-Time Execution For IEC 61499*. International Society of Automation, 2009.
- [46] A. Zoitl, T. Strasser, C. Sünder, and T. Baier. Is IEC 61499 in harmony with IEC 61131-3? *IEEE Industrial Electronics Magazine*, Vol. 3, No. 4, 49–55, 2009.
- [47] A. Zoitl, T. Strasser, and A. Valentini. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study. In *IEEE International Symposium on Industrial Electronics*, pages 3817–3819, Bari, 7 2010.

Модульное тестирование для IEC 61 499 функциональных блоков

Reinhard Hametner
Thales Austria GmbH

Ingo Hegny
Венский технический университет

Alois Zoitl
fortiss GmbH

Содержание

7.1 Введение	151
7.2 Родственная работа	152
7.2.1 Обзор тестового процесса	152
7.2.2 Поблочное тестирование	153
7.2.3 Методы тестовой спецификации	153
7.2.4 Тестирование и проверки	155
7.3 Требования к модульным испытаниям IEC 61499	156
7.3.1 Программные блоки в IEC 61499	156
7.3.2 Наблюдаемое поведение функциональных блоков	157
7.3.3 Спецификация модульного теста для функциональных блоков	158
7.4 Модульные испытания в соответствии с IEC 61499	159
7.4.1 Разделенный тестовый компонент	159
7.4.2 Тестовая спецификация, основанная на данной модели	160
7.4.3 Использование диаграмм последовательности обслуживания для спецификации теста	161
7.4.4 Оценка подходов к спецификации тестов	162
7.5 Структура тестирования на основе результатов	164
7.6 Примеры применения	166
7.6.1 Тестирование функционального блока без сохранения состояния: E_PERMIT	167
7.6.2 Тестирование функционального блока с сохранением информации: E_STU	167
7.7 Выводы и продолжение работы	169

7.1. Введение

Функциональные требования современных систем промышленного производства стремительно возрастают. Кроме того, существует необходимость перенести функциональные реализации с аппаратных компонентов на программные, чтобы иметь гибкие повторно используемые компоненты. Такие компоненты представляют собой инкапсулированные функциональные части, которые необходимы для повышения эффективности проектирования для повторно используемых частей приложения [33]. Поэтому программные компоненты (то есть код промышленной автоматизации) становятся более важными и должны справляться с растущей сложностью современных приложений. Шиперски [31] использует термин «программные компоненты» для программных модулей со некоторыми важными свойствами (например, единица независимого развертывания) и возможностью сторонней композиции, которая не имеет внешних наблюдаемых состояний. Функциональные блоки IEC 61499 (FB) сравнивались с программными компонентами, и аналогичные свойства были определены с помощью Sünder и соавторов [29]. Разработка приложений для промышленных производственных систем становится более трудоемкой, и в короткие сроки приходится сталкиваться с адаптацией управляющего кода.

Важным фактором для эффективного процесса разработки является немедленная реакция на меняющиеся требования на этапе разработки. Изменения механической системы также определяют изменения в электрических и управляющих частях программного обеспечения управляющего приложения [16, 22]. Низкое качество программного обеспечения и более широкое использование непроверенного программного обеспечения в системах автоматизации сопряжено с высокими рисками отказов при выполнении общих требований к качеству системы. Кроме того, повторное использование непроверенных программных компонентов в нескольких проектах автоматизации увеличивает риск. Таким образом, тестирование программного обеспечения помогает повысить и поддерживать высокий уровень качества программного обеспечения в течение нескольких проектов.

Распространенной методикой повышения качества программного обеспечения в области его разработки является разработка через тестирование, которая также известна как разработка в тестовом режиме (TFD). TFD нарушает традиционный рабочий процесс разработки: «кодирование, затем тестирование». Спецификация теста для компонента не может использоваться только для процесса тестирования. TFD также действует как руководство по проектированию и разработке, чтобы помочь разработчику лучше понять проблемы реализации [8]. Таким образом, TFD облегчает автоматическое выполнение тестовых примеров, улучшает взаимодействие между различными заинтересованными сторонами, занимающимися разными дисциплинами, и уменьшает количество дефектов на этапе проектирования и реализации [20].

В первую очередь должны быть указаны все контрольные примеры для тестируемой системы. Инженеры выполняют указанные тестовые примеры для разработанного продукта. Специалисты по системам управления не должны реализовывать указанные контрольные примеры для целевой системы автоматизации. Требуется полностью автоматическое выполнение тестового набора, начиная со спецификации тестового набора [34]. Это помогает компенсировать отсутствие необходимых навыков (например, обширные знания методологии тестирования), а также сокращает циклы внедрения и тестирования.

В этой главе тестовая структура для FB IEC 61499 и осуществимость предложенного подхода демонстрируются на примере реализации.

7.2. Родственная работа

В этом разделе обобщаются методы над процессами тестирования, основанными на методологии TFD, рассматриваются работы по модульному тестированию и объясняется работа, связанная с относящимися к тестированию методами спецификации. Из-за потенциально сложного поведения программных компонентов тесты должны быть полными и всесторонними. Тесты должны быть структурированы иерархически, чтобы повысить понятность.

Распространенным способом структурирования тестов является введение уровней, начиная от нижнего до верхнего уровней. [14, 34]. Самый нижний уровень состоит из тестовых случаев. Они содержатся в тестовых сценариях в ограниченном порядке на основе зависимостей, определенных в поведенческой спецификации. Тестовые сценарии не зависят друг от друга. *Набор тестов* – самый верхний уровень – состоит из одного или нескольких *тестовых сценариев*.

7.2.1. Обзор тестового процесса

Применение стратегий разработки программного обеспечения может помочь повысить качество систем автоматизации за счет улучшения тестируемости и повышения эффективности разработки. Варианты использования определяют последовательность взаимодействия участников с системой и приводят к подходам к тестированию на основе использования. Модели описывают поведение функциональной системы и могут быть использованы для вывода тестовых случаев и кода (то есть это тестирование на основе моделей [8, 34]). Раннее определение тестовых случаев (например, основанных на моделях) приводит к TFD, хорошо известной практике разработки программного обеспечения в гибких (бизнес-) проектах разработки программного обеспечения [2]. На основании спецификаций системы и требований заказчика тестовые примеры указываются до внедрения системы

или компонента. Ранняя спецификация тестового набора позволяет рано обнаружить дефекты в спецификации системы во время создания тестового примера, например, несоответствия между требованиями, а также отсутствующие и/или нечеткие требования. Это представляет собой основу для частых итераций разработки для повышения качества программного обеспечения. Разработка на основе TFD нуждается в тестовых стратегиях [10], которые обеспечивают быструю обратную связь касательно текущего состояния разработки. Shull и соавторы [27] представили исследование, основанное на систематическом обзоре литературы и свидетельствующее об эффективности TFD. Там показано влияние TFD на качество, внутреннее качество и производительность. Мадейски [20] представляет обзор большинства эмпирических исследований, в которых рассматривается TFD и парное программирование в сравнении с тестовым и индивидуальным программированием. Mattesson и соавторы [21] используют TFD и применимый метод для разработки программных проектов, который обеспечивает хорошее повышение эффективности реализации при ограниченных дополнительных усилиях. Эти действия позволяют нам совершенствовать процесс разработки программного обеспечения для систем управления.

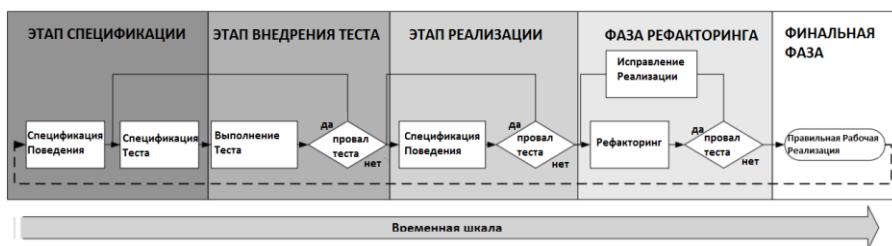


Рисунок 7.1. Процесс тестирования и внедрения процесса разработки в тестовом режиме

На рисунке 7.1 представлен основной рабочий процесс TFD, состоящий из пяти этапов: фаза спецификации, фаза реализации теста, фаза реализации функционального поведения, фаза рефакторинга и финальная фаза. На этапе спецификации будет описано поведение и контрольные примеры для компонента программного обеспечения. На следующем этапе, этапе реализации тестирования, будут реализованы контрольные примеры на основе существующей спецификации. Начиная с этой фазы, тестовые случаи выполняются, но из-за отсутствия реализации поведения выполнение тестового примера должно завершиться неудачей [20]. На третьем этапе, называемом этапом реализации, будет реализовано функциональное поведение, и контрольные примеры могут выполняться многократно, пока не будет достигнуто желаемое поведение.

Если набор тестов (то есть все указанные тестовые примеры) пройден, можно начать фазу рефакторинга. На этом этапе функциональность уже реализована, но качество и структура программного обеспечения могут быть недостаточными. Поэтому реализованное поведение должно быть оптимизировано без изменения функциональности кода. После успешной проверки реорганизованного кода с помощью контрольных примеров достигается финальная фаза. Наконец, если требуется дополнительная функциональность, процесс может быть начат заново с выполнением пяти этапов.

7.2.2. Поблочное тестирование

Программные артефакты без внешних зависимостей используются в качестве основы для модульного тестирования. Их пригодность для использования оценивается с помощью соответствующих ориентированных на интерфейс тестовых случаев, которые проверяют функциональное поведение устройств. Эти тестируемые блоки должны иметь однозначное тестируемое поведение.

Протестированные блоки являются основой для разработки приложений. Следовательно, должно быть проведено испытание как можно большего количества блоков. Поэтому процесс автоматического тестирования, реализованный в процессе TFD, является желательным.

В сообществе разработчиков программного обеспечения тестирование программного обеспечения является хорошо отработанным методом. Несколько методик модульного тестирования оценены и классифицированы в [35]. Поскольку в течение нескольких лет методы гибкого программирования (например, eXtreme Programming) интенсивно используют модульное тестирование, платформы модульного тестирования (например, JUnit) постоянно совершенствуются. Эти структуры модульных тестов обычно требуют, чтобы тесты были реализованы на том же языке, что и тестируемый код.

7.2.3. Методы тестовой спецификации

Для тестирования программных компонентов необходимо указать процедуру или стратегию, сценарии тестирования и контрольные примеры. Тестовые случаи можно классифицировать как нормальные / положительные тестовые наборы, специальные тестовые наборы и ошибочные / негативные тестовые наборы. Нормальные / положительные тестовые случаи необходимы для проверки предполагаемого (заданного) поведения компонентов. Специальные контрольные примеры проверяют поведение компонентов в граничной области предполагаемого поведения (например, верхний или нижний предел). Ошибки / отрицательные тестовые примеры используются

для проверки реализации в случае состояний ошибки, а также поведения при неправильном вводе.

Существует несколько подходов для определения тестового примера. Распространенной методологией тестирования в области разработки программного обеспечения является тестирование на основе моделей, см. [4]. Формальные или полужформальные модели используются для спецификации. Унифицированный язык моделирования (UML) [3], как доминирующий язык моделирования программного обеспечения, является потенциальным инструментом для описания и спецификации функционального и тестового поведения сложных систем управления. Диаграмма состояний UML, являющаяся частью семейства диаграмм UML, в основном используется для структурных процессов тестирования; см. [14, 18, 23]. Выбор полезных и применимых UML-диаграмм для области промышленной автоматизации представлен в [26] и [15]. Чоу [7] описывает метод проверки правильности структур управления на уровне проектирования с помощью конечных автоматов. Следовательно, диаграммы состояний и автоматы используются в процессе тестирования программного обеспечения. Swain и соавторы [30] используют комбинацию диаграмм состояний UML и моделей действий, называемых диаграммой состояний для спецификации и генерации тестовых примеров. Комбинация предоставляет как поток управления, так и информацию об изменении состояния, ориентированного на событие.

Трамбулидис [32] предлагает гибридный подход, который объединяет UML с конструкциями функциональных блоков IEC 61499. Поэтому используется процесс преобразования модели из диаграмм UML в модель проектирования FB.

7.2.4. Тестирование и проверки

Другой подход к соблюдению качества программного обеспечения – формальная проверка (например, проверка модели [17]). При использовании метода проверки программное обеспечение системы управления представляется в виде математических моделей. Они могут быть проанализированы и проверены, если все требования выполняются всегда. Таким образом, правильное поведение проверенного компонента может быть сертифицировано. Несколько подходов к выполнению формальной проверки приложений IEC 61499 были представлены в [6, 9, 25].

Критически важными частями выполнения управляющих приложений являются среда исполнения и аппаратное обеспечение управления. Гербер и Ханиш [11] показали, что среда выполнения может изменить поведение выполнения, и добавили его в свои модели проверки. Однако [28] показал, что полная математическая модель всей среды выполнения и аппаратного обеспечения управления приводит к очень сложным моделям (то есть к неэффективным расчетным усилиям). Для проверки поведения компонентов на реальных устройствах для управления тестирование программного

обеспечения может дополнять методы проверки моделей. Кроме того, методы для тестирования (то есть предполагаемого поведения компонентов) могут поддерживать фазу моделирования для подходов проверки моделей [24].

7.3. Требования к модульным испытаниям IEC 61499

Модульное тестирование – это метод, который помогает обеспечить корректность функционального программного модуля (например, модуля, компонента, класса) в соответствии с его спецификацией. Поэтому первым шагом является определение блоков в IEC 61499. Это включает их интерфейсы, их поведение и спецификации поведения для тестовых случаев.

7.3.1. Программные блоки в IEC 61499

Наибольшее преимущество юнит-тестирования может быть получено, если юниты могут использоваться в различных контекстах без изменения поведения, так как тесты гарантируют, что ложное поведение и возникающие ошибки могут быть вызваны только комбинацией юнитов. Компоненты программного обеспечения, определенные в [31], обладают этой характеристикой. Программный компонент определяется как единица независимого развертывания, стороннего состава и внешне наблюдаемого состояния. Еще один аспект концепции, ориентированной на компоненты, заключается в том, что компоненты должны использоваться без каких-либо знаний об их составе. Чтобы облегчить данный аспект, работа должна быть направлена на тестирование «черного ящика» блоков управляющего программного обеспечения для приложений IEC 61499. Это означает, что для спецификации теста и процесса не требуются знания и предположений о составе устройства. Только публично указанный интерфейс используется для спецификации и выполнения тестов в соответствии с методом тестирования «черного ящика».

Sünder и соавторы [29] показывают, что FB в целом можно рассматривать как программные компоненты при применении концепции программного компонента к IEC 61499. Это особенно верно для базовых FB (BFB), поскольку они идеально инкапсулируют внутренние данные. Содержащиеся алгоритмы и диаграмма контроля выполнения (ЕСС) разрешены только для доступа к внутренним или интерфейсным данным. Для сервисных интерфейсов FB (SIFB) ситуация другая. SIFB содержат скрытые интерфейсы к базовым службам конкретных устройств, что в первую очередь нарушает определение программных компонентов. Во-вторых, эти сервисы должны контролироваться и взаимодействовать с выполняемыми тестами. Обычно это делается во время разработки устройства его поставщиком и требует специальных знаний об устройстве или услуге, предоставленной SIFB.

Наконец, IEC 61499 предоставляет третий выбор, известный как составной FB (CFB). CFB инкапсулирует набор других FB (BFB, CFB или SIFB) и предоставляет эту комбинацию в качестве нового элемента разработчику приложения. Тем не менее для разработчика приложений тип CFB неотличим от любого другого FB (то есть предоставляется интерфейс FB). Поэтому CFB также могут быть блоками для структуры модульного тестирования, рассмотренной в этой главе.

7.3.2. Наблюдаемое поведение функциональных блоков

IEC 61499 определяет FB как пассивные элементы, выполнение которых может быть запрошено входными событиями. Связанные входные данные будут дискретизированы при возникновении инициирующего входного события. Инкапсулированные функциональные возможности будут принимать входные данные, внутренние данные и выходные данные и выполнять свои вычисления, генерируя выходные данные и ноль или больше выходных событий каждый раз с соответствующими выходными данными (см. рисунок 7.2). Это поведение также является отправной точкой для целевых юнит-тестов «черного ящика». С их помощью проверяется наблюдаемое поведение интерфейса FB. Поведение может быть формализовано до функции преобразования FB. Принимается входной вектор $I = (e_i, di_1 \dots di_r)$ и создается выходной векторный набор O ; e_i представляет входное событие, а $di_1 \dots di_r$ представляет все входные данные [13].

$$FB : \vec{I} \mapsto O. \quad (7.1)$$

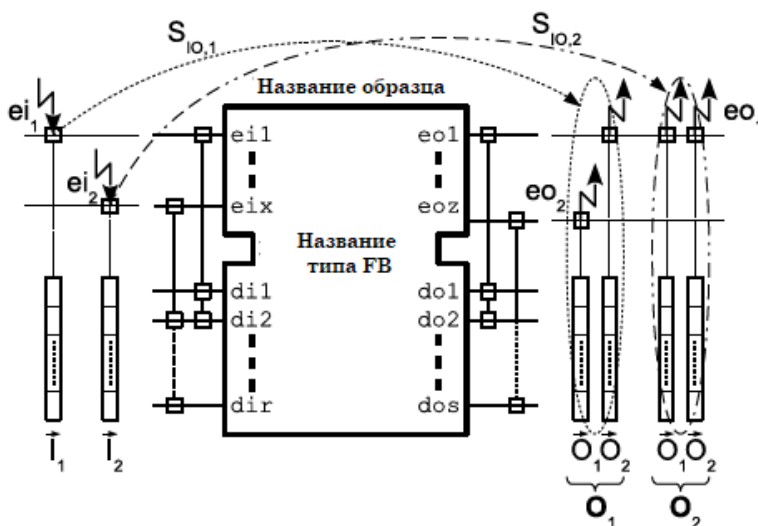


Рисунок 7.2. Наблюдаемое поведение функционального блока IEC 61499 как основы для модульных испытаний

O состоит из нуля или большего количества выходных векторов $\sim \bar{O}_x$

$$\mathbf{O} = (\bar{O}_1 \dots \bar{O}_y), \mathbf{O} = \emptyset : y = 0 \quad (7.2)$$

$$\bar{O}_x = (eo_x, do_1 \dots do_s) \quad (7.3)$$

Подобно входному вектору, eo_x представляет выходное событие, а $do_1 \dots do_s$ представляют все выходные данные.

Отображение в уравнении 7.1 является биективным, только если FB не имеет внутреннего состояния (например, простая функция синуса); y представляет количество выходных векторных элементов. Если $y = 0$, никакого выходного события и никаких выходных данных не существует. Однако, поскольку FB могут иметь внутреннее состояние (то есть состояния ЕСС, внутренние переменные и значения выходных данных), история ранее примененных входных векторов ($\bar{I}_1, \dots, \bar{I}_t$) необходима для определения соответствующего выходного вектора $\mathbf{O}$. Поэтому необходима последовательность кортежей $\bar{I}, \mathbf{O}$ для описания наблюдаемого поведения FB:

$$S_{IO} = \{\{\sim \bar{I}_t, \mathbf{O}_1\}, \dots, \{\bar{I}_t, \mathbf{O}_t\}\} \quad (7.4)$$

7.3.3. Спецификация модульного теста для функциональных блоков

Обычно для выполнения модульных тестов требуется несколько тестовых случаев. Каждый тестовый пример проверяет определенный аспект тестируемого компонента. Такое разделение на контрольные примеры имеет следующее преимущество: тесты легче разрабатывать и поддерживать, и их можно использовать повторно. Кроме того, легче определить причину неудачного теста. Применяя это для тестирования FB IEC 61499, тестовая последовательность будет S_{IO} , как определено в уравнении 7.4. Полный набор тестов U_T представлен набором следующих последовательностей:

$$U_T = \{S_{IO,1}, \dots, S_{IO,u}\} \quad (7.5)$$

Система выполнения модульных тестов под названием «Test Runner» обрабатывает эти данные. Для каждой записи тестовой последовательности применяются входные данные и входное событие для тестируемого FB (FBuT). Кроме того, организатор теста будет отслеживать результаты (события и данные) и сравнивать их с ожидаемыми результатами. Чтобы упростить разработку тестовых случаев, каждая тестовая последовательность предполагает, что FBuT находится в определенном начальном состоянии. Система выполнения модульных тестов должна обеспечить выполнение этого условия.

Для более сложных FB со многими входами и выходами (то есть событиями и данными) эти тестовые последовательности могут стать очень сложными и их трудно определить. Следовательно, необходимы средства, позволяющие описать поведение FB более интуитивно понятным для человека способом, который можно преобразовать в представление модульного теста в

соответствии с уравнением (7.5). Методы спецификации модульных тестов сильно зависят от разработчика модульных тестов. Можно выделить две основные целевые аудитории. Первой являются разработчики управляющих приложений, которые разрабатывают и поддерживают специальные FB для своих приложений. Вторая аудитория – это разработчики библиотеки FB, которые предоставляют общие FB первой аудитории. Их работа в основном сосредоточена на обслуживании, рефакторинге и функциональном улучшении FB, которые являются сложными задачами [36]. Поскольку обе группы работают с моделями IEC 61499, метод спецификации испытаний должен быть знакомым им методом. Продолжая это, Baker и соавторы [1] утверждают, что в тестовой спецификации для эффективного использования должны использоваться те же средства, что и в целевом приложении. Для нашей целевой области это означает, что тестовые спецификации должны быть в лучшем случае моделями IEC 61499 или производными моделями IEC 61499. Исходя из этого, существующие структуры и методы модульного тестирования (например, профиль тестирования JUnit или UML [1]) не могут использоваться напрямую, но они могут служить основой для требуемых функций и возможностей.

7.4. Модульные испытания в соответствии с IEC 61499

На основании выявленных требований ниже подробно описаны и объяснены три потенциальных метода для процесса модульного тестирования по IEC 61499 [12, 13]. Основным отличием этих методов является метод спецификации, поддерживаемый IEC 61499, или внешний метод. Важным моментом является расположение спецификации теста относительно спецификации FBuT. Это еще один отличительный момент.

7.4.1. Разделенный тестовый компонент

Первый естественный подход для структуры модульного тестирования состоит в применении подходов из области разработки программного обеспечения. Это означает, что тесты, например для классов Java, указываются с использованием классов Java, содержащих код теста. Можно показать, что это естественный способ для разработчиков программного обеспечения указывать свои тестовые примеры на том же языке, что и язык разработки или реализации. Применяя этот подход к контрольным случаям для FB IEC 61499, модульные контрольные примеры должны быть реализованы как отдельные и связанные тестовые FB. Эти тестовые FB будут иметь зеркальный интерфейс FBuT (то есть входы станут выходами, и наоборот). Для выполнения тестов оба FB будут подключены, и тесты будут выполнены. Весь процесс разработки и тестирования может быть выполнен в исходной среде разработки IEC 61499

(например, 4DIAC [37]), которая была бы удобна для автоматического управления.

При таком подходе доступны все модели FB IEC 61499, которые можно использовать для реализации теста. ЕСС предоставляет мощные средства для определения контрольных примеров поведения для FBuT. Комбинация ЕСС с несколькими алгоритмами, включенными в FB, может использоваться для определения сложных процедур тестирования. Кроме того, некоторые обычно необходимые процедуры испытаний могут быть предоставлены в отдельных FB. Они могут быть повторно использованы в различных тестовых FB в форме CFB.

Хорошо зарекомендовавшие себя системы тестирования (например, JUnit) автоматически создают тестовые приложения. Напротив, в настоящее время не существует таких генераторов приложений для FB, соответствующих стандарту IEC 61499. Следовательно, тестовые приложения должны разрабатываться вручную и независимо от целевых приложений автоматизации. Такие тестовые приложения нуждаются в контейнерах выполнения IEC 61499 (например, ресурсах) для развертывания и выполнения теста. Из-за разделенных компонентов (то есть тест FB и FBuT) трудно поддерживать изменения в обоих FB (например, изменения интерфейса в FBuT), к тому же это отнимает много времени. Еще одним недостатком является то, что в настоящее время IEC 61499 не имеет средств для выполнения всех требований, необходимых для спецификации испытаний, интерпретации результатов испытаний и представления результатов.

7.4.2. Тестовая спецификация, основанная на данной модели

Второй подход основан на методологии тестирования на основе модели, обычно используемой в разработке программного обеспечения. Основанный на данной модели метод также может быть адаптирован для тестирования промышленных компонентов управления (например, FB IEC 61499). В этом подходе используется отдельная спецификация теста в форме модели программного обеспечения, которая может быть выведена из спецификаций системных требований и позволяет автоматически генерировать код и тестовый набор [34]. Целью этого метода является автоматическое внедрение исполняемых тестовых случаев для FB IEC 61499 из их тестовых спецификаций.

Для моделирования тестовой спецификации семейство диаграмм UML является подходящим решением. Это мощный язык спецификации, широко используемый и принятый в области разработки программного обеспечения. Несколько типов диаграмм UML применимы для спецификации теста [12]. Ограничивающий профиль UML может обеспечить необходимую ясность для реализации тестового примера. Хаметнер и соавторы [14] описывают процесс преобразования модели, который использует информацию тестового примера из диаграммы состояний UML и автоматически генерирует тестовый FB IEC

61499. Это подобно подходу в разделе 7.4.1. Данный FB включает в себя все исполняемые тестовые примеры. ЕСС и алгоритмы на языке структурированного текста используются для реализации теста.

Преимущество по сравнению с предыдущим подходом к тестированию состоит в том, что с изменениями интерфейса и/или изменениями функционального поведения FBuT можно легче справиться, поскольку исполняемые тестовые примеры генерируются автоматически. Этот подход поддерживает регулярные изменения спецификации теста (например, модели UML). Следовательно, усилия по изменению и поддержанию спецификации теста параллельно спецификации поведения уменьшаются. Кроме того, не требуются навыки программирования для определения и поддержки модульных тестов, что повышает приемлемость TFD в области промышленной автоматизации.

Недостатком данного подхода является один из недостатков предыдущего подхода – раздельное обслуживание FBuT и соответствующая спецификация тестирования.

7.4.3. Использование сервисных диаграмм последовательности для спецификации теста.

Диаграммы сервисных последовательностей позволяют задавать последовательности входных событий и результирующих выходных событий. Кроме того, значения входных данных могут использоваться для спецификации теста FB. Кроме того, IEC 61499 допускает спецификацию нескольких последовательностей действий, каждое из которых описывает определенный аспект выполнения FB. Это может быть использовано для определения нескольких последовательностей, которые будут проверять различные аспекты поведения интерфейса FBuT в предварительно определенном порядке. Последовательности обслуживания напоминают общее описание модульного теста FB, как определено в уравнении (7.5). Кроме того, положительные, специальные и отрицательные тестовые случаи могут быть определены и использованы для процесса тестирования.

Основным преимуществом этого подхода является то, что спецификация теста напрямую включена в определение типа FBuT. Поэтому специальная спецификация теста всегда является частью определения FB, и не может возникнуть несоответствия между различными версиями спецификации. Кроме того, метод спецификации теста является родным методом IEC 61499, и для изучения спецификации тестового примера требуется только ограниченное дополнительное обучение IEC 61499.

Для FB со сложными интерфейсами диаграммы последовательности услуг могут стать сложными, но последовательности услуг могут помочь сосредоточиться на конкретных аспектах поведения интерфейса. Кроме того, аналогичные задачи, необходимые в нескольких тестовых последовательностях (например, приведение FBuT в предварительно

определенное состояние), должны быть включены в каждую тестовую последовательность. IEC 61499 не допускает вызова диаграммы последовательности из другой последовательности. Здесь такие расширения, как вызовы последовательностей, циклы или условия, введенные в диаграммы последовательностей UML во втором издании IEC 61499, могут также оказаться полезными для диаграмм последовательности действий.

7.4.4. Оценка подходов к спецификации тестов

Требования, указанные в разделе 7.3, выполняются при помощи трех подходов. Несколько критериев были определены как относящиеся к области промышленной автоматизации. Сравнение подходов представлено в таблице 7.1, которая показывает результаты оценки для шести выбранных свойств. Три различных подхода для сравнения оцениваются тремя значениями (+,0,-):

- + означает, что свойство полностью поддерживается;
- 0 означает, что свойство поддерживается только частично; реализация техники тестирования возможна при некоторых усилиях;
- – показывает, что свойство не поддерживается, поэтому тестирование невозможно.

Сравнив все подходы, можно сделать вывод, что первый подход из раздела 7.4.1 определен как имеющий наибольшую гибкость в отношении спецификации испытания. Этот подход напоминает современные методы модульного тестирования, применяемые в области разработки программного обеспечения. Подход на основе моделей с использованием диаграмм UML (как описано в разделе 7.4.2) является мощным. Однако ограничения (то есть профиль UML) необходимы для однозначного выполнения тестового примера, что также ограничивает гибкость. Что касается гибкости, использование диаграмм последовательности действий в разделе 7.4.3, безусловно, является наиболее ограниченным методом.

Таблица 7.1. Результаты оценки для отдельного компонента теста (7.4.1), спецификации теста на основе модели (7.4.2) и диаграмм последовательности использования службы (7.4.3)

Оценка качества	7.4.1	7.4.2	7.4.3
Гибкость в спецификации теста	+	0	–
Портативность	0	–	+
Читабельность	–	+	0
Целостность	–	0	+
Выразительность	+	0	–
Тестовое обслуживание	–	0	+

Преимущество первого и третьего подходов состоит в том, что образцы, необходимые для тестирования (то есть тестирование FB и диаграммы последовательности тестовых сервисов) могут быть доступны для всех

инженерных инструментов, соответствующих IEC 61499. Переносимость, то есть обмен инженерными данными (например, FB) между инструментами, является важным свойством, определенным в стандарте. Однако обмен диаграммами UML между различными инструментами разработки проблематичен из-за распространенной несовместимости форматов данных.

UML был указан для реализации программного обеспечения. Поэтому удобочитаемость важна для хорошего анализа спецификации поведения компонентов. Диаграммы последовательности обслуживания легко понять. Тем не менее обзор всех тестовых случаев вряд ли можно получить для сложных FB. Относящиеся к тесту свойства (например, ECC, алгоритмы, интерфейс FB) распределены внутри тестового FB. Следовательно, ни один тестовый пример, ни полный набор тестов не могут быть легко поняты с первого взгляда.

В третьем подходе из раздела 7.4.3 информация, относящаяся к тестированию, является частью определения типа FBuT. Поэтому он не может потеряться или забыться при передаче. Ограниченная целостность обеспечивается подходом из раздела 7.4.2, если та же модель UML также используется для спецификации поведения FB, предложенной в [32]. Основным недостатком первого подхода является то, что спецификация и реализация теста полностью отделены от FBuT, что может привести к появлению несоответствий.

Все средства для спецификации и реализации FB могут быть использованы в первом подходе. Второй и третий подходы требуют универсального интерпретатора для спецификации теста. Для однозначной интерпретации спецификаций тестовых примеров должна быть ограниченная выразительность. Диаграммы последовательности услуг IEC 61499, используемые в третьем подходе, обеспечивают лишь ограниченную выразительность. Следовательно, Runner Test может легко достичь однозначной интерпретации указанных тестовых случаев.

Интегрированные методы моделирования, предложенные в третьем и втором подходах (если поведение FB также указано в UML), позволяют легко проверять несоответствия между FBuT и спецификацией теста (например, изменения интерфейса). Эти задачи обслуживания испытаний труднее выполнить с полным разделением испытаний и FBuT, как предлагается в первом и втором подходах.

После оценки критериев оценки в качестве основы для системы модульных испытаний IEC 61499 был выбран третий подход из раздела 7.4.3. Удобство использования диаграмм последовательности действий может быть повышено за счет потенциальных улучшений в IEC 61499 (например, циклы, вызовы последовательностей, условия). Преимущества этого подхода (мобильность, целостность и простота обслуживания теста) перевешивают его недостатки.

7.5. Структура тестирования на основе результатов

Основываясь на подходе спецификации тестирования диаграммы последовательности услуг, реализована тестовая структура, показанная на рисунке 7.3. Тесты в форме диаграмм последовательности действий являются частью файла типа FB, как и все другие определения, специфичные для типа FB.

Тестовый прогон (см. рисунок 7.3 (а)) является основным элементом в представленной тестовой среде и обрабатывает анализ, выбор, выполнение и оценку для тестовых случаев. Test Runner реализован и включен в инженерную среду IEC 61499 на базе Eclipse 4DIAC-IDE [37] в качестве дополнительного подключаемого модуля с открытым исходным кодом. Тесты определяются диаграммами последовательности сервисов (см. рисунок 7.3 (б)), которые хранятся в файле типа FB вместе с интерфейсом и реализацией поведения. Тестовые случаи и интерфейс FB анализируются программой Runner. Собранная информация интерпретируется как набор тестовых случаев, каждый с входным вектором и набором ожидаемых выходных векторов.

На основе определения интерфейса FBuT автоматически создается тестовое приложение для конкретного типа, которое создается для процедуры на удаленном тестовом устройстве. В соответствии со стандартом IEC 61499 используемая среда выполнения – 4DIACRTE [37]. Тестовое приложение состоит из FBuT, FB для связи с организатором тестирования (получателем, отправителем), а также мультиплексными и демультиплексированными FB (MuX, DeMuX). MuX и DeMuX FB являются общими FB; это означает, что среда выполнения может создавать экземпляры FB с переменным числом входов и выходов событий в соответствии с интерфейсом FBuT. Кодирование событий в виде данных позволяет синхронную передачу набора входных данных по однонаправленному соединению от Runner Test к тестовому устройству. Также упрощается синхронная передача выходного вектора, отправленного обратно из тестового устройства в тестовый прогон.

Для связи между Test Runner и тестовым устройством выбран протокол TCP/IP на основе соединения. Номера входов и выходов функций отправителя и получателя получены из интерфейса FBuT. Для передачи закодированных событий добавляется дополнительный ввод данных для отправителя и дополнительный вывод данных для получателя. Необходимые интерфейсы показаны на рисунке 7.3 (с).

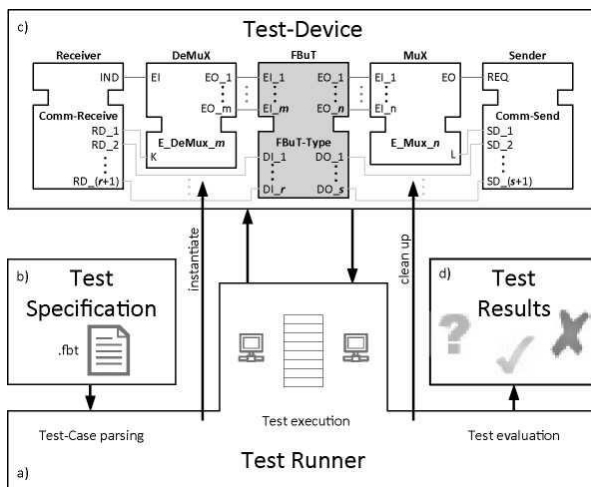


Рисунок 7.3. Общие рамки модульного тестирования: а) Test Runner; б) тесты указываются в файле типа FB вместе с интерфейсом и поведением; в) конкретное тестовое приложение типа FB создается и развертывается в целевой системе управления; д) результаты испытаний сообщаются и визуализируются.

Процесс тестирования выполняется, как описано ниже.

1. Типовое тестовое приложение создается на тестовом устройстве.
2. Test Runner начинает оценку указанных тестовых случаев.
3. Test Runner передает заданный входной набор на тестовое устройство.
4. Выполнение FbUT запускается указанным событием ввода.
5. Полученные выходные события и данные передаются обратно в Runner.
6. Test Runner удаляет тестовое приложение из устройства (этап очистки), анализирует ответы из тестового приложения и оценивает результаты модульного теста (см. рисунок 7.3 (д)).

Определенная тестовая последовательность проходит проверку, если набор принятых выходных векторов равен ожидаемым выходным векторам. И те, и другие взяты из спецификации теста. Как только один из следующих критериев удовлетворяется, последовательность испытаний оценивается как сбой:

- количество полученных выходных векторов не равно указанному количеству;
- в выходном векторе получено неверное событие;
- выходные данные не соответствуют ожидаемым значениям;
- достигнуто заранее заданное значение тайм-аута выполнения теста.

Предстоящие тесты в пределах одной и той же последовательности не оцениваются, поскольку предварительные условия могут не выполняться. После выполнения всех тестов тестовое приложение удаляется с устройства, а результаты сообщаются и визуализируются с помощью средства разработки.

Разделение Test Runner и тестового устройства имеет следующие преимущества.

1. Тесты могут быть выполнены в развернутой среде. Ошибки и изменения при портировании на новые устройства могут быть проверены.

2. Для создания тестового приложения на тестовом устройстве требуются только стандартные FB вместе с FBuT.

3. Любой тип FB (включая SIFB) может быть протестирован на устройствах, для которых существует реализация.

4. Выполнение сетевых тестов позволяет проводить тесты на платформах, которые не предоставляют возможности визуализации или доступа к файловой системе (для сохранения отчетов). Оценка теста в инженерной системе еще больше снижает необходимость предоставления памяти для сбора результатов на тестовом устройстве; при этом тесты могут выполняться даже на небольших устройствах.

5. До тех пор, пока устройства предоставляют стандартные совместимые интерфейсы связи, могут проводиться испытания устройств различных производителей. Предлагаемая тестовая среда также учитывает влияние разной семантики исполнения на разных целевых платформах, которая возникает из-за разных интерпретаций стандарта [6, 5].

7.6. Примеры применения

Для проверки функционального блока выбраны репрезентативные тестовые рамки, которые приемлемы FB. Выбор FB выполняется на основе типичных функций выполнения и свойств FB, которые необходимо учитывать при тестировании. Подробная таксономия функций и свойств исполнения представлена на рисунке 7.4.

В результате классификации и оценки были выбраны два хорошо известных FB, которые определены в Приложении А IEC 61499-1 [19]. FB E_PERMIT и E_STU способны обеспечить понимание спецификации и выполнения модульных тестов для FB.

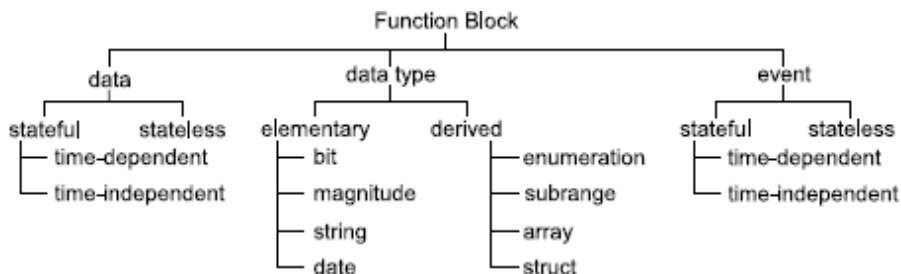


Рисунок 7.4. Таксономия особенностей и свойств выполнения наблюдаемого функционального блока

7.6.1. Тестирование функционального блока без сохранения состояния: E_PERMIT

FB E_PERMIT обеспечивает разрешающее распространение события. Как видно на рис. 7.5 (а), FB имеет один вход события EI и связанный с ним логический вход данных PERMIT. Поведение FB E_PERMIT определено как не имеющее состояния в отношении данных и событий. Наблюдаемое поведение выхода в результате входного события EI зависит только от значения PERMIT. Поскольку это булевы данные, допустимы следующие два вывода:

- PERMIT ложно: никакое выходное событие не должно запускаться;
- PERMIT истинно: одно выходное событие ЕО должно быть запущено.

Поскольку два тестовых примера не зависят друг от друга, они указываются как отдельные тестовые последовательности. Дополнительный тест (то есть отрицательный тест), который должен потерпеть неудачу при выполнении, проверяет функциональность тестовой среды.

Спецификация тестовых случаев редактируется с помощью редактора сервисных последовательностей 4DIAC-IDE; см. рисунок 7.5 (б). Все три теста с соответствующими входными и выходными векторами видны на рисунке. Поскольку сервисные последовательности включены в определение типа FB и используются только методы описания, соответствующие IEC 61499, любой редактор типов FB может использоваться для изменения тестовых случаев.

7.6.2. Тестирование функционального блока с сохранением информации: E_CTU

Второй FB – это E_CTU FB. Его интерфейс включает в себя два входных события (CU, R) и два выходных события (CUO, RO), а также один вход данных (PV 16-разрядного целого типа без знака UINT) и два выхода данных (Q булева типа, CV типа UINT), как показано на рисунке 7.6. Значение CV

увеличивается при возникновении события CU и сбрасывается до 0 при возникновении события R. Поведение FB зависит от значения счетчика, поэтому FB E_CTU независим от времени по отношению к данным.

Использование типа абсолютных данных в качестве входных и выходных данных значительно увеличивает число потенциальных тестовых случаев. Существует 2^{17} возможных входных векторов (то есть 2 не одновременных входа события + 1×16 -битный ввод данных). Кроме того, этот понятный пример уже имеет 2^{18} состояний выходного интерфейса. Следовательно, для исчерпывающего тестирования FB E_CTU потребуется 2^{35} (то есть $2^{17} \times 2^{18}$) тестовых случаев.



Рисунок 7.5. Стандартная библиотека IEC 61499 FB E_Permit и ее тестовая спецификация с 4DIAC

Требование тщательно продуманного выбора тестового примера очевидно. Для типа FB E_CTU был указан репрезентативный набор тестовых случаев (включая специальные тесты и негативные тесты).

Тестовые случаи анализируются программой Runner, когда они запускаются пользователем в пользовательском интерфейсе (плагин в 4DIAC-IDE). В одном и том же пользовательском интерфейсе результаты тестов представляются как в текстовом виде (с подробной информацией о неудачных тестах), так и в графическом виде (с хорошим обзором результатов тестов), как показано на рис. 7.7 (b). При появлении первой ошибки (см. Отрицательный тест) последовательность испытаний останавливается и помечается как неудачная (см. Крестик на рисунке 7.7 (b)). Дальнейшие тестовые случаи в

последовательности неудачных тестов помечаются как непроверенные (см. Вопросительный знак на рис. 7.7 (b)).

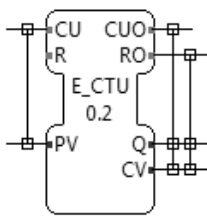


Рисунок 7.6. Спецификация интерфейса стандартной библиотеки IEC 61499 FB E_CTU

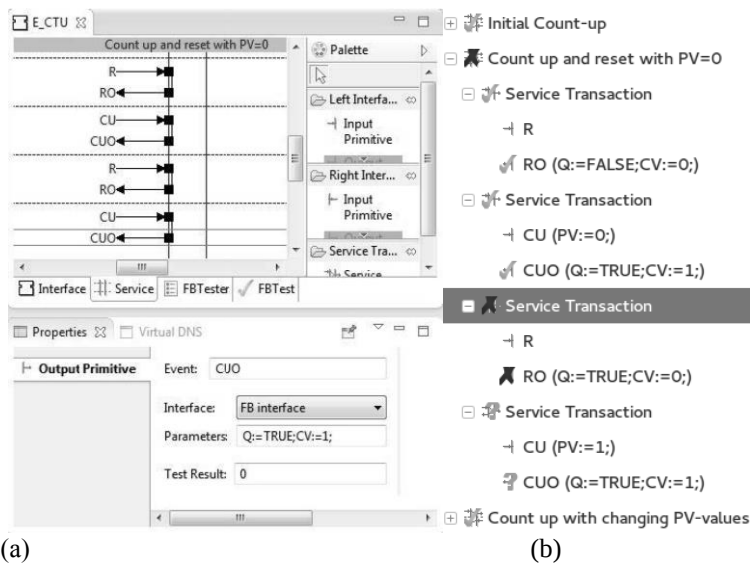


Рисунок 7.7. E_CTU FB: (a) подсчитано и сброшено: с PV = 0; диаграмма последовательности испытаний. (b) Результаты испытаний: сценарии 1 и 3 успешны; сценарий 2 потерпел неудачу из-за отказа сценария 3; сценарий 4 не выполнен в результате; (a) спецификация испытания с диаграммой последовательности; (b) результаты выполнения модульного теста трех сценариев.

7.7. Выводы и продолжение работы

Популяризация использования программного обеспечения для решений промышленной автоматизации требует повышения качества программного обеспечения при одновременном снижении затрат на разработку. Подход TFD

оказался ценным решением этой проблемы в области разработки программного обеспечения.

Структура модульного тестирования, подходящая для промышленной автоматизации, представлена в качестве основы для подхода TFD. FB идентифицируются как программные модули, подлежащие тестированию в IEC 61499. На основе формальной спецификации их внешнего наблюдаемого поведения определяются метод спецификации модульного теста и соответствующая структура выполнения модульного теста. Основными преимуществами представленного решения является то, что оно полностью соответствует стандарту IEC 61499 и может использоваться в любой среде, соответствующей стандарту IEC 61499. С двумя репрезентативными FB, взятыми из стандартной библиотеки IEC 61499, показано использование предложенного подхода и доказана его применимость.

Представленная структура модульного тестирования была добавлена в 4DIAC и доступна во всех текущих выпусках 4DIAC-IDE. Кроме того, два тестовых примера, представленные в этой главе, также доступны в библиотеке 4DIAC FB. Это позволяет заинтересованным исследователям протестировать и оценить пригодность структуры модульного тестирования.

Определены ограничения диаграмм последовательности действий в случае FB со сложными интерфейсами. Последовательности становятся очень большими, и эффективно указано может быть только подмножество. Решением для таких FB может быть моделирование тестовых случаев с использованием UML и автоматическое создание результирующих диаграмм последовательности действий в качестве входных данных для структуры модульного тестирования. Преимущество этого заключается в том, что тестовые случаи могут быть легче разработаны и что тесты хранятся как часть FB.

Чтобы преодолеть эти ограничения, предлагается набор расширений служебных последовательностей IEC 61499, который уменьшает усилия для определения тестов и анализа поведения FB за счет повышения удобства их использования. Первым шагом, безусловно, будет введение последовательностей вспомогательных услуг и механизма вызова, после чего общие задачи, необходимые для нескольких тестовых случаев (например, подготовка FB), могут быть определены единожды и повторно использованы несколько раз. Дополнительными расширениями являются управляющие структуры, условное выполнение и циклы для реализации повторяющихся тестовых последовательностей. Наконец, поскольку приложения для управления ограничены в реальном времени, поведение синхронизации является важной характеристикой. Следовательно, должна быть возможность указать временные характеристики в тестовых последовательностях. Эти требования были представлены комитету по стандартизации IEC 61499 и будут учтены при разработке IEC 61499-5. Представленная тестовая среда является одним из возможных решений для изменения требований в сжатые сроки на этапе разработки, чтобы поддерживать качество программного обеспечения промышленного контроля на высоком уровне.

Библиография

- [1] P. Baker, Z. Dai, and J. Gabrowski. *Model-Driven Testing: Using the UML Testing Profile*. Springer, 2007.
- [2] K. Beck. *Test-Driven Development by Example*. Addison-Wesley, 2002.
- [3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
- [4] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner. *Model-Based Testing of Reactive Systems*. Springer, 2005.
- [5] G. Cengic and K. Åkesson. On formal analysis of IEC 61499 applications, Part B: Execution semantics.
- [6] G. Cengic and K. Åkesson. On formal analysis of IEC 61499 applications, Part A: Modeling. *Industrial Informatics, IEEE Transactions on*, Vol. 6, No. 2, 136–144, May 2010.
- [7] T. Chow. Testing software design modeled by finite-state machines. *Software Engineering, IEEE Transactions on*, Vol. SE-4, No. 3, 178–187, May 1978.
- [8] L. Crispin and J. Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley, 2008.
- [9] V. Dubinin, V. Vyatkin, and H.-M. Hanisch. Modelling and verification of IEC 61499 applications using Prolog. In *IEEE Conference on Emerging Technologies and Factory Automation*, pages 774–781, Sept. 2006.
- [10] M. Duvall, S. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.
- [11] C. Gerber and H.-M. Hanisch. Does portability of IEC 61499 mean that once programmed control software runs everywhere? In *10th IFAC Workshop on Intelligent Manufacturing Systems*, pages 29–34, Lisbon, July 2010.
- [12] R. Hametner. *Test-Driven Software Development for Improving the Quality of Control Software for Industrial Automation Systems*. PhD thesis, Vienna University of Technology, Oct. 2013.
- [13] R. Hametner, I. Hegny, and A. Zoitl. A unit-test framework for eventdriven control components modeled in IEC 61499. In *IEEE Conference on Emerging Technology and Factory Automation*, pages 1–8, Barcelona, Sept 2014.
- [14] R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler, and A. Zoitl. Test case generation approach for industrial automation systems. In *5th International Conference on Automation, Robotics and Applications*, pages 57–62, Wellington, New Zealand, Dec. 2011.
- [15] R. Hametner, D. Winkler, T. Östreicher, N. Surnic, and S. Biffl. Selecting UML models for test-driven development along the automation systems engineering process. In *IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–4, Bilbao, Sept. 2010.
- [16] R. Hametner, A. Zoitl, and M. Semo. Component architecture for the efficient development of industrial automation system. In *6th IEEE Conference on Automation Science and Engineering Proceedings*, pages 156–161, Toronto, Aug. 2010.

- [17] H. M. Hanisch, A. Lobov, J. L. Martinez Lastra, R. Tuokko, and V. Vyatkin. Formal validation of intelligent automated production systems towards industrial applications. *Intl. J. of Manufacturing Technology and Management*, Vol. 8, No. 1, 75–106, 2006.
- [18] T. Hussain and G. Frey. UML-based development process for IEC 61499 with automatic test-case generation. In *IEEE Conference on Emerging Technologies and Factory Automation*, pages 1277–1284, Prague, Sept. 2006.
- [19] IEC 61499-1. *Function blocks. Part 1: Architecture*. International Electrical Commission, 2.0 edition, 2012.
- [20] L. Madeyski. *Test-Driven Development: An Empirical Evaluation of Agile Practice*. Springer, 2010.
- [21] A. Mattsson, B. Lundell, B. Lings, and B. Fitzgerald. Linking modeldriven development and software architecture: A case study. *IEEE Trans. on Software Engineering*, Vol. 35, No. 1, 83–93, Jan./Feb. 2009.
- [22] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren. Reconfigurable manufacturing systems: Key to future manufacturing. *J. of Intelligent Manufacturing*, Vol. 11, No. 4, 403–419, Aug. 2000.
- [23] S. Mouchawrab, L. Briand, Y. Labiche, and M. Di Penta. Assessing, comparing, and combining state machine-based testing and structural testing: A series of experiments. *IEEE Trans. on Software Engineering*, Vol. 37, No. 2, 161–187, Mar./Apr. 2011.
- [24] S. Preusse and H.-M. Hanisch. Specification and verification of technical plant behavior with symbolic timing diagrams. In *3rd International Design and Test Workshop*, pages 313–318, Monastir, Dec. 2008.
- [25] S. Preusse and H.-M. Hanisch. Verifying functional and non-functional properties of manufacturing control systems. In *3rd International Workshop on Dependable Control of Discrete Systems*, pages 41–46, Saarbruecken, June 2011.
- [26] C. Seidner and O. Roux. Formal methods for systems engineering behavior models. *IEEE Trans. on Industrial Informatics*, Vol. 4, No. 4, 280–291, 2008.
- [27] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep, and H. Erdogmus. What do we know about test-driven development? *IEEE Software*, Vol. 27, No. 6, 16–19, Nov./Dec. 2010.
- [28] C. Sünder and V. Vyatkin. Functional and temporal formal modeling of embedded controllers for intelligent mechatronic systems. *Intl. J. of Mechatronics and Manufacturing Systems*, Vol. 2, No. 1/2, 215–235, 2009.
- [29] C. Sünder, A. Zoitl, J. Christensen, H. Steininger, and J. Fritsche. Considering IEC 61131-3 and IEC 61499 in the context of component frameworks. In *6th IEEE International Conference on Industrial Informatics*, pages 277–282, Daejeon, July 2008.
- [30] S. K. Swain, D. P. Mohapatra, and R. Mallc. Test case generation based on state and activity models. *J. of Object Technology*, Vol. 9, no. 5, 1–27, 2010.
- [31] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press, 2nd ed., 2002.

- [32] K. Thramboulidis. Using UML in control and automation: A model driven approach. In *2nd IEEE International Conference on Industrial Informatics*, pages 587–593, June 2004.
- [33] K. Thramboulidis. Model-integrated mechatronics: Toward a new paradigm in the development of manufacturing systems. *IEEE Trans. on Industrial Informatics*, Vol. 1, No. 1, 54–61, Feb. 2005. [34] M. Utting and B. Legeard. *Practical Model-based Testing*. Morgan Kaufmann, 2007.
- [35] S. Vegas, N. Juristo, and V. Basili. Maturing software engineering knowledge through classifications: A case study on unit testing techniques. *IEEE Trans. on Software Engineering*, Vol. 35, No. 4, 551–565, July 2009.
- [36] V. Vyatkin and V. Dubinin. Refactoring of execution control charts in basic function blocks of the IEC 61499 standard. *IEEE Trans. on Industrial Informatics*, Vol. 6, No. 2, 155–165, May 2010.
- [37] A. Zoitl, T. Strasser, and A. Valentini. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study. In *IEEE International Symposium on Industrial Electronics*, pages 3817–3819, Bari, July 2010.

Проверка IEC 61499 приложений

Petr Kadera

Чешский технический университет в Праге

Pavel Vrba

Чешский технический университет в Праге

СОДЕРЖАНИЕ

8.1 Введение	175
8.2 Общая проверка программного обеспечения	177
8.2.1 Динамическая проверка	177
8.2.2 Статическая проверка	178
8.2.2.1 Проверка модели	178
8.2.2.2 Доказательство теорем	179
8.3 Проверка IEC 61131-3	180
8.4 Динамическая проверка IEC 61499	181
8.4.1 Контроль и отладка	181
8.4.2 Блочные тесты	183
8.4.3 Тестирование по ключевым словам	184
8.5 Статическая проверка IEC 61499	185
8.6 Выводы	187

8.1. Введение

В предыдущих главах описаны основные принципы использования стандарта IEC 61499 для разработки распределенных систем управления. Хотя задача создания работающего приложения IEC 61499, конечно, не тривиальна, она относительно проста. Тем не менее создание приложения, которое работает **всегда**, намного сложнее. Как мы можем быть уверены, что разработанное приложение во всех случаях будет соответствовать нашим требованиям? Как мы можем на самом деле описать эти требования? Еще один важный аспект связан с возможностью повторного использования фрагментов IEC 61499. Как мы можем оценить, насколько безопасно повторно использовать ранее разработанный код?

Хензингер и Сифакис [15] следующим образом определили основные проблемы проектирования встроенных систем.

1. Включение разнородных механизмов исполнения и взаимодействия для компонентов системы.

2. Разделение абстракций, которые изолируют подзадачи проектирования, требующие творческого подхода, от тех абстракций, которые можно автоматизировать.

3. Масштабирование путем поддержки композиционных, правильных методов построения.

4. Обеспечение надежности получаемых систем.

Что такое верификация системы? По большому счету в процессе проверки системы нужно выявить, отвечает ли система качественным требованиям, которые были определены. Часто **верификация** и **валидация** являются ошибочными терминами, хотя их значения различны. Определения, представленные в [9], гласят:

Валидация – это процесс оценки программного обеспечения во время или в конце процедуры, направленный на то, чтобы определить, удовлетворяет ли ПО определенным бизнес-требованиям.

Верификация – это процесс оценки рабочих продуктов (не фактического конечного продукта) фазы разработки, направленный на то, чтобы определить, соответствуют ли они определенным требованиям для этой фазы.

Другими словами, цель валидации состоит в том, чтобы продемонстрировать, что продукт выполняет свое предназначение, когда он помещен в необходимую среду, то есть подтвердить, что мы создаем правильный продукт. Цель же верификации состоит в том, чтобы убедиться, что продукты соответствуют указанным требованиям, то есть подтвердить, что мы создаем продукт правильно. В этой главе рассматриваются проблемы верификации.

Необходимость верификации может быть проиллюстрирована на примере четырех известных технологических инцидентов, которых можно было бы избежать, если бы применялись надлежащие верификационные методы.

Радиационная передозировка Therac-25. Радиационное устройство для лечения пациентов, страдающих раком, неправильно измеряло произведенное излучение, и в период с 1985 по 1987 год наблюдались по крайней мере шесть значительных передозировок, что привело к смерти трех пострадавших пациентов. Несчастные случаи произошли, когда мощный пучок электронов был активирован вместо необходимого пучка малой мощности, поскольку пластина распределителя пучка находилась в неправильном положении. Предыдущие модели имели аппаратные блокировки, чтобы предотвратить такое, но в Therac-25 они отсутствовали. Дефект возник из-за переполнения одноканального счетчика во время процедуры тестирования; если оператор будет осуществлять ручной ввод в машину в тот момент, когда счетчик переполнится, блокировка не будет выполнена.

Отказ телефонной сети AT & T. В январе 1990 года 9-часовой выход из строя затронул большую часть телефонной сети США. Предполагаемые убытки, вызванные этим инцидентом, превысили 100 миллионов долларов.

Неожиданной причиной простоя стала неверная интерпретация оператора break в языке программирования C.

Ошибка Pentium FDIV. В 1994 году Intel представила новую версию чипов Pentium. К сожалению, возникли проблемы в блоке с плавающей запятой. При определенных обстоятельствах устройство давало неверные результаты. Общие убытки, связанные с этим вопросом, достигли 500 миллионов долларов.

Бедствие на Ариане 5. Европейское космическое агентство (ESA) в 1996 году столкнулось с аварией, ставшей широко известной. Точная причина крушения так и не была установлена, но наиболее вероятной причиной сбоя системы управления считается неправильное преобразование типа данных из 64-битного с плавающей запятой в 16-разрядное целое число.

Тестирование и отладка IEC 61499 признаны по своей сути сложными из-за характера приложений, являющихся естественно распределенными и параллельными системами, которые могут вести себя недетерминированным образом из-за автономного планирования действий управления отдельными компонентами распределенной системы. Поэтому тестирование является сложным, но необходимым. В этой главе мы рассмотрим эти важные вопросы. Мы предоставим обзор методов и подходов, которые уже были разработаны для общей проверки программного обеспечения. Затем мы кратко упомянем некоторые методы, разработанные для проверки контрольного кода IEC 61131, и, наконец, опишем текущие достижения проверки IEC 61499.

8.2. Общая проверка программного обеспечения

Альтернативой обширным испытаниям недавно разработанных программных и аппаратных решений является использование методов верификации. С ростом сложности компьютерных систем внимание постоянно смещается от конструирования к верификации. Проверка модели программного обеспечения является мощным подходом для формальной верификации программного обеспечения.

8.2.1. Динамическая проверка

Динамическая проверка определяет правильность создания программного обеспечения. В зависимости от объема тесты делятся на следующие категории:

- **блочные тесты:** тесты наименьших фрагментов программы;
- **модульные тесты:** тесты взаимосвязанных блоков.

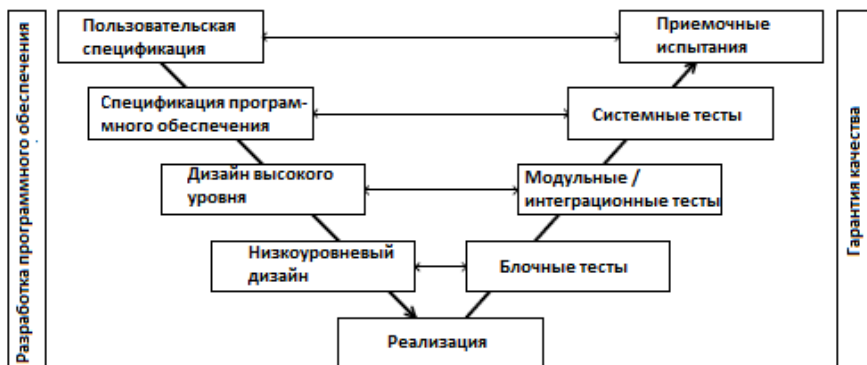


Рисунок 8.1. V-модель процесса разработки программного обеспечения

Интеграционные тесты: тесты взаимосвязанных модулей.

Системные тесты: тесты всей системы.

Тесты могут быть назначены для анализа следующего: функциональность, надежность, производительность, работоспособность, безопасность, совместимость, ремонтпригодность и выносливость. Уровни тестирования и их связь с обеспечением качества конечного программного продукта хорошо проиллюстрированы V-моделью (см. рисунок 8.1). В настоящее время это доминирующий метод проверки программного обеспечения [1] и [8].

8.2.2. Статическая проверка

Формальные методы представляют собой методы верификации, которые не требуют выполнения кода. Исходный код преобразуется в абстрактную форму, которая затем проверяется формальным способом. Двумя основными представителями статической проверки являются проверка модели и доказательство теорем. Подробное сравнение этих двух подходов можно найти в [20].

8.2.2.1. Проверка модели

Авторы [2] определяют проверку модели следующим образом: «Проверка модели – это автоматизированный метод, который, учитывая модель системы с конечным состоянием и формальное свойство, систематически проверяет, выполняется ли это свойство для данного состояния в этой модели». Проверка программного обеспечения на основе моделей была введена в начале 1980-х годов Кларком и соавторами [5] и развилась в сложную методологическую базу. Достижения группы Кларка обобщены в [6].

Были разработаны различные алгоритмы проверки моделей, которые имеют общее свойство: прежде чем проводить проверку, проверяемое программное обеспечение должно быть преобразовано в форму автоматов конечного состояния.

Далее указаны основные проблемы, связанные с проверкой модели.

Построение модели. Первая проблема – это построение модели. Существует значительный семантический разрыв между программой, выраженной в терминах языка программирования (методы, наследование, исключения и т. д.), и моделью такой программы, которая описывается как набор состояний, связанных переходами.

Взрыв состояний: если модель содержит слишком много состояний, это может быть непрактично.

Спецификация свойства: формализация функционального требования с помощью временной логики может быть неуклюжей.

Интерпретация выходных данных: трассировка ошибок может быть очень длинной и трудной для сопоставления с моделью и, следовательно, с исходным кодом. Эта проблема связана с тем же семантическим разрывом, который затрудняет построение модели.

8.2.2.2. Доказательство теорем

Использование доказательств теорем для верификации программного обеспечения было внедрено в 1969 году Хоаром, который построил свой подход на следующем факте: «Компьютерное программирование – это точная наука, в которой все свойства программы и все последствия ее выполнения в любой конкретной среде могут быть выявлены из текста самой программы с помощью чисто дедуктивных рассуждений» [16]. Дедуктивное мышление эквивалентно доказательству теорем и означает, что действительные правила применяются к наборам аксиом. Хоар предложил исчисление, которое описывает поведение программы в виде троек в следующей форме:

$$P \{Q\} R, \quad (8.1),$$

что можно перевести как: «Если предварительное условие P является истинным до начала выполнения программы Q , то условие R будет истинным после завершения». Эта концепция получила дальнейшее развитие у Дейкстры в виде понятия «преобразователи предикатов», Учитывая программу Q и постусловие R , идентифицируется набор предварительных условий, которые должны соблюдаться для удовлетворения постусловия [7]. Основные проблемы подхода доказательства теорем, подробно описанного в [17], заключаются в следующем.

Изобретение лемм и новых концепций. Возможно, самой большой проблемой является первый шаг: построение заданной формулы из аксиом по правилам вывода.

Примеры и контрпримеры. Сложно преобразовать набор примеров в логическое доказательство, а также трудно автоматически создать контрпример из несостоятельного доказательства.

Аналогия, обучение и извлечение данных. Пользователи часто обвиняют доказателей теорем в повторении старых доказательств с новыми символами вместо нахождения обобщенной леммы. К сожалению, обобщение – это сложная проблема, требующая осторожного подхода, и поэтому изучение и извлечение данных для доказательства теорем не легки.

Открытая архитектура. Сложно спроектировать осуществляющий проверку элемент, который вел бы себя автономно (то есть самостоятельно пробовал бы различные стратегии проверок) и мог бы быть легко перенастроен, чтобы исключить некоторые приемы проверок.

Параллельные и совместные средства доказательства теорем. Распараллеливание является крайне востребованной функцией для средств проверки. Это важно для (1) разложения большой задачи на более мелкие подзадачи и (2) интеграции уже доказанных теорем в новые решения.

Пользовательский интерфейс и интерактивное управление. Поиск подходящей формы представления результатов проверки своим пользователям является еще одной проблемой, поскольку форма должна представлять все важные факты и не отвлекать пользователей множеством бессмысленных деталей.

Образование. Количество людей, способных использовать автоматические рассуждения, ограничено. Обычные курсы бакалавриата по исчислению предикатов и логике не обеспечивают достаточного понимания всех аспектов доказательства теорем.

Проверенный доказатель теоремы. Трудно гарантировать, что проверяющий всегда обеспечивает правильные результаты.

8.3. Проверка IEC 61131-3

Лестничная логика (LL) является одним из наиболее часто используемых обозначений для программирования PLC, определенных в IEC 61131-3. Это графическое обозначение, основные элементы которого основаны на аналогии с физическими релейными диаграммами. Это сделало LL доступным для инженеров, которые не знакомы с компьютерным программированием. Однако программы LL сложно отлаживать и проверять, потому что их графическое представление логики переключения скрывает последовательную, зависящую от состояния логику, присущую разработке программы [10]. Метод проверки программ, разработанных в LL, предложен Бендером и соавторами [3]. Этот метод представляет собой модельно-ориентированный подход для формальной проверки программ LL посредством проверки моделей. Модель разработана в соответствии с метамоделью LL, способной моделировать подмножество языка LL, которое преобразуется во временные сети Петри, которые затем формально проверяются. Авторы сосредоточились на проверке общих, то есть

независимых от модели свойств, учитывая отсутствие условий гонки в программах LL. Однако применимость этого подхода ограничена, потому что моделируется только контроллер, а завод-изготовитель и его модель не рассматриваются. Как заявляют авторы, учет установки в конечном итоге может привести к взрыву в пространстве состояний, что делает анализ невозможным.

Проверка программ, написанных в Список инструкций (IL), который является другим текстовым стандартом языка программирования PLC, определенным IEC 61131-3, рассматривается в [13]. Чтобы иметь возможность применять основанную на сети Петри систему верификации в программах IL, авторы представили семантику сети Петри для обозначения IL. Ни предыдущая проверка LL, ни эта работа не моделируют поведение установки. Поэтому авторы расширили свою работу и предложили расширенную версию своего метода в [14].

Результаты этих экспериментов показали, что верификация на основе моделей применима для малых и средних приложений для управления. Большие приложения были преобразованы в гигантские модели с огромным количеством состояний и переходов между состояниями, которые были трудны для вычисления.

8.4. Динамическая проверка IEC 61499

В этом разделе представлен обзор подходов, разработанных для тестирования и верификации правильности поведения работающего приложения IEC 61499. Во-первых, вводится система мониторинга и отладки. Во-вторых, описано применение модульного теста для IEC 61499.

8.4.1. Контроль и отладка

Мониторинг и отладка являются необходимыми функциями любой системы промышленной автоматизации, поскольку они позволяют оперативно реагировать на непредвиденные ситуации и сбои и определять корни обнаруженных проблем. Согласно [26], ключевым аспектом проектирования надлежащей системы мониторинга является выбор звука наблюдаемых информационных входов, потому что, если собрано слишком много информации, возникают проблемы с производительностью. Напротив, если собирать слишком мало информации, поведение системы не может быть полностью реконструировано.

Инфраструктура мониторинга и отладки для распределенных систем управления в соответствии с IEC 61499, которая предоставляется как часть реализации 4DIAC с открытым исходным кодом [24], представлена в [34]. Ее важной характеристикой является то, что она обеспечивает понимание программы для управления, выполняемой на уровне приложений, то есть

централизованного представления системы управления, которая может быть распределена по нескольким вычислительным ресурсам. Это требует получения данных от различных ресурсов и их синхронизацию для предотвращения несоответствия данных. Авторы решают вопросы, связанные с распределенным мониторингом, путем внедрения менеджера центрального мониторинга (СММ). СММ является единственной точкой подключения для инструмента мониторинга и отладки действий. Таким образом, количество каналов связи привязано к одному соединению через устройство. Кроме того, диспетчер центрального мониторинга устройства может быть создан в независимом контексте выполнения (то есть в процессе или задаче), уменьшая помехи для частей управляющего приложения в устройстве. Архитектура этого решения показана на рисунке 8.2.

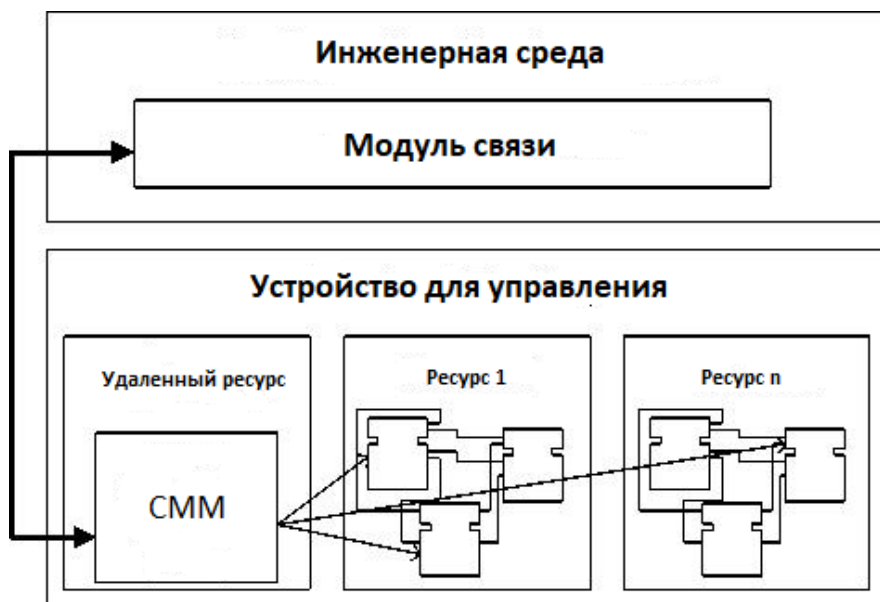


Рисунок 8.2. Диспетчер мониторинга как единая точка связи

Система мониторинга состоит из нескольких расширений стандартного интерфейса управления ИЕС 61499.

Добавить наблюдение: подписка на прослушивание изменений события или порта данных.

Удалить наблюдение: обратная операция.

Чтение наблюдений: обновление всех отслеживаемых портов.

Принудительная запись: пишем команду, которая гарантирует, что записанное значение порта данных не будет перезаписано.

Запись на порт события: запускается выполнение события.

Список расширений показывает, что функциональность этого решения основана только на повышении осведомленности о текущем событии и значениях порта данных. Контрольные точки, которые являются общими частями отладчиков для процедурных языков, таких как Java или C ++, не поддерживаются.

8.4.2. Блочные тесты

Поскольку тестирование законченных сложных приложений является очень сложным, был предложен подход, основанный на ранних тестах отдельных программных компонентов. Проще говоря, процесс начинается с модульного тестирования, где основные блоки, то есть отдельные функциональные блоки, тестируются изолированно, чтобы гарантировать, что каждый из них ведет себя правильно.

Применение метода первой тестовой разработки (TFD) повышает качество программных блоков кода, позволяя как можно скорее обнаружить возможные проблемы. Этот подход был широко принят инженерами-программистами, и теперь он находит свое применение в области управления. Его основной принцип заключается в том, что процесс разработки начинается с определения запрашиваемого поведения конкретного фрагмента приложения. Затем следует разработка, и код постоянно тестируется на соответствие указанным требованиям. Каждый этап проверки заканчивается, когда все требования выполнены. Применение этого подхода для разработки IEC 61499 было представлено Хаметнером и соавторами, которые разработали структуру модульного тестирования для управляемых событиями компонентов, смоделированных в IEC 61499 [11]. Занимаясь данными разработками, авторы должны были ответить на следующие вопросы.

1. Что такое тестируемые программные блоки в IEC 61499?
2. Каков подходящий способ задания тестовых примеров и модульных тестов в IEC 61499, позволяющий генерировать входные данные для системы выполнения модульных тестов?
3. Как мы можем выполнить тесты и автоматизировать их выполнение независимо от целевого устройства управления?

Ответ на первый вопрос: тестируемая программная единица – это общая концепция, выходящая за пределы стандарта IEC 61499. Шиперски определяет программный компонент как единицу независимого развертывания, стороннего состава и наблюдаемого состояния [25]. Эти характеристики хорошо согласуются с концепцией функциональных блоков, и, таким образом, функциональный блок является тестируемым программным блоком.

Ответ на второй вопрос: авторы разработали метод автоматического генерирования последовательностей событий для тестирования функциональных блоков со многими входами. Последовательности воздействий и механизмы вызова были введены для определения нескольких

тестовых случаев (например, подготовки FB) и многократного их использования.

Ответ на третий вопрос: авторы разработали автономную систему модульных испытаний, которая подходит для промышленной автоматизации в качестве основы для подхода TFD. Кроме того, блочные тесты и отчеты о результатах могут быть созданы автоматически.

Подробное описание инфраструктуры для блочных испытаний IEC 61499 можно найти в главе 7.

8.4.3. Тестирование по ключевым словам

Альтернативой общеизвестным методам верификации является тестирование по ключевым словам (KDT) [35]. Использовать его для тестирования приложений IEC 61499 было предложено [12]. Принцип KDT состоит в том, чтобы описывать каждый тестовый случай как можно более абстрактно, делая его достаточно точным, чтобы интерпретировать и выполнять тестовые примеры с помощью инструмента тестирования [27].

KDT четко разделяет разработку программного обеспечения и определение контрольного примера. Это позволяет нам определять тестовые случаи на ранних этапах разработки, как ожидается в подходе TFD. Термин «ключевые слова» относится к словам, которые управляют обработкой конкретного приложения. Таким образом, наборы ключевых слов, предназначенных для тестирования приложений для настольного ПК и приложений промышленного управления, будут разными.

Тестовая структура, представленная в [12], содержит следующие компоненты.

Точность тестового набора зависит от отдельных тестовых сценариев, тестовых наборов и базового набора ключевых слов в соответствии со стандартами реализации.

При **управлении тестированием** координируется и контролируется выполнение тестового набора. Устройство находится между системой управления тестированием и тестируемой системой.

Анализ проводится при отдельных прогонах теста, генерируется отчет о результатах теста.

Компонент Fixture обеспечивает гибкость, которая позволяет нам сравнивать поведение двух систем, разработанных по-разному. Авторы [12] иллюстрируют эту возможность при параллельном тестировании двух систем, разработанных в соответствии со стандартами IEC 61134-3 и IEC 61499.

8.5. Статическая проверка IEC 61499

Если функциональные блоки ведут себя должным образом, когда они разделены, можно выполнить интеграционное тестирование или тестирование модуля, при котором блоки внутри модуля тестируются вместе.

Шнакенбург [22] попытался смоделировать поведение системы управления IEC 61499, чтобы доказать безопасность контроллера, и предложил формальный метод, основанный на переводе кода функционального блока на синхронный язык SIGNAL [19]. Метод требует выражения ожидаемого поведения управляющего кода в терминах языка SIGNAL, чтобы использовать методы автоматической проверки для подтверждения того, что система управления соответствует указанным свойствам.

Также предложена методология для моделирования поведения функционального блока IEC 61499 с использованием синхронизированных автоматов [23]. Данная работа предоставляет метод, который способен связывать базовые синхронизированные автоматы с каждым функциональным блоком и составлять их для проверки выполнения сети функциональных блоков.

Структура верификации системы управления была разработана Вяткиным и Пангом [32]. Предлагаемая методология разработки основана на проектировании компонентов автоматизированных производственных систем из интеллектуальных мехатронных компонентов. Авторы называют свой подход замкнутым моделированием, поскольку оно учитывает управляемую систему. Основная цель этой структуры состоит в интеграции разнородных нотаций и инструментов моделирования (например, Unified Modeling Languages или Simulink), а его ключевой особенностью является неотъемлемая поддержка формальной проверки с использованием автоматического преобразования между различными моделями системы.

Альтернативный подход к верификации, предложенный в [33], основан на переводе контрольного кода IEC 61499 на язык Esterel [4]. Основной строительный блок программы Esterel – это модуль, который определил свои входные и выходные сигналы. Каждый сигнал может состоять из статуса и/или значения. Программы Esterel выполняются с дискретными временными шагами, называемыми тактами; выполнение внутри такта происходит мгновенно. Соотношение между функциональными блоками и концепциями Esterel обобщено в таблице 8.1.

Вяткин и Ханиш разработали метод моделирования поведения функциональных блоков и связанных объектов (замкнутых контуров) с использованием языка моделирования Net Condition / Event Systems (NCES) [31]. NCES повторно использует основные концепции сетей Петри и расширяет их для сбора событий и потока данных. Таким образом, сеть простых и составных функциональных блоков может быть сопоставлена со структурой сетевых модулей условий / событий (NCEM). Этот подход к моделированию не учитывает задержек, вызванные инфраструктурой связи, соединяющей отдельные функциональные блоки. Это является важным

ограничивающим фактором, который уменьшает потенциальное применение этого метода для реального промышленного использования, поскольку задержки не могут быть устранены из модели.

Лапп и соавторы [18] предложили решение для преодоления этой проблемы. Данное решение основано на введении большего количества временных свойств, касающихся времени выполнения, в формальную модель, чтобы учитывать их при проверке. Ключевым улучшением является введение правил преобразования, которые преобразуют в NCEM функциональные блоки, а также функциональные блоки служебного интерфейса (SIFB), которые не вносят непосредственного вклада в управление, но обеспечивают связь между несколькими вычислительными ресурсами. Поведение SIFB определяется сервисными примитивами в соответствии с ISO-IEC-10731. Сервисные примитивы интуитивно описывают зависимости сигналов связи, и, таким образом, они также отображаются в соответствующие NCEM. Созданные модели NCES должны быть уточнены, то есть дополнены количественной информацией, такой как время выполнения алгоритмов в функциональных блоках или время отклика интерфейсов связи. Следовательно, соответствующие части моделей должны быть количественно расширены, например, с помощью мощностей, взвешенных дуг, начальных отметок или временных интервалов. Значения для этих параметров должны быть определены из технических характеристик оборудования и обоснованных оценок, касающихся ограничений безопасности.

Таблица 8.1. Отображение между функциональным блоком и Esterel Concepts

Элемент функционального блока	Esterel
Функциональный блок	Модель
События	Чистые сигналы интерфейса
Данные	Только ценные сигналы интерфейса
Внутренние переменные	Только ценные локальные сигналы
Алгоритмы	Мгновенные модули
Сеть функциональных блоков	Параллельная композиция модулей

Комплексная работа по формальному анализу IEC 61499 была начата Cengic и Åkesson и описана в двух статьях, посвященных моделированию [29] и семантике исполнения [30]. Предшествующий вклад состоял в обеспечении формальной структуры IEC 61499 для математического моделирования и сравнения различных семантик выполнения, а теперешний иллюстрирует, как три семантики выполнения (буферизованная модель последовательного выполнения, невыгруженная многозадачность и циклическая буферизованная модель выполнения) могут быть математически формализованы. Эти модели можно использовать для анализа и сравнения поведения приложения при работе с использованием другой семантики.

Введенный формализм может быть интегрирован в модель контролируемой установки для получения модели с обратной связью. В таком случае модель установки предоставляет входы для контроллера и получает управляющие сигналы.

Кроме того, формализмы и математические модели, представленные в этой работе, были использованы в качестве основы для реализации среды выполнения Фубера [28] и программного инструмента Moger в составе Supremica [21], которая генерирует модели, подходящие для формальной проверки.

8.6. Выводы

Широкий спектр обозначений, методов и инструментов, упомянутых в этой главе, иллюстрирует сложность области верификации программного обеспечения. Некоторому ПО требуются пользователи со специальными навыками, что ограничивает его применение в производственной практике. Это не относится к системам мониторинга и отладки, которые в настоящее время интегрированы в среду разработки для IEC61499 и поэтому могут использоваться напрямую. Модульные тесты представляют собой многообещающие подходы для проверки фрагментов управляющего кода, но их природа не позволяет использовать их для моделирования и проверки сложного поведения, возникающего при взаимодействии нескольких функциональных блоков. Методы же статической проверки способны сделать это, но требуют создания базовой модели.

Принимая во внимание, что было предложено несколько способов преобразования схемы функциональных блоков в модель, автоматическое моделирование аппаратного обеспечения и плана управления остается проблемой. В настоящее время эти модели должны быть под контролем инженеров по системам управления, так как их использование может быть сложной и трудоемкой задачей, требующей специальных навыков.

Библиография

- [1] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.
- [2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
- [3] D. Bender, B. Combemale, X. Cregut, J. Farines, B. Berthomieu, and F. Vernadat. Ladder metamodeling and PLC program validation through time Petri nets. In *Model Driven Architecture: Foundations and Applications*, pages 121–136. Springer, 2008.

- [4] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Sci. Comput. Program.*, Vol. 19, No. 2, 87–152, 1992.
- [5] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite state concurrent system using temporal logic specifications: A practical approach. In *Proceedings of 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 117–126, New York, 1983.
- [6] E. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
- [7] E. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, Vol. 18, No. 8, 453–457, 1975.
- [8] A. Engel. *Verification, Validation, and Testing of Engineered Systems*. John Wiley & Sons, 2010.
- [9] M. Fisher. *Software Verification and Validation: An Engineering and Scientific Approach*. Springer, 2007.
- [10] A. Guasch, J. Quevedo, and R. Milne. Fault diagnosis for gas turbines based on the control system. *Engineering Applications of Artificial Intelligence*, Vol. 13, No. 4, 477–484, 2000.
- [11] R. Hametner, I. Hegny, and A. Zoitl. A unit-test framework for eventdriven control components modeled in IEC 61499. In *Proceedings of IEEE Conference on Emerging Technology and Factory Automation*, pages 1–8. IEEE, 2014.
- [12] R. Hametner, D. Winkler, and A. Zoitl. Agile testing concepts based on keyword-driven testing for industrial automation systems. In *Proceedings of 38th Annual Conference of IEEE Industrial Electronics Society*, pages 3727–3732, 2012.
- [13] M. Heiner and T. Menzel. Instruction list verification using Petri net semantics. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics.*, volume 1, pages 716–721, 1998.
- [14] M. Heiner and T. Menzel. Time-related modelling of PLC systems with timeless petri nets. In R. Boel and G. Stremersch, editors, *Discrete Event Systems*, pages 275–282. Springer, 2000.
- [15] T. Henzinger and J. Sifakis. The discipline of embedded systems design. *Computer*, Vol. 40, No. 10, 32–40, 2007.
- [16] C. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, Vol. 12, No. 10, 576–580, Oct. 1969.
- [17] M. Kaufman and S. Moore. Some key research problems in automated theorem proving for hardware and software verification. In *Revista de la Real Academia de Ciencias Exactas, Físicas y Naturales. Serie A: Matemáticas*, Vol. 98, No. 1, 181, 2004.
- [18] H. Lapp, C. Gerber, and H. M. Hanisch. Improving verification and reliability of distributed control systems design according to IEC 61499. In *Proceedings of IEEE Conference on Emerging Technologies and Factory Automation*, 2010.
- [19] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with signal. *Proceedings of IEEE*, Vol. 79, No. 9, 1321–1336, 1991.

- [20] M. Ouimet and K. Lundqvist. Formal software verification: Model checking and theorem proving. Technical report, Massachusetts Institute of Technology, 2008.
- [21] K. Åkesson, M. Fabian, H. Flordal, and R. Malik. Supremica: An integrated environment for verification, synthesis and simulation of discrete event systems. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 384–385, 2006.
- [22] C. Schnakenbourg, J.-M. Faure, and J.-J. Lesage. Towards IEC 61499 function blocks diagrams verification. In *IEEE International Conference on Systems Man and Cybernetics*, pages 1–6, 2002.
- [23] M. Stanica and H. Guéguen. Using timed automata for the verification of IEC 61499 applications. In *Proceedings of 7th IFAC Workshop on Discrete Event Systems*, pages 375–380. Elsevier, 2005.
- [24] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, and A. Martel. Framework for distributed industrial automation and control (4DIAC). In *Proceedings of 6th IEEE International Conference on Industrial Informatics*, pages 283–288, 2008.
- [25] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.
- [26] H. Thane. *Monitoring, Testing and Debugging of Distributed Real-Time Systems*. PhD thesis, Royal Institute of Technology, Stockholm, 2000.
- [27] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2010.
- [28] G. Cengic and K. Åkesson. Definition of the execution model used in the Fuber IEC 61499 runtime environment. In *Proceedings of 6th IEEE International Conference on Industrial Informatics*, pages 301–306, 2008.
- [29] G. Cengic and K. Åkesson. On formal analysis of IEC 61499 applications. Part A: Modeling. *IEEE Transactions on Industrial Informatics*, Vol. 6, No. 2, 136–144, 2010.
- [30] G. Cengic and K. Åkesson. On formal analysis of IEC 61499 applications. Part B: Execution Semantics. *IEEE Transactions on Industrial Informatics*, Vol. 6, No. 2, 145–154, 2010.
- [31] V. Vyatkin and H. M. Hanisch. A modeling approach for verification of IEC1499 function blocks using net condition/event systems. In *Proceedings of 7th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 261–270, 1999.
- [32] V. Vyatkin, H. M. Hanisch, C. Pang, and C.-H. Yang. Closed-loop modeling in future automation system engineering and validation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 39, No. 1, 17–28, 2009.
- [33] L. Yoong and P. Roop. Verifying IEC 61499 function blocks using Esterel. *IEEE Embedded Systems Letters*, Vol. 2, No. 1, 1–4, March 2010.
- [34] A. Zoitl, G. Ebenhofer, and M. Hofmann. Developing a monitoring infrastructure for IEC 61499 devices. In *Proceedings of 18th IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–6, 2013.

[35] A. Zylberman and A. Shotten. Test language: Introduction to keyword driven testing. *Methods & Tools*, Vol. 19, , 15–21, 2010.

Отказоустойчивые приложения IEC 61499

Mario de Sousa

Университет Порту

СОДЕРЖАНИЕ

9.1 Введение	191
9.2 Предпосылки	193
9.2.1 Системная надежность	193
9.2.2 Отказы, ошибки и сбои	195
9.2.3 Достижение надежности	195
9.3 Репликация в приложениях IEC 61499	197
9.3.1 Распространение сообщения	199
9.3.2 Семантика исполнения платформы	204
9.3.3 Функциональные блоки голосования	205
9.3.4 Функциональные блоки связи	207
9.4 Платформа репликации на FORTE	208
9.4.1 Семантика выполнения среды FORTE	208
9.4.2 Функциональные блоки связи	209
9.4.3 Функциональные блоки выбора	210
9.5 Пример реплицированного применения IEC 61499	211
9.5.1 Пример приложения	211
9.5.2 Добавление репликации	212
9.5.3 Обработка выбора	214
9.6 Количественная оценка надежности системы	214
9.7 Выводы	216

9.1. Введение

Любой, кто читает эту книгу с самого начала, наверняка знает, что IEC 61499 был создан с целью поддержки приложений распределенной автоматизации и управления. Нераспределенные приложения IEC 61499 возможны и могут даже иметь смысл во многих ситуациях. Тем не менее в этой главе основное внимание будет уделено приложениям, работа которых распределена по нескольким исполнительным устройствам, будь то PLC (программируемые логические контроллеры), встроенные платформы или

полноценные компьютеры. При этом нужна сеть передачи данных, связывающая эти исполнительные устройства и позволяющая различным подпрограммам, распределенным среди исполнительных устройств, обмениваться данными и событиями синхронизации.

При развертывании распределенного приложения разработчик должен учитывать возможность частичных сбоев. Под частичными сбоями подразумевается ситуация, когда одно из исполнительных устройств просто по какой-то неожиданной причине прекращает работу, в то время как остальные исполнительные устройства продолжают функционировать. Это сценарий, который никогда не происходит в централизованном приложении, работающем на одном исполнительном устройстве, где сбой исполнительного устройства приведет к полному завершению работы управляющего приложения. Поэтому распределенные приложения с ограничениями надежности должны учитывать эти новые режимы сбоев.

Обратите внимание, что этот режим отказа не является специфическим для приложений IEC 61499, но фактически является общим для всех распределенных приложений независимо от используемой базовой структуры. Даже распределенные приложения IEC 61131-3 подвержены этому явлению, но в этом случае семантика циклического выполнения IEC 61131-3 обеспечивает простое средство обнаружения частичного отказа. Типичное приложение распределенного управления, основанное на IEC 61131-3, состоит из множества отдельных приложений, каждое из которых работает на отдельном PLC, наряду с конкретной конфигурацией сети связи, используемой для обмена данными. Все локальные программы постоянно запускают цикл обработки событий. Аналогично сеть связи обычно конфигурируется для периодического обмена четко определенными данными между PLC. Периодическая природа обмена данными по сети позволяет любому PLC определить, что удаленный PLC больше не активен – будь то из-за сбоя самого PLC или из-за проблем с сетью, когда удаленный PLC не может установить связь во время одного из коммуникационных циклов. Упомянутый метод служит для обнаружения сбоев, а процедуры для обработки этих сбоев обычно либо выполняются непосредственно в рамках каждой из локально выполняемых программ, либо учитываются в конфигурации сети связи.

С другой стороны, управляющие приложения IEC 61499 имеют совершенно отличную семантику выполнения: активация FB основана на передаче событий, а передача данных и/или событий между исполнительными устройствами не обязательно происходит с периодичностью. В этом сценарии обнаружение сбоя удаленного исполнительного устройства не может быть основано на сбое в получении события и/или данных от этого устройства. Тем не менее мы видели, что приложения IEC 61499 также подвержены частичным сбоям, и если они должны использоваться в ситуациях с высокими требованиями к надежности, то тогда необходимо решение, позволяющее допустить эти частичные отказы.

В этой главе представлен образец конструкции для допуска частичных неисправностей в приложениях IEC 61499. Этот шаблон проектирования

представляет собой адаптацию традиционных подходов, уже с высокой степенью надежности применяемых в области приложений, а для понимания его свойств и простоты изложения в первом разделе этой главы представлен обзор часто используемых выражений из этой области. В следующих разделах представлен шаблон проектирования и показано, как он может быть реализован в существующей среде выполнения IEC 61499 и как могут быть определены некоторые свойства надежности конечного приложения IEC 61499.

9.2. Предпосылки

Во введении к этой главе использовались слова «надежность» и «отказ». Для сообщества, занимающегося системами, критически важными для безопасности, эти и многие другие термины (синтаксис) имеют очень четко определенные значения (семантику). Это значительно упрощает процесс обмена идеями между людьми, согласившимися на общий синтаксис и семантику, а также обеспечивает рамки, на которых строятся идеи и подходы для создания надежных приложений. В этом подразделе мы объясним некоторые из этих терминов и их значения, а также общие подходы к достижению надежных систем.

9.2.1. Системная надежность

Надежность системы была определена рабочей группой IFIP 10.4 по достоверным вычислениям и отказоустойчивости как «[...] надежность вычислительной системы, которая позволяет обоснованно полагаться на работу, которую она выполняет [...]». Другое подобное, но несколько более простое определение дали Avizienis и соавторы [2]: «способность предоставлять услуги, которым можно оправданно доверять».

Эти определения, однако, очень общие и могут для разных людей означать разное. Кроме того, они не очень помогают, если свойства не могут быть количественно измерены. Как только мы сможем оценить число надежных систем, можно будет провести значимые сравнения между конкурирующими решениями. Чтобы упростить эту задачу, надежность была дополнительно определена как система, которая в той или иной степени обладает некоторыми или всеми из следующих свойств:

- 1) доступность;
- 2) надежность;
- 3) безопасность;
- 4) пригодность для обслуживания (ремонтпригодность);
- 5) целостность;
- 6) конфиденциальность.

Доступность используется для измерения доли времени, в течение которого система доступна для выполнения своих функций. Это, очевидно, числовое значение, при этом значение 99 % считается типичным для многих приложений. Обратите внимание, что 99-процентная доступность, которая иногда расценивается как очень хорошая для обычных поставщиков услуг (например, поставщиков интернет-услуг и компаний по распределению электроэнергии), на самом деле не очень высокий показатель, так как он означает более 3 полных дней простоя в год.

Более формально доступность может быть определена как «вероятность того, что элемент выполнит требуемую задачу [...] в указанный момент времени» [3], и определяется как фиксированное значение вероятности.

Следующий атрибут – надежность – определяется как «вероятность того, что элемент выполнит требуемую от него функцию [...] в течение указанного промежутка времени» [3]. Фактически, это показатель того, как долго система может непрерывно работать без перебоев, но формально задается в качестве функции вероятности для каждого временного интервала. Например, если надежность жесткого диска составляет 50 % в течение 5 лет, это означает, что вероятность непрерывной работы без перерыва в течение 5-летнего периода составляет всего 50 %. Вероятность непрерывной работы уменьшится еще больше, если рассматривать более длинные интервалы. Если бы мы построили график вероятности непрерывной работы этого же диска в течение определенного периода t , мы бы получили функцию надежности $R(t)$.

Обратите внимание, что доступность и надежность характеризуют различные свойства. Встроенная система, которая автоматически перезагружается каждый час и требует 100 мс для каждой перезагрузки, будет иметь доступность более 99,99 %, но очень низкую надежность. С другой стороны, производственный станок, который может работать непрерывно в течение всего года, но всегда останавливается на техническое обслуживание на 1 неделю в течение следующего года, будет иметь гораздо более высокую надежность, но доступность не намного больше, чем 98 %.

Безопасность определяется как «отсутствие катастрофических последствий для пользователя (-ей) и окружающей среды» [2]. То, что считается катастрофическим, должно быть определено для каждой конкретной системы. Еще раз обратите внимание, что ни доступность, ни надежность не являются синонимами безопасности. Система с доступностью 0 % может быть совершенно безопасной (например, самолет, который никогда не покидает землю). Система с низкой надежностью также может быть безопасной, если она построена таким образом, что она всегда дает сбой, который никогда не приведет к катастрофическим последствиям (например, барьер для железнодорожного переезда, который всегда будет закрываться при возникновении любого отказа).

Ремонтопригодность определяет, насколько легко устранить любые сбои в системе.

Конфиденциальность – это отсутствие непреднамеренного раскрытия информации, а целостность – отсутствие непреднамеренных изменений в

системе. Безопасность часто определяется как свойство вычислительных систем и рассматривается как то, что одновременно обеспечивает доступность, конфиденциальность и целостность.

9.2.2. Отказы, ошибки и сбои

В контексте надежных систем *отказ*, *ошибка* и *сбой* имеют очень четкие значения.

Отказ происходит, когда система не выполняет свою задачу, то есть, когда она не выполняет желаемую функцию. Например, в нужное время PLC не может деактивировать привод.

Считается, что отказ вызван ошибкой. Ошибка обычно объясняется как неправильное внутреннее состояние системы, которое может привести к сбою. В предыдущем примере с PLC ошибка будет появляться на выходном реле PLC, которое застревает в закрытом положении. Обратите внимание, что эта ошибка может существовать, но никто не будет знать о ее наличии. Ошибка вызывает сбой PLC только тогда, когда этот конкретный выход необходимо деактивировать.

Сбой является причиной ошибки. Застывание контактов выходного реле могло быть вызвано чрезмерным током на этом выходе. Эта последовательность обобщается следующей последовательностью событий:

Сбой → ошибка → отказ.

Обратите внимание, что классификация конкретной ситуации, определенной как неисправность, ошибка или сбой, будет зависеть от границ рассматриваемой системы. Например, индикатор питания PLC может иметь сбой (перестать светиться), но это же событие считается просто ошибкой для PLC, если оно не влияет на выполнение программы PLC.

9.2.3. Достижение надежности

Существует два основных подхода к достижению надежности:

- 1) предотвращение сбоев;
- 2) отказоустойчивость.

Подход предотвращения сбоев состоит в том, чтобы построить систему таким образом, чтобы предотвратить возникновение сбоев. В программном проекте это будет означать следование строгим правилам разработки, которые уменьшат вероятность появления «багов» (программных ошибок) в программном обеспечении. В аппаратном проекте это может быть достигнуто путем использования деталей более высокого качества или путем разработки системы с более высоким запасом прочности.

Второй подход — отказоустойчивость — предполагает, что невозможно предсказать все возможные сценарии сбоев во время проектирования, и допускает, что сбои в конечном итоге произойдут в системе, когда она будет

использоваться. Поэтому система должна быть построена таким образом, чтобы допускались некоторые сбои и система продолжала обеспечивать правильное обслуживание даже при наличии неисправностей.

Именно такой подход используется в шаблоне проектирования, представленном в этой главе. Отказоустойчивость будет достигнута путем введения избыточности. Это означает, что некоторые конкретные компоненты системы будут продублированы (или даже утроены, увеличены в четыре раза и т. д.); таким образом, отказ одного подкомпонента не приводит к отказу всей системы. Говорят, что самолет с двумя двигателями, для которого требуется только один двигатель, способен выдержать отказ одного двигателя.

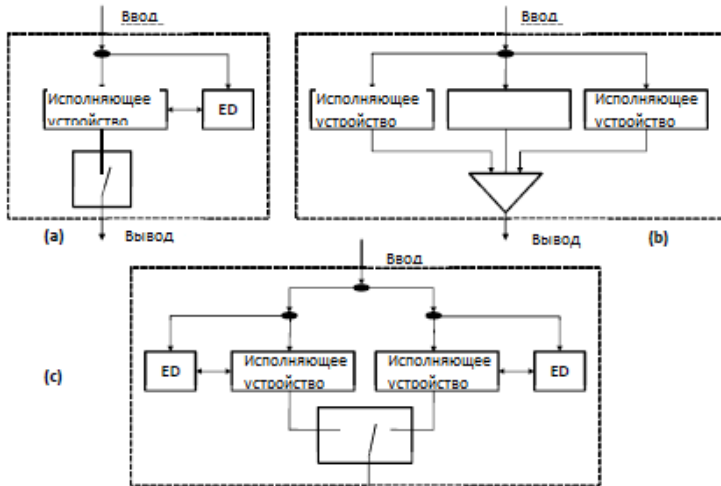


Рисунок 9.1. Отказоустойчивые архитектуры

Архитектуры для введения отказоустойчивости могут быть классифицированы с выделением трех основных подходов, которые лучше всего объяснить с помощью диаграммы (см. рисунок 9.1). Два подхода, основанные на использовании детекторов ошибок (ED), страдают недостатком, заключающимся в том, что не всегда возможно идентифицировать наличие ошибки, и любой переход с одного процессора на другой обычно требует дополнительной задержки, которая может быть неприемлемой в приложениях для управления, чувствительным к времени. По этой причине схема проектирования, предложенная в этой главе, следует архитектуре (c), в которой избиратель определяет консолидированный выход избыточных реплик.

9.3. Репликация в приложениях IEC 61499

Как упоминалось во введении, распределенные управляющие приложения IEC 61499 подвержены сбоям одного или нескольких исполнительных устройств, на которых работает распределенное приложение. В централизованных приложениях, работающих на одном PLC, традиционный подход к устранению сбоев заключается во введении еще одного (или более) PLC, который идентичен (как с точки зрения аппаратного, так и программного обеспечения) исходному PLC.

Продельвание того же самого для каждого исполнительного устройства, на котором запущено распределенное приложение, можно считать излишним, так как количество исполнительных устройств, подлежащих репликации, может быть довольно большим.



Рисунок 9.2. Распределение реплицированных FB между исполнительными устройствами

Возможное решение состоит в том, чтобы использовать распределенную природу самого IEC 61499: каждое исполнительное устройство может быть сконфигурировано для запуска копий подпрограмм, работающих на других исполнительных устройствах той же системы. Например, каждое исполнительное устройство может быть сопряжено с еще одним исполнительным устройством, и каждая пара исполнительных устройств может быть сконфигурирована для выполнения всех подпрограмм этой пары. Это может привести к ситуации, аналогичной реплицированным PLC, где каждая реплика запускает такой же набор программного обеспечения, что и ее пара.

Этот подход все еще довольно неудобен и неоправданно ограничивает свободу проектирования и разработки приложений. Лучший подход – это позволить разработчику приложения решать, какие программные компоненты распределенного приложения необходимо реплицировать. Критерии этого выбора на данный момент не актуальны, но очевидным соображением будет копирование подкомпонентов, которые оказывают более сильное влияние на требуемые свойства всей системы (надежность, доступность, безопасность и т.

д.), и их выполнение на исполнительных устройствах, которые более подвержены сбоям.

Таким образом, предлагаемый шаблон проектирования репликации допускает репликацию только частей приложения ИЕС 61499 и не накладывает ограничений на то, как они выделяются другим – новым или существующим – исполнительным устройствам. Для этого отказоустойчивого шаблона проектирования единицей репликации является FB. Это означает, что внутренние части составного FB не могут быть частично реплицированы: либо весь составной FB реплицируется полностью, либо не воспроизводится вообще. Это связано с тем ограничением, что составные FB не могут быть распределены между несколькими исполнительными устройствами, что, в свою очередь, связано с тем, что составные FB поддерживают внутреннее состояние и должны храниться в одном централизованном месте. Подпрограммы ИЕС 61499 могут быть частично реплицированы, так как они не имеют такого же ограничения.

На рисунке 9.2 показан пример того, как части распределенного приложения ИЕС 61499 могут быть реплицированы, а реплики распределены между исполнительными устройствами. Обратите внимание, что реплики не обязательно должны находиться в одном исполнительном устройстве. Приемлемо иметь одну реплику конкретного субприложения, исполняющуюся на одном устройстве, и другую реплику этого же субприложения, распределенную между двумя или более исполнительными устройствами. В этом случае, однако, применяются некоторые ограничения.

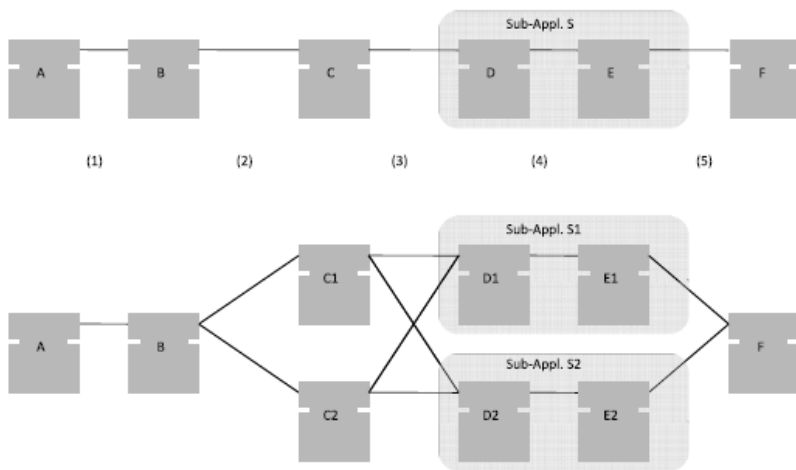


Рисунок 9.3. Сценарии взаимодействия с реплицированным FB

Теперь возможно идентифицировать несколько сценариев взаимодействия (см. рисунок 9.3) и, в зависимости от этого, реплицировать. Обратите внимание, что другой возможный сценарий взаимодействия возникает, когда

субприложение реплицируется полностью. Подпрограмма содержит соединения с данными и событиями внутри себя, и эти соединения также реплицируются, когда реплицируется полное подприложение. Это составляет другой отличный сценарий взаимодействия, обозначенный как (4).

В каждом из этих сценариев избиратели будут размещены на входе любого FB, который получает данные из реплицированного FB. Рассматривая каждый из сценариев взаимодействия по очереди, потребуется один избиратель сценария взаимодействия перед FB F (5) и один избиратель перед каждой из реплик подпрограммы сценария S (3) (см. рисунок 9.4). Особо следует отметить реплицированного избирателя в сценарии (3). Когда FB реплицируется, любые избиратели, предоставляющие данные этому FB, также должны реплицироваться, и каждый из этих избирателей должен быть назначен тому же исполнительному устройству, что и каждая из реплик. Основанием для этой архитектуры является устранение отдельных точек сбоев (упомянуто в разделе 9.3.3).

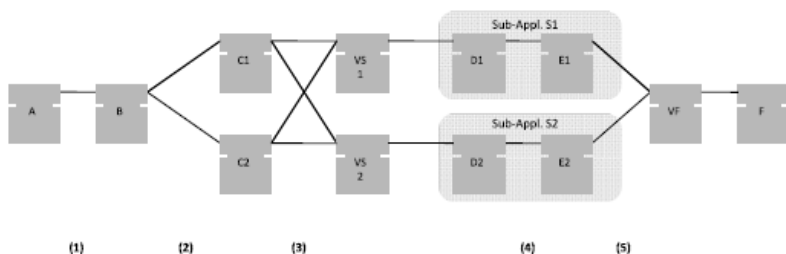


Рисунок 9.4. Сценарии взаимодействия и размещения избирателей

Если исходное нереплицированное приложение содержит ситуацию, в которой один FB обрабатывает события и/или данные, поступающие от двух восходящих FB (FB G, H и I на рисунке 9.5), или отправляет данные и/или события двум нисходящим FB (FB J, K и L на рис. 9.5), сценарии репликации останутся такими же, как те, что указаны на рис. 9.3, но каждый из них будет реализовываться несколько раз в одном и том же приложении.

9.3.1. Распространение сообщения

Использование реплик обычно имеет смысл, только если все реплики ведут себя одинаково при представлении с одинаковым набором входных значений. Другими словами, каждая реплика одного и того же FB должна гарантированно принимать одинаковые решения при получении одинаковых входных данных и/или событий. Внутренние алгоритмы могут, например, не зависеть от использования случайной величины, которая принимает различное значение в каждой из реплик. Предположение, что ни одна из реплик не имеет внутреннего сбоя, позволит репликам всегда представлять одинаковые значения на их выходных данных и, следовательно, предоставлять избирателям согласованные входные данные для работы.

Обратите внимание, что для того, чтобы реплики поддерживали одни и те же выходные данные и события, также необходимо, чтобы их внутренние переменные делали то же самое. Если внутренняя переменная не совпадает между репликами, это может привести к возможному несоответствию выходов в будущем, даже если выходы совпадают, тогда как внутренняя переменная становится несовпадающей. Набор текущих значений всех выходов и внутренних переменных часто называют внутренним состоянием FB. Свойство синхронизации этого внутреннего состояния между репликами называется детерминизмом реплик.

Чтобы поддерживать детерминизм реплик, также необходимо гарантировать, что все реплики получают одинаковые входные данные и события в одинаковом порядке. Взяв в качестве примера сценарий взаимодействия (2) на рисунке 9.3, данные, отправляемые FB В, должны поступать в том же порядке в каждую реплику С. Это может показаться тривиальным, но мы также должны учитывать, что FB С может иметь другие источники входных данных, кроме FB В. Например, когда реплицируется FB L на рисунке 9.5 (L1 и L2), обе реплики должны получать одинаковые входные данные в одинаковом порядке от обоих FB – J и K. Это должно быть гарантировано независимо от того, какая сеть связи используется.

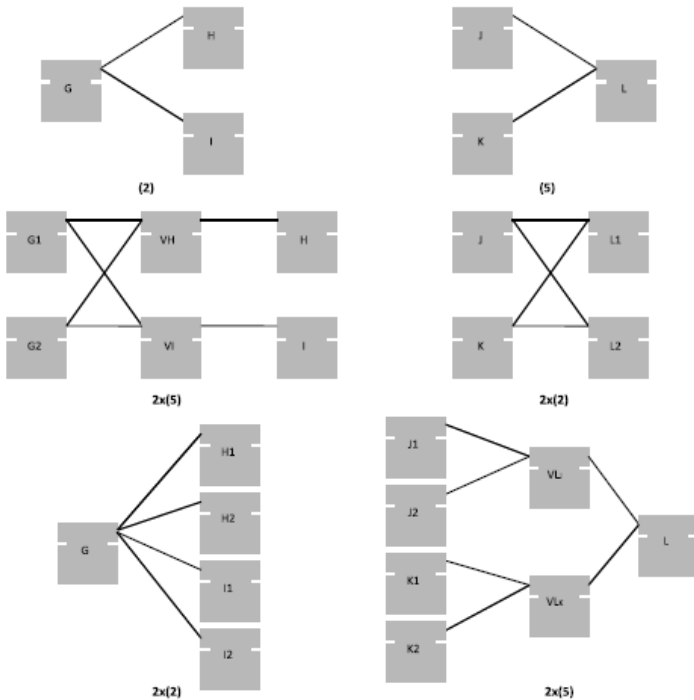


Рисунок 9.5. Репликация FB с несколькими восходящими и нисходящими соединениями

Обычно сети не предоставляют гарантий упорядочения, и поэтому необходима стратегия, гарантирующая, что все сообщения поступают в одинаковом порядке на все реплики. Было разработано несколько коммуникационных алгоритмов, которые могут предоставить эти гарантии, но они обычно требуют обмена несколькими сообщениями между всеми FB, чтобы те могли прийти к общему согласию относительно того, какой порядок рассмотреть (см. [4] для обзора). Этих решений недостаточно для выполнения общих жестких временных требований к приложениям для управления. По этой причине мы предлагаем использовать протокол синхронизированных сообщений [7]. Этот подход основан на времени и предполагает, что часы всех исполнительных устройств, используемых для запуска реплицированного приложения, синхронизированы.

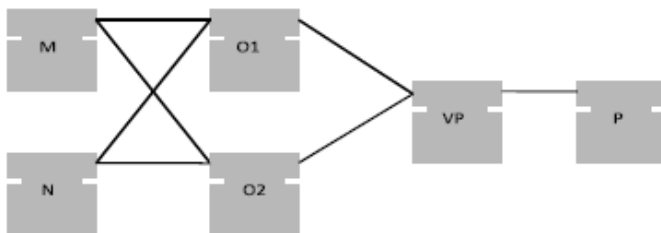


Рисунок 9.6. Синхронизированные сообщения

Используя этот протокол, каждое сообщение между двумя FB (где реплицируется хотя бы одно из них) несет в себе постоянное время – будущее время, когда сообщение считается действительным. Отправляющий FB рассчитывает это будущее время как текущее время, в которое отправляется сообщение, плюс фиксированный временной сдвиг, характерный для сообщения. Фиксированное смещение определяется в автономном режиме и должно иметь значение, превышающее все возможные задержки, которые могут возникнуть при отправке сообщения в FB назначения.

Рассмотрим, например, FB M и N, отправляющие сообщения на реплики O1 и O2 (рисунок 9.6). Фиксированные смещения, используемые каждым из отправляющих FB (M и N), не должны быть одинаковыми; фактически, если один FB имеет сетевое соединение с более высокими задержками (например, потому что он использует WiFi), он, вероятно, потребует использования более высокого значения смещения. Необходимо гарантировать, что каждое сообщение поступит как в O1, так и в O2 до истечения срока его действия. Каждая реплика (O1 и O2) затем буферизует все полученные сообщения и освобождает их только в то время, когда они становятся действительными. Это гарантирует, что каждое сообщение будет выпущено во всех репликах в том же порядке и в одно и то же время.

Эта точная синхронизация времени, предоставляемая протоколом синхронизированных сообщений, также используется, чтобы позволить голосующим FB знать, когда голосовать. Помните, что при наличии отказа одной из реплик эта реплика не будет отправлять никаких выходных

сообщений. После того, как принимающий FB (VP избирателя на рисунке 9.6) получил сообщение от одной из рабочих реплик, он должен знать максимальное время, в течение которого ему следует ожидать поступления соответствующих сообщений от других реплик. Только после того, как это максимальное время ожидания истекло, он может решить, что оставшаяся реплика вышла из строя, и поэтому может продолжить голосование, используя данные, которые он получил до этого момента времени.

Для этого каждая реплика O1 и O2 при отправке выходного сообщения избирателю добавляет к этому сообщению время, в которое сообщение можно считать действительным. Это время действия определяется путем добавления фиксированного смещения к времени действия события, которое начало цепочку выполнения. Давайте рассмотрим пример: FB M отправляет сообщение в обе реплики FB O (O1 и O2). Каждая из этих реплик будет обрабатывать сообщение, поступающее от M, в результате чего каждая реплика будет отправлять новое сообщение VP избирателя. В этом случае время достоверности сообщений, отправляемых O1 и O2, будет определяться путем добавления фиксированного смещения к времени достоверности сообщения, поступающего от M. Значение фиксированного смещения, добавляемого O1 и O2, должно быть одинаковым, таким образом, срок действия обоих сообщений, отправленных O1 и O2, будет одинаковым. Фиксированное смещение будет определяться в автономном режиме и должно учитывать не только максимальное время передачи сообщений, отправляемых O1 и O2, но также время выполнения самих FB O1 и O2.

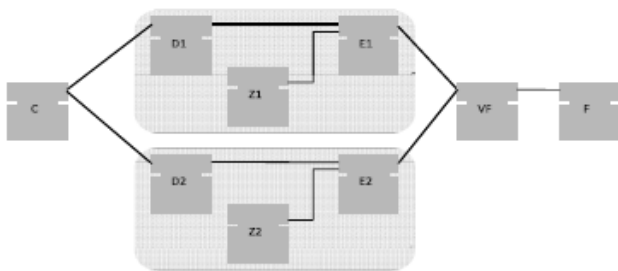


Рисунок 9.7. Синхронизация внутренних событий

Время действия сообщений, поступающих избирателю VP, также используется для их группировки. Избиратель проголосует только за сообщения, срок действия которых идентичен, поскольку только эти сообщения являются результатами одного и того же исходного события. Не имеет смысла позволять избирателю VP голосовать за одно сообщение, прибывающее из O1, которое было получено в результате обработки сообщения от M, вместе с другим сообщением, прибывающим из O2, но исходящим из-за сообщения из N. Хотя возникновение такого случая маловероятно, тем не менее это возможно: например, мы можем рассмотреть

большой разброс возможных сетевых задержек в сложных сетях, используя протокол связи на основе UDP.

Следует также отметить, что приложения IEC 61499 обычно не работают изолированно, и они обычно требуют ввода в физическое пространство. В IEC 61499 это делается с использованием FB сервисного интерфейса (SIFB), который предоставляет значение физического входа, подключенного к исполнительному устройству. При представлении репликации может также возникнуть необходимость реплицировать физические входные данные, особенно если репликация была введена, поскольку мы не можем достаточно доверять исполнительному устройству, к которому подключен вход. В этой ситуации один и тот же физический вход будет подключен к двум или более исполнительным устройствам, и SIFB, предоставляющие значение этого же физического входа на каждом из исполнительных устройств, должны рассматриваться как копии одного и того же SIFB – FB Z1 и Z2 на Рисунок 9.7.

Обратите внимание, что соединения внутри реплицированных подпрограмм на рисунке 9.7 представляют собой взаимодействие, обозначенное как тип (4) на рисунке 9.3. Точная синхронизация времени протокола синхронизированных сообщений также важна для взаимодействий этого типа. Здесь может быть необходимо гарантировать, что физические входные данные считываются в одно и то же время в каждой реплике. Мы можем форсировать одновременное чтение входных данных с помощью протокола синхронизированных сообщений в случае, когда запрашивается чтение физического ввода (при условии использования запрашивающего SIFB). В этом случае фиксированное смещение времени будет определяться максимальным временем выполнения FB, которые будут выполняться с предыдущей точки синхронизации (в этом случае с прибытия сообщения от C, которое высвобождается одновременно в обеих репликах) до чтения физического ввода (прибытие события в FB Z на рисунке 9.7). Поскольку физический вход всегда читается каждой репликой в один и тот же момент, соответствующее значение выходного сигнала FB Z также всегда будет одинаковым в каждой реплике. Поэтому вывод Z может безопасно использоваться любым другим FB в реплике, в то же время гарантируя синхронизацию между репликами.

Обратите внимание, что этой синхронизации может быть недостаточно для обеспечения синхронизации внутреннего состояния между репликами, если точное время, в которое SIFB читает физический вход, не находится под контролем приложения. Это тот случай, когда отвечающий SIFB используется в качестве интерфейса к физическим входам, поскольку этот FB может считывать физический вход и генерировать выходные события в моменты времени, которые контролируются только средой исполнения IEC 61499. Эти ситуации лучше избегать, если требуется очень строгий детерминизм реплик. Когда исключение невозможно, необходимо разделить между репликами все входные значения, считанные каждой из реплик, и разрешить каждому одному избирателю (используя тот же алгоритм голосования) использовать значение.

Этот подход имеет недостаток, заключающийся в добавлении дополнительных сообщений в сеть и больших задержках в последовательности обработки.

9.3.2. Семантика исполнения платформы

Как указывалось ранее, чтобы реплики поддерживали синхронизированное внутреннее состояние, выполнение каждой реплики должно быть идентичным. Это объяснялось как детерминизм реплик, причем было наложено ограничение: алгоритмы FB должны быть детерминированными. Однако FB обычно создаются средой выполнения IEC 61499, и для детерминизма реплики также требуется, чтобы среды выполнения также выполняли каждую реплику точно таким же образом.

Это особенно важно, когда реплики являются подпрограммами или составными FB, где выполнение реплики включает в себя распространение событий между FB в сети FB каждой реплики. Порядок выполнения событий в сети FB должен быть одинаковым для каждой реплики. Поэтому следует соблюдать особую осторожность, когда одна реплика полностью выполняется в одном исполнительном устройстве, тогда как другая реплика распределяется между вторым и третьим исполнительными устройствами. В этом последнем случае невозможно гарантировать детерминизм реплик.

Следует отметить, что время, необходимое каждой реплике для выполнения, не имеет значения – исполнительные устройства не должны иметь одинаковую вычислительную мощность и не должны прилагать одинаковую нагрузку для обработки. Протокол синхронизированного времени гарантирует, что исходные события обрабатываются в каждой реплике в одно и то же время, и когда это событие освобождается, одни и те же данные становятся доступны на входах FB всех реплик. С другой стороны, семантика исполнения IEC 61499 гарантирует, что когда FB активируется событием, оно начнется с чтения всех соответствующих данных, представленных на входах данных FB. Даже если входные данные изменяются, когда FB обрабатывает событие (поскольку сообщение, передающее новое значение данных, достигло своего времени действия), эти изменения просто игнорируются исполняющим FB.

Эта семантика не применяется к подпрограммам, которые не сохраняют копии значений на своих входах, когда они начинают обрабатывать входное событие. В этом случае больше невозможно гарантировать внутреннюю синхронизацию состояния. Хотя в каждой реплике данные станут действительными в одно и то же время, нельзя дать никаких гарантий относительно времени, в которое значения будут фактически считываться (поскольку исполнительные устройства могут иметь разные скорости обработки и нагрузки). Таким образом, синхронизация состояний между репликами просто невозможна при репликации подпрограмм.

Когда требуется синхронизация состояний между реплицированными FB, также важно гарантировать, что каждый FB обрабатывает только одно событие

за раз. Реплика, которая уже обрабатывает одно входное событие и получает второе входное событие, должна обрабатывать второе событие только после того, как первое было полностью обработано. По сути, выполнение FB реплики должно быть сериализуемым. Это ограничение является обязательным, поскольку реплики не гарантированно выполняются с одинаковой скоростью, и может возникнуть ситуация, когда одна реплика выполняет два события, одно за другим, в то время как другая реплика будет выполнять эти два события одновременно.

Также важно отметить, что исполнительное устройство, которое начинает отставать в обработке полученных сообщений, должно хранить эти сообщения и обрабатывать их в порядке, в котором они достигают своего срока действия. Рассмотрим пример исполнительного устройства, которое получает два сообщения со временем действия 100 и 110 единиц и значением данных со временем действия 120 единиц. При $t = 100$ устройство начинает обработку первого события, но завершает эту обработку только при $t = 140$. В это время оно начнет обработку второго события, но оно должен сделать это без учета данных, которые стали действительными при $t = 120$.

9.3.3. Функциональные блоки голосования

FB для голосования используются для объединения в одно выходное значение различных выходных значений, генерируемых репликами одного и того же FB. Возможны многие стратегии и алгоритмы голосования, и выбор используемого алгоритма должен учитывать желаемую семантику приложения.

Вероятно, наиболее распространенным алгоритмом голосования является мажоритарный избиратель, который выдает на выходе входное значение, которое встречается чаще всего (то есть представляет большинство). Этот алгоритм обычно имеет два возможных варианта. Один из вариантов переходит к голосованию, как только будет получено то же значение от большинства реплик, даже если некоторые из реплик еще не отправили свои значения. Другой вариант ожидает получения значения от каждой реплики, прежде чем приступить к голосованию. Эта последняя версия также должна быть настроена с тайм-аутом, чтобы не приходилось вечно ожидать значения, которое никогда не будет получено из-за сбоя реплики. Этот второй подход не имеет смысла для шаблона проектирования репликации, адаптированного для IEC 61499, поскольку данные, поступающие избирателям, считаются действительными только во время действия, содержащегося в самом сообщении. Голосование всегда происходит именно в это время, при этом любые значения могут быть получены до этого времени. Любые сообщения, поступающие избирателю со временем, относящимся к действию в прошлом, должны просто игнорироваться, поскольку они считаются недействительными – реплика, отправившая эти сообщения, считается сбойной.

Стратегия мажоритарного голосования не всегда может использоваться (например, при голосовании по значениям с плавающей запятой). Как известно, обработка значений с плавающей запятой может отличаться в разных аппаратных платформах как в незначительных, так и в важных деталях. Возможно также, что компилятор для каждой платформы переводит одно и то же математическое выражение в несколько разные операции сборки, что может привести к тому, что значения, за которые проголосовали, будут неидентичны. Также вариация может возникнуть тогда, когда каждая реплика считывает свой собственный физический ввод того же аналогового значения, это обязательно должно включать некоторую ошибку преобразования, которая может привести к немного отличающимся входным значениям в каждой реплике. Эти ситуации требуют использования алгоритмов голосования, которые не требуют, чтобы входные значения были идентичными.

Другие альтернативные стратегии голосования: наличие усредняющего избирателя, который выдает в качестве выходных данных среднее значение всех входных данных, и наличие медианного избирателя, который выбирает выходной сигнал в зависимости от того, какой вход находится посередине. Смесь этих двух стратегий заключается в отбрасывании экстремальных значений (максимальных и минимальных) и определении среднего из оставшихся значений. Еще одна альтернатива состоит в том, чтобы рассматривать в качестве отвергнутых только те значения, которые отличаются от медианы более, чем на настраиваемую константу (например, любое значение, которое отличается от медианы на 30 %, считается отвергнутым).

Усредняющий избиратель требует, чтобы входные значения были числовыми, в то время как медианный избиратель требует только, чтобы входные значения могли быть упорядочены. Когда входные значения не удовлетворяют ни одному из этих условий, может использоваться мажоритарный избиратель, поскольку единственное требуемое свойство входных значений состоит в том, что они могут сравниваться – равны они или нет.

Независимо от стратегии голосования, обратите внимание, что избиратель будет находиться на том же исполнительном устройстве FB, получающем сообщения. Это означает, что когда FB реплицируется, связанный избиратель не реплицируется. Таким образом, избиратель будет представлять собой единую точку отказа. В последнем разделе этой главы будет показано, что надежность результирующего реплицированного приложения никогда не может быть выше надежности любого отдельного нереплицированного компонента, включая избирателя. По этой причине нереплицированные избиратели и FB, которые обрабатывают свои данные, должны иметь высокую надежность, иначе они должны быть реплицированы. Репликация избирателя подразумевает репликацию FB, получающего данные избирателя, и приведет к сценарию связи (3) на рисунке 9.3.

Система управления обязательно будет включать в себя некоторые FB, которые активируют физический выход. Обычно не представляется возможным, чтобы два или более исполнительных устройства управляли

одним и тем же физическим выходом. Это подразумевает, что по крайней мере один нереплицированный избиратель должен существовать в конце последовательности обработки FB для записи в этот физический выход, что ограничивает надежность получаемого приложения. Однако обратите внимание, что в крайних случаях физические выходы могут быть реплицированы, и окончательный избиратель может быть встроен в аппаратное обеспечение, например, путем размещения реле последовательно или параллельно, в зависимости от стратегии голосования, требуемой для приложения. В очень экстремальных случаях могут быть воспроизведены даже исполнительные механизмы, которые будут подключены к одному физическому устройству. В этом случае можно сказать, что последний избиратель – это физическое устройство, которое соединяет все исполнительные механизмы и поэтому суммирует все силы, приложенные исполнительными механизмами.

Таким образом, стратегия голосования не является оптимальной для всех ситуаций. Следовательно, шаблон дизайна репликации требует, чтобы разработчики приложений явно внедряли избирателей. Голосующие FB могут быть выбраны из библиотеки FB, которые были тщательно протестированы и утверждены и даже, возможно, одобрены внешним органом. IEC 61499 не поддерживает FB с переменным числом входов, поэтому каждая стратегия голосования требует нескольких FB – каждая версия будет иметь установленное количество входов и сможет обрабатывать фиксированное количество реплик. Поэтому количество реплик должно быть фиксированным во время разработки.

9.3.4. Функциональные блоки связи

Приложения IEC 61499 получают доступ к коммуникационной сети через коммуникационные SIFB. Стандарт IEC 61499 допускает множество различных реализаций этих FB, причем каждая реализация обеспечивает поддержку определенного стека протоколов связи или сети. Однако стандарт IEC 61499 определяет общий интерфейс для всех этих коммуникационных SIFB, поэтому распределенное приложение можно легко перестроить для работы в отдельной сети, просто заменив коммуникационные SIFB.

Эта адаптивность архитектуры IEC 61499 используется для поддержки шаблона проектирования репликации. Предоставляя конкретные реализации коммуникационных SIFB, которые поддерживают протокол синхронизированных сообщений, разработчик приложения освобождается от необходимости реализовывать этот протокол. Шаблон проектирования предполагает две различные пары SIFB-связи для публикации и подписки. Одна пара используется для распространения сообщений типа «многие-к-одному», тогда как другая пара используется для распространения сообщений типа «один-ко-многим».

«Один-ко-многим» используется в сценарии взаимодействия (2) на рисунке 9.3 и отображается как «1» на рисунке 9.8. Пара «многие-к-одному» используется в сценарии (5) и представляется как «1tom». Сценарий 3 использует комбинацию этих двух пар.

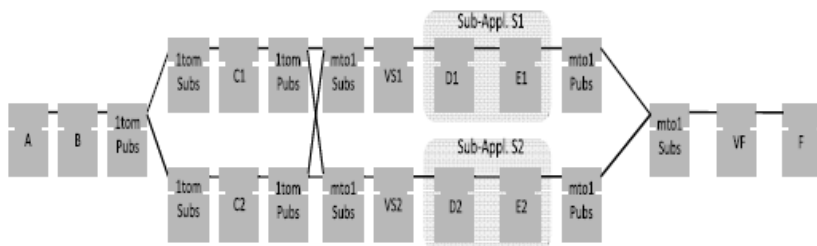


Рисунок 9.8. SIFB: сценарии взаимодействия, публикация и подписка

Обратите внимание, что если в реплицированном субприложении требуется какая-либо временная синхронизация (например, поскольку оно считывает физический ввод, как объяснено в разделе 9.3.1), можно использовать пару SIFB-связи «один-ко-многим» для публикации и подписки. В этом случае каждый подписчик будет настроен на получение сообщений от одного издателя, поскольку он не должен обмениваться данными между репликами. Пары SIFB-связи для публикации и подписки просто вставляются для синхронизации считывания входных данных.

9.4. Платформа репликации на FORTE

Поддержка описанного шаблона разработки репликации была реализована в среде выполнения FORTE IEC 61499 [1].

9.4.1. Семантика выполнения среды FORTE

Как объяснено выше, чтобы реплики поддерживали синхронизированные внутренние состояния, семантика среды исполнения IEC 61499 должна быть детерминированной. Другими словами, для каждой активации исходного события (то есть события, которое генерируется автономно с помощью SIFB без получения входного события, например FB подписки или периодического FB таймера) последовательность активированных FB всегда должна быть такой же, что и во всех репликах. FORTE относительно хорошо подходит для поддержки шаблона проектирования репликации, поскольку его семантика выполнения почти гарантирует детерминированное выполнение последовательностей событий.

Экземпляр среды выполнения FORTE реализует один или несколько ресурсов, которые мы до сих пор называли исполнительными устройствами.

Поскольку каждый ресурс в FORTE обрабатывается автономно, без общения, в дальнейшем нами будет рассматриваться только один ресурс на виртуальную машину FORTE.

Все события внутри ресурса обрабатываются одним фоновым потоком и нулем, одним или несколькими потоками реального времени. Каждый из этих потоков поддерживает очередь ожидающих событий, которые были освобождены и ожидают обработки. Каждое исходное событие явно отображается в одном потоке в реальном времени (то есть помещается в очередь событий этого потока) или оставляется для выполнения одним фоновым потоком. Каждое событие обрабатывается индивидуально в порядке «первым пришел – первым вышел». При обработке любого события новые события, которые генерируются во время его выполнения, добавляются в конец очереди. Это означает, что все другие события, сгенерированные во время обработки данного события, обрабатываются тем же потоком.

Обратите внимание, что FB, который принимает входные события, поступающие от двух разных SIFB, будет выполняться в потоках, с которыми связаны SIFB. Поэтому FB может работать в двух отдельных потоках, возможно, даже одновременно (в планировщике из потока реального времени с более высоким приоритетом будут вытесняться потоки с более низким приоритетом). Это разрушит сериализуемость обработки FB, поэтому необходимо соблюдать осторожность при настройке ассоциации SIFB с потоками выполнения, чтобы гарантировать, что каждая реплика будет работать в одном потоке.

Хотя это еще не было указано явно, мы можем видеть, что описанный шаблон проектирования репликации гарантирует, что все данные и события, поступающие в реплику, должны проходить через протокол временных сообщений. Следовательно, реплики могут быть свободно выделены любому потоку в ресурсе (если они всегда выполняются только в одном потоке), поскольку любые данные и события, которые они получают, будут защищены протоколом временных сообщений.

9.4.2. Функциональные блоки связи

Шаблон проектирования репликации требует использования двух пар FB, которые реализуют протокол синхронизированных сообщений (пары «один-многим» и «многие-к-одному»). В принципе, конкретная версия этих пар FB потребуется для поддержки каждого протокола связи. Однако, поскольку все коммуникационные SIFB-модули FORTE уже реализованы с использованием многоуровневого шаблона проектирования, необходимо было добавить только два новых уровня репликации.

При добавлении публикации или подписки FB в приложение FORTE требуется, чтобы у этого FB был входной параметр, в котором перечислены все протоколы связи, которые этот конкретный FB будет использовать. Стандартные протоколы связи следуют многоуровневому подходу, и это дает

разработчику полный контроль над тем, какие протоколы использовать. Поэтому разработчику приложения, который добавляет репликацию к существующей репликации, просто нужно создать новые реплики и добавить один из протоколов синхронизации («один-ко-многим» или «многие-к-одному») в стек протоколов всех коммуникационных SIFB, отправляющих и получающих данные из реплик.

Помните, что протоколы синхронизации должны быть настроены с соответствующим фиксированным значением смещения. Разработчик приложения должен определить правильное значение для этого фиксированного смещения, поскольку оно будет зависеть от максимальных ожидаемых сетевых задержек текущей настройки, а также от максимального времени выполнения некоторых реплицированных FB.

Также обратите внимание, что при использовании пары «многие-к-одному» SIFB-подписки должны предоставлять набор выходных данных для каждой реплики. Все эти выходы затем подключаются к избирателю, который объединяет их в единый набор выходных данных. Например, если каждый из издателей «многие-к-одному» отправляет два значения данных и используются три реплики, у каждого издателя будет два ввода данных для каждого значения данных, а у подписчика будет шесть выходов данных (по два на реплику).

Реализация уровней репликации «многие-к-одному» и «один-ко-многим» отправляет вместе с данными время, в которое данные и связанное с ними событие станут действительными. На стороне подписки оба слоя будут буферизовать все полученные сообщения до тех пор, пока не будет достигнут срок действия. Обратите внимание, что многотональным FB требуется доступ к времени действия исходного события, как показано на рисунке 9.6. Данные о времени должны перемещаться вместе с событиями, поскольку они распространяются в реплицированном составном FB, пока не достигнут публикуемого SIFB. Стандартная реализация FORTE просто отправляет события и не может связать время действия с распространяемым событием. Пересмотренная реализация FORTE, поддерживающая шаблон проектирования репликации, была изменена, поэтому время действия автоматически распространяется вместе с событиями в сети FB.

9.4.3. Функциональные блоки выбора

Пока еще не было приложено больших усилий для реализации полного набора FB для выбора для пересмотренной реализации FORTE, поддерживающей шаблон проектирования репликации, в основном из-за большого количества перестановок типов данных, алгоритмов выбора и количества входов в каждом избирателе. Предоставляются только очень простые и базовые избиратели. Внедрение новых избирателей, тем не менее, относительно тривиально; это то же самое, что и внедрение любого другого базового FB в среде разработки FORTE и 4DIAC. Поэтому любому реплицированному разработчику приложений рекомендуется внедрять

избирателей, которые обслуживают сложившуюся ситуацию, принимая во внимание режимы отказов, которые необходимо допустить.

9.5. Пример реплицированного применения ИЕС 61499

В этом разделе будет представлен пример того, как применить репликацию в приложении ИЕС 61499, используя ранее описанный шаблон проектирования. Данное приложение очень простое, и основное внимание будет уделено тому, что необходимо добавить для обеспечения отказоустойчивости.

9.5.1. Пример применения

Приложение, которое будет служить примером, взято из более крупного приложения, которое управляет распределением заготовок на заводе на базе конвейеров. Для простоты применения рассматриваются только два типа конвейеров: линейный и вращающийся. Конвейеры обоих типов относительно короткие и в любое время могут быть заняты только одной деталью. Каждый конвейер имеет единственный датчик, который обнаруживает заготовку, центрированную на конвейере. Конвейеры монтируются в соответствии с компоновкой на рисунке 9.9.

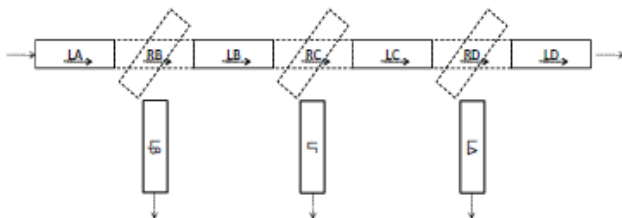


Рисунок 9.9. Конвейерная компоновка

Заготовки доставляются слева и должны быть перенесены на любую из обрабатывающих станций внизу. Каждый конвейер управляется одним экземпляром FB соответствующего типа (контроллера линейного или вращающегося типа). Каждый конвейер также работает благодаря своему собственному независимому исполнительному устройству (или PLC), к которому подключены исполнительные механизмы и датчики соответствующего конвейера.

Алгоритмы управления каждым FB очень просты. Линейные конвейеры просто пытаются протолкнуть заготовку на следующий конвейер, как только она освободится. Вращающиеся конвейеры получают заготовку с конвейера слева и протолкнут ее на один из следующих конвейеров (справа или снизу), в зависимости от того, что освободится первым.

Из-за возможного проскальзывания заготовки при перемещении с одного конвейера на другой конвейер считается свободным для приема новой заготовки только тогда, когда текущая заготовка достигла следующего конвейера. Таким образом, каждый линейный конвейер имеет четыре основных внутренних состояния:

- 1) свободен и остановлен;
- 2) свободен и работает (возможно получение заготовки с предыдущего конвейера);
- 3) занят и остановлен;
- 4) занят и работает (перенос заготовки на следующий конвейер).

Вращающиеся конвейеры будут иметь немного более сложную диаграмму состояний, поскольку им необходимо решить, какой конвейер получит следующую заготовку.

Сеть FB, управляющая последовательностью конвейеров, изображена на рисунке 9.10. Обратите внимание, что каждый конвейер должен получать и отправлять события на каждый из следующих конвейеров и предыдущий конвейер. Данные, отправляемые на каждый из следующих конвейеров, используются для того, чтобы сказать этому конвейеру о том, что надо принять новую заготовку. Данные, полученные от каждого из следующих конвейеров, предназначены для выяснения того, свободен ли этот конвейер для приема заготовки или нет.

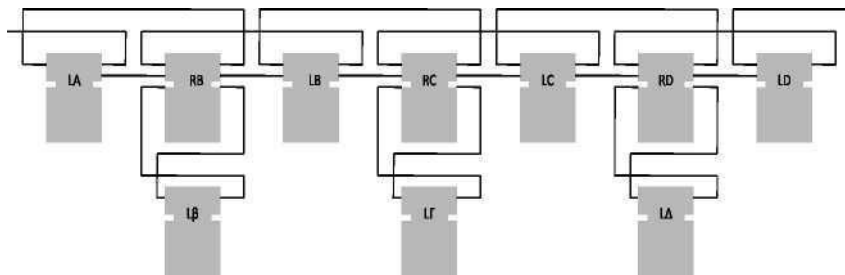


Рисунок 9.10. Пример не реплицированного приложения

9.5.2. Добавление репликации

Нужно признать, что это приложение не достигло требуемой надежности, и поэтому было решено добавить репликацию на контроллеры конвейеров вдоль горизонтальной линии. Также было решено, что каждое исполнительное устройство будет управлять двумя конвейерами одновременно, как показано на рисунке 9.11.

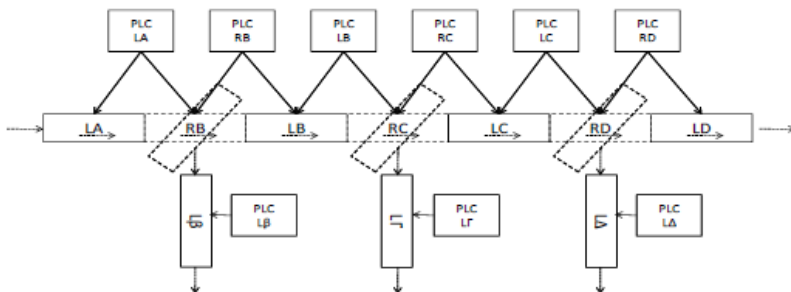


Рисунок 9.11. Выделение реплик для PLC

Реплицированная система управления, включая требуемые публикации и подписки, станет довольно сложным, поскольку в этом приложении каждый FB должен одновременно передавать и принимать события для каждого из окружающих FB. На рисунке 9.12 представлено реплицированное приложение с учетом только коммуникационного SIFB, необходимого для передачи информации в следующий FB (чтобы тот начинал прием заготовки). Канал связи, который идет назад к предыдущему FB (для сообщений о готовности принять заготовку), не представлен.

Обратите внимание, что эта ситуация предусматривает взаимодействие типа (3) (представленное на рисунке 9.3) между RB и LB и тип взаимодействия (5) между RB и L. По этой причине используются как пара «один-ко-многим», так и пара «многие-к-одному». Все подписчики будут типа «многие-к-одному».

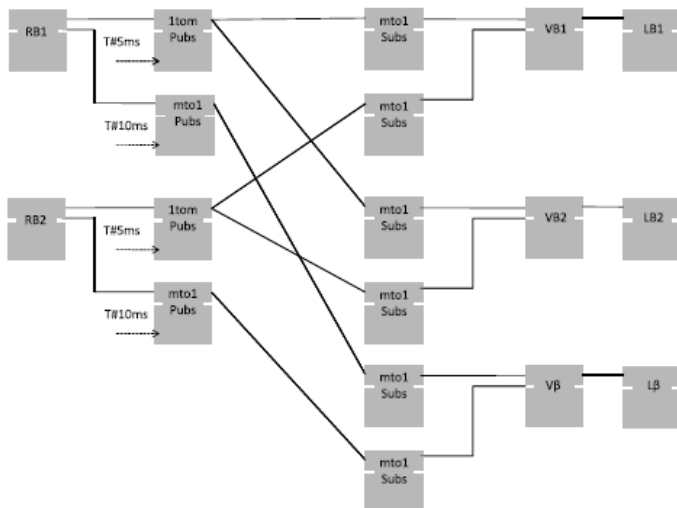


Рисунок 9.12. Пример реплицированного приложения

Обратите также внимание и на то, как каждый публикующий SIFB должен быть настроен с дополнительным параметром задержки. Этот параметр задержки должен быть одинаковым у каждого из издателей, передающих сообщение в одну и ту же реплику, чтобы избиратель мог сгруппировать поступающие сообщения.

9.5.3. Обработка выбора

Поскольку каждый FB в этом приложении имеет только две реплики, используемые избиратели будут предполагать, что любое полученное сообщение является правильным и поэтому выберут первое сообщение, которое должно поступить, для размещения на его выходе. Голосование на физических выходах также необходимо, поскольку каждый физический привод управляется двумя PLC, а не одним. В этом приложении предполагается, что неисправный PLC установит для всех своих цифровых выходов значение false (то есть каждый PLC следует модели аварийной остановки в случае сбоя). Это означает, что выходы обоих PLC могут быть соединены параллельно, чтобы управляемый двигатель был активирован до тех пор, пока хотя бы один из PLC не активирует связанный с ним выход. Как минимум, это один избиратель, построенный в виде электрической цепи.

9.6. Количественная оценка надежности системы

Надежность системы $R(t)$ – это вероятность правильной работы в течение заданного периода времени. Предполагается, что система построена из нескольких подкомпонентов, надежность которых известна. Поэтому определение надежности всей системы является относительно простым вычислением вероятностей. Хороший способ сделать это – использовать метод блок-схемы надежности (RBD) [6].

Система моделируется диаграммой блоков, с входом слева и выходом справа. Блок добавляется для каждого подкомпонента, и соединения между блоками следуют логической функции, которую они выполняют (а не их физическому расположению). Например, чтобы система работала правильно при том условии, что все подкомпоненты работают правильно, они размещаются последовательно между узлами входа и выхода. А чтобы система работала правильно при условии, что работает только один из подкомпонентов, блоки размещаются параллельно.

Предполагая, что вероятность отказа каждого подкомпонента является независимой, надежность системы, состоящей из последовательных блоков, определяется произведением надежности всех подкомпонентов. Для системы, состоящей из параллельных блоков, расчеты проще проводить на основе ненадежности подкомпонентов (определяется как $1 - R(t)$). В этом случае

ненадежность системы определяется произведением ненадежности всех подкомпонентов.

Надежность системы, состоящей из сложной сети последовательных и параллельных подкомпонентов, может быть определена путем последовательного сведения каждой последовательности блоков параллельно или последовательно к одному блоку эквивалентной надежности. В качестве примера рассмотрим тривиальную систему, состоящую из одного не реплицированного входа, трижды реплицированного блока обработки, за которым следует нереплицированный блок выбора и вывода. Предполагая, что каждый блок выполняется отдельным исполнительным устройством и что избиратель использует алгоритм, который требует для получения вывода, чтобы функционировала только одна из реплик обработки, эта система может моделироваться RBD, как показано на рисунке 9.13. Глядя на эту модель RBD, легко понять, что надежность избирателя снизит общую надежность системы (так как модель размещена последовательно со всеми другими блоками), поэтому важно, чтобы избиратель обладал высокой надежностью. Если избиратель реализован внутри того же исполнительного устройства, что и конечное устройство вывода, он может быть удален из модели RBD, и это не повлияет на общую надежность системы.

Следует, однако, отметить, что не все системы могут быть описаны сетью независимых серий и параллельных компонентов. Если избиратель использует мажоритарное голосование и, следовательно, требует, чтобы n из m реплик оставались функционирующими, простое размещение блоков параллельно не будет работать, и специальные узлы, которые идентифицируют семантику n из m (например, 2 из 3), должны использоваться в RBD. Другим примером, в котором изображение RBD является более сложным, является пример из предыдущего раздела (см. рисунок 9.11), в котором один и тот же PLC используется несколькими репликами, и эти PLC не могут быть организованы в непересекающиеся группы. В этом случае становится необходимым иметь несколько блоков в RBD, представляющих одно и то же исполнительное устройство, что делает недействительным использование ранее упомянутого метода уменьшения модели RBD и вычисляет результирующую надежность, поскольку этот метод действителен только до тех пор, пока все блоки независимы (то есть вероятность отказа одного блока в RBD полностью не зависит от вероятности отказа другого блока). В этих случаях могут использоваться более сложные алгоритмы, которые из-за их сложности обычно не подходят для ручного вычисления [3, 5]. В этих случаях предпочтительны программные средства, которые выполняют анализ RBD.

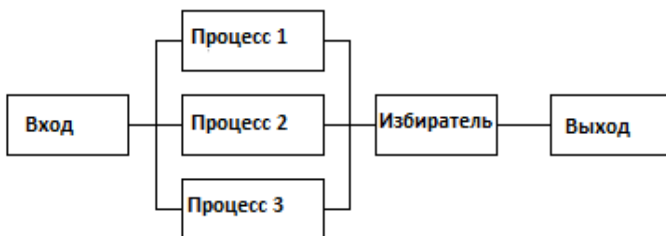


Рисунок 9.13. Простая блок-схема надежности

Другая ситуация, которая может потребовать программных средств, – это когда надежность блока не постоянна, а изменяется во времени в соответствии с некоторой функцией надежности ($R(t)$). В этом случае обычно используется экспоненциальная функция, поскольку она представляет систему с постоянной вероятностью отказа. Другими словами, вероятность неудачи в будущем всегда одинакова и не зависит от того, как долго система работала правильно, без каких-либо ошибок. Экспоненциальная функция выглядит так:

$$R(t) = e^{-\lambda t}$$

где λ – постоянная частота отказов. В этом сценарии умножение надежности последовательных блоков (умножение экспоненциальных функций) приводит к простому добавлению частоты отказов, но ситуация для параллельных блоков, к сожалению, более сложная.

Читатель также должен понимать, что в предыдущих примерах учитывается только надежность аппаратного обеспечения. В действительности дополнительное программное обеспечение, добавляемое для поддержки репликации, может содержать непреднамеренные ошибки, и это также может быть смоделировано функциями надежности. В этом случае RBD будет состоять из блоков, которые представляют программные функции, а не из аппаратных средств.

9.7. Выводы

При разработке распределенных приложений возникает необходимость учитывать возможность частичных сбоев. Меры, которые допускают большую вероятность появления неисправностей, могут потребоваться, если работа системы требует достижения определенного уровня надежности. Когда к приложению, работающему в реальном времени, также выдвигается такое требование, используемые отказоустойчивые меры должны быть ограничены теми, которые используют подход прямого восстановления.

Добавление репликации модуля с голосованием на выходах реплик является одной из таких мер, но многие алгоритмы выбора требуют, чтобы все реплики были синхронизированы. В этой главе был проведен анализ того, как

этого можно достичь для приложений IEC 61499 ограничений, которые необходимо наложить на среды исполнения IEC 61499.

Можно сделать вывод: хотя стандартные шаблоны проектирования для реплик и выбора могут быть легко адаптированы для приложений IEC 61499, следует обратить внимание на семантику выполнения IEC 61499, реализованную исполнительными устройствами. Они должны использовать детерминированные алгоритмы, а выполнение событий должно быть сериализуемым. Глобальный порядок выполнения событий также должен соблюдаться, поэтому был предложен протокол с синхронизированными сообщениями в качестве средства достижения этой цели с небольшими накладными расходами на обмен сообщениями. Поддержка этого протокола была добавлена в среду исполнения FORTE, но на момент написания этой главы еще не была включена в основной дистрибутив данного программного пакета.

Автор хотел бы поблагодарить организации, которые финансировали эту работу, а именно Европейский фонд регионального развития (ERDF) через Программу COMPETE (оперативная программа по повышению конкурентоспособности), а также Fundação para a Ciência e a Tecnologia (Португальский фонд науки и техники), проекты FCOMP-01-0124-FEDER-037281 и FCT EXPL / EEIAUT / 2538/2013.

Библиография

- [1] 4DIAC and FORTE project. <http://www.fordiac.org/>.
- [2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, pages 11–33, 2004.
- [3] A. Birolini. *Reliability Engineering: Theory and Practice*. Springer Verlag, 2014.
- [4] X. Defago, A. Schiper, and P. Urbanr. Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Computing Surveys*, Vol. 36, No. 4, 372–421, 2004.
- [5] S. Distefano and A. Puliafito. Dependability Evaluation with Dynamic Reliability Block Diagrams and Dynamic Fault Trees. *IEEE Transaction on Dependable and Secure Computing*, Vol. 6, No. 1, 4–17, 2009.
- [6] M. Modarres, M. Kaminskiy, and V. Krivtsov. *Reliability Engineering and Risk Analysis: A Practical Guide*. CRC Press, 2009.
- [7] S. Poldena, A. Burns, A. Wellings, and P. Barret. Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems. *IEEE Transaction on Computers*, Vol. 49, No. 2, 100–111, 2000.

Разработка функциональных блоков интерфейса связи IEC 61499 в приложениях распределенного управления и автоматизации

Georgios Sfiris

Университет Аристотеля в Салониках

George Hassapis

Университет Аристотеля в Салониках

СОДЕРЖАНИЕ

10.1 Введение	220
10.2 Программирование IEC 61499 и коммуникационная семантика	223
10.2.1 Общая семантика	223
10.2.2 Функциональный блок интерфейса службы связи	223
10.2.2.1 Инициализация CSIFB	225
10.2.2.2 Однонаправленный обмен данными	225
10.2.2.3 Двухнаправленный обмен данными	226
10.2.2.4 Деинициализация CSIFB	227
10.2.3 Профили соответствия	228
10.3 Пример распределенного приложения	229
10.4 UDP/IP и протоколы связи TCP/IP в IEC 61499	231
10.4.1 Протокол пользовательских дейтаграмм / протокол интернета	231
10.4.2 Протокол управления передачей / протокол интернета	232
10.4.3 Пример приложения UDP/IP и реализации TCP/IP	232
10.5 Пример системы SCADA	235
10.5.1 Пример применения протокола UDP/IP в системе SCADA	238
10.6 Протокол связи Modbus в IEC 61499	238
10.6.1 Реализация протокола Modbus в IEC 61499	240
10.6.2 Конфигурирование CSIFB для использования протокола Modbus	240
10.6.3 Пример приложения реализации Modbus	243
10.7 Реализации других протоколов связи в IEC 61499	245
10.7.1 Коммерческие инструменты IEC 61499	246
10.7.1.1 ISaGRAF	246

10.7.1.2 nxtSTUDIO	246
10.7.2 Открытые инструменты IEC 61499	247
10.8 Пример программирования	247

10.1. Введение

Типичные промышленные автоматизированные системы управления для обрабатывающей промышленности параллельно выполняют множество задач, таких как считывание данных с датчиков, выполнение алгоритмов вычислительного управления, вырабатывающих выходные данные для регулирования параметров процесса и обновления отображения и передачи данных процесса в информационные системы предприятия. Эти задачи, в зависимости от их количества и сложности, реализуются иерархическими и распределенными компьютерными архитектурами, состоящими из множества вычислительных узлов. Первая такая промышленная автоматизированные системы управления была анонсирована компанией Honeywell в середине 1970-х годов под торговой маркой TDC 2000/3000. С момента своего появления эта концепция широко использовалась в системах управления электростанций.

Использование локальных сетей для соединения компьютеров и других устройств в рамках промышленной автоматизированной системой управления стало популярным с 1980 года, и сегодня это норма для всех систем, установленных в большинстве отраслей промышленности. Существуют два подхода к ее реализации. Первый использует запатентованные проекты и архитектуры, а второй рассматривает открытые распределенные архитектуры с коммуникацией через сети типа Ethernet или сети, построенные в соответствии с общедоступными стандартами. Говоря об открытых архитектурах, можно сказать, что любая промышленная автоматизированная система управления обычно имеет структуру, включающую несколько иерархических уровней. Каждый уровень имеет различные потребности в коммуникации и свою функциональность, которая предъявляет различные требования к сети коммуникации. На рисунке 10.1 показан пример иерархии в промышленной автоматизированной системей управления.

Самым нижним уровнем иерархии является полевой уровень, который включает в себя такие устройства, как исполнительные механизмы и датчики. Задача сети на этом уровне состоит в том, чтобы передавать данные с датчиков производственного процесса на контроллеры, которые выполняют алгоритмы для регулирования производственного процесса в конкретных рабочих точках и передают результаты алгоритмов на исполнительные механизмы. Это большая категория сетей, характеризующихся короткими размерами сообщений и предсказуемым временем отклика в диапазоне от 0,001 до 0,01 секунд. В целом, эти сети соединяют интеллектуальные устройства, которые предлагают диагностические и конфигурационные

возможности за счет большего интеллекта, вычислительной мощности и высокой цены.

На следующем уровне сети – на рисунке 10.1 это уровень управления – информационный поток в основном заключается в односторонней связи между контроллерами и компьютерными системами, используемыми для интерфейса человек–машина (HMI), архивации истории, диспетчерского управления, синхронизации машин и обработки. Для последних задач сетевые требования отличаются от требований первых задач, поэтому необходима адаптивная сегментация сообщений. Размеры сообщений и файлов для диспетчерского и синхронизирующего управления относительно велики, а время отклика их передачи находится в диапазоне от 0,5 до 2 секунд. Что касается мгновенного отображения и хронологических графиков, то тут размер сообщений даже больше, чем в случае с диспетчерским и синхронизирующим управлением, и для их передачи требуется время отклика в диапазоне от 2 до 6 секунд. Многие компании разработали собственные сети как для полевого уровня, так и для уровня контроля. Сети полевого уровня обычно называют полевыми шинами, и они отличаются такими характеристиками, как размер сообщения и время отклика. Известные названия сетей этого уровня – FOUNDATION Fieldbus [6], CANopen [2], DeviceNet [33] и Profibus-DP [19]. На уровне управления некоторые из полевых шин также могут использоваться, но в большинстве случаев предпочтительными являются сети с более высокой скоростью передачи данных, лучшей защитой данных и гарантированной синхронизацией времени, а также с не очень требовательным откликом в реальном времени. Известные названия сетей этого уровня – ControlNet [34], Profinet [20] и Modbus [29, 28, 27]. Кроме того, часто используется коммутируемый или промышленный Ethernet с TCP/IP [26].



Рисунок 10.1. Иерархическая система промышленной автоматизации

Верхний сетевой уровень промышленной автоматизированной системы управления – это информационный уровень, который собирает

управленческую информацию с уровня управления и управляет всеми функциями автоматизации предприятия. На этом уровне существуют крупномасштабные сети, например Ethernet и WAN, для планирования процесса и обмена управленческой информацией, а сети Ethernet могут выступать в качестве шлюзов для соединения с другими промышленными сетями.

Растущее развертывание промышленных автоматизированных систем управления с открытой распределенной компьютерной архитектурой предъявляет высокие требования к инструментам и методам для разработки распределенного программного обеспечения производительным и эффективным способом. Предыдущие усилия по стандартизации, которые привели к стандарту IEC 61131-3 [18], предлагали языки программирования для управления и автоматизации с конструкциями, независимыми от аппаратного обеспечения, на котором будут выполняться их исполняемые коды, но это не учитывало распределенную природу программного обеспечения. Эта аппаратная независимость сохраняется только для логики управления. Однако функции считывания значений с датчиков и настройки исполнительных механизмов и обмена данными по полевой шине или сети уровня управления требуют реализации конкретных конструкций, в зависимости от характеристик сети и технологии датчиков и исполнительных механизмов.

Полевые шины и сети уровня управления характеризуются протоколами, смоделированными в соответствии со стандартом OSI [21]. Однако в случае полевых шин и сетей управления используется сокращенный и упрощенный стек протоколов, включающий только уровни 1, 2 и 7 модели OSI [35]. Разработчик должен выбрать сетевой протокол полевой шины, который может подходить для рассматриваемого приложения и устройств, которые имеют те же характеристики физического уровня, что и рассматриваемая сеть. Очевидно, что программирование обмена данными между сетевыми устройствами, которые требуются для функций управления, не является простой задачей, если только нет библиотек в форме сервисов, которые могут вызываться программой управления через абстрактный интерфейс, который должен быть совместим с конструкциями языка программирования логики управления.

Вышеуказанные проблемы в определенной степени решаются стандартом IEC 61499 [16], в котором предлагается компонентный подход к разработке распределенного программного обеспечения. Что касается логики управления, то это, вероятно, уместно для аппаратно-независимых абстрактных программных разработок. Тут предоставляется шаблон для разработки необходимых библиотек связи для передачи данных логике управления от узлов, имеющих различные характеристики и требующих, чтобы эта передача происходила в соответствии с правилами рассматриваемого сетевого протокола. Для проектировщика приложений эти библиотечные компоненты делают передачу прозрачной, тем не менее они должны быть разработаны специалистами и стать доступными для среднего разработчика приложений.

Далее будет объяснен механизм, предложенный стандартом IEC 61499 для разработки таких библиотек и включения их в среду разработки приложения. Также простой пример покажет, как прикладное программное обеспечение может быть написано с использованием библиотеки сетевого сервиса. Но сначала мы дадим краткий обзор основных концепций стандартных коммуникационных конструкций IEC 61499 и объясним, как предоставляемые механизмы библиотек связи можно интегрировать в другие конструкции программного обеспечения, созданные в соответствии с рекомендациями IEC 61499.

10.2. Программирование IEC 61499 и коммуникационная семантика

10.2.1. Общая семантика

Стандарт IEC 61499 вводит серию обозначений моделирования, состоящих из функционального блока, приложения, ресурса, устройства и системы.

Функциональный блок (FB) инкапсулирует локальные данные и алгоритмическое поведение с четко определенным интерфейсом. Такие компоненты могут быть связаны событиями и линиями данных и в конечном итоге могут быть распределены по разным узлам и устройствам распределенной компьютерной системы. Приложение представляет собой программный объект, состоящий из сети взаимосвязанных FB, которые решают проблему управления. Ресурс – это функциональная единица, содержащаяся в устройстве. Устройство представляет собой автономный аппаратный элемент, содержащий процессор, память и интерфейсы связи. Система представляет собой совокупность устройств, соединенных между собой и связывающихся друг с другом посредством сетей связи.

10.2.2. Функциональный блок интерфейса службы связи

В каждое устройство промышленной автоматизированной системы управления с распределенной компьютерной архитектурой должно быть загружено программное обеспечение, реализующее протокол сети, к которой оно будет подключено. Это программное обеспечение должно быть в форме исполняемого кода, предназначенного для запуска на определенном оборудовании. Данное программное обеспечение также должно иметь интерфейс, позволяющий осуществлять его вызов из приложения, которое будет запрашивать чтение или передачу данных по сети. Специальный тип функционального блока, называемый функциональным блоком интерфейса службы связи (CSIFB), предусмотрен IEC 61499 для выполнения вышеуказанной цели. Коммуникационные механизмы абстрагированы внутри этих специальных функциональных блоков, скрывающих базовую реализацию, в то же время открывая интерфейс связи для приложения.

CSIFB можно настроить в соответствии с конкретными характеристиками сетевого протокола. Чтобы настроить CSIFB, параметры сети должны быть известны, и программное обеспечение, реализующее протокол, должно быть доступно для включения в алгоритмический раздел CSIFB. Стандарт предлагает четыре типа CSIFB для применения различных моделей связи: PUBLISH, SUBSCRIBE, CLIENT (КЛИЕНТ) и SERVER (СЕРВЕР) FB (см. рисунок 10.2). CSIFB позволяют приложению IEC 61499 взаимодействовать с коммуникационной инфраструктурой устройства, либо получая к нему доступ, либо прерываясь им.

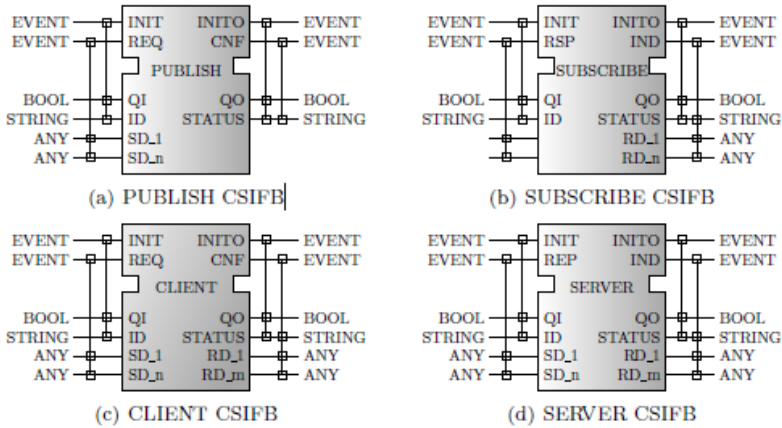


Рисунок 10.2. Четыре наиболее часто используемых функциональных блока интерфейса связи IEC 61499

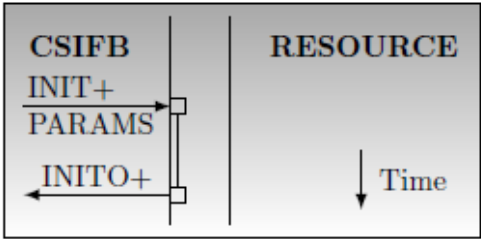


Рисунок 10.3. Диаграмма последовательности услуг при инициализации CSIFB

В дополнение к спецификации данного типа, IEC 61499 определяет другую нотацию, чтобы лучше показать временные и последовательные взаимосвязи между CSIFB и базовым оборудованием или CSIFB, размещенным в другом устройстве. Эти обозначения называются диаграммами последовательности услуг и взяты из стандартов связи. Пример приведен на рисунке 10.3. Мы

будем использовать эти обозначения, чтобы лучше объяснить, как CSIFB инициализируются и соединяются для обеспечения связи между устройствами.

10.2.2.1. Инициализация CSIFB

На рисунке 10.3 дана диаграмма последовательности действий инициализации CSIFB. Горизонтальные стрелки показывают входящие и исходящие события, а две вертикальные линии показывают границы домена. Левая часть является доменом CSIFB, в то время как правая часть изображает базовый домен ресурсов. Время увеличивается вниз. Чтобы вызвать и инициализировать CSIFB, его вход события INIT должен запускаться с событием INIT+ (событие INIT с входом QI, установленным в TRUE) вместе с соответствующими сетевыми параметрами. При успешной инициализации CSIFB генерирует событие INITO+ (событие INITO с выходом QO, установленным в TRUE). Событие INITO+ происходит в результате события INIT+, поэтому первое находится ниже второго. Вертикальная линия указывает на связь между двумя событиями.

10.2.2.2. Однонаправленный обмен данными

PUBLISH FB используются для отправки данных в сеть без ожидания какого-либо подтверждения, в то время как SUBSCRIBE FB получают информацию из сети без отправки какого-либо подтверждения. Это позволяет нескольким PUBLISH FB отправлять сообщение одному и тому же пункту назначения или нескольким SUBSCRIBE FB и прослушивать один и тот же источник. PUBLISH и SUBSCRIBE FB чаще всего используются для реализации многоадресной технологии и обычно спарены в распределенных приложениях IEC 61499.

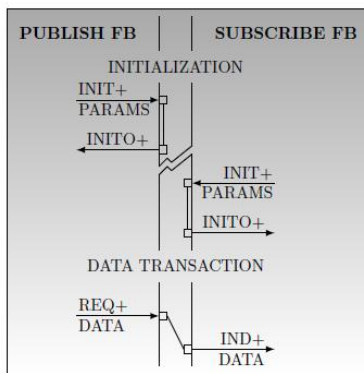


Рисунок 10.4. Диаграмма последовательности действий транзакции PUBLISH–SUBSCRIBE

На рисунке 10.4 показана диаграмма последовательности услуг транзакции PUBLISH–SUBSCRIBE. Процесс инициализации PUBLISH FB не связан с процессом инициализации SUBSCRIBE FB. Оба FB должны быть инициализированы, прежде чем может произойти какая-либо транзакция данных. После того как оба FB инициализированы, чтобы установить свои соединения с базовой сетью, они готовы обрабатывать столько транзакций данных, сколько необходимо. После получения события REQ + PUBLISH FB получает доступ к базовому сетевому ресурсу, который, в свою очередь, запускает SUBSCRIBE FB, чтобы выдать событие IND+ вместе с полученными данными. Переданные данные поступают на входы данных SD_1 – SD_n PUBLISH FB (рисунок 10.2) и становятся доступными на выходах данных RD_1 – RD_n SUBSCRIBE FB после завершения транзакции.

10.2.2.3. Двухнаправленный обмен данными

КЛИЕНТЫ используются для установления двухточечных соединений. КЛИЕНТЫ ожидают подтверждения. С другой стороны, СЕРВЕРЫ принимают соединения от источника и отвечают в соответствии с его запросами. КЛИЕНТ и СЕРВЕР используют подтверждения и чаще всего применяются для реализации наиболее детерминированной технологии master / slave, причем КЛИЕНТ и СЕРВЕР играют роли мастеров и подчиненных соответственно. В распределенных приложениях IEC 61499 КЛИЕНТ и СЕРВЕР обычно также сопряжены.

На рисунке 10.5 показана диаграмма последовательности обслуживания транзакции КЛИЕНТ–СЕРВЕР. В отличие от инициализации PUBLISH–SUBSCRIBE, пара КЛИЕНТ–СЕРВЕР чаще всего имеет определенную последовательность инициализации. Сначала необходимо инициализировать СЕРВЕР, устанавливая сервероподобное соединение с базовой сетью, а затем инициализировать КЛИЕНТА, устанавливая соединение с СЕРВЕРОМ через базовые сетевые ресурсы. После инициализации обоих FB они готовы обрабатывать столько транзакций данных, сколько необходимо. После получения события REQ+ КЛИЕНТ получает доступ к базовому сетевому ресурсу, отправляя запрос СЕРВЕРУ, который в свою очередь генерирует событие IND+ вместе с полученными данными. Операция еще не закончена. СЕРВЕР должен ответить с подтверждением, указывающим, что все данные были получены, или переносом запрошенных данных обратно в КЛИЕНТ. Это реализуется путем запуска СЕРВЕРА с событием RSP+, когда СЕРВЕР отправляет подтверждение обратно в КЛИЕНТ через базовые сетевые ресурсы. После получения подтверждающего сообщения КЛИЕНТ генерирует событие CNF+.

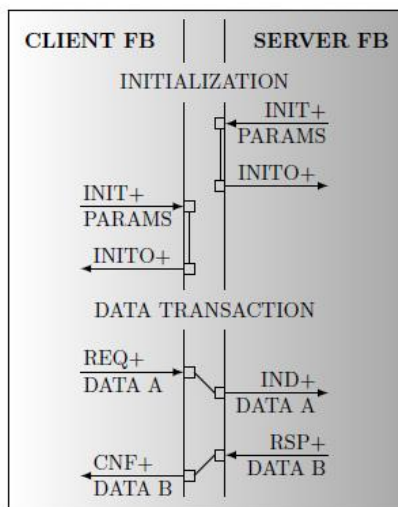


Рисунок 10.5. Диаграмма последовательности обслуживания транзакции КЛИЕНТ–СЕРВЕР

Обратите внимание, что как запрос, так и сообщение с подтверждением могут передавать данные по сети. Переданные данные, прикрепленные к сообщению запроса, поступают на входы данных SD_1 – SD_n КЛИЕНТА (рисунок 10.2) и становятся доступными на выходах данных RD_1 – RD_n СЕРВЕРА. Переданные данные, прикрепленные к сообщению подтверждения, поступают на входы данных SD_1 – SD_n СЕРВЕРА и становятся доступными на выходах данных RD_1 – RD_n КЛИЕНТА. Таким образом, если КЛИЕНТ запрашивает данные, эти данные будут доступны на его выходах RD_1 – RD_n, когда транзакция будет завершена.

10.2.2.4. Деинициализация CSIFB

Чтобы отключить соединение, установленное CSIFB с базовой сетью, CSIFB должен быть деинициализирован. На рисунке 10.6 показана диаграмма последовательности услуг завершения CSIFB. CSIFB запускается с событием INIT- (событие INIT с входом QI, установленным в FALSE). После успешного завершения CSIFB освобождает все системные ресурсы и генерирует событие INITO (событие INITO с выходом QO, установленным в FALSE). Это важно в некоторых случаях соединения КЛИЕНТ–СЕРВЕР, когда КЛИЕНТ должен отключить свое соединение от СЕРВЕРА, чтобы другой КЛИЕНТ мог к нему подключиться.

10.2.3. Профили соответствия

Разработка службы CSIFB зависит от среды разработки программного обеспечения, в которой разрабатываются блоки IEC 61499. Обычно среда предоставляет шаблоны и средства CSIFB для вставки алгоритмов, реализующих сетевой протокол, и определения последовательностей действий. Если шаблон представляет собой программу на Java, C++ или другом языке, при компиляции программы будет сгенерирован исполняемый код CSIFB, готовый для использования в качестве программной конструкции через графический интерфейс в любой прикладной программе.

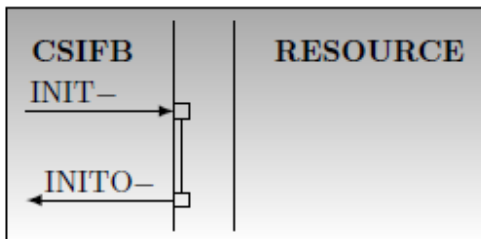


Рисунок 10.6. Диаграмма последовательности обслуживания CSIFB

Чтобы оставаться адаптивным к новым технологиям программирования и связи, IEC 61499 избегает явного определения того, как CSIFB используются для реализации конкретного протокола связи. Вместо этого стандарт определяет механизм расширения стандарта и определения пользовательских реализаций протоколов связи и других функций. Механизм называется профилем соответствия [17] и пытается достичь соответствия между различными платформами и средами программирования, разработанными разными поставщиками.

Поставщик, реализующий функцию, не определенную стандартом, должен опубликовать профиль соответствия, описывающий подробности его платформо-зависимых решений для связи или другой проблемы. Другие поставщики могут обратиться к этому профилю соответствия для разработки совместимых решений, чтобы устройства, платформы или среды программирования с различными внутренними компонентами могли отвечать требованиям совместимости и взаимодействовать друг с другом. Однако поставщики часто не выполняют эти инструкции и поэтому игнорируют распределенную природу стандарта IEC 61499, как описано в разделе 10.7. С другой стороны, сообщество open source продвигается по-своему, публикуя открытые профили соответствия, предлагающие решения некоторых проблем IEC 61499.

В следующем разделе приведен пример распределенного приложения. Этот пример используется в последующих разделах, чтобы описать, как CSIFB, которые управляют связью приложений, развертываются в сети FB.

10.3. Пример распределенного приложения

Управляемая событиями природа приложений IEC 61499 позволяет распространять их на несколько устройств. Каждый FB может быть назначен определенному ресурсу, находящемуся в определенном устройстве, в соответствии с возможностями ресурса и положением устройства внутри завода. На рисунке 10.7 показан системный вид приложения, распределенного на трех устройствах. Мы можем видеть каждое устройство, подключенное к сегменту сети, системному соединению и конкретному ресурсу, содержащемуся в каждом устройстве. MONITOR управляет датчиками и исполнительными механизмами, контролирующими физический процесс (например, нагрев печи), а CONTROL руководит управлением вышеупомянутого процесса (например, запуском алгоритма PID), тогда как SAFETY наблюдает за соблюдением требований безопасности (например, тревога в случае чрезвычайной ситуации). На рисунке 10.8 показана сеть FB распределенного приложения. Мы можем видеть каждый FB, назначенный определенному системному устройству.

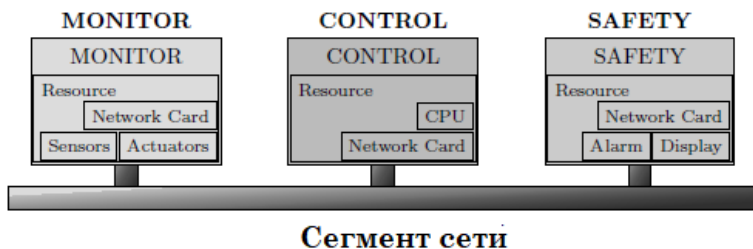


Рисунок 10.7. Распределенная система из трех устройств. MONITOR контролирует физический процесс и содержит ресурс, оснащенный датчиками и исполнительными механизмами. CONTROL осуществляет управление физическим процессом и содержит ресурс с программируемым процессором. SAFETY отвечает требованиям безопасности и содержит ресурс с системой сигнализации. Все ресурсы содержат сетевые карты, поэтому они могут физически подключаться к сети.

MONITOR содержит SENSOR и ACTUATOR (привод). SENSOR подключен к физическому датчику (например, тепловому датчику). При запуске с событием REQ он считывает текущее значение датчика и выдает его вместе с событием CNF. ACTUATOR подключен к физическому приводу (например, к нагревательному резистору). При запуске с событием REQ он устанавливает или изменяет значение привода (например, мощность нагревательного резистора) в соответствии с его входным значением VALUE. При запуске с событием OFF привод отключается, и FB выдает событие CNF вместе с файлом журнала, содержащим последние значения привода, из сообщений безопасности.

CONTROL содержит PID FB (идентификатор процесса), который управляет физическим процессом, отслеживаемым SENSOR-ом и ACTUATOR-ом. При запуске с событием REQ он считывает заданное значение датчика и запускает некоторый алгоритм (например, контур PID) для вычисления соответствующего значения привода, которое, в свою очередь, излучает вместе с событием CNF.

Значение датчика также передается на устройство SAFETY для проверки безопасности. Там CHECK FB при запуске с событием REQ проверяет, находится ли значение в пределах допустимых пределов безопасности. Если значение превышает эти пределы или значение не предоставляется в течение критического периода, FB генерирует событие ERROR.

Это событие запускает ALARM FB для включения красного предупреждения, а также запускает ACTUATOR FB устройства MONITOR для отключения физического привода. Если привод успешно отключен, ACTUATOR FB генерирует событие CNF вместе с файлом журнала, как описано выше. Событие CNF приводит к тому, что аварийный FB устройства SAFETY дает желтый предупреждающий сигнал и выводит полученный файл журнала на монитор, что завершает аварийное управление. Пунктирные линии на рисунке 10.8 обозначают переходы между устройствами. Связь между FB в разных устройствах осуществляется путем вставки в эти позиции стандартных CSIFB IEC 61499 (см. [25, с. 64–67]). В конце концов, FB в одном устройстве может запускать FB в других устройствах.

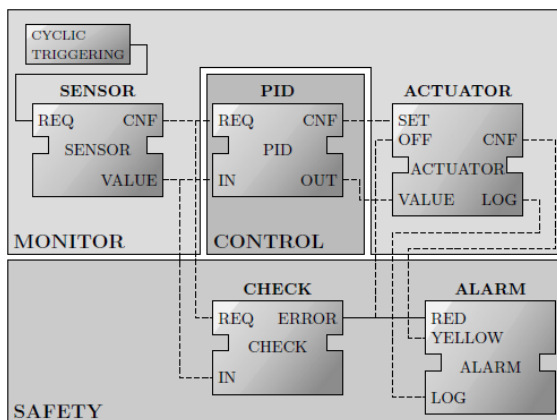


Рисунок 10.8. Приложение распространяется на три устройства. SENSOR и ACTUATOR назначены для MONITOR, PID назначен для CONTROL, а CHECK и ALARM назначены для SAFETY. Пунктирные линии обозначают соединения, которые проходят через границы устройств.

10.4. UDP/IP и протоколы связи TCP/IP в IEC 61499

Общедоступный Профиль соответствия IEC 61499 для технико-экономических обоснований, поддерживаемый Holobloc Inc. [10], пытается сопоставить однонаправленные и двунаправленные модели связи, описанные в IEC 61499, с протоколом пользовательских дейтаграмм [22] по интернет-протоколу [23] (UDP/IP) и протоколом управления передачей [24] через интернет-протокол [23] (TCP/IP). Реализации UDP/IP и TCP/IP в IEC 61499 представлены в следующих подразделах.

10.4.1. Протокол пользовательских дейтаграмм / протокол интернета

Краткое описание реализации UDP/IP в IEC 61499 приведено в [38]. UDP/IP используется для PUBLISH и SUBSCRIBE. Для связи друг с другом как FB, так и PUBLISH и SUBSCRIBE получают один и тот же параметризованный ввод данных ID:

multicastAddress:port

multicastAddress устанавливает многоадресный интернет-адрес, используемый FB PUBLISH / SUBSCRIBE для отправки и получения переданных данных.

port устанавливает номер интернет-порта, используемого для вышеупомянутой многоадресной передачи.

Например, ввод идентификатора PUBLISH и SUBSCRIBE, осуществляющих связь по многоадресному IP-адресу и номеру порта 230.0.0.1:4445, может быть представлен как «230.0.0.1:4445». Данный адрес многоадресной рассылки в Интернете и номер порта служат ключом, связывающим два CSIFB вместе, так что все данные, передаваемые FB PUBLISH, принимаются соответствующим FB SUBSCRIBE.

Может быть несколько PUBLISH и SUBSCRIBE с одним и тем же вводом данных ID. Любые данные, отправленные с использованием определенного идентификатора любым PUBLISH FB, будут получены любым SUBSCRIBE FB с использованием того же идентификатора. UDP/IP является ненадежным протоколом с учетом отраслевых требований, поскольку в нем отсутствуют функции квитирования и подтверждения связи, что приводит к потере пакетов. Если он должен использоваться в промышленном приложении, программист должен интегрировать решения уровня приложения. Однако UDP/IP – это готовый к реализации протокол, который обычно предпочтительнее TCP/IP в простых приложениях с низкими требованиями к надежности и высокими требованиями к синхронизации.

10.4.2. Протокол управления передачей / протокол интернета

Краткое описание реализации TCP/IP в IEC 61499 приведено в [38]. TCP/IP используется для пары КЛИЕНТ и СЕРВЕР. СЕРВЕР получает следующий идентификатор ввода данных:

internetAddress:portNumber

internetAddress, установленный на IP-адрес СЕРВЕРА TCP/IP или «localhost» или «127.0.0.1», или имя устройства в сети

portNumber устанавливает порт прослушивания СЕРВЕРА TCP/IP

Например, СЕРВЕР, работающий на устройстве с именем m51568 с IP-адресом «161.153.19.227», может иметь входной идентификатор, представленный как «m51568: 1499», «161.153.19.227:1499», «localhost: 1499» или «127.0.0.1 : 1499». Соответствующий КЛИЕНТ имеет следующий идентификатор ввода данных:

internetAddress:portNumber

internetAddress, установленный на IP-адрес или имя устройства СЕРВЕРА FB

portNumber, установленный на номер порта прослушивания, открытый СЕРВЕРОМ

КЛИЕНТ, обращающийся к СЕРВЕРУ в вышеприведенном примере, может иметь ввод идентификатора, представленный как «m51568: 1499» или «161.153.19.227:1499». Имея эту информацию, КЛИЕНТ может начать обмен данными, подключившись к СЕРВЕРУ.

Очевидно, что КЛИЕНТ и СЕРВЕР работают парами, устанавливая надежную связь друг с другом. Это не означает, что время получения любого сообщения гарантировано, но сообщение будет отправлено повторно до получения. Это повышает надежность транзакции, но не дает никаких гарантий относительно времени отклика. Использование TCP/IP ограничено из-за этой функции.

10.4.3. Пример приложения UDP/IP и реализации TCP/IP

Распределенное приложение, показанное на рисунке 10.8, будет использоваться в качестве примера применения протоколов UDP/IP и TCP/IP в приложении IEC 61499. Как объяснено ранее, пример относится к приложению, которое контролирует и защищает физический процесс. Три устройства – MONITOR, CONTROL и SAFETY – соответственно выполняют эти функции. В системе есть два сетевых события и потоки данных. Одно управляет связью между устройствами MONITOR и CONTROL, а другое – защитным обменом. Чтобы лучше контролировать физическую процедуру, SENSOR FB должен часто запускаться для получения огромного числа значений; поэтому простой протокол UDP/IP будет использоваться для

обработки связи MONITOR–CONTROL, что позволит сэкономить пропускную способность сети.

С другой стороны, устройство SAFETY должно быть способно надежно отключать ACTUATOR (ПРИВОД) MONITOR-а, когда это необходимо. Более того, это не частая транзакция, она осуществляется только в случае ошибки. Таким образом, более надежный протокол TCP/IP будет использоваться для управления связью SAFETY–MONITOR. Связь CONTROL–SAFETY также будет обрабатываться соединением UDP/IP.

Полное приложение вместе со вставленными CSIFB, которые обрабатывают связь системы, можно увидеть на рисунке 10.9. Процесс инициализации CSIFB опущен, но подробное описание процедуры можно найти в разделе 10.2.2. Обратите внимание, что каждый набор CSIFB имеет свой собственный ключ ID. Также обратите внимание, что функция многоадресной рассылки протокола UDP/IP используется для отправки данных датчика на устройства CONTROL и SAFETY. Системные транзакции заключаются в следующем. Сначала запускается SENSOR FB, вызывая PUBLISH_A FB для публикации данных датчика в сети. И SUBSCRIBE_A, и SUBSCRIBE_A2 могут принимать значения датчика. Использование разных ключей ID позволяет CSIFB различать релевантные и неактуальные пакеты UDP в сети. SUBSCRIBE_B FB не будет принимать данные, опубликованные PUBLISH_A FB. Когда SUBSCRIBE_A FB получает значение датчика, он вызывает PID FB, чтобы рассчитать соответствующее значение для ACTUATOR-а. Когда PID FB имеет готовое значение, он вызывает PUBLISH_B FB, чтобы опубликовать значение привода в сети. Только SUBSCRIBE_B FB получит значение привода, опубликованное PUBLISH_B, и, в свою очередь, вызовет ACTUATOR, чтобы установить соответствующее значение для физического привода.

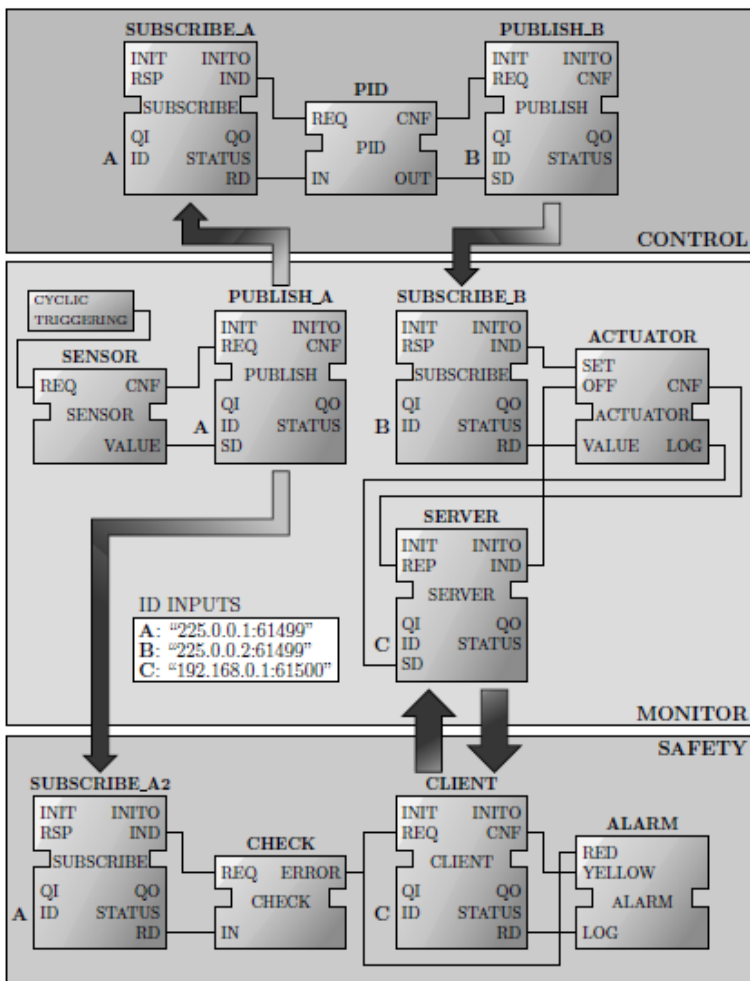


Рисунок 10.9. Развертывание CSIFB в приложении, показанном на рисунке 10.8. PUBLISH и SUBSCRIBE используют протокол UDP/IP, а КЛИЕНТ и СЕРВЕР используют протокол TCP/IP

Как обсуждалось выше, протокол UDP/IP страдает от потери пакетов, но в нашем случае это не проблема, поскольку потерянное значение датчика будет без ущерба заменено последующим значением, отправленным PUBLISH_A FB, при условии, что SENSOR FB и, следовательно, PUBLISH_A FB запускаются достаточно часто.

Транзакции безопасности запускаются, когда SUBSCRIBE_A2 FB получает значение датчика от PUBLISH_A FB. Затем он вызывает CHECK FB, чтобы проверить, находится ли значение датчика в допустимых пределах. При

обнаружении ошибки CHECK FB запускает ALARM FB (для выдачи красного предупреждения) и КЛИЕНТА. КЛИЕНТ просит устройство MONITOR отключить привод. Для этого он отправляет запрос СЕРВЕРУ, который, в свою очередь, обрабатывает запрос, вызывая ACTUATOR FB и фактически выключая привод. ACTUATOR отправляет файл журнала обратно на СЕРВЕР, а СЕРВЕР отвечает КЛИЕНТУ файлом журнала. КЛИЕНТ получает подтверждение вместе с запрошенными данными, тем самым завершая транзакцию КЛИЕНТ–СЕРВЕР. Наконец, КЛИЕНТ запускает ALARM (аварийный сигнал) FB, чтобы сделать желтое предупреждение, поскольку система приостановлена, но не находится в опасности. Файл журнала распечатывается на дисплее устройства SAFETY.

Процедура выключения привода обрабатывается более надежным соединением КЛИЕНТ–СЕРВЕР, поскольку оно не может потерять пакет. У него есть только один критически важный пакет для доставки, а протокол TCP/IP имеет соответствующие механизмы, обеспечивающие доставку пакета. Более того, соединение КЛИЕНТ–СЕРВЕР идеально подходит для характера транзакции запрос-ответ, поскольку устройство SAFETY запрашивает устройство MONITOR, чтобы отключить привод, а также отправить обратно файл журнала. Опубликованный в открытом доступе профиль соответствия IEC 61499 для технико-экономических обоснований [10], обеспечивающий реализации UDP/IP и TCP/IP в IEC 61499, предложил простое и удобное решение для обработки сообщений в IEC 61499. Однако эти протоколы не всегда обеспечивают решения для крупномасштабных распределенных систем, которые требуют более сложного центрального контроля связи. Следующая глава обсуждает и представляет такую систему, а затем представляет решение проблемы связи, основанное сначала на UDP/IP, а затем на протоколе Modbus.

10.5. Пример системы SCADA

Большинство распределенных автоматизированных систем управления определенного размера требуют централизованного управления всеми взаимодействующими подсистемами. Специальная система диспетчерского контроля и сбора данных (SCADA) установлена над всеми подсистемами для сбора и анализа данных в режиме реального времени с целью мониторинга и управления устройствами в различных подсистемах. Типичной операцией системы SCADA является параметризация удаленных устройств, сбор данных для диаграмм HMI и управление центральной системой связи. Обозначения IEC 61499 подходят для описания такой системы. На рисунке 10.10 показан системный вид простой системы автоматизации, включающей блок SCADA. Каждое устройство MONITOR_n контролирует физический процесс (например, движение конвейерной ленты) с помощью датчика. Каждое устройство CONTROL_n имеет возможность управлять физическим процессом, отслеживаемым соответствующим устройством MONITOR_n,

путем запуска алгоритма управления и предоставления результирующих данных в привод, содержащийся в каждом устройстве CONTROL_n. SCADA используется для сбора всех данных датчика в файл журнала для целей HMI. На рисунке 10.11 показана действенная сеть приложения FB и назначение каждого FB соответствующему системному устройству.

Каждое устройство MONITOR_n содержит FB типа SENSOR. Каждый SENSOR_n FB связан с физическим датчиком (например, спидометром). При запуске с событием REQ каждый SENSOR_n FB считывает текущее значение датчика и выдает его вместе с событием CNF.

Каждое устройство CONTROL_n содержит FB типа PID и FB типа ACTUATOR, которые управляют физическим процессом, контролируемым соответствующим SENSOR_n FB. При запуске с событием REQ каждый FB PID_n считывает данное значение датчика и запускает алгоритм (например, цикл PID) для вычисления соответствующего значения исполнительного механизма, отправляя его вместе с событием CNF в соответствующий ACTUATOR_n FB. Каждый ACTUATOR_n FB связан с физическим приводом (например, двигателем конвейера). При запуске с событием REQ он устанавливает значение исполнительного механизма (например, электрическую мощность двигателя) равным значению, записанному на входе данных VALUE.

Каждое значение датчика, генерируемое соответствующим SENSOR_n FB, также предоставляется соответствующему LOGGER_n FB, содержащемуся в устройстве SCADA, которое будет добавлено в файл журнала.

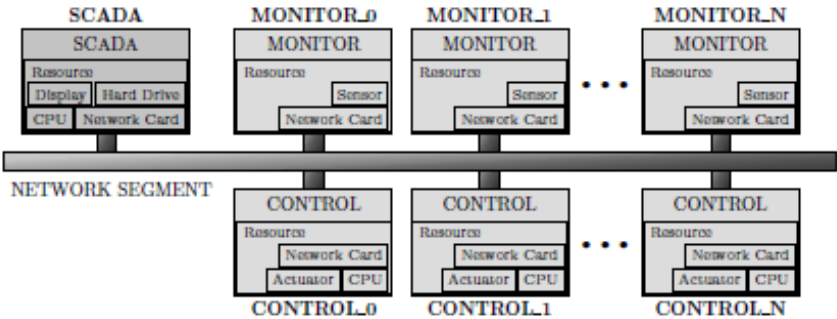


Рисунок 10.10. Распределенная система нескольких контрольных пар, контролируемых устройством SCADA. Каждое устройство подключено к одному сегменту сети, таким образом, достигается системное соединение. Каждое устройство содержит определенный ресурс, предоставляющий средства для управления работой устройства.

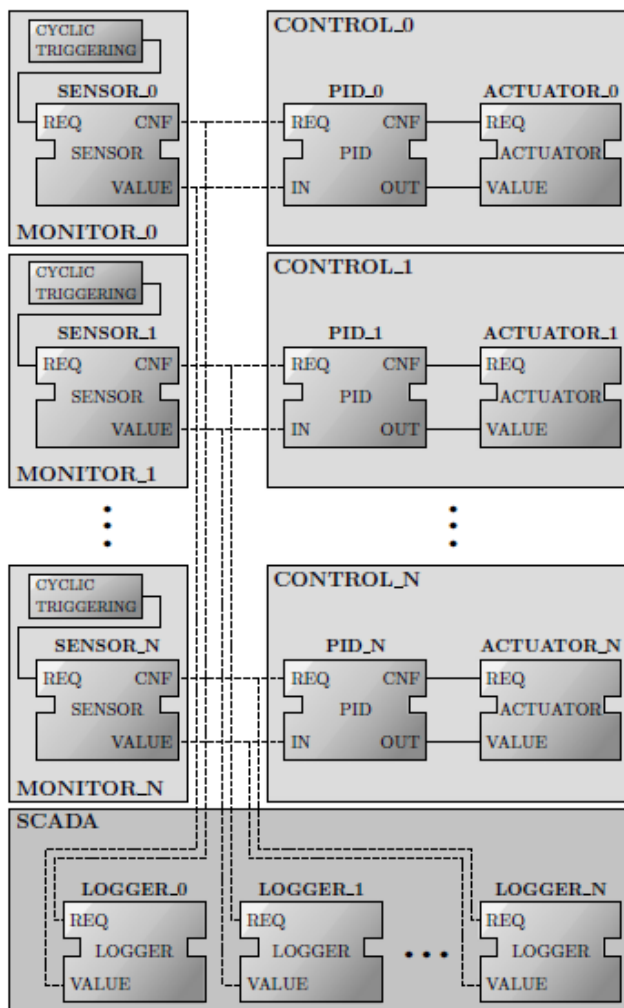


Рисунок 10.11. Приложение распространяется на несколько устройств вместе с устройством SCADA. $SENSOR\_n$ FB назначены соответствующим устройствам $MONITOR\_n$. PID_n и $ACTUATOR\_n$ назначены соответствующим устройствам $CONTROL\_n$, в то время как все $LOGGER\_n$ FB назначены устройству SCADA. Пунктирные линии обозначают соединения, которые проходят через границы устройств.

10.5.1. Пример применения протокола UDP/IP в системе SCADA

Применение протокола UDP/IP в примере системы SCADA на рисунке 10.11 выглядит довольно просто. PUBLISH FB вводится после каждого SENSOR_n FB для широковещательной передачи соответствующего значения датчика в сеть, тогда как SUBSCRIBE FB вводится перед каждым PID_n FB для сбора соответствующего значения датчика из сети. Другой SUBSCRIBE FB вводится перед каждым LOGGER_n FB, чтобы также собирать соответствующие данные датчика из сети.

Полное приложение вместе с введенными CSIFB, которые управляют связью системы, можно увидеть на рисунке 10.12. Системные транзакции реализуются следующим образом: сначала каждый SENSOR_n FB циклически запускается, вызывая соответствующий PUBLISH_n FB для публикации соответствующих данных датчика в сети. Оба соответствующих SUBSCRIBE_n и SUBSCRIBE_nb могут принимать значение датчика. Использование разных ключей ID позволяет CSIFB различать релевантные и неактуальные пакеты UDP в сети. Когда каждый SUBSCRIBE_n FB принимает соответствующее значение датчика, он вызывает соответствующий PID_n FB, чтобы вычислить соответствующее значение для соответствующего ACTUATOR_n FB, который, в свою очередь, соответствующим образом устанавливает соответствующий физический привод, таким образом реализуя соответствующий цикл управления. Каждый SUBSCRIBE_nb FB получает соответствующие данные датчика и вызывает соответствующий LOGGER_n FB, чтобы добавить данные в объединенный файл журнала.

Чтобы лучше контролировать физическую процедуру, SENSOR_n FB должны часто запускаться, создавая огромное количество значений. Протокол UDP/IP использует простую систему обмена сообщениями с низкой пропускной способностью, но иногда он не способен обрабатывать эту связь, так как сеть переполняется передаваемыми данными, особенно когда число пар управления значительно увеличивается. Попытка решить эту проблему путем снижения частоты циклического срабатывания датчиков может снизить эффективность контура управления. Центральная система мониторинга необходима для использования доступной пропускной способности сети без переполнения системы.

В следующем разделе представлены наиболее часто используемые SCADA-приложения и протокол связи Modbus. Данный же выше пример приложения далее используется для демонстрации того, как CSIFB развертываются в сети FB для обработки его требований к связи с использованием протокола Modbus.

10.6. Протокол связи Modbus в IEC 61499

Modbus является одним из старейших, самых простых и наиболее распространенных протоколов связи в промышленной автоматизации ([3, 12,

36], [4, с. 508-510)). Его спецификации были открыто опубликованы в [28, 27, 29] и благодаря своей простоте, надежности и эффективности де-факто стали стандартом.

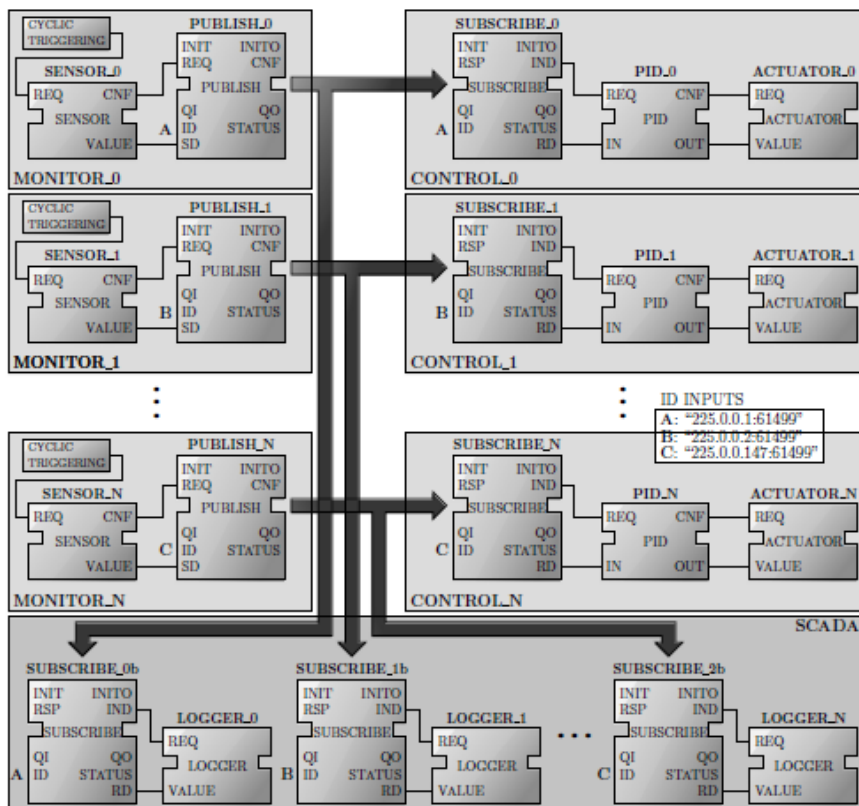


Рисунок 10.12. Развертывание CSIFB в приложении, показанном на рисунке 10.11, для обработки связи с использованием протокола UDP/IP

Протокол Modbus основан на главной / подчиненной сети. В каждой сети Modbus может существовать только один ведущий, но можно подключить несколько ведомых. Каждый ведомый запускает серверную службу и ожидает запросов. Ведущий отправляет запросы на чтение данных и/или запись данных во внутреннюю память ведомого устройства. Только ведущее устройство может инициировать транзакцию, а ведомые отправляют данные в сеть только по требованию главного устройства. Таким образом, вся связь обрабатывается одним ведущим устройством.

10.6.1. Реализация протокола Modbus в IEC 61499

Простая реализация протокола Modbus в соответствии с IEC 61499 представлена в [38, с. 508-510]. Предлагается использовать КЛИЕНТ для ведущего устройства Modbus и СЕРВЕРА для ведомого устройства Modbus. Но здесь представлена новая реализация в пользу абстрактной коммуникационной модели IEC 61499. Применяется более гибкий и простой в использовании подход КЛИЕНТ–PUBLISH/SUBSCRIBE. Ведущий реализован с использованием КЛИЕНТА, а обмен данными в подчиненном устройстве обрабатывается PUBLISH и SUBSCRIBE.

На рисунке 10.13 показаны три разных транзакции данных реализации. На рисунке 10.13 (а) показана операция публикации подчиненного устройства. Как упоминалось ранее, ведомое устройство запускает службу сервера Modbus. Вместо того, чтобы отправлять данные по сети, запущенный PUBLISH FB записывает данные в специальную таблицу данных Modbus, делая их доступными для службы сервера Modbus, без участия каких-либо сетевых операций.

На рисунке 10.13 (b) показана операция чтения главного устройства. При правильном запуске, КЛИЕНТ на главном устройстве отправит запрос на чтение в службу сервера Modbus на ведомом устройстве, запрашивая данные. Имея данные, доступные через операцию публикации, описанную выше, служба сервера Modbus отвечает обратно на ведущее устройство, отправляющее запрошенные данные, тем самым реализуя операцию чтения.

На рисунке 10.13 (c) показана операция записи главного устройства. При правильном запуске, КЛИЕНТ на главном устройстве отправит запрос на запись в службу сервера Modbus на ведомом устройстве. Служба сервера Modbus отвечает, записывая данные в таблицу данных Modbus, в то же время вызывая любые SUBSCRIBE FB, связанные с этими данными, и отправляет подтверждение на главное устройство.

Обратите внимание, что смешанный запрос на чтение и запись также может быть обработан реализацией путем объединения обеих операций. Также обратите внимание, что несколько PUBLISH и SUBSCRIBE могут быть связаны с одной службой Modbus Sever и функционировать одновременно на стороне ведомого устройства, в то время как несколько КЛИЕНТОВ могут существовать и функционировать одновременно на стороне главного устройства.

10.6.2. Конфигурирование CSIFB для использования протокола Modbus

Modbus поставляется с тремя вариантами: TCP, RTU и ASCII. Вариант TCP работает поверх сети Ethernet TCP/IP, а варианты RTU и ASCII работают поверх последовательной линии (RS232 или RS485). Обозначение требуется для различения CSIFB, использующих разные варианты Modbus или разные

протоколы в целом. В [8] предлагается использовать одни и те же CSIFB (см. рисунок 10.2) для всех протоколов и изменить ввод данных пользовательских ID, чтобы установить использование предпочтительного протокола. Этот подход используется здесь и расширен для решения проблемы различия вариантов Modbus. Строка передается на ввод данных идентификатора CSIFB, чтобы правильно настроить ее для обработки транзакций Modbus в соответствии с определенным вариантом. Чтобы сконфигурировать КЛИЕНТА для обработки основных функций Modbus, строка имеет следующий формат:

modbus[variant:slaveAddress:(function:readAdds:sendAdds):(params)]

modbus настраивает КЛИЕНТА FB для использования протокола Modbus
variant устанавливает вариант Modbus; допустимыми значениями являются tcp, rtu и ascii

slaveAddress устанавливает адрес подчиненного устройства, которому адресован запрос; для варианта TCP значение представляет собой IP: адрес порта подчиненного устройства в сети TCP/IP, а для вариантов RTU и ASCII значение представляет собой число от 1 до 247, что соответствует последовательному адресу подчиненного устройства, или 0 для режима ширококвещательной передачи

function устанавливает код функции Modbus, который будет использоваться; он является необязательным и используется только для обратной совместимости с устройствами, не использующими IEC 61499

readAdds перечисляет адреса таблицы данных ведомого устройства, которые должны быть прочитаны; используется только для обратной совместимости

sendAdds перечисляет адреса таблицы данных ведомого устройства, которые должны быть перезаписаны; используется только для обратной совместимости

params устанавливает параметры протокола и варианта, что необязательно; значения по умолчанию используются, когда значения опущены.

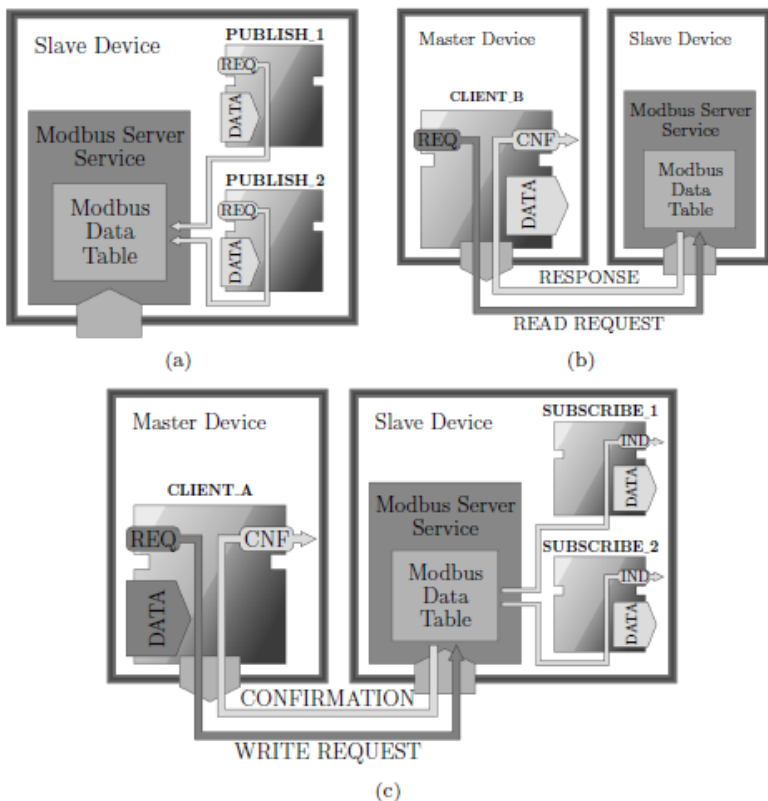


Рисунок 10.13. Три операции транзакции данных в реализации Modbus: (a) PUBLISH FB на ведомой стороне Modbus записывает некоторые данные в таблицу данных Modbus службы сервера Modbus. (b) КЛИЕНТ на ведущей стороне Modbus отправляет запрос на чтение службе сервера Modbus на ведомой стороне Modbus. (c) КЛИЕНТ на ведущей стороне Modbus отправляет запрос на запись в службу сервера Modbus на ведомой стороне Modbus.

PUBLISH / SUBSCRIBE FB сконфигурированы для работы с подчиненными функциями Modbus с использованием следующего ввода данных ID:

modbus[variant:slaveAddress:(dataType:dataAdds):(params)]

modbus устанавливает PUBLISH / SUBSCRIBE FB для использования протокола Modbus

variant устанавливает вариант Modbus; допустимыми значениями являются TCP, RTU и ASCII

slaveAddress устанавливает адрес устройства; для варианта TCP значением является IP: адрес порта устройства в сети TCP/IP, а для вариантов RTU и ASCII значением является число от 1 до 247, соответствующее последовательному адресу устройства

dataType устанавливает соответствующий тип данных Modbus для записи / прослушивания; используется только для обратной совместимости

dataAddrs устанавливает соответствующие адреса данных Modbus для записи / прослушивания; используются только для обратной совместимости

params устанавливает параметры протокола и варианта, что необязательно; значения по умолчанию используются, когда значения опущены

Все необязательные опции и параметры обратной совместимости играют важную роль в настройке сети связи Modbus, но здесь для краткости мы не стали углубляться в детали.

10.6.3. Пример приложения реализации Modbus

Распределенное приложение, показанное на рисунке 10.11, будет использоваться в качестве примера применения протокола Modbus в приложении IEC 61499 для установки механизма контроля связи системы. Как объяснялось выше, пример относится к системе, состоящей из нескольких пар управления и устройства SCADA.

Полное приложение вместе с введенными CSIFB, которые управляют связью системы с использованием протокола Modbus, можно увидеть на рисунке 10.14. Обратите внимание, что пары MONITOR_n – CONTROL_n остаются неизменными, при этом изменяются только входы идентификаторов пар PUBLISH_n – SUBSCRIBE_n, так как они настроены на использование протокола Modbus. С другой стороны, устройство SCADA значительно модифицировано, чтобы играть руководящую роль в схеме связи.

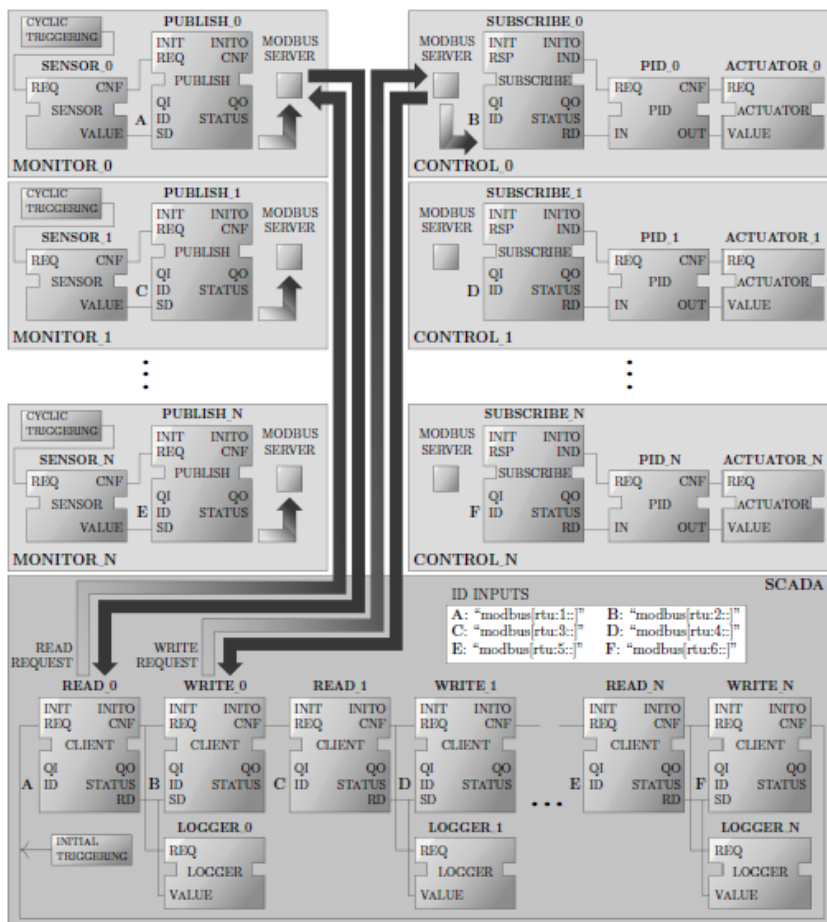


Рисунок 10.14. Развертывание CSIFB в приложении, показанном на рис. 10.11, для настройки контролирующего управления связью с использованием протокола Modbus

Все SENSOR_n FB запускаются одновременно и циклически, вызывая соответствующие PUBLISH_n FB для записи соответствующих значений датчика в соответствующие внутренние таблицы данных Modbus серверов Modbus. Таким образом, устройства MONITOR_n не имеют реального доступа к сети. Все транзакции выполняются устройством SCADA следующим образом. Сначала READ_0 FB запускается с начальным событием – маленькая стрелка на рисунке 10.14 соответствует логическому OR (или) между строками событий, приходящими из поля INITIAL TRIGGERING и WRITE_N FB. READ_0 FB получает данные датчика от устройства MONITOR_0 (ранее сохраненные на сервере Modbus с помощью PUBLISH_0 FB). Когда

транзакция завершена, данные датчика добавляются в файл журнала блоком `LOGGER_0` и записываются на сервер Modbus на устройстве `CONTROL_0` блоком `WRITE_0`. Когда запись завершена, запускается `PID_0` FB, чтобы вычислить соответствующее значение для `ACTUATOR_0` FB, и запускается `READ_1` FB, чтобы выполнить следующую транзакцию, связанную с парой `MONITOR_1 – CONTROL_1`. На рисунке 10.14 показаны только транзакции, связанные с первой парой управления. Все последующие транзакции выполняются одинаково до тех пор, пока не будет подана пара `MONITOR_n – CONTROL_n`, что завершает цикл связи и запускает блок `READ_0` FB, чтобы начать новый цикл связи.

Выполнение этих коммуникационных циклов один за другим позволяет использовать всю пропускную способность сети без проблем с перегрузкой или переполнением, поскольку отдельные устройства не создают помехи для системы связи по собственной инициативе. Аналогичным образом CSIFB, использующие протокол Modbus, могут быть сконфигурированы для применения операций SCADA в крупномасштабных распределенных системах, содержащих несколько удаленных устройств и подсистем.

Реализации UDP/IP, TCP/IP и Modbus в IEC 61499 были широко приняты научным сообществом, поскольку они предоставляют простые и готовые к использованию решения в рамках IEC 61499. По мере того, как исследователи и поставщики стали более активно использовать стандарт IEC 61499, они начали предлагать различные решения и реализации стандарта открытых коммуникационных проблем IEC 61499.

10.7. Реализации других протоколов связи в IEC 61499

Стандарт IEC 61499 был сформирован с использованием стандарта полевой шины IEC 61158 [19], и многие концепции IEC 61158 были приняты во внимание при его разработке (см. [25]). Вот почему связь и распространение стали очень важными аспектами стандарта IEC 61499. Предполагалось, что приложения будут распределяться по нескольким устройствам, а замена устройств будет простой, независимой от поставщика задачей. Несколько протоколов связи полевой шины были реализованы в IEC 61499, но, поскольку промышленность обратилась к связи на основе Ethernet [30], последовало исследование IEC 61499. В настоящее время средства разработки IEC 61499 используют различные реализации IEC 61499 протоколов связи на основе Fieldbus и Ethernet, но, насколько известно авторам, только несколько профилей соответствия [17] были опубликованы. Реализации UDP/IP и TCP/IP были опубликованы в [10] и широко использовались в проектах (4DIAC [1] и FBDK [9]), а реализация общепромышленного протокола (CIP) [32] была опубликована в [39], но, возможно, она уже устарела. Была также предпринята попытка интегрировать связь стандарта IEC 61850 в энергетической отрасли [15] в стандарт IEC 61499 в качестве своего коммуникационного направления [7].

10.7.1. Коммерческие инструменты IEC 61499

10.7.1.1. ISaGRAF

ISaGRAF [14] – инструмент промышленного развития, в настоящее время продвигаемый ICS Triplex ISaGRAF. Сочетая как стандарты IEC 61131, так и IEC 61499, но сосредоточившись на первом, ISaGRAF менее эффективен при создании распределенных приложений IEC 61499. Платформа оценки ISaGRAF включает три сегмента сети: ETCP для TCP/IP, ISARSI для каналов RS232 и HSD (драйвер хост-системы) для обмена данными между ресурсами на одном устройстве. ISaGRAF не предоставляет представление ресурсов распределенных приложений и автоматически обрабатывает связь. Пользователь не предоставляется и не должен использовать CSIFB. После добавления определенного сегмента сети в представление развертывания системы ISaGRAF автоматически добавляет соответствующий код связи для каждого события и линии подключения к данным, которая пересекает границы устройства (более подробное представление см. в [37]). Это не является отступлением от правил IEC 61499, так как стандарт предлагает, чтобы инженерные инструменты могли автоматически вводить CSIFB в распределенные приложения. Проблема заключается в том, что не существует общедоступного профиля соответствия для реализации этих сообщений ICS Triplex ISaGRAF (профиль соответствия упоминается только в [13]), поэтому ни один другой поставщик или исследователь не может разрабатывать совместимые решения или расширения.

10.7.1.2. nxtSTUDIO

nxtSTUDIO [31] – инструмент для промышленного проектирования, принадлежащий австрийской компании nxtControl GmbH. Он в основном сфокусирован на стандарте IEC 61499, но также поддерживает языки стандарта IEC 61131. Это демо-версия поставляется вместе со многими типами сегментов сети, такими как Ethernet, ControlNet, DeviceNet и Profibus. nxtControl создал несколько библиотек CSIFB для реализации этих протоколов связи. В отличие от ISaGRAF, nxtSTUDIO предлагает отдельное представление распределенных приложений по ресурсам, но аналогичным образом предоставляет автоматический способ вставки необходимых CSIFB, которые обрабатывают связь. Хотя nxtSTUDIO поддерживает профиль соответствия IEC 61499 для технико-экономических обоснований [10], опубликованных профилей соответствия всех других реализаций не существует, что не позволяет другим поставщикам или исследователям тестировать или расширять решения, предлагаемые nxtControl.

10.7.2. Открытые инструменты IEC 61499

В качестве альтернативы недавно появился инструмент разработки с открытым исходным кодом для приложений IEC 61499, названный 4DIAC [1] (Фреймворк для распределенной промышленной автоматизации и управления), дополняющий FBDK Holobloc [9] (Kit для разработки функциональных блоков) в качестве первой исследовательской платформы IEC 61499. 4DIAC был недавно принят в качестве официального проекта Eclipse [5], в то время как Holobloc недавно опубликовал вторую версию FBDK [11].

Оба проекта основаны на собственной системе исполнения, FORTE и FBRT для 4DIAC и FBDK соответственно, но могут работать совместно. FBRT и FORTE также используют одни и те же принципы в отношении реализации различных протоколов связи. Открытый характер обоих проектов привлек внимание исследовательского сообщества, в частности, в ряде публикаций, касающихся внедрения коммуникаций в IEC 61499.

Как уже упоминалось, в [8] предлагается использовать одни и те же CSIFB для всех протоколов и изменить ввод данных идентификатора пользователя, чтобы установить использование предпочтительного протокола, что даст общее решение проблемы различия CSIFB между протоколами. Кроме того, как уже говорилось, профиль соответствия IEC 61499 для технико-экономических обоснований [10] представил реализации протоколов UDP и TCP в IEC 61499, в то время как реализация протокола CIP была опубликована в IEC 61499, в профиле соответствия для функциональных блоков связи на основе CIP [39]. Описание реализации CIP также дано в [40], где отмечается, что его мультимастерная природа делает его наиболее подходящим для распределенной природы IEC 61499, в то время как он предлагает режимы для поддержки циклического, последовательного и параллельного исполнения моделей IEC 61499.

10.8. Пример программирования

Пример, представленный в этом разделе, демонстрирует, как можно запрограммировать систему управления экзоскелета коленного сустава, используемого в задачах реабилитации, с использованием инструмента разработки с открытым исходным кодом 4DIAC [1].

Механизм экзоскелета соответствует ноге пользователя, к которой экзоскелет прикреплен с помощью ремней. Механизм приводится в действие исполнительным механизмом, и на него воздействует усилие человека, создаваемое мышцами нижней конечности, воздействующими на коленный сустав. Предполагается, что пользователь находится в положении сидя и выполняет движения сгибания и разгибания коленного сустава без учета контакта с землей. Крутящий момент, создаваемый экзоскелетом, регулируется для отслеживания желаемой траектории, определенной с учетом

намерений пользователя или в соответствии с определенными упражнениями. Носимое устройство оснащено датчиками, которые определяют приложенный крутящий момент пользователя, а также угол и угловую скорость коленного сустава. Значение крутящего момента используется для оценки желаемой траектории, в то время как угловые данные предоставляются контроллеру в другом устройстве для вычисления мощности помощи, которая должна быть предоставлена механизмом экзоскелета для колена, чтобы достичь желаемого угла. Третье устройство используется в качестве панели для вставки данных для задач, которые должны быть применены к коленному суставу или в качестве альтернативы для выбора автоматического управления, при которой экзоскелет выполняет команду пользователя. Представление конфигурации системы 4DIAC приложения показано на рисунке 10.15.

Каждому устройству сети присваивается IP-адрес, и он загружается ресурсом EMB_RES, способным хранить приложение IEC 61499. Устройства подключены через сеть Интернет, и части приложений, которые находятся на каждом устройстве, обмениваются данными с PUBLISH-SUBSCRIBE CSIFB, используя протокол UDP/IP.

Представление приложения показано на рисунке 10.16. Как предложено стандартом IEC 61499, представление приложения не включает CSIFB, используемые приложением для обмена данными. CSIFB появляются только в отдельном представлении ресурсов распределенного приложения, показанном на рисунках 10.17, 10.18 и 10.19. Там можно увидеть PUBLISH и SUBSCRIBE FB, которые обрабатывают связь, а также специальный START FB, который инициирует приложение в каждом ресурсе. Таким образом, алгоритмическая часть приложения отделена от прикладной коммуникации. Более того, IEC 61499 предлагает автоматически вводить CSIFB для применения определенного протокола связи, что ожидается в следующих версиях 4DIAC.

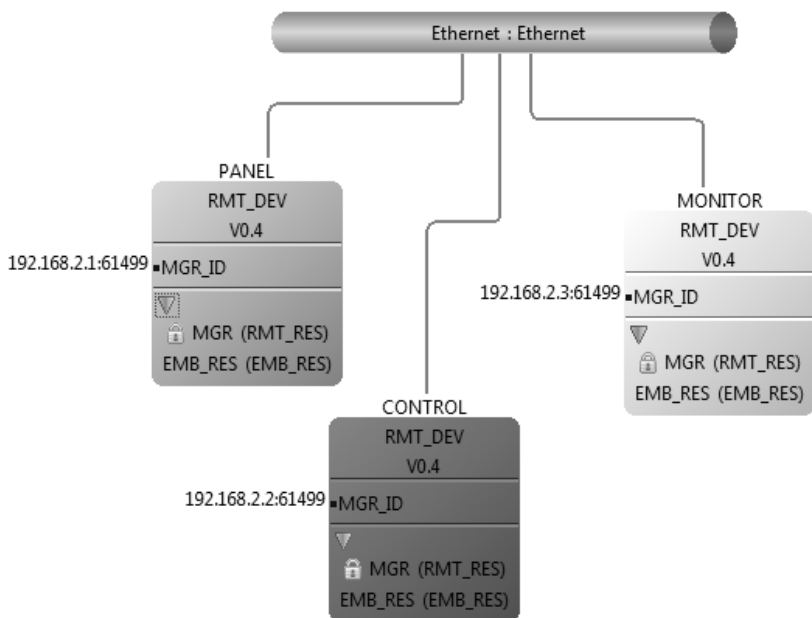


Рисунок 10.15. Конфигурация системы 4DIAC приложения распределенного управления механизмом экзоскелета

Алгоритмическая и коммуникационная работа приложения осуществляется следующим образом. Устройство MONITOR активирует привод, который заставляет механизм экзоскелета прикладывать крутящий момент, необходимый для добавления механизма экзоскелета к усилиям пользователя. Он также считывает измерения датчика приложенного крутящего момента, угла и угловой скорости коленного сустава. Угловые данные публикуются в устройстве CONTROL, которое рассчитывает соответствующий крутящий момент, который должен применять механизм экзоскелета. Это значение крутящего момента отправляется на устройство MONITOR. Прилагаемый крутящий момент пользователя вместе с угловыми данными публикуется на устройстве PANEL, которое оценивает желаемый угол и угловую скорость для коленного сустава. Расчетные значения публикуются на устройстве CONTROL для использования при расчете вспомогательного крутящего момента экзоскелета. Обратите внимание, что угловые данные публикуются как на устройстве CONTROL, так и на устройстве PANEL с использованием многоадресной передачи UDP/IP. Устройство PANEL также используется для выбора ручного режима работы, когда требуемые угловые данные не оцениваются и вынуждены решать определенные задачи конечности.

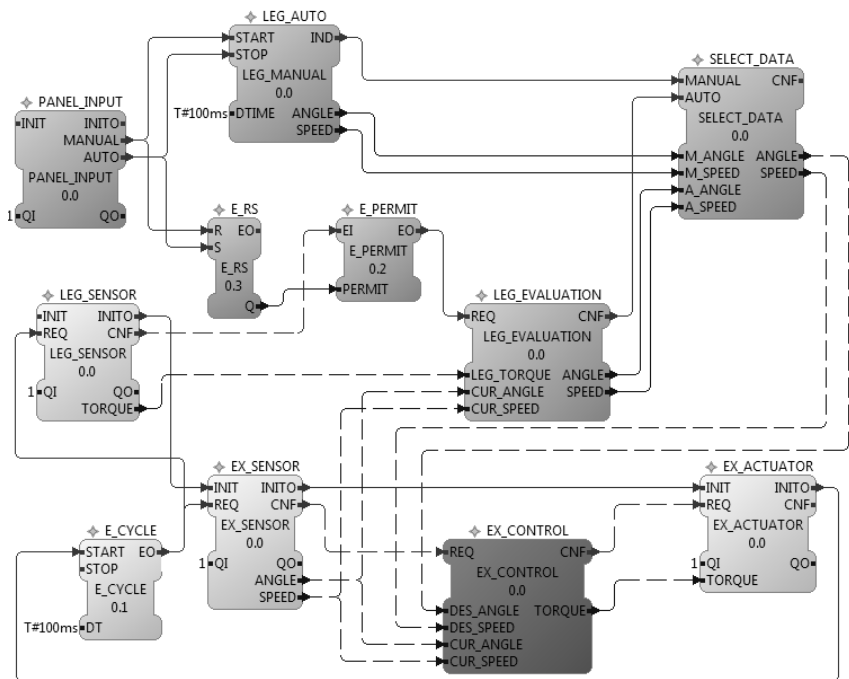


Рисунок 10.16. Приложение 4DIAC: вид программы

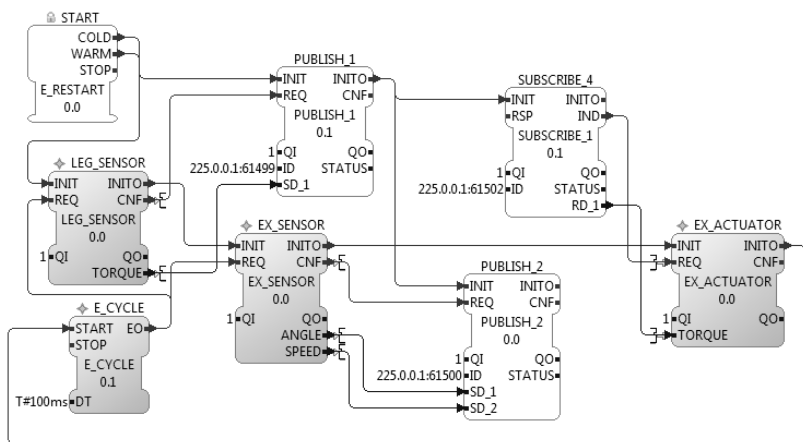


Рисунок 10.17. Частичное приложение распространяется на устройство MONITOR

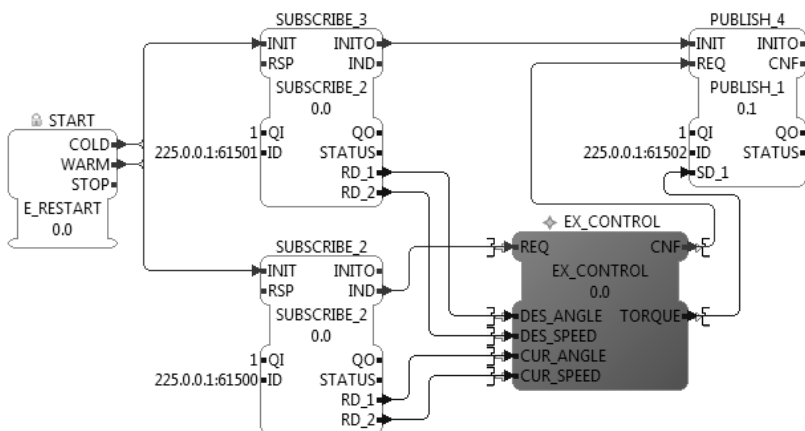


Рисунок 10.18. Частичное приложение распространяется на устройство CONTROL

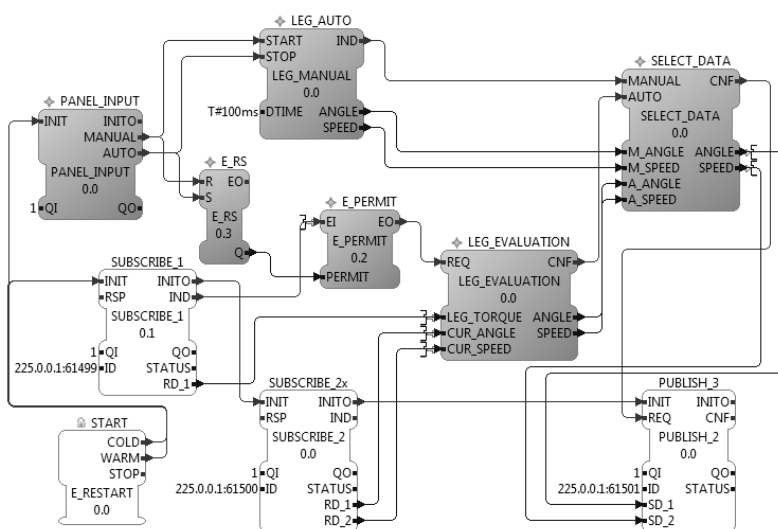


Рисунок 10.19. Частичное приложение распространяется на устройство PANEL

Библиография

- [1] 4DIAC Consortium. 4DIAC Framework for Distributed Industrial Automation and Control, Open Source Initiative.
<http://www.fordiac.org/14.0.html>, 2014.
- [2] CAN in Automation. CiA 404 CANopen: Device Profile for Measuring Devices and Closed Loop Controllers.
<http://www.can-cia.org/standardization/technical-documents/>, 2002.
- [3] M. De Sousa and P. Portugal. Modbus. In B. M. Wilamowski and J. D. Irwin, editors, *The Industrial Electronics Handbook: Industrial Communication Systems*. CRC Press, Boca Raton, 2011.
- [4] B. Drury. *Control Techniques Drives and Controls Handbook*. Institution of Engineering and Technology, Stevenage, United Kingdom, second edition, 2009.
- [5] Eclipse Foundation, Inc. 4DIAC Framework for Distributed Industrial Automation and Control. <https://projects.eclipse.org/projects/iot.4diac>, 2015.
- [6] Fieldbus Foundation. Foundation Fieldbus Technical Overview, 2003.
- [7] N. Higgins, V. Vyatkin, N. Nair, and K. Schwarz. Distributed power system automation with IEC 61850, IEC 61499, and intelligent control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 41, No. 1, 81–92, 2011.
- [8] M. Hofmann, M. Rooker, and A. Zoitl. Improved communication model for an IEC 61499 runtime environment. In *16th IEEE International Conference on Emerging Technologies & Factory Automation*, pages 1–7, 2011.
- [9] Holobloc Inc. FBDK Function Block Development Kit.
<http://www.holobloc.com/doc/fbdk/>, 2011.
- [10] Holobloc, Inc. IEC 61499 Compliance Profile for Feasibility Demonstrations. <http://www.holobloc.com/doc/ita/index.htm>, 2013.
- [11] Holobloc Inc. FBDK 2.3: Function Block Development Kit.
<http://www.holobloc.com/fbdk2/index.htm>, 2015.
- [12] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi. Attack taxonomies for the modbus protocols. *International Journal of Critical Infrastructure Protection*, Vol. 1, 37–44, 2008.
- [13] ICS Triplex ISaGRAF Inc. ISaGRAF 5.1 assessment according to IEC 61499. http://www.isagraf.com/get/isagraf_5-1_1499-tuvreport.pdf, 2007.
- [14] ICS Triplex ISaGRAF Inc. ISaGRAF – IEC 61131 and IEC 61499 software. <http://www.isagraf.com/>, 2014.
- [15] International Electrotechnical Commission. Communication networks and systems for power utility automation. Part 6: Configuration description language for communication in electrical substations related to IEDs, 2009.
- [16] International Electrotechnical Commission. Function blocks. Part 1: Architecture, 2012.
- [17] International Electrotechnical Commission. Function blocks. Part 4: Rules for compliance profiles, 2013.
- [18] International Electrotechnical Commission. Programmable controllers. Part 3: Programming languages, 2013.

- [19] International Electrotechnical Commission. Industrial communication networks: Fieldbus specifications. Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series, 2014.
- [20] International Electrotechnical Commission. Industrial communication networks: Fieldbus specifications. Part 6-10: Application layer protocol specification: Type 10 elements, 2014.
- [21] International Standards Organization. Open systems interconnection basic reference model: The basic model, 1994.
- [22] Internet Engineering Task Force. Request for Comments 768: User Datagram Protocol. <https://www.ietf.org/rfc/rfc768.txt>, 1980.
- [23] Internet Engineering Task Force. Request for Comments 791: INTERNET PROTOCOL. <https://www.ietf.org/rfc/rfc791.txt>, 1981.
- [24] Internet Engineering Task Force. Request for Comments 793: TRANSMISSION CONTROL PROTOCOL. <https://www.ietf.org/rfc/rfc793.txt>, 1981.
- [25] R. Lewis. *Modeling Control Systems Using IEC 61499: Applying Function Blocks to Distributed Systems*. Control Engineering Series. Institution of Engineering and Technology, London, 2001.
- [26] P. S. Marshall and J. S. Rinaldi. *Industrial Ethernet*. ISA, Research Triangle Park, NC, 2005.
- [27] Modbus Organization, Inc. Modbus Messaging on TCP/IP Implementation Guide V1.0b. http://www.modbus.org/docs/modbus_messaging_implementation_guide_v1_0b.pdf, 2006.
- [28] Modbus Organization, Inc. Modbus over serial line specification and implementation guide V1.02. http://www.modbus.org/docs/modbus_over_serial_line_v1_02.pdf, 2006.
- [29] Modbus Organization, Inc. Modbus Application Protocol Specification V1.1b3. http://www.modbus.org/docs/modbus_application_protocol_v1_1b3.pdf, 2012.
- [30] P. Neumann. Communication in industrial automation – What is going on? *Control Engineering Practice*, Vol. 15, No. 11, 1332–1347, 2007.
- [31] nxtControl GmbH. nxtSTUDIO: Engineering tool for the planning of control, visualization, process integration, simulation, testing and documentation. <http://www.nxtcontrol.com>, 2014.
- [32] Open DeviceNet Vendors Association, Inc. Volume One: Common Industrial Protocol (CIP) Specification. <http://www.odva.org/default.aspx?tabid=79>.
- [33] Open DeviceNet Vendors Association, Inc. Device Net Technology Overview. <https://www.odva.org/default.aspx?tabid=107>, 2015.
- [34] Rockwell Automation. Controlnet Network. <http://www.ab.com/en/epub/catalogs/12762/2181376/214372/1809768/>, 2015.
- [35] T. Sauer. Fieldbus system fundamentals. In R. Zurawski, editor, *Industrial Communication Technology Handbook*. CRC Press, Boca Raton, 2015.

[36] J.-P. Thomesse. Fieldbus technology in industrial automation. *Proceedings of the IEEE*, Vol. 93, No. 6, 1073–1101, 2005.

[37] V. Vyatkin and J. Chouinard. On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations. In *6th IEEE International Conference on Industrial Informatics*, pages 289–294, 2008.

[38] V. Vyatkin, M. De Sousa, and A. Zoitl. Communication aspects of IEC 61499 architecture. In B. M. Wilamowski and J. D. Irwin, editors, *The Industrial Electronics Handbook: Industrial Communication Systems*. CRC Press, Boca Raton, 2011.

[39] F. Weehuizen and A. Zoitl. IEC 61499 Compliance Profile for CIP based Communication Function Blocks.

<http://www.ist.massey.ac.nz/functionblocks/cipcompliance.htm>, 2006.

[40] F. Weehuizen, A. Zoitl, et al. Using the CIP protocol with IEC 61499 communication function blocks. In *5th IEEE International Conference on Industrial Informatics*, volume 1-3, pages 261–265, 2007.

Адаптированная методология проектирования на IEC 61499 для распределенного управления коммуникационными устройствами

Carlos Catalan

Университет Сарагосы

Alfonso Blesa

Университет Сарагосы

Felix Serna

Университет Сарагосы

Jose Manuel Colom

Университет Сарагосы

СОДЕРЖАНИЕ

11.1 Краткая мотивация	256
11.2 Введение	257
11.3 Программное обеспечение системы управления для AMS: стандарт IEC 61499	258
11.4 Соединение устройств с IEC 61499: сфера применения платформы COSME	260
11.5 Цели проектирования платформы COSME	261
11.5.1 Как облегчить проектирование приложений	261
11.5.2 Соответствие принципам распределенных систем	262
11.5.3 Объединение функций с добавленной стоимостью	265
11.6 Модель COSME FB	265
11.7 Архитектура платформы COSME	267
11.8 Процесс проектирования COSME	270
11.9 Проблемы реализации	272
11.10 Выводы	273

11.1. Краткая мотивация

Разработка системы управления для станков является нетривиальной и актуальной областью при разработке промышленного программного обеспечения. Это нетривиально, потому что это программное обеспечение должно учитывать множество различных технологий для реализации интегрированного управления механическими, электромеханическими, электрическими и электронными компонентами. Кроме того, существуют функции, строго не связанные с управлением, которые должны быть интегрированы в конечное приложение, которое должно быть реализовано в распределенной среде. Эта сложность порождает такие нюансы, как внимание к жестким ограничениям в реальном времени и необходимость интеграции с другими информационными системами внутри компании, и это, конечно, нужно учитывать. Все это приводит к необходимости методологий, которые способны решать проблемы проектирования с междисциплинарной точки зрения. Некоторые из методологий разработки программного обеспечения могут быть адаптированы к этому контексту. Но учитывая давнюю традицию в области станков и экономическую важность данной области, эти методологии должны интегрировать, по крайней мере, частично, с существующими способами проектирования, а также унаследованными компонентами. В этой главе описывается такая методология (или руководство по проектированию), адаптированная к этой области, чтобы помочь устранить разрыв между стандартом IEC 61499 и промышленной реальностью. Здесь нужно сказать о COSME, платформе исполнения, разработанной для поддержки этой методологии, которая учитывает особенности проблемы станков в контексте IEC 61499. Наконец, в качестве тематического исследования в работе кратко изложены характеристики стола для резки плоского стекла и аспекты, которые влияют на процесс проектирования его управляющего программного обеспечения (например, время цикла, количество осей и т. д.).

11.2. Введение

Обрабатывающая промышленность должна быстрее реагировать на изменения на рынке, но при этом сохранять свою экономичность. Такое понимание вызвало изменение производственных стратегий в сторону меньших адаптированных серий для удовлетворения меняющихся требований. Гибкие производственные системы (FMS) были технологическими ответами на этот сценарий. Они основаны на использовании стандартизированных станков общего назначения (например, компьютерного числового управления), которые могут производить различные детали, изделия и транспортные линии для их перемещения.

Большинство систем управления станками реализуются с использованием программируемых логических контроллеров (PLC). PLC являются моделью

централизованного и циклического управления, то есть получение входных переменных, реализация алгоритмов управления и обновление выходных переменных. IEC 61131-3 [11], широко используемый в промышленности, предоставляет стандарт для систем PLC на основе моделей. Этот стандарт определяет типы данных, функциональные блоки (FB) и языки для приложений управления и является важным шагом вперед в обеспечении переносимости приложений между различными моделями и платформами разработки.

Новые требования отрасли требуют системы управления с новыми функциями; например, легко приспосабливаемый, реконфигурируемый (даже динамический) и многократно используемый стандарт, однако упомянутый стандарт не соответствует этим новым требованиям [28]. Вне области автоматизации при разработке программного обеспечения используются технологии, которые облегчают разработку программного обеспечения. Одной из таких технологий является ориентация компонентов, которая обеспечивает гибкость и повторное использование программного обеспечения; она постепенно была принята в области приложений управления. Другим важным событием стало продвижение в области коммуникаций, которое облегчило переход от централизованной парадигмы к распределенной.

Предыдущие требования и проблемы были рассмотрены в рамках стандарта IEC 61499 [12]. Приложения IEC 61499 – это в основном распределенные сети FB. Для определения модели необходимо учитывать следующее: (a) альтернативные алгоритмы выбора, (b) модели планирования и коммуникации, (c) время отклика, (d) реализацию на основе событий, (e) повторное использование программного обеспечения и (f) ловкость [12]. Этот стандарт не предоставил методологии или руководящих принципов для проектирования сетей FB, и в некоторых работах в качестве примеров были предложены шаблоны проектирования, такие как хорошо известный шаблон прокси-сервера [5] и шаблон управления отказами [20].

IEC-61499 был прорывом в разработке систем управления, потому что, помимо прочего, он позволяет проектировать распределенные системы и имеет модель исполнения, основанную на событиях. В настоящее время ведутся дебаты о преимуществах и недостатках этой модели исполнения. Проблемы, связанные с предсказуемостью времени выполнения, не были изучены в отрасли [26]. Несколько работ [28, 35] указывают на возможные действия, необходимые для содействия его принятию.

В этой главе представлены модель FB и платформа распределенного управления, названная COSME, предназначенная для облегчения проектирования программного обеспечения распределенного управления на основе IEC 61499. Эти концепции также обеспечивают более простую реализацию и хорошую масштабируемость. Оба приспособлены к программному обеспечению управления станком в контексте гибких производственных систем (AMS). Остальная часть главы организована следующим образом. В разделе 11.3 приведены соответствующие основы

стандарта IEC 61499. Раздел 11.4 описывает область действия платформы COSME, а раздел 11.5 – цели проектирования COSME. В разделах 11.6, 11.7 и 11.8 представлены модель COSME, архитектура платформы и процесс проектирования соответственно. Наконец, в разделах 11.9 и 11.10 представлены реализация, тесты и выводы.

11.3. Программное обеспечение системы управления для AMS: стандарт IEC 61499

Правила управления программным обеспечением на основе PLC установлены стандартом IEC 61131 [11] и широко приняты и используются в промышленности. Чтобы соответствовать новым производственным требованиям, появился IEC 61499 [12, 28]. Этот стандарт определяет архитектуру и модели, которые стремятся облегчить разработку реконфигурируемых и распространяемых систем управления. Наиболее значимой моделью является FB, определенный в предыдущем стандарте IEC 61131. Приложение IEC 61499 состоит из сети функциональных блоков (FBN). Эта модель FB включает управляемое событиями выполнение, которое более эффективно, чем выполнение сканирования по IEC 61131. Функциональные блоки IEC 61499 выполняют алгоритмы управления только тогда, когда события указывают на наличие новых данных для обработки, тогда как функциональные блоки IEC 61131 выполняют алгоритмы управления циклически, даже если новых данных нет.

С другой стороны, распределенная система управления состоит из нескольких контроллеров, связанных между собой сетями, без центрального контроллера, в котором вопросы связи и синхронизации играют центральную роль. Модели и архитектура IEC 61499 были определены с самого начала для разработки таких систем. FBN могут быть развернуты в разных ресурсах (то есть во время выполнения), которые размещены на устройствах (то есть контроллерах). Каналы событий и данных между контроллерами абстрагируются функциональными блоками интерфейса службы связи (CSIFB) с моделями публикации / подписки и клиент-сервера.

Цель реконфигурируемого программного обеспечения может быть достигнута с помощью парадигмы компонентного программного обеспечения. Функциональные блоки IEC 61499 можно рассматривать как программные компоненты [14]. Переконфигурирование возможно посредством изменений в FBN путем создания, удаления, замены и присоединения FB мгновенно, а также развертывания и установки новых типов FB. Стандарт обеспечивает модель управления ресурсами, устройствами и функциональными блоками для поддержки проблем реконфигурации.

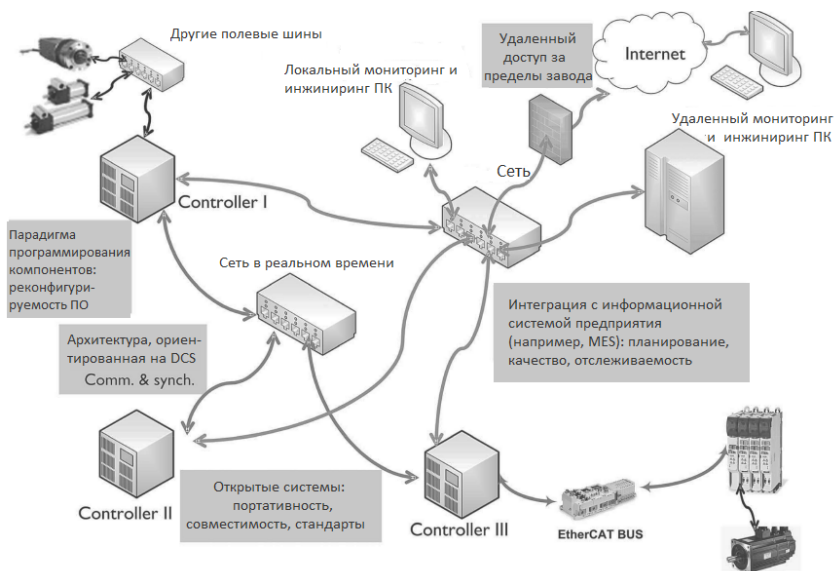


Рисунок 11.1. Сценарий межконтроллерной связи

В отличие от IEC 61131, отрасль еще не приняла новый стандарт. В некоторых ранних исследованиях [25] и [9] были проанализированы возможные причины этого, в основном незрелость стандарта [28] что указывает на действия, предпринятые для содействия его принятию (следующее появление новой версии, первый промышленный опыт и т. ж.). Существует несколько инструментов и платформ для IEC 61499. FBDK и его исполняемая среда FBRT [10] реализованы на Java и не могут соответствовать ограничениям реального времени, однако были ссылки на стандарт во многих научных работах. Важной инициативой с открытым исходным кодом является 4DIAC [35], с инструментом, основанным на платформе Eclipse и средах выполнения для PC и ARM. Другие предложения можно увидеть в [28]; эта работа также указывает на то, что с помощью этих научных инструментов трудно разрабатывать приложения. Однако важно иметь инструменты и платформы, которые могли бы иметь практическое применение инженерами-контролерами за пределами исследовательских лабораторий. Наконец, два коммерческих предложения появились от производителей оборудования для автоматизации: а) ISaGRAF, основанный на предыдущей структуре для IEC 61131 [29], можно рассматривать как сравнимый с FBDK и FBRT, и б) NxtStudio [17]. Первый промышленный опыт разрабатывается с ними.

11.4. Соединение устройств с IEC 61499: сфера применения платформы COSME

Эта работа сосредоточена на станках, которые являются важными элементами любой производственной системы. В общем виде станок выполняет такие операции, как резка, сверление, фрезерование и т. д. Эти операции требуют определенного количества перемещений деталей, обрабатывающих инструментов и готовых изделий. Для выполнения этих задач программное обеспечение системы управления должна учитывать: а) точное позиционирование, б) загрузку и выгрузку деталей и готовых изделий и в) режимы работы и управление отказами.

Коммуникационные аспекты являются ключевыми в станках. То есть они должны быть интегрированы на разных уровнях заводских сетей связи. Чтобы выполнить эту интеграцию, программное обеспечение системы управления должна учитывать взаимодействие: а) с другими элементами того же рабочего процесса на заводе и б) с внешними приложениями (например, диспетчерское управление, сбор данных и управление производством). Поскольку операции обработки могут изменяться в контексте гибкого производства, другим аспектом, который следует учитывать, является реконфигурируемость программного обеспечения. На рисунке 11.1 показан сценарий, рассмотренный в этой главе для распределенного управления станками.

Важной особенностью этих станков является управление отказами, которое должно учитывать как их собственные сбои, так и ошибки, вызванные другими машинами в том же рабочем процессе. Чтобы установить политику ответных действий (то есть управление отказами), можно рассмотреть несколько режимов работы, таких как нормальный и удовлетворительный режимы работы и сбой. Также необходимо учитывать особенности профилактического обслуживания.

В некоторых работах по использованию IEC 61499 для станков [31] представлена структура и методология адаптивных сборочных операций с использованием FB для динамических операций в цехе. Хагге и Вагнер [8] используют FB для реализации моделей состояний управления с открытой модульной архитектурой (ОМАС) для упаковочных машин. Была предложена архитектура для открытой и распространяемой системы CNC на основе STEPNC и FB [16], которая была протестирована в лабораторном прототипе с FBDK и FBRT. Xuemei [32] рассмотрел возможность реконфигурируемого управления для PLC и CNC, но не представил никаких экспериментальных результатов, за исключением краткого тематического исследования. Ни одно из этих предложений не было проверено в реальных приложениях.

Реальный станок может иметь десятки осей и независимых движений, поэтому его управление на основе IEC 61499 может привести к большим и реконфигурируемым FBN с сотнями FB. В первом приближении проект FBN может начинаться с разделения управления на основные части (например, управление осями). Эти части появляются неоднократно и могут быть реализованы с одним sub-FBN. Затем управление может быть реализовано

путем составления этих подчиненных FBN и добавления некоторой бизнес-логики. Масштабируемость FBN является важной целью для поддержания хорошей производительности и сокращения времени проектирования. То есть эти проблемы не должны зависеть от количества осей или движений. Авторы определили контрольные части конкретного станка и предложили рекомендации по проектированию [21].

Эти элементы являются достаточно универсальными, чтобы быть использованными на любом станке: а) управление: точное позиционирование осей, б) автоматизация: последовательности перемещений, в) управление отказами, г) бизнес-логика: специфическая функциональность и режимы работы, е) процесс взаимодействия: датчики и исполнительные механизмы по полевой шине, ф) внешняя связь: шлюз с внешним оборудованием.

Что касается распределенного управления, FBN, состоящий из составных частей управления sub-FBN, может быть развернут (то есть распределен) от одного до десятков контроллеров, в зависимости от степени детализации распределения. То есть один контроллер может управлять одной осью или всей машиной. Опять же в первом приближении этот распределенный FBN может устанавливать связи между контроллерами с использованием CSIFB.

11.5. Цели проектирования платформы COSME

11.5.1. Как облегчить проектирование приложений

С точки зрения компонента парадигмы программирования типы FB должны быть тщательно спроектированы, чтобы можно было повторно использовать их в различных контекстах (то есть они должны взаимодействовать с другими FB). FBN – это больше, чем монолитный дизайн, разделенный на части, которые реализованы как FB; он должен быть разработан с использованием методологий разработки программного обеспечения. IEC 61499 не устанавливает методологии или руководящих принципов, которые описывают типы FB или определяют взаимодействия FB. Программное обеспечение, основанное на модельно-управляемых методологиях, связанных с IEC 61499, обобщено [28]. Другие авторы сосредоточились на шаблонах проектирования [19, 20, 22] и руководствах [3] для стандарта.

Недостатком является то, что инженеры систем управления не знакомы с парадигмами IEC 61499 (то есть с компонентами, управляемыми событиями и программным обеспечением), или с методологиями разработки программного обеспечения. Поэтому эти предметы должны быть включены в учебные программы по инженерному контролю (например, см. [25] и [36]).

Подход COSME состоит в том, чтобы предоставить инженерам систем управления адаптированную модель FB, универсальную и установленную для конкретных видов применения. Эта адаптация пытается предопределить

события и алгоритмы, чтобы упростить разработку приложений и сократить время разработки.

11.5.2. Соответствие принципам распределенных систем

С точки зрения стандарта, приложение распределенного управления может быть разработано в два этапа [30]: 1) FBN спроектирован без учета проблем распределения и 2) FBN разделен на зависимые части, которые впоследствии развертываются на ресурсах и соединяются по CSIFB. Этот процесс направлен на увеличение возможности повторного использования FB и деталей FBN. Соответственно, вопросы распределения (например, количество или тип контроллеров) и топология FBN не влияют на дизайн. Проектирование распределенных FBN не так просто реализовать [1, 24, 27]. Например, проектирование невозможно без учета задержек связи и синхронизации между FB [30]. Чтобы решить эти проблемы, дизайнеры должны добавить модификации в исходный FBN. Распределенное управление на основе IEC 61499 может осуществляться с учетом общепринятых принципов проектирования распределенных систем [15]: прозрачность, надежность, производительность, масштабируемость, безопасность, гибкость и неоднородность. На этом этапе мы сосредоточимся на первых пяти, потому что последние два являются неотъемлемой частью стандарта.

Таблица 11.1. Типы коммуникационных устройств

Тип связи	Типовая задержка	Модель исполнения	Надежность	Объем данных	Пример
Контроль приложений	100 мс	Управляемый событием	Да	Байты	Режимы работы. Управление отказами
Контроль над процессом	1 мс	Управляемый циклом	Да	Байты	Точное позиционирование оси с помощью PID-регулятора
Команда процесса	100 мс	Управляемый событием	Да	Байты	Последовательность команд движения: «go XY»
Синхронизация процессов	100 мс	Управляемый событием	Да	Байты	Высокие семантические команды: «available piece»

Человеко-машинный интерфейс	100–500 мс	Управляемый событием и управляемый циклом	Нет / да	Килобайты	Надзор / сигнализации
Управление данными приложения	500 мс	Управляемый циклом	В основном	Килобайты	Профилактическое обслуживание
Управление производственными данными	Несколько секунд	Управляемый циклом	Да	Мегабайты	Связь с MES

Прозрачность. Этот принцип считается скрытым для базовых распределенных систем. IEC 61499 не обеспечивает прозрачность, потому что разработчики должны внести изменения в FBN, упомянутые выше, а затем ввести CSIFB в линии связи. Предлагается решение [30], с помощью которого CSIFB автоматически вводятся инструментом разработки, когда сопоставление между FB и ресурсами уже установлено. Другое предложение [7] заменяет CSIFB уровнем промежуточного программного обеспечения. Для достижения этой цели подход COSME предопределяет линии связи, что облегчает автоматическую вставку CSIFB и позволяет избежать дальнейших изменений в FBN. Ранее типы связи были определены и охарактеризованы из конкретных областей применения [2]; это похоже на случаи общения, представленные в [23]. В таблице 11.1 показаны идентифицированные типы, которые характеризуются несколькими особенностями: задержка, режим выполнения, надежность и объем передаваемых данных. Эти типы включены в модель COSME FB, повышая уровень абстракции распределения (то есть принцип прозрачности) для программного обеспечения распределенного управления, основанного на IEC 61499.

Надежность. Надежность в FMS рассматривается, в частности, с точки зрения аппаратного обеспечения, поскольку традиционные управляющие приложения относительно просты, и поэтому программное обеспечение также простое. PLC предназначены для поддержания высокой степени надежности и доступности в течение длительных периодов в суровых промышленных условиях. Хотя надежность должна повышаться за счет использования уже протестированных FB, требования к гибким производственным системам (AMS) увеличивают сложность программного обеспечения, затрудняя разработку систем, критичных для безопасности. В частности, распределенное программирование и функции IEC 61499, управляемые событиями, затрудняют обеспечение детерминированного поведения приложений и, следовательно, уменьшают их надежность [1]. Одним из способов обеспечения надежности является использование методов верификации и валидации, но их сложно применять в этом контексте [28]. Адаптированная модель COSME FB с предопределенными событиями и алгоритмами приводит к модели исполнения (см. рисунок 11.6), которая облегчает определение

поведения системы во время разработки, способствуя надежности. Кроме того, эта модель избегает проектирования с нуля, что также способствует достижению названной цели.

Производительность. Хотя, с точки зрения инженеров систем управления, управляемое событиями выполнение более эффективно, чем сканирование, его реализация добавляет накладные расходы, которые снижают производительность [28]. Это влияет на время выполнения в наихудшем случае (WCET), что важно для определения соответствия ограничениям в реальном времени. Расчет WCET в FBN IEC 61499 сложен из-за его управляемого событиями выполнения (сильно зависит от его реализации во время выполнения) и функции реконфигурируемости. Некоторые работы посвящены этой проблеме. Например, в [13] выполнен статический анализ на основе проверки модели, а в [7] вычисляется WCET, чтобы показать производительность конкретной платформы. Полное исследование выполнения в реальном времени и исчисления WCET в системах IEC 61499 было представлено в [34]. Эта работа перечисляет установленные руководящие принципы, которые ограничивают гибкость при проектировании FBN, чтобы улучшить WCET и производительность.

Модель выполнения COSME устанавливает простую и регулярную топологию FBN, которая стремится получить хорошую производительность и простой расчет WCET.

Масштабируемость. Желательно, чтобы распределенная система имела производительность, по крайней мере эквивалентную централизованной системе, и была масштабируемой. На масштабируемость приложений IEC 61499 влияют два фактора: а) количество контроллеров и б) размер sub-FBN на каждом контроллере. Существуют работы, в которых изучались приложения IEC 61499 с большим количеством контроллеров [4, 33], но нам не известны работы с большими sub-FBN на контроллер. В целом, масштабируемость зависит от коммуникационных аспектов и параллелизма базовой платформы; для больших sub-FBN вторая проблема является наиболее важной. Параллелизм связан с параллельным выполнением FB и неявно присутствует в его модели, управляемой событиями. На практике трудно реализовать масштабируемую реализацию FBN, поскольку для реальных приложений (например, станков) может потребоваться большое количество FB, работающих одновременно [9], и, следовательно, требуется использование многих ресурсов платформы. Эти ресурсы связаны главным образом с контекстами выполнения (например, потоками), которые были проанализированы для различных элементов IEC 61499 [24]. Эта работа предлагает концепцию цепочки событий для достижения эффективного выполнения FBN, но не учитывает проблему масштабируемости. Для достижения этой цели модели исполнения COSME требуется ограниченное количество потоков, независимо от размера и сложности FBN.

Безопасность. Современные производственные системы интегрированы в коммуникационные сети, которые выходят за рамки заводских приложений. Если безопасность очень важна в любой сети коммуникации, то в

производственных системах это является обязательным, поскольку эти системы могут влиять на людей, оборудование (то есть, безопасность) и скрытую информацию. К аспектам безопасности, которые необходимо учитывать, относятся целостность, конфиденциальность, доступность и аутентификация. IEC 61499 не учитывает ни один из них, поскольку они зависят от реализации и абстрагируются от CSIFB. Тем не менее в некоторых работах этот вопрос рассматривался в общепринятом виде [6, 18]. Предопределенные каналы связи позволяют COSME устанавливать конкретные механизмы безопасности. Например, связь в реальном времени между контроллерами, поддерживаемыми выделенной проводной локальной сетью, может не требовать шифрования данных; связь в режиме реального времени между контроллерами и внешними приложениями, поддерживаемыми невыделенной сетью, обязательно требует шифрования данных.

11.5.3. Объединение функций с добавленной стоимостью

Как упомянуто в разделе 11.4, эти приложения, помимо функций управления, имеют дополнительные функциональные требования, такие как профилактическое и прогностическое обслуживание. Платформа COSME включает в себя услуги для их удовлетворения. В первом случае адаптированная модель может включать непрерывный учет использования (время или количество циклов элементов машины). Во втором случае она может включать в себя FB, которые определяют сбои (например, по правилам) контрольных элементов машины [19, 22]. Управляющие приложения должны решать проблемы интеграции на всех уровнях управления производством, например, в системах управления производством (MES). Эти системы управляют обязательными функциями, такими как планирование производства, контроль качества и отслеживание продукта. COSME включает функции, которые позволяют инженерам управлять этими функциями.

В таблице 11.2 сравниваются подходы COSME и IEC 61499. Там рассматриваются область приложения, компонентная парадигма, архитектура для поддержки распределенного управления и дополнительные функции.

В таблице 11.2 приведены основные характеристики подхода COSME по сравнению с общим подходом IEC 61499.

11.6. Модель COSME FB

IEC 61499 указывает, что FB должны учитывать множество алгоритмов, выбираемых по событиям или условиям. Модель COSME FB (см. рисунок 11.2) основана на предварительно определенных алгоритмах, учитывающих общие ситуации в управляющих приложениях, такие как инициализация,

нормальный режим работы, сбой и т. д. В таблице 11.3 приведено более подробное описание.

Входные и выходные события предопределены, а входные и выходные данные – нет, и они зависят от функциональности типа FB. Алгоритмы генерируют событие, когда они выполнены. Это позволяет составлять FBN для создания топологии гирляндной цепочки, которая связывает события одного и того же типа алгоритма. Расширение распределенных систем управления этой модели может быть просто сделано путем включения CSIFB в цепочки между контроллерами. Эти CSIFB должны быть типами публикации и подписки, потому что они подходят для распространения событий вдоль контроллеров.

Как было сказано выше, модель COSME нацелена на включение типов связи станков. Для этого (см. рисунок 11.2) типы сопоставляются с предварительно определенными цепочками следующим образом: цепочки COLD_INIT, WARM_INIT, FINALIZE и FAILURE RECOVERY соответствуют типу связи управляющего приложения. Только события распространяются, поэтому CSFIB без данных необходимы.

FAILURE также соответствует типу связи управляющего приложения, но имеет связанную информацию о состоянии отказа; в этом случае нужен CSIFB с данными.

Таблица 11.2. Сравнение подходов COSME и IEC 61499

	Подход IEC 61499	Подход COSME
Домен приложения	Системы автоматизации общего назначения	Специальное назначение, станки для строительных материалов (например, стекла, дерева и т. д.)
Компонентная парадигма программирования		
Модель FB	Управляемый событием. Разработчики определяют события и данные I/O, а также ECC	Управляемый событием. Предопределенные события и алгоритмы (нормальный режим работы и сбой, инициализация и т. д.), адаптированные к предметной области
Модели выполнения	Вызов FB: последовательный или циклический; контексты выполнения: FB, ресурс или цепочка событий; непрерывный многопоточный ресурс; параллельно	«Гирлянда из ромашек» (на основе модели цепочки событий)
Распределенный FBN	Разработчики должны иметь дело с деталями (например, CSIFB, модификациями FBN, вызванными проблемами синхронизации)	Распределенная гирляндная цепочка с помощью предопределенного CSFB. FBN не должен быть изменен из-за проблем синхронизации
Проектирование FBN	Методологии, основанные на моделях, с небольшой поддержкой IEC 61449	Адаптированная модель FB (предварительно приготовленная). Проектировщики только

		определяют I/O данных и предопределенный код алгоритмов в адаптированной модели FB, затем устанавливают последовательные цепочки
Архитектура, ориентированная на поддержку распределенного управления		
Прозрачность	Вставьте CSIFB вручную или используя промежуточное ПО	Инструмент разработки COSME автоматически вводит CSIF на основе предопределенных типов связи
Надежность	Сложно обеспечить детерминированное поведение	Модель COSME FB облегчает определение
Производительность и масштабируемость	Сильно зависит от сложности FBN, моделей исполнения и реализации платформы	Модель исполнения COSME и ее реализация требуют ограниченного количества ресурсов, независимо от размера и сложности FBN. Хорошо: сеть из 200 FB на реальном приложении
Безопасность	Зависит от платформы	Предопределенные типы связи допускают определенные механизмы безопасности: (1) синхронизация процесса; (2) управление приложениями и производственными данными
Дополнительные функции	–	Платформа COSME обеспечивает такие функции, как профилактическое и прогнозное обслуживание или интеграция с информацией о корпоративной системе

11.7. Архитектура платформы COSME

COSME – это уровень между управляющими приложениями и базовой системой (то есть аппаратным обеспечением, операционной системой и связью). Типичная среда COSME показана на рисунке 11.3. Для выполнения платформа следует концепции цепочки событий (ЕС) [24]. Реализация осуществляется путем связывания этих ЕС с различными приоритетными потоками в соответствии с моделью, называемой «Путь к событию», для каждой задачи, приоритет на основе пути события [7]. Алгоритмы управления должны быть выполнены в короткие сроки, потому что они блокируют систему для ЕС с более низким приоритетом.

Из-за ограниченного количества ЕС и потоков реализация легко масштабируется. Таким образом, эта модель особенно подходит для приложений с большим количеством FB, а также облегчает определение поведения системы во время разработки.

Таблица 11.3. Алгоритмы FB COSME

Название	Описание
Холодный Init	Устанавливает начальное состояние FB. Он выполняется аperiodически с наивысшим приоритетом.
Теплый Init	Он выполняет действия возобновления после запланированной остановки. Он выполняется аperiodически с наивысшим приоритетом.
Согласование	Выполняет действия по завершению. Он выполняется аperiodически с наивысшим приоритетом. Типичным действием может быть отключение исполнительных механизмов.
Нормальный RT	Он выполняет действия с ограничениями в реальном времени. Это выполняется периодически с высоким приоритетом. Типичным действием может быть петля PID. Если возникает ненормальное состояние, он может генерировать предупреждения об ухудшении работы или сбое других FB.
Нормальный NRT	Он выполняет действия с ограничениями не в реальном времени. Периодически выполняется с меньшим приоритетом, чем обычный RT. Типичным действием может быть надзор или связь с HMI. Как и обычный RT, он может генерировать событие, чтобы уведомить о ненормальной работе или запросить фоновое действие.
Предпосылки	Он выполняет действия в фоновом режиме. Это время от времени выполняется с самым низким приоритетом. Типичным действием может быть расчет траекторий CNC.
Сбой	Он выполняет действия режима отказа. Это выполняется периодически с высоким приоритетом. Типичными действиями могут быть установка безопасного режима и регистрация данных об ошибках.
Восстановление после сбоя	Выполняет действия по восстановлению из режима сбоя. Типичные действия могут быть в обратном порядке.
Внешнее событие	Он обрабатывает внешние события. Он выполняется аperiodически с наивысшим приоритетом. Типичным действием является обработка данных с полевой шины I/O.

Элементами COSME, которые выполняют работу, являются сервер выполнения, сервер имен и загрузчик приложений. Загрузчик загружает типы FB, создает FB и устанавливает внутри- и межконтроллерные соединения, затем заказывает загрузку выполнения. Платформа имеет постоянный элемент хранения для запуска приложений.

Хотя текущая версия COSME сфокусирована на аспектах разработки и распространения приложений, она позволяет просто реконфигурировать FBN во время выполнения с помощью команд (создание экземпляров, переключатель и т. д.). В новой версии COSME эти команды будут отправлены внешним менеджером реконфигурации, который получит данные от управления и от MES для реализации динамической реконфигурации. Элементами COSME, которые выполняют эту функцию, являются сервер компонентов, сервер имен и шлюз. Межконтроллерные каналы связи основаны на режиме публикации и подписки. Сообщения содержат номер сообщения, имена издателя и подписчика, отметку времени и данные (при необходимости). Связь между контроллерами использует проводную выделенную сеть и коммутируемое устройство Ethernet. Эти устройства гарантируют детерминированные задержки без специального аппаратного или программного обеспечения. Политика аутентификации используется для гарантии безопасности.

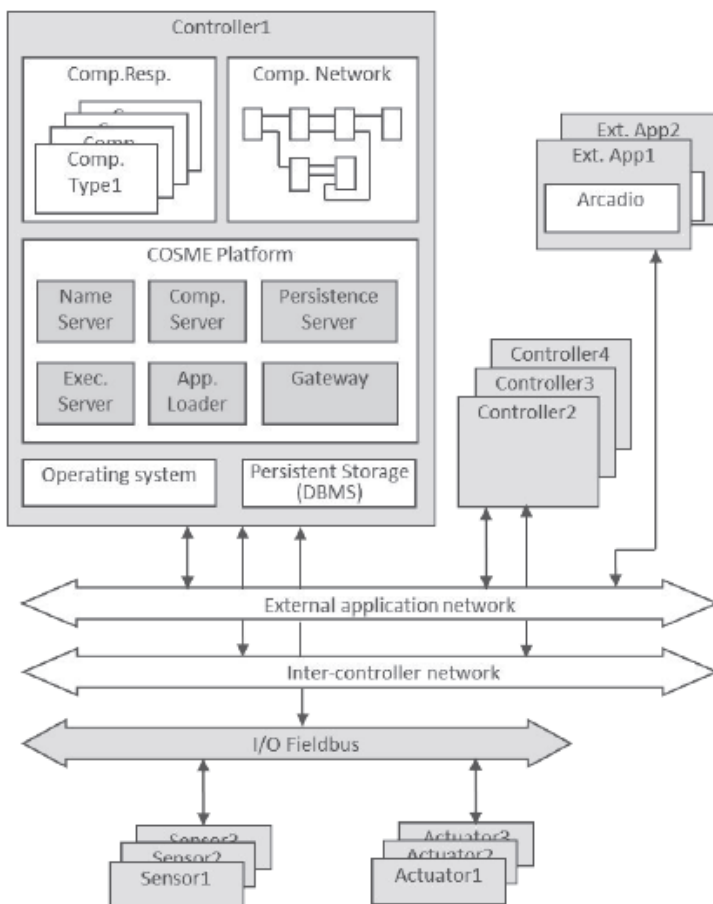


Рисунок 11.3. Типичная среда COSME. Существует четыре контроллера и два внешних приложения: HMI и управление производством. Каждая платформа COSME имеет сервисы (имя, приложение, сервер постоянства и компонентов, загрузчик и шлюз), хранилища FB и сеть FB.

Связь с внешними приложениями осуществляется с клиентом и сервером или публикует и подписывает модели обмена сообщениями. В обоих случаях сообщения имеют тот же формат, что и в случае межконтроллера. Библиотека Arcadio [21] осуществляет связь во внешних приложениях, скрывая от разработчиков детали реализации. Здесь используются не выделенные сети, поэтому в целях безопасности соединения используют аутентификацию и шифрование. Во время загрузки загрузчик приложения устанавливает все каналы связи, и ими позже управляет элемент шлюза. Наконец, взаимодействие с интерфейсом процесса выполняется полевой шиной.

11.8. Процесс проектирования COSME

Вкратце, процесс состоит из сопоставления каждой части управления приложениями, идентифицированной одним FB или sub-FBN, а затем их составления. Sub-FBN может состоять из нескольких типов FB. Например, точное позиционирование оси имеет один контур PID-регулирования, один генератор заданного значения и одну коробку передач. Каждый тип FB должен иметь следующее: а) интерфейс FB посредством выходных и входных данных; б) состояние FB с помощью внутренних переменных и в) поведение FB путем установки действий в определенных алгоритмах.

Для создания FBN правилами составления COSME являются: а) соединение выходных и входных данных между взаимосвязанными sub-FBN; б) установление последовательных цепочек с входными и выходными событиями с учетом зависимостей данных (то есть это должно гарантировать, что производительные данные FB выполняются раньше, чем потребительские); в) установление CSIFB в распределенных последовательных цепях, соответствующих ранее указанным типам связи. Чтобы облегчить процесс проектирования, COSME включает среду программирования Domiciano [2]. Domiciano – это интегрированная среда разработки (IDE), которая позволяет разрабатывать управляющие приложения для платформы COSME. Domiciano был разработан с учетом характеристик COSME, таких как создание последовательных цепочек из порядка выполнения FB, заданного разработчиком. Затем при необходимости вводится CSIFB.

(например, протокола передачи листов) и контроля. Наконец, хотя это не показано на рисунке 11.4, было использовано иерархическое управление отказами. Это основано на схеме проектирования, представленной авторами в [20].

Резюмируя, можно сказать, что гирлянда COSME является промежуточной моделью между событиями и сканированием. Дизайнеры должны написать код ответа для ситуаций, например, управляемых событиями, но ситуации должны быть предопределены. COSME был разработан с учетом программного обеспечения для управления станками, хотя его можно использовать и в других приложениях. Правила композиции просты и могут применяться инженерами с классическим образованием в области управления.

Таблица 11.4. Коммуникационные задержки контроллеров

E_CYCLE	Max.	Min.	Среднее	Стандартное среднее	Медианное
50	10,19	1,99	2,58	0,65	2,40
100	10,49	2,01	2,46	0,57	2,30
300	9,57	2,15	2,19	0,51	2,16

Примечание: все измерения даны в миллисекундах

11.9. Проблемы реализации

Текущая реализация COSME работает на ПК с Linux с интерфейсом приложений реального времени (RTAI) и Java SE6. RTAI использовался для реализации сервера компонентов, сервера имен и сервера выполнения. Другие элементы были частично реализованы с помощью Java и RTAI. COSME предоставляет коммуникационную библиотеку Arcadio сторонним внешним приложениям, чтобы они могли подключаться к COSME для использования приложений, связанных с пользовательским интерфейсом, отладкой, задачами журналирования, удаленным мониторингом, интеграцией MES и т. д. Arcadio и Domiciano полностью реализованы на Java. COSME использовался управляющими инженерами производителя станков TUROMAS-TECNOCAT Group для разработки управляющего программного обеспечения новой линии резки стекла [21].

Стол для резки имеет девять осей, точность резки которых меньше одного миллиметра, ускорение до 22 м/с². Эта линия имеет загрузчики и хранит листовое стекло и готовые изделия. Существуют два внешних приложения: NMI и управление производством, оба реализованы на Java. Упрощенное описание этого тестового приложения приведено в разделе 11.8. Реальное протестированное приложение было включено с целью дать некоторые показатели производительности при реализации работы COSME в реальной среде, эффективности подхода и заинтересованности в использовании COSME в сравнении с более общими платформами.

Станок для резки стекла может иметь до 35 типов FB, 200 экземпляров и 70 автоматических последовательностей. Прикладное оборудование состоит из двух ПК-контроллеров B & R Automation PC 620 с процессором Intel Celeron 1,1 ГГц, каждый из которых имеет два сетевых интерфейса по 100 МБ и одну мастер-карту PCI-7841Adlink CANopen. Кроме того, имеется два удаленных ввода-вывода X20 B & R, два вентильных островка s0700 SMC, три стандартных электропривода Sdrive Power Stages и два интеллектуальных гигабитных коммутатора CISCO SLM200.

Периодическая цепь NORMAL_RT, отвечающая за точное позиционирование и последовательность перемещения, имеет RT_E_CYCLE (2 мс) для достижения требуемой точности резки. Чтобы установить время цикла NORMAL_NRT, задержки связи между контроллерами были измерены с различным временем E_CYCLE (см. Таблицу 11.4). Результаты зависят от производительности Java, но имеют аналогичные значения для этой конфигурации. Для удовлетворения требований был выбран E_CYCLE на 50 мс.

Основными факторами производительности и масштабируемости, которые следует учитывать, являются количество осей и количество последовательностей движения. Эти факторы значительно увеличивают сложность и размер FBN. Количество осей является ключевым фактором, потому что для его реализации требуются более сложные sub-FBN. Таблица 11.5 показывает максимальное и минимальное время выполнения и характеристики масштабируемости для трехосных чисел. Обратите внимание, что результаты близки к линейному поведению.

Таблица 11.6 показывает время выполнения NORMAL_RT соответствующих частей приложения.

11.10. Выводы

В этой главе показаны трудности проектирования приложений распределенного управления AMS на основе стандарта IEC 61499. Предлагаемое решение характеризует распределенное управление конкретными областями (например, в области станков). Благодаря нему можно успешно решать задачи разработки приложений, которые должны соответствовать требованиям реального времени или параллелизма (со многими осями). Технологические изменения не должны изменять архитектуру приложения (например, изменять шину fieldbus, обслуживающую I/O), и, наконец, этот подход позволяет адаптировать данное решение к изменениям, вызываемым рынком (то есть легко изменять функциональные и нормативные ограничения).

Это позволяет нам определить адаптированную модель FB и модель исполнения. Данный подход основан на методологии (которая абстрагирует систему как набор компонентов и сеть) и четко определенной семантике (поддерживаемой в ее цикле выполнения) в модели связи и определении компонента поведения. Это предложение сопровождается техническим

инструментом, который поддерживает методологию с предопределенной семантикой и адаптируется и/или расширяется для технологических изменений.

Таблица 11.5. Время выполнения контроля оси

Количество осей		Max. (μs.)	Min. (μs.)
3		646,35	276,59
6		1285,60	601,78
9		1670,72	821,54

Таблица 11.6. Время выполнения частей приложения

Функция	Max. (μs.)	Min. (μs.)
PID-контроль Axis	8,72	3,69
Последовательность загрузки	45,63	8,68
Привод CANopen (3 оси)	179,48	104,35
Кондиционирование I/O	0,90	0,33

Для проверки ее эффективности была внедрена платформа COSME. Результаты, полученные в реальных и сложных приложениях, показывают хорошую производительность и масштабируемость. Кроме того, эти модели снижают сложность конструкции, облегчая тем самым работу инженерам систем управления.

Библиография

[1] R. Brennan. Toward real-time distributed intelligent control: A survey of research themes and applications. *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, Vol. 37, No. 6, Sept. 2007.

[2] C. Catalán, F. Serna, A. Blesa, J. Rams, and J. Colom. Communication types for manufacturing systems: A proposal to distributed control system based on IEC 61499. In *Automation Science and Engineering, IEEE Conference on*, pages 767–772, Aug. 2011.

[3] C. Catalán, F. Serna, T. Civera, A. Blesa, and J. Rams. Control design for machine tools using Domiciano, an IDE based on software components. In *Emerging Technologies Factory Automation, IEEE 16th Conference on*, pages 1–4, Sept. 2011.

[4] J. Chouinard, J. Lavallée, J.-F. Laliberté, N. Landreadud, K. Thramboulidis, P. Bettez-Poirier, F. Desy, F. Darveau, N. Gendron, and T. C.-D. An IEC 61499 configuration with 70 controllers: Challenges, benefits and a discussion on technical decisions. In *IEEE Conference on Emerging Technologies and Factory Automation*, Sept. 2007.

- [5] J. H. Christensen. Design patterns for systems engineering with IEC 61499. *Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung*, 2000.
- [6] I. M. Delamer and J. L. Martínez. Information security for reconfigurable manufacturing systems using networked embedded controllers. In A. D. M. E. Pereira, editor, *Information Control Problems in Manufacturing 2006*, pages 129–134. Elsevier, Oxford, 2006.
- [7] G. Doukas and K. Thramboulidis. A real-time-linux-based framework for model-driven engineering in control and automation. *IEEE Transactions on Industrial Electronics*, Vol. 58, No. 3, 914–924, March 2011. [8] N. Hagge and B. Wagner. Implementation alternatives for the OMAC state machines using IEC 61499. In *Emerging Technologies and Factory Automation, IEEE International Conference on*, pages 215–220, Sept. 2008.
- [9] K. Hall, R. Staron, and A. Zoitl. Challenges to industry adoption of the IEC 61499 standard on event-based function blocks. In *IEEE Conference on Industrial Informatics*, 2007.
- [10] Holobloc. <http://www.holobloc.com/doc/fbdk/index.htm>, 2012.
- [11] International Electrotechnical Commission. *IEC 61131-3 Edition 2.0: Programmable Controllers. Part 3: Programming Languages.*, 2003.
- [12] International Electrotechnical Commission. *IEC 61499-1, Function Blocks. Part 1: Architecture*, 2.0 edition, 2005. [13] M. Kuo, L. Yoong, S. Andalam, and P. Roop. Determining the worstcase reaction time of IEC 61499 function blocks. In *IEEE Conference on Industrial Informatics*, July 2010.
- [14] R. Lewis. *Modelling Control Systems Using IEC 61499: Applying Function Blocks to Distributed Systems*. Number 59. Institution of Engineering and Technology, 2001.
- [15] M. Liu. *Distributing Computing: Principles and Applications*. Pearson-Addison Wesley, 2004.
- [16] M. Minhat, V. Vyatkin, X. Xu, S. Wong, and Z. Al-Bayaa. A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks. *Robotics and Computer-Integrated Manufacturing*, Vol. 25, No. 3, 560–569, June 2009.
- [17] NXTControl. <http://www.nxtcontrol.com>, 2012.
- [18] C. Schwab, M. Tangermann, and A. Lüder. Integration of scalable safety and security actions in IEC 61499 control applications. In *IFAC Conference on Field Bus Systems and Their Applications*. Nov. 2005.
- [19] F. Serna, C. Catalán, A. Blesa, J. Colom, and J. Rams. Predictive maintenance surveyor design pattern for machine tools control software applications. In *Emerging Technologies Factory Automation, IEEE 16th Conference on*, pages 1–7, Sept. 2011.
- [20] F. Serna, C. Catalán, A. Blesa, and J. Rams. Design patterns for failure management in IEC 61499 function blocks. In *Emerging Technologies and Factory Automation, IEEE Conference on*, pages 1–7, Sept. 2010.
- [21] F. Serna, C. Catalán, A. Blesa, J. Rams, and J. Colom. Control software design for a cutting glass machine tool based on the COSME platform. case study.

In *Automation Science and Engineering, IEEE Conference on*, pages 501–506, Aug. 2011.

[22] F. Serna, C. Catalán, A. Blesa, J. Rams, and J. Colom. Preventive maintenance manager design pattern for component based machine tools. In *Industrial Informatics, 9th IEEE International Conference on*, pages 615–620, July 2011. [23] S. Sierla, J. Peltola, and K. Koskinen. Real-time middleware for the requirements of distributed process control. In *Industrial Informatics, 3rd IEEE International Conference on*, pages 1–6. Aug. 2005.

[24] T. Strasser, A. Zoitl, J. Christensen, and C. Sünder. Design and execution issues in IEC 61499 distributed automation and control systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, Vol. 41, No. 1, 41–51, Jan 2011.

[25] K. Thramboulidis. IEC 61499 in factory automation. In K. Elleithy, T. Sobh, A. Mahmood, M. Iskander, and M. Karim, editors, *Advances in Computer, Information, and Systems Sciences, and Engineering*, pages 115–124. Springer, 2006.

[26] K. Thramboulidis. IEC 61499: Back to the well proven practice of IEC 61131? In *Emerging Technologies & Factory Automation, IEEE 17th Conference on*, pages 1–8, 2012.

[27] V. Vyatkin. The IEC 61499 standard and its semantics. *Industrial Electronics Magazine, IEEE*, Vol. 3, No. 4, 40–48, Dec. 2009.

[28] V. Vyatkin. IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. *Industrial Informatics, IEEE Transactions on*, Vol. 7, No. 4, 768–781, Nov. 2011.

[29] V. Vyatkin and J. Chouinard. On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations. In *Industrial Informatics, 6th IEEE International Conference on*, pages 289–294, July 2008.

[30] W. Wagner, J. Bohl, and G. Frey. An IEC 61499 interpretation and implementation focused on usability. In *IEEE Conference on Emerging Technologies and Factory Automation*, Sept. 2008.

[31] L. Wang, S. Keshavarzmanesh, and H. Feng. Design of adaptive function blocks for dynamic assembly planning and control. *Journal of Manufacturing Systems*, Vol. 27, 45–51, 2008.

[32] H. Xuemei. Distributed and reconfigurable control of lower level machines in automatic production line. In *8th World Congress on Intelligent Control and Automation*, pages 2339–2344, July 2010.

[33] J. Yan and V. Vyatkin. Distributed execution and cyber-physical design of baggage handling automation with iec 61499. In *Industrial Informatics, 9th IEEE International Conference on*, pages 573–578, July 2011. [34] A. Zoitl. *Real-Time Execution for IEC 61499*. O3NEIDA and Instrumentation Society of America), 2009.

[35] A. Zoitl, T. Strasser, and A. Valentini. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC case study. In *Industrial Electronics, IEEE International Symposium on*, pages 3817–3819, July 2010.

[36] A. Zoitl and V. Vyatkin. Face to face: IEC 61499 architecture for distributed automation: The glass half full view. *IEEE Industrial Electronics Magazine*, Vol. 3, No. 4, 7–23, 2009.

Часть III

Пример промышленной реализации

Гибкое и допускающее повторное использование приложение для управления производственным процессом

Gernot Kollegger
nxtControl GmbH

Arnold Kopitar
nxtControl GmbH

СОДЕРЖАНИЕ

12.1 Введение	280
12.2 Ожидания решений по автоматизации на основе IEC 61499	281
12.3 Требования к приложениям на основе IEC 61499	283
12.3.1 Требования к приложению автоматизации	283
12.3.2 Реальные компоненты как компоненты программного обеспечения	283
12.4 Мощность атрибутов	284
12.5 САТ: составной тип автоматизации	285
12.5.1 Вложенные компоненты программного обеспечения	285
12.5.2 Подключение аппаратных приложений к I/O	286
12.6 Приложение для управления процессом	288
12.7 Понятие и компоненты библиотек управления процессом	288
12.7.1 Основные важные компоненты	289
12.7.2 Общие режимы работы и режим САТ	289
12.7.3 Усовершенствованные компоненты	290
12.7.4 Внутренняя структура компонента	291
12.8 Применение конвейерных лент	298
12.8.1 Определение и использование адаптера IBeltDrive	299
12.8.2 Экземпляры САТ транспортной линии	300
12.8.3 Отображение приложения	300
12.9 Распределенный подход управления последовательностью	301
12.9.1 Функциональные блоки DSC	302
12.9.2 Отображение приложения DSC	306

12.9.3 Визуализация приложений DSC	306
12.9.4 Приложения DSC в данном проекте	306
12.10 Дозирование и реакторное приложение	307
12.11 Приложение для конфигурации и мониторинга оборудования	307
12.12 Выводы	311

12.1. Введение

Концепция Industry 4.0, широко обсуждаемая в Европе, ставит новые задачи для промышленной автоматизации. Производство станет более гибким, адаптивным, интегрированным и даже «пересечет границы» компании. Все больше компонентов становятся интеллектуальными и интегрируются в широкие сети связи [1]. Автоматизация имеет решающее значение для обеспечения требуемой гибкости, автономности и согласованности данных. Решения для сложных задач в машиностроении также крайне важны, они позволят внедрить в данную сферу современные системы.

При автоматизации таких сложных систем требуется гибкое и простое распределение логики или управления нескольким контроллерам. Именно здесь IEC 61499 демонстрирует свою силу. Кроме того, отделение оборудования от прикладного программного обеспечения является большим плюсом с точки зрения гибкости [2]. Становится возможным независимое от топологии и аппаратное обеспечение. Инструмент разработки должен поддерживать инженера для решения задач распределенного интеллекта, легкой адаптации процессов и согласованности данных от уровня поля до уровня облака.

Объектно-ориентированное проектирование – это ключ к решению проблемы. Поскольку визуализация, такая как человеко-машинный интерфейс (HMI) и/или диспетчерский контроль и сбор данных (SCADA), более или менее востребована во всех проектах, она должна быть интегрирована вместе с подключением I/O из программного приложения к физическому аппаратному обеспечению. Представьте себе инженера, которому нужно запрограммировать все каналы связи между всеми распределенными контроллерами. Интегрированная документация, тестирование, симуляция и поддержка при вводе в эксплуатацию – это другие функции, помогающие инженерам справиться со сложностью проектирования.

Проблема возрастающей важности – подключение к IT-системам. На уровне поля запрашиваются более точные данные, которые должны предоставляться вышестоящим системам управления. С другой стороны, автономные и самонастраивающиеся блоки управления делают производство более эффективным и гибким. Управление процессом и предоставление огромных объемов его данных с помощью интернет-технологий становится важным навыком.

В этой главе будет показано, как объектно-ориентированное и аппаратно-независимое проектирование предназначены для поддержки решений для

самых последних задач и разработки повторно используемых программных объектов для поддержания разумных затрат на проектирование, несмотря на растущие проблемы.

12.2. Ожидания решений по автоматизации на основе IEC 61499

Каковы ожидаемые преимущества при использовании приложений на основе IEC 61499?

Чтобы ответить на этот вопрос, рассмотрим современное состояние для программирования проектов автоматизации. Какие возможности у нас есть для создания программного обеспечения систем автоматизации?

- Запрограммируйте специальную прошивку с нуля для какого-либо оборудования.
- Используйте стандартизированную среду программирования IEC 61131, если она поддерживается производителем оборудования.
- Используйте проприетарную среду программирования от конкретного производителя оборудования.

Все эти решения основаны на восходящем подходе. Это означает, что проекты начинаются с выбранной части оборудования, а прикладное программное обеспечение относится к этому оборудованию. Программное обеспечение может быть повторно использовано только на том же типе оборудования. Аппаратно-независимые части программного обеспечения могут быть повторно использованы в одном семействе оборудования. В некоторых случаях возможно повторное использование одного и того же программного обеспечения на разных аппаратных платформах разных производителей. Стандарт IEC 61131 очень помог установить общий язык программирования для широкого спектра устройств автоматизации. Однако получающееся в результате программное обеспечение не является аппаратно-независимым, и программы автоматизации необходимо адаптировать к другому базовому оборудованию.

Программные инструменты от производителей оборудования сильно различаются, и не существует взаимозаменяемого формата. Даже если они основаны на стандарте IEC 61131, созданный код часто не может быть повторно использован в другой среде программирования или для других аппаратных платформ без переписывания хотя бы его части.

Благодаря поддержке настраиваемости и переносимости, IEC 61499 помогает преодолеть упомянутые выше ограничения [3]. Другой проблемной областью является эффект наложения каналов связи между устройствами автоматизации с циклическим приводом. Ненадежное качество данных, также возникающая в результате этого задержка (по меньшей мере на один цикл) и необходимость запрограммированного механизма для обеспечения качества приводят к очень редкому использованию этого решения в качестве

распределенной системы. Даже если распределенная система будет лучше соответствовать промышленным требованиям, использование централизованной системы часто предпочтительнее. Используются мощные устройства централизованной системы автоматизации в сочетании с системами полевых шин и децентрализованной средой I/O.

Исходя из этой ситуации, каковы ожидания для нового стандарта IEC 61499?

Чтобы перевести существующую кодовую базу в новый стандарт или начать с нового стандарта автоматизации, помимо существующих возможностей, необходимо сосредоточиться на темах, выходящих за рамки уровня техники.

Системный интегратор заинтересован в эффективном проектировании и надежных приложениях. Оба аспекта в значительной степени поддерживаются стандартом и программным обеспечением автоматизации nxtControl.

Для эффективного технического повторного использования уже созданных решений ключевыми факторами являются интеграция нескольких задач автоматизации в одном инструменте разработки и полный набор функций, поддерживающих работу инженера. Распределение контроля увеличивает сложность проектирования даже больше, если визуализация через HMI или SCADA является частью проекта. Инженерные инструменты должны помочь справиться со сложностью в таких приложениях.

Важным шагом к эффективности является объектно-ориентированный подход в машиностроении. Инкапсуляция нескольких аспектов автоматизации (логика управления, визуальное представление, связность I/O, тестирование, моделирование и документирование) реальных устройств и функций в программных объектах имеет важное значение. Эта интеграция, представленная в инструменте проектирования, является основным фактором, способствующим повышению эффективности проектирования, и такие задачи, как HMI и SCADA, устанавливаются очень быстро и легко. Опять же инженерный инструмент создаст пути связи между контроллерами и клиентами визуализации. Второй важный момент: аппаратно-независимая природа этих программных объектов. Благодаря IEC 61499 эта аппаратная независимость легко достигается. Третий момент: это простой способ распределения логики управления на несколько контроллеров простым перетаскиванием. Нет необходимости программировать перекрестную связь между контроллерами. Это делается с помощью инженерного инструмента с учетом методов IEC 61499.

Надежность приложения в значительной степени поддерживается функциями и концепциями, упомянутыми выше. Плавная интеграция различных инженерных задач в инструмент проектирования исключает взаимодействие между различными этапами пирамиды автоматизации. Автоматически устанавливаемые каналы связи между распределенными контроллерами и между контроллерами и несколькими клиентами

визуализации снижают риск ошибок. Наконец, возможность тестировать и моделировать приложение на офисном столе перед запуском гарантирует надежные приложения. Комплексные функции наблюдения и отладки обеспечивают эффективное обнаружение сбоев при вводе в эксплуатацию. Автоматически обновляемая документация приложения обеспечивает надежность в течение всего жизненного цикла.

12.3. Требования к приложениям на основе IEC 61499

12.3.1. Требования к приложению автоматизации

Проект автоматизации должен соответствовать нескольким требованиям.

- Система визуализации должна быть включена.
- Визуализация должна включать элементы управления пользователями, элементы мониторинга и контроля, управление аварийными сигналами, журналирование, анализ тенденций и доступ к историческим данным.
- Несколько клиентов визуализации должны быть подключены параллельно к различным устройствам.
- Приложения должны состоять из легко используемых компонентов.
- Компоненты прикладного программного обеспечения должны быть подключены к различным аппаратным системам I/O.
- Приложение должно быть свободно распространяемым среди меняющегося количества устройств автоматизации.
- Распределенное приложение должно иметь структуру программы самовосстановления.
- Части программы должны быть запрограммированы как логика управления последовательностью.

Некоторые из требований являются только частично выполнимыми на основе концепций IEC 61499. Подход, ориентированный на события, является предпосылкой для запуска распределенных и интегрированных приложений.

12.3.2. Реальные компоненты как компоненты программного обеспечения

Необходимым условием для достижения ожидаемых возможностей является преобразование реальных компонентов в программные компоненты. Эти компоненты построены как функциональные блоки, и они должны объединять все свойства, входы и выходы – такие, как интерфейс и возможности конфигурации, – чтобы отражать реальный компонент во всех ожидаемых состояниях работы. Это включает в себя необходимость эксплуатации и мониторинга состояния объекта реального мира. Подключение к системе визуализации должно быть интегрирующим аспектом в программном компоненте. Тем не менее программные компоненты не должны

зависеть от аппаратной и физической структуры I/O, на которой программное обеспечение выполняется.

Аппаратно-независимые программные компоненты

Чтобы преодолеть ограничение использования программных компонентов, представляющих компоненты реального мира, на выделенном аппаратном обеспечении, функциональные блоки должны реализовать физический уровень интерфейса подключения I/O. Все сигналы должны быть преобразованы в общий диапазон, не связанный с аппаратным обеспечением. Программный компонент в своей структуре должен быть гибким.

Многоразовые программные компоненты

При различных вариантах параметров программные компоненты должны быть настраиваемыми, чтобы отражать различные типы однотипных объектов реального мира без необходимости создавать новый программный компонент, если отличаются только некоторые параметры. В качестве примера, регулируя параметры, программный компонент должен быть способен отражать различные стадии расширения клапанов с концевыми выключателями или без них или изменяя время открытия или закрытия.

Основная работа заключается в создании и следовании концепции библиотеки программных компонентов, которая позволяет нам создавать более сложные программные компоненты путем объединения базовых программных компонентов. Это включает в себя определение соглашений об именах для функциональных блоков и переменных, определения диапазона сигналов, режимы работы, концепцию инициализации программы, частоту обновления состояний и сигналов, временное поведение и т. д. Аналогичная работа должна быть проделана для части визуализации, чтобы определить концепцию пользовательского интерфейса и общее удобство использования с различными типами объектов, включая цветовые схемы, рекомендации по проектированию, рабочие процессы и другие.

12.4. Мощность атрибутов

Атрибуты определены в стандарте IEC 61499. Они используются для объявления метаданных на разных уровнях функционального блока. Эта информация используется, например, для того, чтобы специализировать функциональные возможности функциональных блоков, определять переменные для дополнительного использования и конфигурационную информацию для функциональных блоков, где они используются для обеспечения простого и интуитивно понятного рабочего процесса для пользователя при программировании приложения автоматизации. Фактически атрибуты используются для расширения функциональности и повышения производительности без нарушения правил и определений стандарта.

12.5. CAT: составной тип автоматизации

CAT – это специализированный функциональный блок, применяемый nxtControl. С точки зрения стандарта IEC 61499, он реализует составной функциональный блок. nxtStudio использует этот тип для определения нескольких аспектов программного компонента и объединяет их в один тип. По сути, CAT не ограничивает количество аспектов, которые он объединяет. В настоящее время логика управления (сеть функциональных блоков IEC 61499), визуализация (с символами и лицевыми панелями), аппаратное соединение (концепция SYMLINK) и документация (стандартизированный формат HTML и docbook) включены в CAT.

Для интеграции части визуализации в определение CAT автоматически добавляется SIFB HMI, представляющий интерфейс данных и событий между средой выполнения IEC 61499 и средой HMI. Поверх этого интерфейса создаются символы и лицевые панели с помощью приложений *.Net GUI*. CAT можно использовать иерархически, в структуре CAT-in-CAT.

12.5.1. Вложенные компоненты программного обеспечения

Сложные программные компоненты, такие как двигатели, контуры управления и дозирующие линии, объединяют или интегрируют свои функции иерархическим образом. Они выстроены с использованием более основных функциональных блоков в иерархическую структуру объектов в объектах.

Для создания легко используемых программных компонентов различные типы функциональных блоков объединяются в сети функциональных блоков. Сеть функциональных блоков также может содержать самый основной функциональный блок. Внутренняя реализация базового FB состоит из диаграммы управления выполнением (ECC) (см. рисунок 12.1), которая включает в себя переходы и состояния с действиями и вызовами алгоритмов, определенных в FB. ECC также определяет события, инициируемые в различных состояниях основного FB.

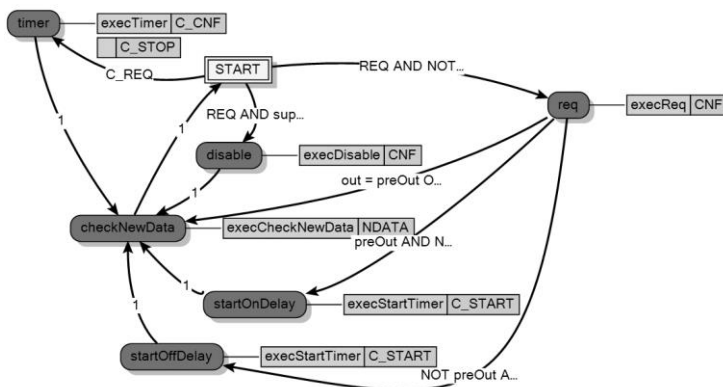


Рисунок 12.1. Базовая диаграмма управления событиями FB

12.5.2. Подключение аппаратных приложений к I/O

Чтобы достичь наивысшей степени повторного использования функциональных блоков, необходим интерфейс для подключения переменных через все иерархии вложенных функциональных блоков, без необходимости выставлять этот интерфейс на каждой границе вложенной иерархии. Иначе это увеличит количество переменных до огромного числа в самом верхнем функциональном блоке.

Тем не менее система автоматизации требует физического I/O. Подключение к I/O связано с аппаратным обеспечением. nxtControl разработал концепцию уровня абстрагирования оборудования, чтобы сконфигурировать сигналы I/O, относящиеся к оборудованию, как восходящую конфигурацию, предоставляя их приложению автоматизации. На рисунке 12.3 (а) показаны аппаратные сигналы I/O. Они могут быть локальными входами непосредственно на устройстве или в настроенной системе полевой шины, такой как EtherCAT, ModBUS или т. д. Каждый элемент оборудования может быть представлен в виде CAT, который позволяет осуществлять мониторинг, ручную настройку (для ввода в эксплуатацию) и настройку в системе визуализации (рисунок 12.3 (b)).

Функциональные блоки интерфейса сервиса Symlink

Для доступа к физическим I/O используется механизм, позволяющий легко связать программное приложение с этими физическими сигналами. Сложные компоненты объединяют функциональные блоки иерархически. Соответствующие сигналы I/O могут потребоваться на разных уровнях вложенности. Эта конструкция требует виртуального символического метода доступа для подключения приложения к аппаратным сигналам в рамках проекта автоматизации. nxtControl реализовал концепцию символической ссылки. Семейство SYMLINK FB – это универсальные сервисные функциональные блоки. Это означает, что число и тип переменных в интерфейсе можно динамически настраивать во время разработки для каждого экземпляра в соответствии с потребностями приложения (рисунок 12.2).

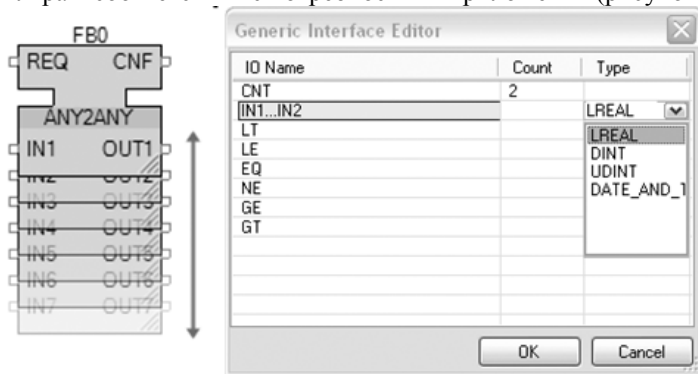
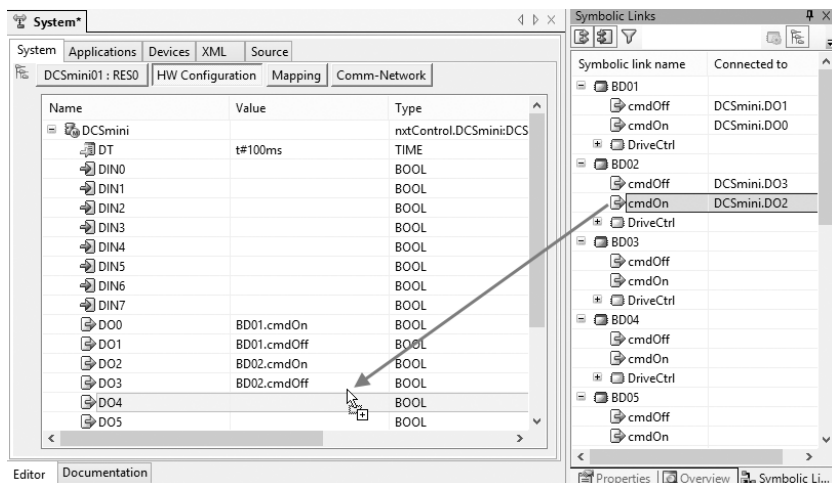
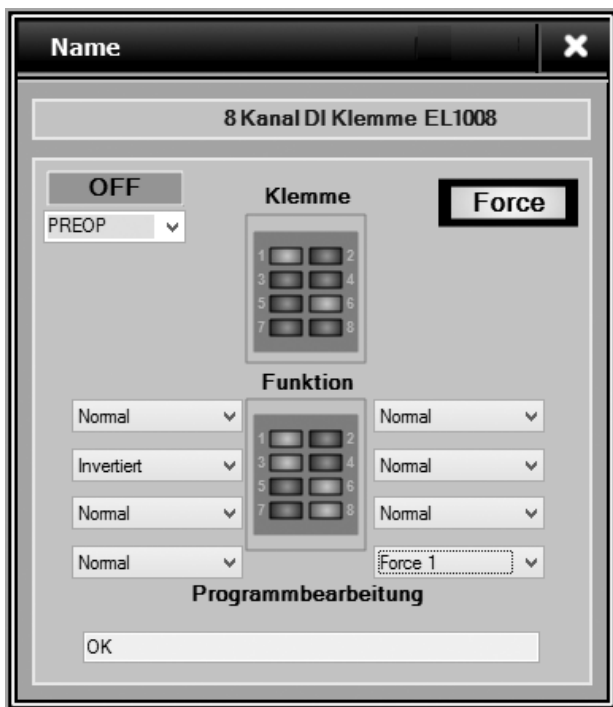


Рисунок 12.2. Общий интерфейс и конфигурация FB

Экземпляры функциональных блоков, использующие сервисный интерфейс FB семейства SYMLINK, будут заполнять все переменные NAME значениями $\${PATH}$ в их списке символических ссылок, где они могут соединять точки данных по имени друг с другом (рисунок 12.3 (а)). Чтобы назначить открытую аппаратную точку приложения физическому I/O, перетащите сигнал из списка символических ссылок на физический I/O.





(a)

(b)

Рисунок 12.3. Конфигурация и мониторинг абстрагирования оборудования:
(a) вид конфигурации; (b) мониторинг

12.6. Приложение для управления процессом

Конструкция состоит из проектов автоматизации для конвейера с лентой, линий управления несколькими дозаторами, реакторного блока с регулировкой температуры, насосных станций с резервуарными блоками и контроллером уровня. Сложные компоненты построены поверх библиотечных элементов, предназначенных для блоков управления процессом.

12.7. Понятие и компоненты библиотек управления процессом

Библиотека состоит из специальной библиотеки управления процессом и библиотеки основных компонентов. Мы рассмотрим некоторые компоненты, чтобы дать общее представление о концепции и структурах этих библиотек.

12.7.1. Основные важные компоненты

Основные функциональные блоки

modeBase используется для вычисления и отображения состояний режима работы.

pt1EC используется как фильтр PT-1 для аналоговых значений.

pidSC предоставляет алгоритм PID для контуров управления.

Композитные функциональные блоки

PLCstart указывает на запуск PLC, обеспечивает процедуру последовательного запуска и предотвращает пакетное событие при запуске ресурса.

TIMEMONITOR используется для контроля временного диапазона, например для контроля времени перехода состояний для двигателей.

CAT-ы

MODE используется в качестве вспомогательного CAT во многих других CAT. Он предоставляет пользовательский интерфейс для мониторинга и управления режимами работы двигателей, клапанов, аналоговых контроллеров и других компонентов.

BINGETVAL, **BINSETVAL** – это очень простые и базовые CAT для мониторинга и контроля двоичных значений. Они используются для простых задач, таких как мониторинг одного бита или установка значения параметра, и не имеют дополнительных возможностей управления.

ANAGETVAL, **ANASETVAL**, **ANASETVALEXT** – это очень простые и базовые CAT для мониторинга и контроля аналоговых значений. Они используются для простых задач, таких как мониторинг или установка аналоговых значений; не имеют дополнительных возможностей управления.

MATHCURVE может вычислить выходное значение с помощью математического выражения с одним параметром.

12.7.2. Общие режимы работы и режим CAT

Общая концепция и простое понимание режимов работы важны для принятия библиотек пользователями. Режим работы определяет, как компонент работает вообще и как он работает в приложении. Библиотека процессов определяет различные режимы работы для мониторинга состояний компонентов и управления компонентами. Некоторые из режимов работы могут быть выбраны оператором; другие предоставляют информацию и не могут быть изменены оператором.

Информативные режимы

LOOV – **локальное переопределение**. Программные функции компонента переопределяются сигналами в реальных компонентах. Это может быть в том

случае, если на электрическом шкафу есть физический выключатель для включения двигателя. Программное управление не имеет никакого эффекта. Доступные сигналы обратной связи могут использоваться для отслеживания реального состояния в программном компоненте.

EMAN – принудительное руководство. Управляющий сигнал переводит компонент в определенное состояние. Например, если клапан закрыт, двигатель не должен запускаться и принудительно отключаться.

EMAN – внешнее руководство. Состояния уставок компонента контролируются физическими сигналами вручную и не могут быть изменены оператором с помощью системы визуализации.

Режимы, выбираемые оператором

MAN – руководство. Состояния вывода компонента устанавливаются оператором с помощью системы визуализации. Алгоритмы управления не выполняются, а ограничения сигнала и функции блокировки активны.

AUTO – автоматический. Выходные состояния компонента рассчитываются по алгоритму компонента. Заданное значение для рассчитанного выхода обеспечивается оператором с помощью системы визуализации.

CAS – каскад. Выходные состояния компонента рассчитываются по алгоритму компонента. Заданное значение для рассчитанного выхода обеспечивается программным сигналом из другого функционального блока компонента.

12.7.3. Усовершенствованные компоненты

Основные функциональные блоки

stateCheckBin вычисляет текущее состояние двоичного управляемого компонента, такого как двигатель или клапан. Можно использовать до восьми входных сигналов, чтобы отразить максимум пять различных состояний двоичного компонента.

stateControlBin рассчитывает выходные сигналы двоичного управляемого компонента, такого как двигатель или клапан. Можно настроить до восьми выходных сигналов для максимум пяти различных состояний двоичного компонента.

binDev1SEC вычисляет запрошенное заданное значение двоичного управляемого компонента, такого как двигатель или клапан. Включает в расчет режим работы, разрешенные состояния, сигналы блокировки и местного переопределения.

StateDevAnaEC рассчитывает аналоговый выходной сигнал. Включает в себя режим работы, блокировки, пределы выхода и настройки скорости изменения выхода.

CAT-ы

DSCBINDEV1S использует функциональные блоки для двоичных компонентов для управления мониторингом режима работы, для сигнализации и мониторинга перехода и объединяет их в CAT для управления и контроля двоичных устройств с одним активным состоянием. Такими устройствами могут быть двигатели (выкл. / вкл.) и клапаны (закрыть / открыть).

DSCBITCTRL просто управляет двоичным выходным сигналом без контроля изменения состояния. Он использует функциональные блоки для контроля двоичных сигналов, а также для мониторинга и управления режимом работы.

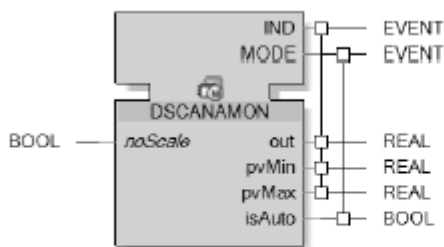
BITMON контролирует двоичные входные сигналы. Он использует функциональные блоки для контроля двоичных сигналов, сигнализации и для мониторинга и управления режимом работы.

DSCANAMON, **DSCANACTRLPID**, **DSCANALIMITMON**, **DSCANASP** и **DSCANAPULSCOUNT** отслеживают и контролируют аналоговые значения. Они используют функциональные блоки для расчета аналоговых сигналов, контроля и управления рабочим режимом и сигнализацией и объединяют их для создания CAT в целях контроля и мониторинга значений аналоговых процессов.

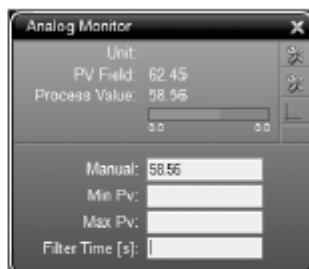
12.7.4. Внутренняя структура компонента

Структура и реализация логики управления будут объяснены на основе CAT **DSCANAMON** и **DSCBITCTRL**. Другие компоненты библиотеки следуют той же концепции и правилам реализации.

Аналоговый монитор CAT DSCANAMON



(a) Интерфейс FB



(b) Мониторинг лицевой панели

Рисунок 12.4. CAT **DSCANAMON**

На рисунке 12.4 (a) показан внешний интерфейс и связь между данными и событиями для функционального блока CAT, а также лицевая панель HMI для мониторинга и управления. Цель этого CAT – считывать аналоговые значения

из физического источника I/O и масштабировать исходное значение до правильного диапазона, чтобы отразить физически измеренное значение. Таблица 12.1 дает обзор переменных интерфейса и событий функционального блока CAT.

Внутренняя логика CAT использует функциональный блок PLStart для последовательной инициализации времени выполнения во время фазы запуска устройства. Для считывания значения процесса используется служебный функциональный блок SYMLINK, для которого в качестве параметра имени используется $\${PATH}\backslash vPct$. Функциональный блок ioScale используется для масштабирования нормализованного значения процесса от процентов (0–100 %) до настроенного диапазона денормализованных значений процесса, например, от 0 до 10 бар. Значения диапазона берутся из функционального блока DSCANAMON_HMI, который является интерфейсом связи между средой выполнения HMI и средой выполнения IEC 61499. Вычисленное значение процесса затем передается в функциональный блок фильтра PT1. PT1 управляет циклическим таймером, чтобы запускать или останавливать циклический расчет значения $PT1.out$. Чтобы уменьшить события в системе, вычисление PT1 останавливается, как только отклонение от вычисленного $PT1.out$ по сравнению со значением процесса $PT1.pv$ составляет $< 0,01\%$. MODE CAT используется в качестве вспомогательного CAT в DSCANAMON для обеспечения общих режимов работы. Для этого разрешены CAT, MAN и AUTO. В зависимости от выбранного режима вычисленное и отфильтрованное значение процесса $PT1.out$ или ручное значение $IThis.manPV$ из функционального блока интерфейса HMI передаются в переменную out CAT.

Таблица 12.1. Переменные интерфейса DSCANAMON

Направление	Название	Тип	Описание
Вход (Input)	noScale	BOOL	Параметр; масштабирование не требуется, поскольку необработанное значение уже является значением процесса в физических единицах
Выход (Output)	IND	EVENT	Событие генерируется при любом изменении вычисленного значения аналогового монитора
Выход (Output)	MODE	EVENT	Событие запускается при любом изменении режима работы
Выход (Output)	out	REAL	Вычисляет аналоговое значение, используемое другими функциональными блоками в сети функциональных блоков
Выход (Output)	pvMin, pvMax	REAL	Настройка диапазона аналоговых значений может использоваться другими

			функциональными блоками, например для автоматической настройки рабочего диапазона контура управления PID
Выход (Output)	isAuto	BOOL	Один указывает, что функциональный блок работает в автоматическом режиме, а выходное значение выводится из значения физического процесса

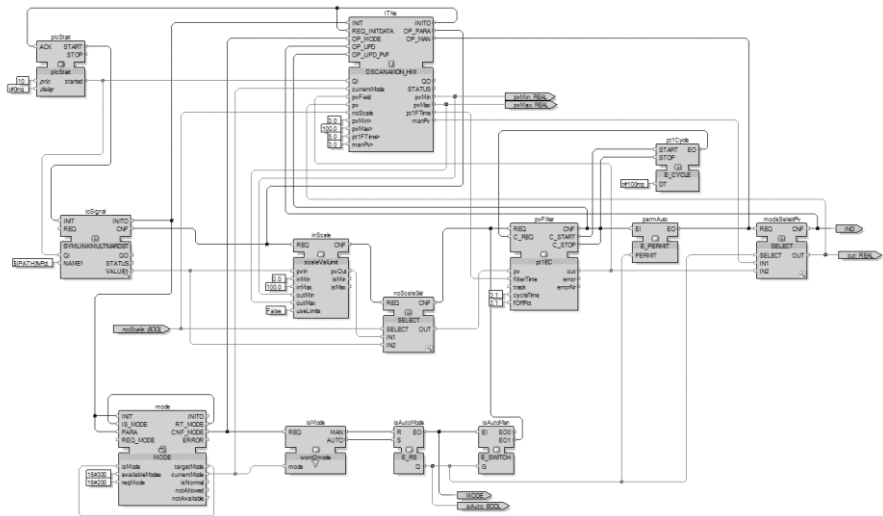
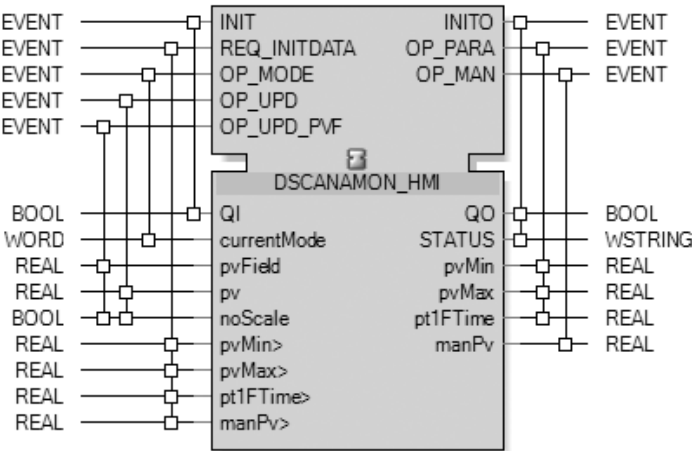


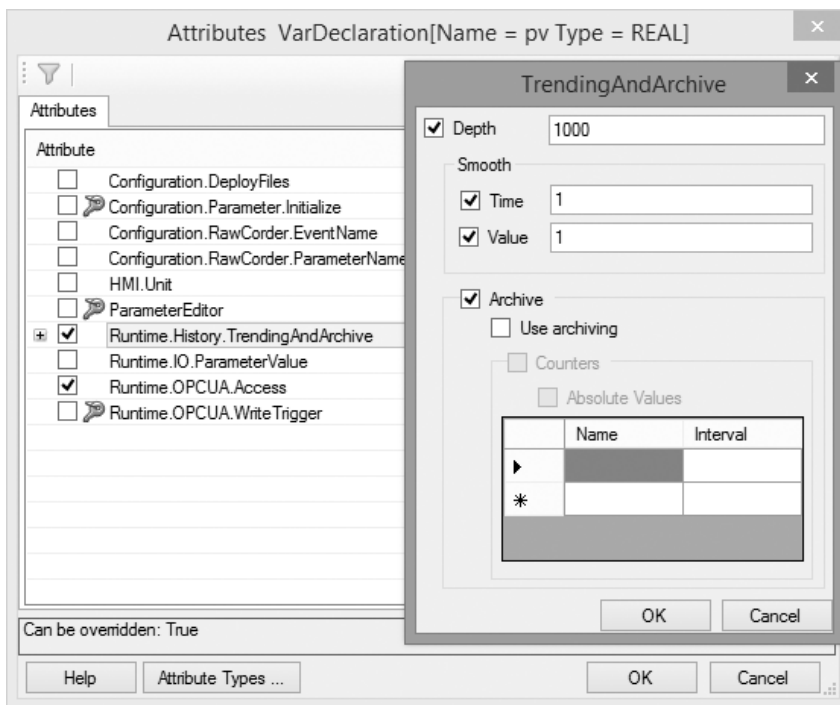
Рисунок 12.5. Логика внутреннего контроля CAT DSCANAMON
Функциональный блок HMI Service Interface внутри CAT

Функциональный блок интерфейса HMI CAT является точкой входа связи между клиентом визуализации и средой выполнения IEC. Он используется не только для целей визуализации, но также для настройки постоянных пользовательских параметров и дополнительной метаинформации для системы подсчета времени IEC 61499 и инженерной среды. Связь с HMI строго следует модели, ориентированной на события. Это видно по входным событиям (см. рисунок 12.6 (а)) для изменения режима (OP_MODE), контроля значений процесса (OP_UPD_PVF), контроля расчетных значений (OP_UPD), а также выходных событий для настройки параметров (OP_PARA) и работы ручных значений (OP_MAN). Переменные в интерфейсе могут быть определены как постоянные; это позволит установить значения по умолчанию (если не существует постоянного значения) с переменными `%inputname%` и загрузить постоянные сохраненные значения для переменных при запуске устройства. Когда оператор изменяет значения через HMI, эти значения сохраняются и перезагружаются при загрузке устройства.

Инженер может определить атрибуты для каждой переменной (см. рисунок 12.6 (б)). В этом примере данные активируются для переменной *pv*. Значения атрибута, определенного на уровне типа, могут быть переопределены в каждом отдельном экземпляре DSCANAMON CAT. Это позволяет хранить выборочные данные и помогает рассчитать необходимое хранилище базы данных и пропускную способность связи в системе автоматизации.

Лицевые панели (рисунок 12.4 (б)) и символы (рисунок 12.9 (а)) для HMI разработаны в верхней части этого интерфейса и могут использоваться для создания представлений процесса для мониторинга и управления.





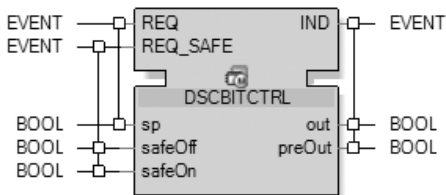
(a)

(b)

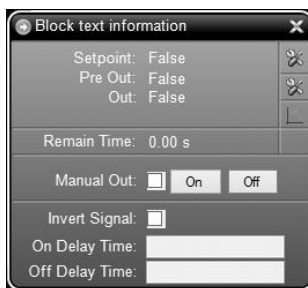
Рисунок 12.6. Интерфейс CAT DSCANAMON HMI: (a) интерфейс CAT HMI; (b) атрибуты CAT

Двоичное командное управление CAT DSCBITCTRL

На рисунке 12.7 (a) показан внешний интерфейс и связь данных / событий для этого функционального блока CAT, а также лицевая панель HMI для мониторинга и управления. Целью этой CAT является контроль двоичных сигналов. В таблице 12.2 представлен обзор переменной интерфейса и событий функционального блока CAT.



(a)



(b)

Рисунок 12.7. Приложение и представление HMI CAT DSCBITCTRL: (a) интерфейс CAT; (b) лицевая панель CAT

Внутренняя логика CAT, показанная на рисунке 12.8, имеет структуру, аналогичную описанному ранее DSCANAMON. Он использует функциональный блок PLCstart для последовательной инициализации времени выполнения во время фазы запуска устройства. Вход SP используется в качестве заданного значения в автоматическом режиме. *BinMonBasicFB* используется для вычисления контрольного значения. Время задержки может быть параметризовано для расчета выходного значения. Предварительное выходное значение рассчитывается без каких-либо ограничений по времени задержки. MODE CAT используется в качестве вспомогательного CAT в DSCBITCTRL для обеспечения общих режимов работы. Для CAT разрешены режимы MAN и AUTO. В зависимости от выбранного режима заданное значение для расчета выхода берется из переменной *sp* или из ручного значения *IThis.manPV* из функционального блока интерфейса HMI. Функциональный блок HMI, как обычно, используется для связи с системой визуализации и конфигурирования постоянных значений и определения атрибутов для архивирования и других функций.

Таблица 12.2. Переменные интерфейса DSCANAMON

Направление	Название	Тип	Описание
Вход (Input)	REQ	EVENT	Если запущено, вызвать нормальный алгоритм управления
Вход (Input)	REQ_SAFE	EVENT	Если запущено, вызвать алгоритм для операций безопасности
Вход (Input)	sp	BOOL	Заданное значение используется в автоматическом режиме
Вход (Input)	safeOff	BOOL	Переводит CAT в рабочий режим FMAN и отключает управление битами
Вход (Input)	safeOn	BOOL	Переводит CAT в рабочий режим FMAN и включает управление битами
Выход (Output)	IND	EVENT	Событие генерируется при любом изменении рассчитанных выходных значений управления битами
Выход (Output)	out	BOOL	Вычисляет выходное значение (после любого заданного времени задержки) для контроля битов CAT

Выход (Output)	preOut	BOOL	Вычисляет предварительное выходное значение (настроенное время задержки не учитываются) для управления битами CAT

Этот CAT и большинство других из библиотеки не содержат функциональных блоков SYMLINK для доступа к аппаратным значениям. CAT разработаны в обобщенном виде и подготовлены для использования в качестве вспомогательных CAT в более специализированных CAT. Например, клапан может иметь один или два физических выходы, может иметь физические входы, такие как концевые выключатели, или может реализовывать логику для внешнего ручного управления и т. д. Но базовой логикой управления является CAT DSCBINDEVIS, используемый как вложенный компонент в специализированной CAT. Выделенная реализация использует блоки служебных функций SYMLINK для чтения и записи в значения физических процессов.

12.8. Применение конвейерных лент

Линии ленточного конвейера различаются по размеру и количеству сегментов. Каждый сегмент содержит двигатель для привода ремня. Несколько сегментов сгруппированы в зону. Если ремни зоны запускаются, предупреждающий звуковой сигнал и свет работают перед каждым запуском. Зона состоит как минимум из одного сигнала безопасности для остановки ремней в случае аварии.

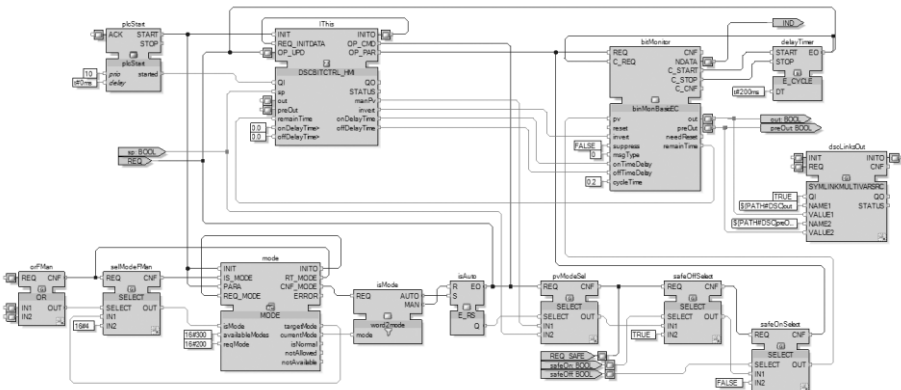


Рисунок 12.8. Логика внутреннего контроля CAT DSCBITCTRL



Рисунок 12.9. Примеры символов CAT

Транспортерная линия с ременным приводом должна быть запущена последовательно. Это означает, что последний ремень должен быть запущен первым, и после того, как этот ремень будет запущен, может запускаться предыдущий ремень и т. д. Если по какой-либо причине ремень останавливается, все предыдущие ремни необходимо немедленно остановить.

Поскольку расстояние ременной транспортировки может быть большим, необходимы распределенные электрические шкафы для питания ременных двигателей. Вместо использования централизованного PLC с распределенной системой I/O, каждый шкаф должен содержать небольшой PLC для управления и контроля двигателей, назначенных шкафу. Следовательно, система управления должна поддерживать распределенный подход. Кроме того, решение должно быть легко расширяемым и модифицируемым. Помимо этих требований, должны быть реализованы следующие программные компоненты.

CAT BELT_CONTROL контролирует линию ремня и имеет индикации запуска, остановки и состояния.

CAT BELT_DRIVE — это ременный двигатель с выделенными физическими сигналами I/O. Внутренне он использует CAT DSCBINDEV1S в качестве вспомогательного CAT для управления и контроля двигателя.

CAT BELT_SAFESTOP реагирует на запрос безопасной остановки. Кроме того, состояние программного обеспечения отслеживается и принудительно выполняется в соответствии с запросом безопасной остановки.

CAT BELT_STARTSIG используется для представления единицы сигнала для зоны конвейерной ленты. Перед каждым запуском звуковой сигнал звучит определенное время. Кроме того, сигнальные лампы используются для индикации состояния зоны пояса для работников в опасной зоне.

FB beltStartState использует основной функциональный блок для генерации сигналов для звукового сигнала и сигнальных огней и используется в CAT BELT_STARTSIG.

Адаптер IBeltDrive соединяет ремни CAT друг с другом, чтобы построить конвейерную линию.

Ремни CAT разработаны таким образом, чтобы их можно было легко использовать и подключать вместе с адаптерами. Интерфейс адаптера состоит из всех данных, необходимых между CATS BELT для построения логики управления линией. Каждый компонент содержит полную логику управления для управления собой и предоставления запрошенных данных соседним компонентам. Это приводит к автоматически программируемому и, следовательно, неограниченному масштабируемому приложению.

12.8.1. Определение и использование адаптера IBeltDrive

Основным решением является использование адаптера для соединения отдельных компонентов. Можно представить соединение адаптера как своего рода виртуальную шинную систему между программными компонентами. Сигналы транспортируются и пересылаются из одного компонента в другой. Каждый компонент считывает и/или изменяет данные или производит и активно отправляет сигналы на шину. На рисунке 12.10 показаны подробности реализации того, как интерфейс адаптера используется в компоненте BELT_DRIVE.

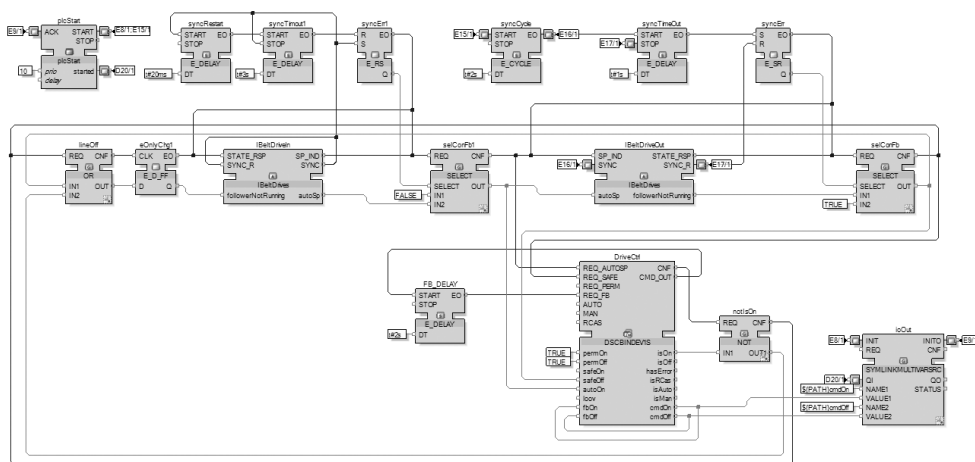


Рисунок 12.10. Логика внутреннего контроля CAT BELT_DRIVE

CAT содержит FB PLCstart для последовательной инициализации времени выполнения во время фазы запуска устройства. Адаптер состоит из заданного значения и информации о состоянии, а также событий синхронизации. События синхронизации очень важны, так как они используются для создания распределенного приложения, которое будет отображаться на нескольких устройствах. Поскольку состояние устройств в сети и работоспособность сети не могут быть гарантированы, необходим своего рода механизм поддержания активности и синхронизации. Если соединение связанных устройств потеряно, ленточным двигателям необходимо перейти в безопасное состояние, чтобы предотвратить повреждение транспортной линии.

Благодаря использованию компонентов библиотеки остальные части логики просты и состоят из SYMLINK для сигналов управления; обратная связь создается внутренне, и DSCBINDEVIS CAT используется для управления двигателем.

12.8.2. Экземпляры CAT транспортной линии

Используя реализованные программные компоненты, легко построить конвейерную линию. Необходимо создать хотя бы один элемент каждого CAT. BELT_CONTROL должен быть первым, за которым следует один или несколько экземпляров типа BELT_DRIVE. Последние два экземпляра должны иметь тип BELT_SIGSTART, за которым следует тип BELT_SAFESTOP. Тем не менее все экземпляры этого типа могут быть размещены в любом месте линии для создания зон безопасности. Экземпляр типа BELT_SIGSTART или BELT_SAFESTOP всегда отвечает за все предыдущие экземпляры типа BELT_DRIVE. Рисунок 12.11 должен прояснить структуру конвейерной ленты.

Экземпляры этого приложения отображаются в трех блоках управления и взаимодействуют как распределенное приложение. Благодаря модели выполнения, управляемой событиями, необходимая загрузка процессора и пропускная способность сети очень низки. Приложение HMI (Рисунок 12.16 (b)) позволяет оператору контролировать линию транспортировки ленты.

12.8.3. Отображение приложения

Приложение управления конвейерной лентой сопоставлено с тремя устройствами (см. рисунок 12.11, Блоки управления с 1 по 3). Коммуникационная сеть между устройствами создается автоматически с помощью инструмента разработки. Все приложение будет проанализировано, когда сопоставление будет выполнено. Если идентифицировано соединение двух функциональных блоков, сопоставленных с различными устройствами, в приложение для задачи связи должна быть добавлена дополнительная логика.

Связь происходит по протоколу Ethernet UDP между устройствами. Инструмент разработки автоматически добавляет в приложение функциональные блоки отправки и получения для соединений данных и событий и настраивает их в соответствии с сетями распределенных функциональных блоков. Конфигурация обеспечивает передачу данных и событий только в том случае, если интерфейс остается неизменным с обеих сторон. Подписи для событий и проводов данных являются частью протокола связи. Если подписи не совпадают, сообщение будет отброшено, чтобы предотвратить ненадежную или непоследовательную передачу данных и событий между устройствами. Ориентированный на события подход IEC 61499 и реализация из nxtControl дает возможность передавать события и данные через распределенную среду без эффектов наложения, а также надежным и быстрым способом.

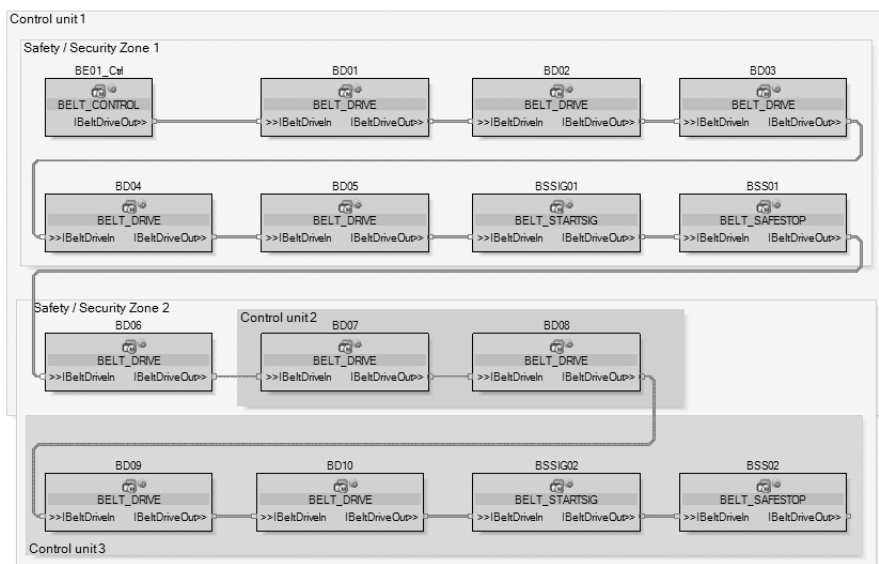


Рисунок 12.11. Применение пояса

Для программы управления конвейерной лентой было легко распределить приложение по нескольким устройствам. Однако для обеспечения надежной среды в функциональные блоки была включена некоторая логика контроля времени ожидания. Эта логика постоянно проверяет связь между компонентами, и, если время ожидания слишком велико, ременные приводы переходят в безопасное состояние, чтобы предотвратить повреждение. Как только связь между программными компонентами может быть снова проверена, приложение переходит в автоматический режим, как определено для логического управления конвейерной лентой.

12.9. Распределенный подход управления последовательностью

Общей задачей программ автоматизации является последовательное выполнение команд для клапанов, приводов, регуляторов и других компонентов. Для таких задач стандарт IEC 61131 предлагает язык SFC. Он может использоваться для отдельных монолитных функциональных блоков, но нет способа распределить SFC. Чтобы использовать возможности SFC в распределенном подходе, необходима концепция разделения аспектов SFC на функциональные блоки, которые будут использоваться в качестве частичных сетей и, следовательно, могут распространяться поверх множества устройств.

Цель состояла в том, чтобы создать логику распределенного последовательного управления (DSC), которую можно частично сравнить с реализацией SFC. Должна быть возможность формализовать алгоритмы последовательного управления, охватывающие несколько устройств, для создания программы последовательного управления. Также должна быть возможность соединить частично созданные DSC в законченную программу DSC путем соединения частей DSC с адаптерами. Концепция DSC должна быть реализована так, чтобы последовательная программа инициализировалась во время запуска, чтобы определить структуру всей реализации DSC.

12.9.1. Функциональные блоки DSC

Следующие FB в основном используются для создания программы DSC.

CAT DSC_HEAD является основным функциональным блоком управления для приложения DSC.

CAT DSC_STEP представляет собой шаг в программе DSC. Он также содержит прямую оценку транзакции, которая должна следовать за шагом.

CAT DSC_TRANSACTION представляет альтернативную транзакцию для DSC. Он используется для запуска альтернативной ветви программы DSC.

CAT DSC_ACTION_ENTER представляет действие для выполнения. FB выполняется, когда шаг активирован. Несколько действий могут быть добавлены к одному шагу.

Composite cpDscParallelAlloc используется для создания параллельной ветви в программе DSC.

Composite cpITransJumpBackCmd создает точку входа для операции возврата назад (эквивалент структуры цикла).

Адаптер IDscChain соединяет шаги DSC с FB управления потоком DSC.

Адаптер IDscParallelAllocChain соединяет параллельные ветви FB.

Адаптер IDscParallelMergeChain соединяет параллельные ветви объединения FB.

Адаптер IAction соединяет функциональные блоки с соответствующим шагом.

Адаптер ITransaction соединяет альтернативные транзакции FB с соответствующим шагом.

Адаптер ITransJumpBack создает операцию возврата.

Интерфейсы адаптера активно используются для построения структуры последовательности и связи в DSC. FB используют SIFB, чтобы определить, подключены ли интерфейсы адаптера к интерфейсу FB. В зависимости от результата логика строит последовательность управления и поток сообщений через программу DSC.

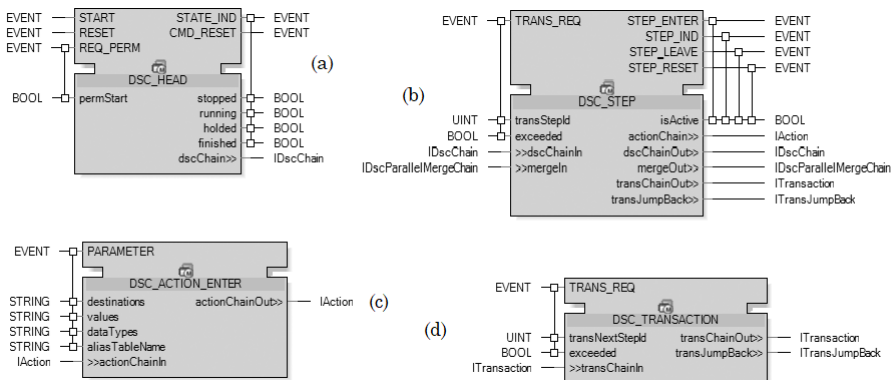
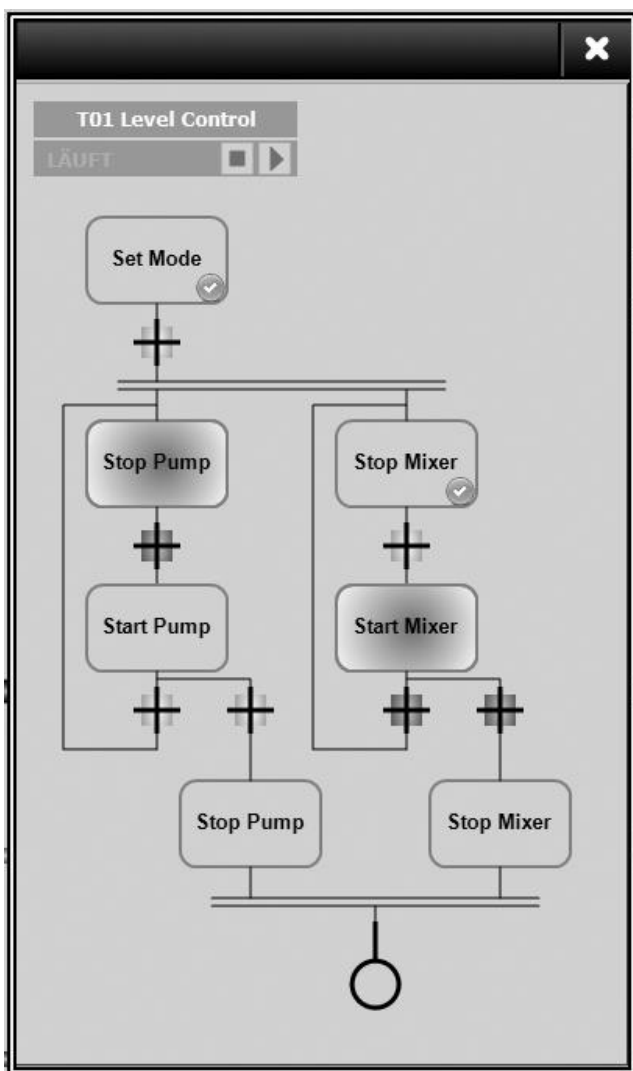


Рисунок 12.12. Интерфейс функционального блока DSC CAT: (a) DSC Head FB; (b) DSC Step FB; (c) DSC Action FB; (d) DSC Transaction FB

FB DSC_HEAD является первым функциональным блоком в программе DSC и используется для инициализации структуры последовательности и управления рабочим состоянием последовательности. Инициализация означает подсчет количества функциональных блоков и проверку правильности структуры последовательности. Несколько альтернативных символов и лицевых панелей используется для мониторинга и контроля рабочих состояний последовательности. Интерфейс FB имеет события и входы данных для управления состояниями от других FB или для предотвращения запуска последовательности. Последующие FB подключаются к выходу адаптера *dscChain*.

Шаг DSC определяется FSC DSC_STEP. Он имеет интерфейс ввода и вывода адаптера *dscChain* для подключения других шагов или композитов управления потоком. Композиты управления потоком также могут использовать адаптеры *mergeIn*, *mergeOut*, *transChainOut* и *transJumpBack*. Наиболее важным для выдачи команд шагов является адаптер *actionChain*, с помощью которого FB DSC_ACTION подключаются и выполняются одним шагом. В будущем могут появиться другие FB DSC_ACTION; на данный момент доступен только FSC DSC_ACTION_ENTER. Этот FB будет вызываться и выполняться, когда подключенный шаг активирован. Через *actionChainIn* и *Action-ChainOut* несколько FB DSC_ACTION_ENTER могут быть подключены к шагу FB. Позиция в цепочке действий также является порядком, в котором будут выполняться действия FB. Помимо адаптера, SYMLINK FB являются другими основными компонентами DSC. Объекты подготавливаются с внутренним интерфейсом (в основном той же идеей, что и для аппаратных подключений) для выдачи команд и чтения состояний из компонентов с поддержкой DSC. Например, DSCBINDEV1S состоит из интерфейса DSC для *setMode*, *rcasOn*, *currentMode*, *hasError*, *isOff* и *isOn*. Это позволяет программе последовательности DSC записывать управляющие команды и считывать информацию о состоянии для определений транзакций.



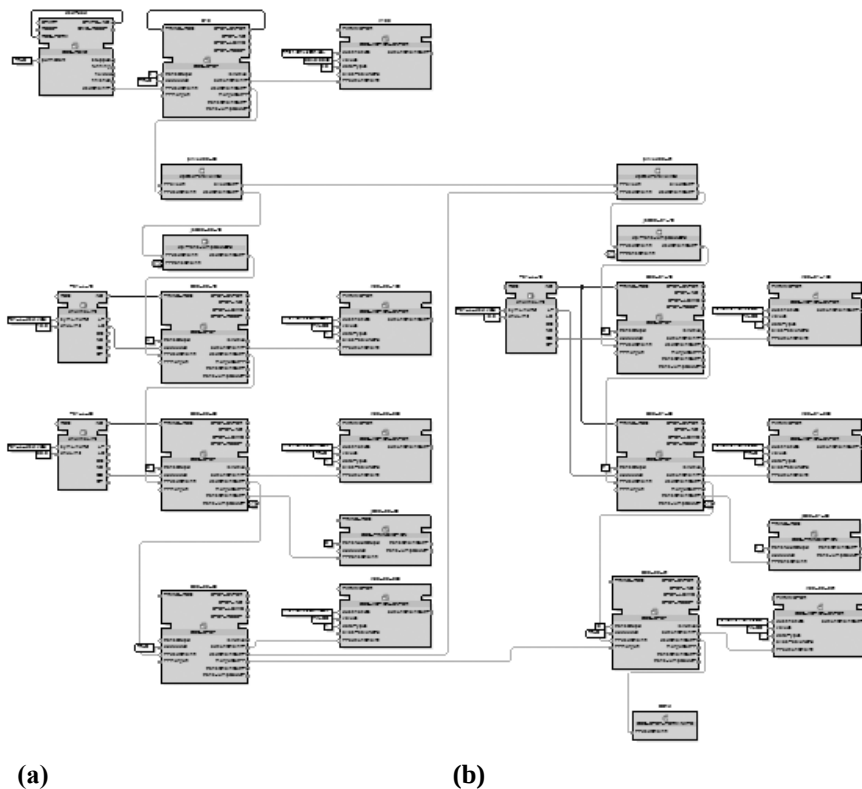


Рисунок 12.13. Последовательность DSC, HMI и логическое представление: (a) лицевая панель; (b) сеть

Переменные интерфейса *destinations*, *values*, *dataTypes* определяют команды действий, которые будут отправлены. Все переменные являются разделенными запятыми списками строк. Каждый элемент соответствует команде. Количество элементов – это количество команд, отправленных этим FB. Все три переменные должны иметь одинаковое количество элементов. Переменная адресаты – это список адресов, на которые должны быть отправлены команды.

Например, `P01.DriveCtrl.setMode;T01_M01.DriveCtrl.setMode` отправит команды режима изменения в `TP01.DriveCtrl` и `T01_M01.DriveCtrl`.

Переменная *values* содержит список значений, которые следует отправить элементу, указанному в переменной назначения. Значение указывается в виде строки и должно быть преобразовано в целевой формат. Чтобы найти правильный формат, переменная *dataTypes* определяет пронумерованные типы данных, отвечающие за значение, которое будет соответствующим образом преобразовано и впоследствии отправлено целевому объекту.

Также возможно создать полностью повторно используемую последовательность DSC, но для этого требуется интерактивная замена

переменных экземпляра для *destinations*. Для этого доступна дополнительная переменная *aliasTableName*. При использовании этого подхода можно использовать заполнители в форме *%VALVE%.setMode* в определении мест назначения. Заполнитель будет решен путем запроса пары ключ / значение, на которую ссылается имя, указанное в *aliasTableName*. Это расширенное использование программы DSC не встречается в описанном проекте.

12.9.2 Отображение приложения DSC

Взаимосвязь между функциональными блоками приложения DSC осуществляется с помощью событий, данных и подключений адаптера. Исходя из этого, можно легко распространять программу, просто сопоставив функциональный блок программы DSC с одним или несколькими устройствами. Перекрестные коммуникации генерируются автоматически с помощью инструмента разработки *nxtStudio*. Существует ограничение, согласно которому части приложения DSC с *SYMLINK* должны быть сопоставлены с устройством, на котором находится целевой объект *SYMLINK*. Ссылки *SYMLINK* работают только на одном устройстве.

12.9.3. Визуализация приложений DSC

Как обычно в *nxtStudio*, CAT для последовательностей DSC также имеют реализации в виде символов и лицевых панелей. Чтобы создать графическое представление программы DSC, нам нужно только добавить символы на холст и нарисовать соединительные линии для представления последовательности программ. Если программа DSC инкапсулирована в новый CAT с FB DSC в качестве sub-CAT, лицевая панель или символ могут быть созданы для этого CAT таким же образом. В этом случае последовательность можно отслеживать и контролировать с помощью символов и лицевых панелей (рис. 12.13 (a)).

12.9.4. Приложения DSC в данном проекте

В этом проекте последовательности DSC используются для контроля подачи извести в известковый резервуар до того, как он достигнет производственной линии. Другая последовательность DSC контролирует процесс дозирования извести, воды и химикатов, процесс нагревания и охлаждения, а также последовательность слива из реактора. Управляющая последовательность DSC запускается, останавливается и контролирует последовательности как упрощенный пакетный процесс.

12.10. Дозирование и реакторное приложение

Даже если программа управления процессом использует индивидуальную логику и не может быть повторно использована в разных проектах, части процесса основаны на одних и тех же концепциях и операциях. Эти части можно использовать повторно, дублируя их для аналогичной работы. В этом проекте подпроцесс дозирования используется несколько раз для производства конечного продукта. Поэтому логично создать собственный CAT, который можно использовать несколько раз для отдельных составляющих. Объектно-ориентированный подход функциональных блоков обеспечивает основные технологии для этой работы.

Применение дозирующего устройства

Приложение дозирования CAT *DOS_UNIT* является стандартизированным компонентом, который включает датчики и исполнительные механизмы в качестве вспомогательных CAT. Для каждого экземпляра дозирования была создана отдельная последовательность управления DSC. *DOS_UNIT* состоит из датчика расходомера и концевых выключателей, счетчика количества импульсов и клапана. Счетчик количества импульсов также может обеспечить логику управления процессом дозирования, поскольку он обеспечивает управление заданным значением. Логика IEC 61499 для датчиков и механизмов включает в себя все логические и управляющие алгоритмы, необходимые для работы дозирующего устройства. Помимо автоматического режима, каждый компонент имеет ручной режим, позволяющий операторам использовать каждый отдельный элемент вручную и в удаленном автоматическом режиме, который используется распределенным управлением последовательностью для использования компонентов в приложении DSC. На рисунке 12.16 (b) показана часть приложения с визуализацией и лицевые панели для счетчика и контрольной диаграммы DSC.

Программа DSC рассчитывает дозы для каждого отдельного ингредиента и переносят его в реактор. Программа последовательности используется в реакторе для управления температурой и дозированием, для добавления правильного количества ингредиентов и контроля времени их обработки в реакторе.

12.11. Приложение для конфигурации и мониторинга оборудования

Две трудоемкие задачи в рамках проекта – это определение конфигурации оборудования контроллера и подсистемы I/O. Обычно определение делается на ранней стадии проекта. В рамках распределенного проекта, использующего IEC 61499 и *nxtControl*, эту задачу не нужно выполнять на первом этапе. Однако концепция распределенного проекта требует разделения приложения технологической системы и определения аппаратного обеспечения. Концепция

SYMLINK позволяет инженеру подключать аппаратное обеспечение к системному приложению для взаимодействия с реальным миром.

Определение I/O является перевернутой концепцией. По этой причине конфигурация оборудования добавляется к ресурсу устройства. Например, для поддерживаемых аппаратных устройств, шинного соединителя и зажимов I/O, CAT доступен в выделенных аппаратных библиотеках.

Этот проект построен на подсистеме I/O Profibus. В проект была добавлена библиотека для Profibus, благодаря которой поддерживаемые модули были доступны для настройки и добавления в программу. На рисунке 12.14 показан вид конфигурации аппаратного обеспечения инженерной системы. Варианты I/O состоят из обычных цифровых и аналоговых модулей ввода и вывода, а также специальных модулей для пускателя двигателя и датчиков веса. Аппаратное обеспечение основано на модульной системе I/O с соединителем Profibus и зажимами для различных сигналов. Каждый соединитель настраивается путем выбора параметров для каждого зажима в соответствии с добавленными модулями. Полученный файл конфигурации используется на этапе запуска контроллера мастером Profibus для взаимодействия с подсистемой I/O Profibus.

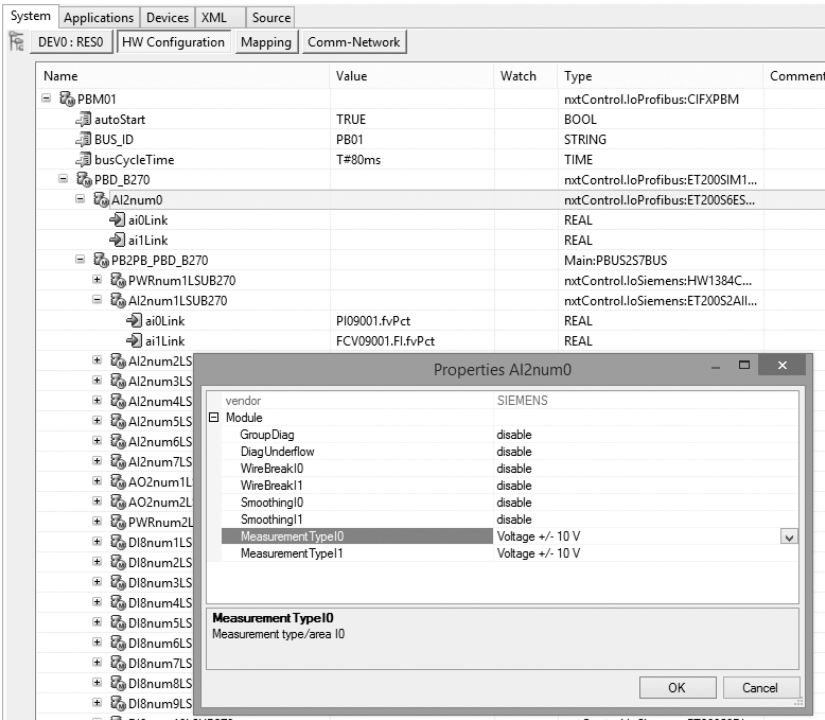


Рисунок 12.14. Представление конфигурации оборудования DSC

После добавления аппаратных модулей в проект можно создать графическое представление, добавив холсты для аппаратного представления и разместив символы модулей I/O на холсте. Аппаратные САТ представляют физическую точку I/O, они помогают системному интегратору легко настроить аппаратную среду, а техническим специалистам помогают на этапе ввода проекта в эксплуатацию.

Для этого аппаратные САТ-файлы состоят из символов и лицевых панелей. Лицевые панели используются для контроля сигналов, входных зажимов и сигналов управления выходными зажимами. Кроме того, можно интерактивно устанавливать параметры для манипулирования подключенными сигналами I/O, например инвертировать цифровые входы. На этапе ввода в эксплуатацию проекта аппаратный САТ может использоваться для быстрой проверки соединений I/O в реальном мире путем мониторинга цифровых и аналоговых входных сигналов и моделирования сигналов приложения для цифровых и аналоговых выходных сигналов. Для проверки оборудования проекта требуется только приложение визуализации (рисунок 12.15) для оборудования. Техники могут просто использовать визуализацию для своей работы.

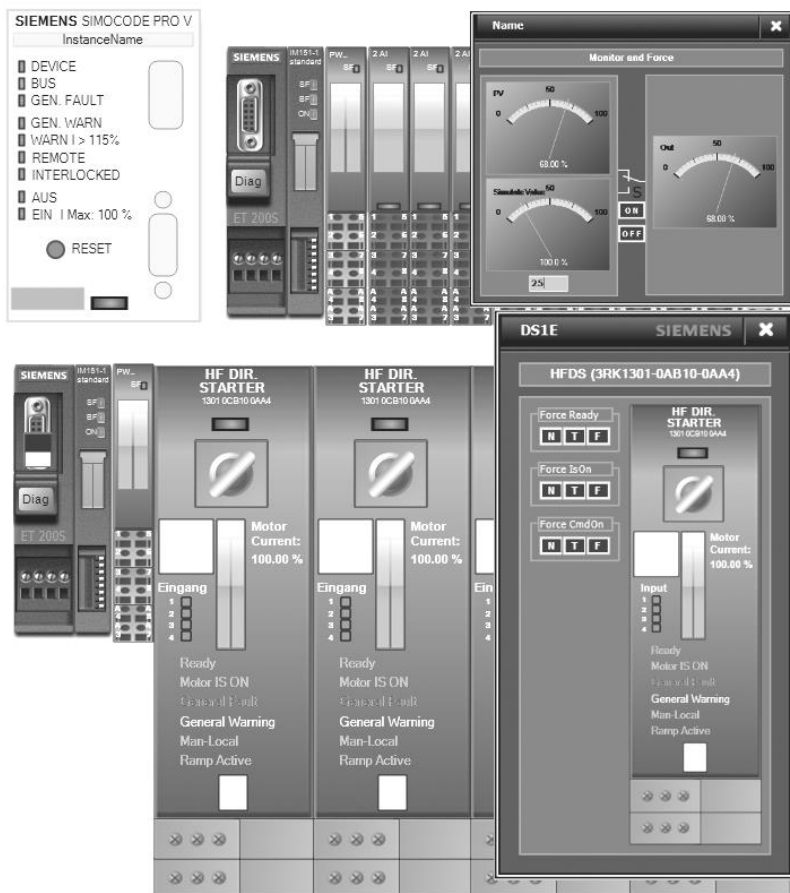


Рисунок 12.15. Мониторинг просмотра аппаратных компонентов DCS

12.12. Выводы

Промышленная автоматизация меняется. Более гибкие решения и бесшовная интеграция с IT-системами, как обсуждалось в Industry 4.0 и в Интернет вещей, окажут свое влияние. Более интеллектуальные устройства будут распределены в системах автоматизации, и разработка таких систем станет более сложной. Описанный выше проект является лишь одним примером, к которому IEC 61499 может быть адаптирован для конкретных нужд заказчика и легко разработан для инженера. Объектно-ориентированный инженерный подход хорошо подходит для адаптации увеличивающихся сложностей. В заключение сделаем такой вывод: IEC 61499 можно использовать везде, где сегодня применяется IEC 61131, поскольку он предлагает гораздо больше возможностей. Это верно для распределенных систем управления, где преимущества очевидны.

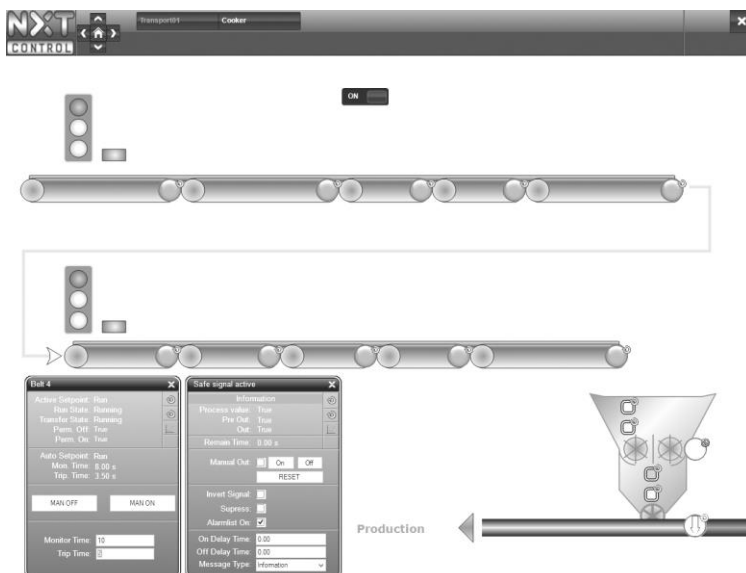


Рисунок 12.16 (а). Проектная визуализация: ременная линия и лицевые панели для управления двигателем и для контроля двоичного сигнала

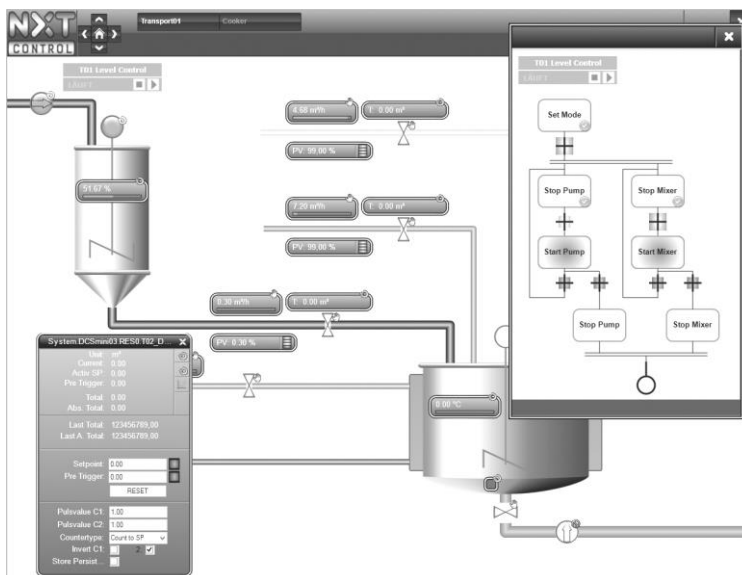


Рисунок 12.16 (b). Проектная визуализация: реактор HMI и лицевые панели для счетчика и детализации последовательности

Библиография

- [1] Forschungsunion und Deutsche Akademie der Technikwissenschaften. *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0, Abschlussbericht des Arbeitskreises Industrie 4.0*. Forschungsunion und Deutsche Akademie der Technikwissenschaften, 2013.
- [2] B. Vogel-Heuser, C. Diedrich, A. Fay, S. Jeschke, S. Kowalewski, M. Wollschlaeger, and P. Göhner. Challenges for software engineering in automation. *Journal of Software Engineering and Applications*, pages 440–451, 2014.
- [3] V. Vyatkin. *IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review*. IEEE, 2011.

Разработка системы автоматизации

Gernot Kollegger

nxtControl GmbH

Arnold Kopitar

nxtControl GmbH

СОДЕРЖАНИЕ

13.1 Введение	313
13.2 Требования к приложениям управления зданием	314
13.2.1 Как скрыть сложность с интерфейсами адаптера	315
13.2.2 Масштабируемые приложения	316
13.3 Система управления	317
13.3.1 Описание работы	317
13.3.2 Создание приложения для управления и приложения для мониторинга	317
13.3.2.1 Приложения для управления помещением	318
13.3.2.2 Масштабируемость сегментов помещения	322
13.3.3 Приложения HVAC и управления поставками	322
13.3.3.1 Архивация исторических данных и аварий	325
13.3.4 Аппаратное подключение в рамках проекта	325
13.4 Выводы	327

13.1 Введение

По оценкам, 40 % глобально генерируемой энергии потребляется в зданиях [2]. Европейское Сообщество устанавливает цель по сокращению выбросов CO² на 20 % до 2020 года [1]. Новые правила по энергоэффективности зданий помогут достичь этой цели. Помимо структурных мер, автоматизация зданий является ключом к энергоэффективным зданиям. Непрерывная интеграция и коммуникация различных технологий является проблемой. Сложность автоматизации зданий очень часто недооценивается.

Несколько технологий от нескольких поставщиков могут быть реализованы в здании. Для системного интегратора взаимодействие является сложной задачей, потому что инженерные усилия должны компенсировать отсутствие взаимодействия. Устойчивые решения для автоматизации зданий должны

позволять эффективно использовать энергию и эффективные инженерные решения, чтобы справиться со сложностью.

Проекты по автоматизации зданий присуждаются через тендеры. Сравнение продуктов легко, а сравнение систем – нет. Вот почему инновационные решения для автоматизации зданий внедряются недостаточно часто. Выбирается наиболее экономичное решение, основанное на сравнении продуктов, без учета того, что решение может увеличить затраты на проектирование. Для лучшей автоматизации зданий прозрачность затрат должна стать более распространенной или автоматизация зданий должна стать более простой. Способ сделать автоматизацию здания проще - скрыть сложность.

В следующей главе будет показано, как можно снизить сложность для системного интегратора с помощью объектно-ориентированного проектирования, основанного на концепциях IEC 61499, для содействия упрощению и ускорению проектирования средств автоматизации зданий.

13.2. Требования к приложениям управления зданием

Приложения становятся все более и более сложными, когда в один проект необходимо интегрировать несколько технологий. Приложение управления зданием по своей природе является распределенной системой. Поэтому среда распределенного управления, такая как nxtControl и IEC 61499, наилучшим образом соответствует потребностям такого приложения.

В предыдущей главе мы описали, как использовать преимущества распределенных систем управления в среде управления процессами. Система управления зданием предъявляет особые требования к различным уровням здания. Существуют особые требования к процессу проектирования, для работы и взаимодействия с пользователями, а также для управления отдельными уровнями здания.

В области отопления, вентиляции и кондиционирования воздуха (HVAC) требования аналогичны требованиям приложений управления процессом с другой степенью детализации распределения. Например, основная вентиляционная установка должна управляться и контролироваться в общей технологической среде. Однако воздушный поток в различные области здания часто децентрализован и изолирован от основной системы вентиляции, хотя основной задачей остается контроль температуры и качества воздуха.

Дополнительный уровень контроля требуется в занятых областях, где освещение и затенение, температура, качество воздуха и безопасность имеют решающее значение. Люди живут и работают в этих областях, и системы автоматизации должны поддерживать их индивидуальные потребности и выполнять свои задачи без предварительного уведомления. Во многих ситуациях оба типа технологий работают бок о бок.

Для управления освещением распространенной технологией является система цифрового адресуемого интерфейса освещения (DALI). Затеняющие двигатели управляются стандартными интерфейсами двигателя (SMI).

Коммутаторы и комнатные блоки управления управляются стандартизированным сетевым протоколом связи на основе OSI для интеллектуальных зданий (KNX) или элементами, подключенными напрямую.

Эта смесь систем не была проблемой в прошлом, хотя системы работали независимо. Однако теперь необходимы новые концепции систем автоматизации. Подсистемы должны быть интегрированы для совместной работы. Поток данных между системами имеет решающее значение для обеспечения последовательного контроля здания на всех уровнях.

Руководствуясь этими требованиями, необходимы новые концепции систем автоматизации управления. Подсистемы должны работать вместе. Поток данных между системами является обязательным условием для достижения общего контроля над зданием на всех уровнях.

13.2.1. Как скрыть сложность с интерфейсами адаптера

Компонентно-ориентированные системы инкапсулируют алгоритмы в функции или функциональные блоки с интерфейсами для обеспечения компонентов параметрами. Если инженер использует такую функцию в других функциях в качестве подкомпонента, эти параметры должны быть определены. Когда системы становятся сложными и к компоненту применяются несколько уровней вложенности, интерфейсы часто становятся слишком сложными, чтобы обрабатываться одним функциональным блоком. Кроме того, это становится еще сложнее, если цель состоит в интеграции подсистем в общий уровень автоматизации.

Интерфейсы адаптера, розетки и вилок

Стандарт IEC 61499 включает определение интерфейсов адаптера. Концепции nxtControl для приложений автоматизации для автоматизации зданий интенсивно их используют.

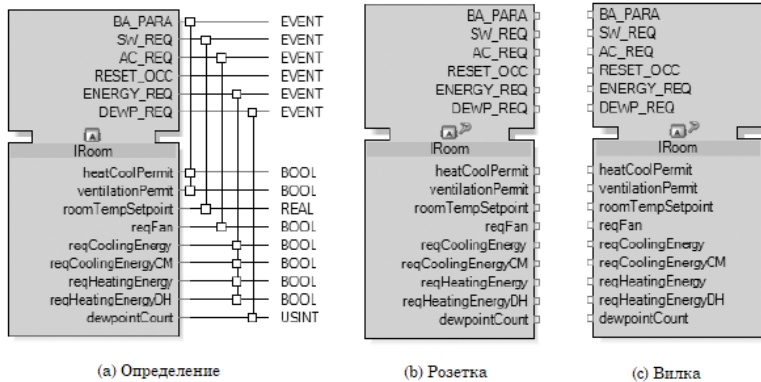


Рисунок 13.1. Вид интерфейса адаптера

Интерфейсы адаптера обеспечивают двунаправленную связь между функциональными блоками. Они могут использоваться слева (входная сторона) в качестве разъемов или справа (выходная сторона) в качестве разъемов функционального блока. На рисунке 13.1 (а) показано определение, на рисунке 13.1 (b) – представление в функциональном блоке при использовании в качестве разъема, а на рисунке 13.1 (c) - в качестве разъема. Интерфейс адаптера состоит из событий и входов и выходов данных и их связей, определенных с помощью конструкции WITH. Связь между исходным и целевым адаптерами обеспечивает согласованную передачу данных в гранулярности конструкции набора данных события и связи. Даже если связь выходит за границы устройства, согласованность данных задается в распределенном приложении. Наборы данных немедленно передаются адаптеру назначения, и события планируются для выполнения. Это уменьшает задержку выполнения события до минимума.

Соккрытие сложности

Адаптеры помогают снизить сложность для техников, использующих и комбинирующих функциональные блоки в своих проектах. Тем не менее, создание библиотеки является более сложным, но для удовлетворения потребностей рынка, чтобы интегрировать широкий спектр функциональных возможностей и систем в один общий набор объектов автоматизации. Создание библиотек является экспертной задачей и должно проверяться на качество, проверяться и контролироваться версиями.

Интерфейсы, построенные из событий и переменных данных, используются почти во всех взаимодействующих функциональных блоках. Адаптеры сводят интерфейс к одной точке подключения. Не все сигналы должны быть объединены в одном адаптере, но могут быть созданы домены переменных для соединения определенных типов функциональных блоков друг с другом одной линией.

Источник и назначение адаптеров должны быть совместимы, чтобы их можно было подключать. Это требование упрощает мир специалистов по интеграции, поскольку уменьшает вероятность неправильных соединений функциональных блоков; они ограничены реализацией правильных интерфейсов. Например, адаптер, управляющий освещением, не может быть неправильно подключен к адаптеру, предназначенному для управления шторами. Очевидно, что программные компоненты должны быть предназначены для выполнения только ожидаемой функции.

13.2.2. Масштабируемые приложения

Создание приложений на уровне управления помещением может быть высоко стандартизировано в функциях управления. Однако преимущество стандартизации высокого уровня уменьшается из-за несовместимости различных технологий управления комнатой. Для создания масштабируемых

приложений используются адаптеры для наиболее гибкого соединения этих различных технологий и освобождения основных приложений управления помещениями от аппаратных зависимостей. Аппаратные операции должны быть разделены на специализированные компоненты и подключены через адаптеры к основному управляющему приложению. Интерфейс в форме адаптеров может освоить сочетание технологий и объединить их в эффективное приложение.

Для приложений, ориентированных на спрос, масштабируемость является важным фактором в структуре программы. В энергосберегающем здании ключевым требованием является обеспечение отопления и охлаждения на основе спроса. Однако программирование такой системы не должно мешать простой настройке приложения здания. Более того, оно должно быть законченным приложением и компонентом основных программных компонентов.

13.3. Система управления

13.3.1. Описание работы

Работа состоит из нескольких приложений управления, таких как управление вентиляцией, систем отопления и охлаждения, геотермальной системы отопления и охлаждения, также из автоматизации управления помещением для HVAC и затенения. Здание, которое будет контролироваться этим приложением, содержит три этажа с более чем сотней комнат. В комнатах стандартные офисы, конференц-залы и лаборатории.

Системы отопления и охлаждения должны контролироваться с учетом рассчитанных заданных значений. По возможности, геотермальная энергия должна использоваться для отопления и охлаждения.

Для лабораторий требуется более продвинутый контроль и мониторинг. Поскольку в некоторых лабораториях могут использоваться химические вещества, в этих помещениях необходимо контролировать давление. Кроме того, контроль выхлопа необходим во всех областях, где исследователи работают с химикатами, и в помещениях для хранения химикатов.

13.3.2. Создание приложения для управления и приложения для мониторинга

Для создания управляющего приложения использовались четыре контроллера. Один контроллер на этаж для автоматизации управления помещением для отопления, охлаждения и затенения. Для специальных лабораторных комнат были добавлены расширенные алгоритмы управления. Четвертый контроллер работал с системами отопления и охлаждения.

13.3.2.1. Приложения для управления помещением

Комнатная температура

В здании требовалось несколько уровней сложности управления комнатами. Средства управления помещением были запрограммированы как САТ и включали в себя все необходимые детали для автоматизации помещений. В каждой комнате было специальное устройство управления, подключенное через последовательный порт Modbus к устройству автоматизации отопления для пола. Терморегуляторы были установлены в каждой комнате. Для достижения желаемой температуры было использовано несколько исполнительных механизмов. В каждой комнате был НМІ, чтобы управлять и контролировать свою автоматизацию (Рисунок 13.20). Примеры приводов:

Радиатор регулируется пульсирующим клапаном.

Блок подачи воздуха для обеспечения потока воздуха, регулируемого по объему, из системы HVAC.

Нагревательный змеевик расположен в вентиляционной шахте непосредственно перед регулируемой коробкой подачи воздуха.

Потолки отопления и охлаждения могут использоваться для любых целей, в зависимости от температуры наружного воздуха.

Охлаждающие агрегаты используются в некоторых лабораториях для обеспечения дополнительного охлаждения.

Несколько типов комнат используются в проекте и требуют различных комбинаций типов приводов. Комнатный блок управления, подключенный к Modbus, используется для информирования пользователей о состоянии управления помещением. Пользователи также могут настроить заданное значение температуры в помещении в предварительно заданном диапазоне.

В помещениях с потолками для обогрева и охлаждения имеются дополнительные датчики точки росы и оконные контакты для предотвращения образования конденсата, когда система находится в режиме охлаждения. Эти датчики подключены к существующим входным каналам на устройстве управления помещением, а сигналы доступны в подсистеме ввода-вывода Modbus.

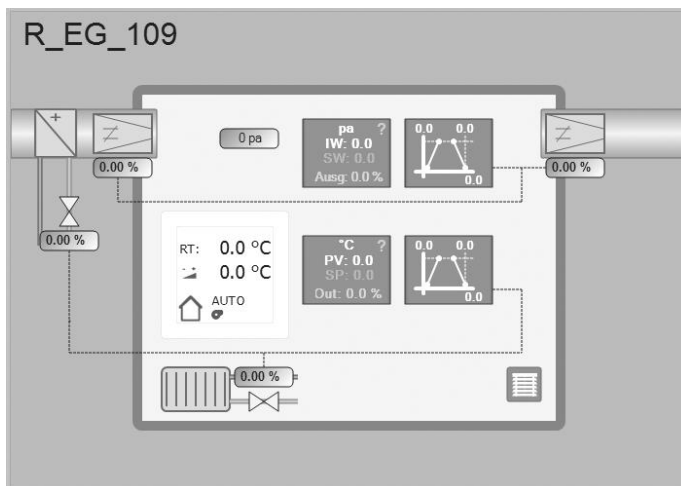


Рисунок 13.2. Расширенное управление помещением для различных источников затенения

Если система находится в режиме охлаждения, и открывается окно, событие вызывает закрытие потолочного клапана. Сгенерирован информационный сигнал, а состояние устройства управления помещением обновлено. Дополнительно пользователь комнаты информируется о ситуации. Комната не контролируется системой, пока окно остается открытым. Когда окно закрыто, система возвращается в автоматический режим и пытается регулировать температуру в соответствии с заданным значением.

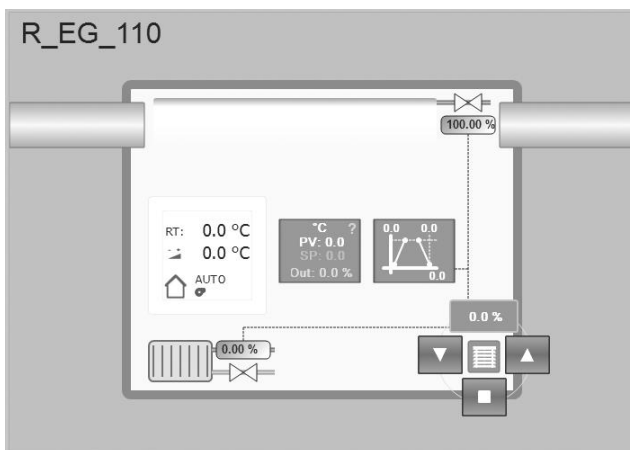


Рисунок 13.3. Основные элементы управления обогревом и затенением

Запрос энергии на основе спроса

Каждая комната рассчитывает свою потребность в энергии в зависимости от заданного значения и отклонений от него. Для подключения запроса энергии к системе отопления, вентиляции и кондиционирования воздуха в каждой комнате предусмотрены адаптеры для запроса подачи энергии в виде розеток и вилок. Комнаты соединены друг с другом через этот адаптер. В CAT управления комнатами сигналы других комнат считываются из розетки и сравниваются с вычисленными запросами текущей комнаты. Если необходимо настроить значения, новые значения передаются на штекер.

Этот расчет выполняется только в том случае, если в системе изменяется заданное значение или измеренная температура изменяется больше, чем настроенное отклонение. Конструкция CAT и способ реализации адаптеров сводят события к минимуму для каждого изменения.

Каждый этаж здания может быть индивидуально отрегулирован с точки зрения режима управления HVAC и базовых уставок для контроллеров комнатной температуры. Режимы работы «Выкл.», «Защита», «Экономия» и «Комфорт» могут зависеть от времени. Настраиваемые планировщики используются для активации режимов: один для ежедневной настройки, а другой для особых дней. Уставка CAT реализует тот же адаптер, что и CAT комнатного управления, и используется в качестве первого CAT в структуре программы этажа.

На рисунке 13.4 показана упрощенная структура системы для расчета заданных значений, определяемых спросом. Уставка и все CAT сегмента помещения соединены с адаптером IRaum и, в конце, с композитом anRaumEnergySupply для каждого этажа. Композиты связаны с помощью адаптера IEnergySupply. Системы энергоснабжения подключены к этому адаптеру, а заданные значения, собранные в помещениях, передаются в системы вентиляции, централизованное снабжение, охлаждающую машину и геотермальную подсистему.

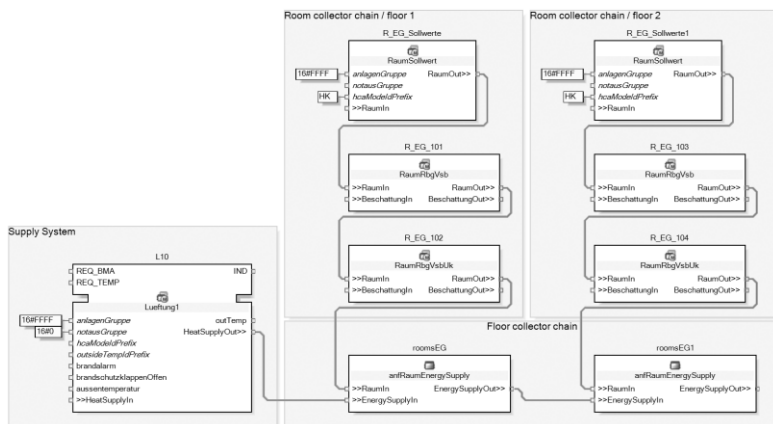


Рисунок 13.4. Расчет заданного значения, выполняемый по запросу

Система управления затенением

Помимо контроля температуры, каждая комната содержит логику для управления шторами. Оттенки имеют разные уровни контроля:

Блокировка метеостанции означает, что в зависимости от скорости ветра шторы могут открываться и блокироваться автоматически, чтобы предотвратить повреждение.

Автоматические программы затенения позволяют позиционировать оттенки по времени в зависимости от положения солнца. Каждый оттенок может быть назначен группе, связанной с основным направлением. Для каждой группы создается индивидуальная программа управления

Прямое управление со стороны пользователя позволяет вручную управлять шторами с помощью проводных переключателей.

Прямой контроль на уровне комнаты HMI позволяет обслуживать или контролировать шторы комнаты от HMI.

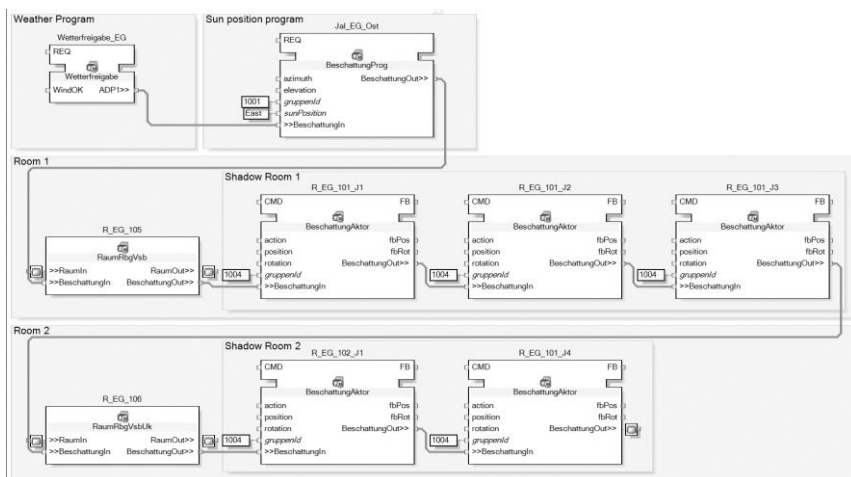


Рисунок 13.5. Структура программы управления тенью

Прямое управление и автоматическое управление имеют одинаковый приоритет и могут перекрывать друг друга. Метеостанция имеет приоритет над всеми другими командами по соображениям безопасности.

Каждый элемент управления оттенком инкапсулирован в САТ того же типа *BeschattungAktor* и реализует адаптер *ISunblindCtrl*. Несколько оттенков в комнате связаны и связаны с комнатой. В комнатах также реализован *ISunblindCtrl*, и они подключены для получения команд управления от программ погоды или групповой автоматизации. На рисунке 13.5 показана уменьшенная структура для управления затенением.

13.3.2.2. Масштабируемость сегментов помещения

На основе CAT для автоматизации помещений и управления затенением легко работать с различным количеством оттенков отдельно на комнату, на этаж и на здание. Чтобы добавить шторы в комнату, просто подключите их с помощью адаптера *ISunblind-Ctrl*. Чтобы соединить комнаты с полом, установите композит *anfRaumEnergySupply* и адаптер *IRaum*. Наконец, с помощью адаптера *IEnergySupply* FB запроса энергии для пола подключаются к системе питания.

13.3.3. Приложения HVAC и управления поставками

Для обеспечения энергии правильного количества и качества для здания, несколько единиц HVAC внедрены. Различные блоки питания разрабатываются как CAT для повторного использования и сводят к минимуму пусконаладочные работы для интеграции на месте.

HVAC система

Для проекта требовалась стандартная система вентиляции и вытяжка. На рисунке 13.6 показана система вентиляции CAT. Большая часть необходимой функциональности уже включена, но некоторые дополнительные параметры и логика должны быть определены, чтобы разрешить использование одного и того же оборудования для различных целей. Для заданного значения CAT реализует адаптер *IEnergySupply* для управляемой по требованию работы системы.

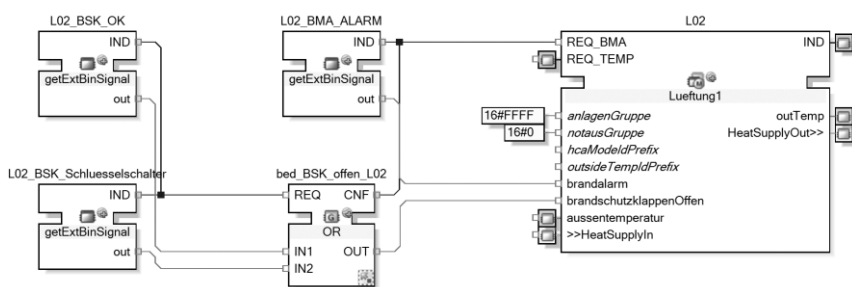


Рисунок 13.6. Логическая сеть системы вентиляции

Для работы вентиляционных агрегатов глобальный режим рассчитывается на основе температуры наружного воздуха. Концепция SYMLINK используется для переключения между режимами вентиляции для зимы, лета и времени перехода. Также в комнатах с потолками для обогрева и охлаждения информация о режиме используется для переключения между операциями обогрева и охлаждения.

Типы вентиляции и вытяжки оснащены стандартными CAT для клапанов, двигателей, аналоговых и цифровых датчиков, регуляторов и простых

универсальных функциональных блоков. Они связаны со стандартным соединением событий и данных IEC 61499. Концепция CAT с ее вложенной структурой позволяет создать полную систему вентиляции в виде единой CAT. Интегрированная визуализация и аппаратная абстракция позволяют легко добавить CAT в проект. Оставшаяся работа для системного интегратора заключается в перетаскивании CAT-типа и создании экземпляра в редакторе IEC 61499 и в редакторе HMI (рис. 13.7), а также для подключения аппаратных точек I/O к созданному снизу аппаратному обеспечению. контроллер.

Системы отопления и охлаждения

В рамках проекта было использовано несколько систем отопления и охлаждения. Геотермальная энергия была объединена с компонентами централизованного теплоснабжения и охлаждения. Были установлены системы буферного хранения тепла и холода, чтобы оптимизировать эффективность геотермальной установки. Программа автоматизации была необходима для управления геотермальной установкой и ее максимально широкого использования, чтобы уменьшить потребность в выработке энергии из районной системы.

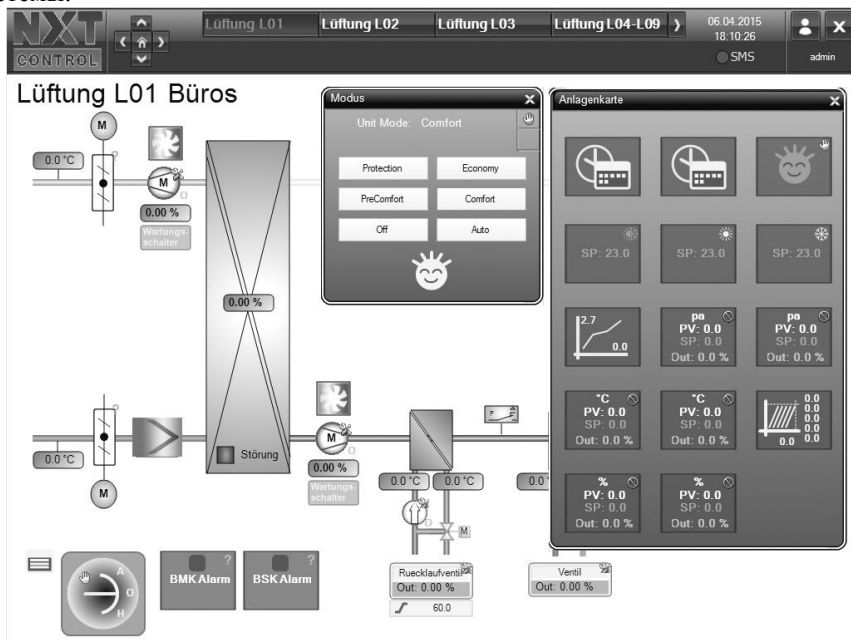


Рисунок 13.7. Система вентиляции HMI

Эта комбинация требовала контроля сложных взаимодействий между различными подразделениями. Одна система может разрушить возможность повторного использования программного кода. Инженеры сочли необходимым спроектировать домены, которые можно было бы заключить в компоненты,

способные взаимодействовать с каждым из них, а также работать как автономные системы. Это облегчало подключение отдельных компонентов. Эти компоненты таковы:

- контроль охлаждающего буфера;
- геотермальный контроль буфера;
- свободное управление охлаждением;
- охлаждающая машина;
- геотермальный контроль.

Чтобы обеспечить более простую интеграцию в структуру автоматизации, компоненты соединяются (рисунок 13.8) с адаптерами для управления буфером IPuffer и адаптером IEnergySupply для установки заданных по требованию заданных значений из комнат. Была добавлена некоторая логика (FreecoolingWT) для задачи свободного охлаждения, а также добавлены адаптеры IFreecooling и IFreecoolingK к компонентам геотермальной и охлаждающей машин.

Компоненты используют базовые CAT для датчиков и блоков, чтобы создать необходимую функциональность. Соединения данных и событий используются для потока сигналов. Функциональные блоки SYMLINK добавляются для обеспечения точек доступа для аппаратного подключения. Параметры для настройки системы также требуют базовых CAT. У CAT есть символы и лицевые панели, которые используются на полотнах HMI. На этапе ввода в эксплуатацию параметры вводятся через HMI. Значения параметров сохраняются на контроллере и автоматически восстанавливаются при каждом перезапуске устройства.

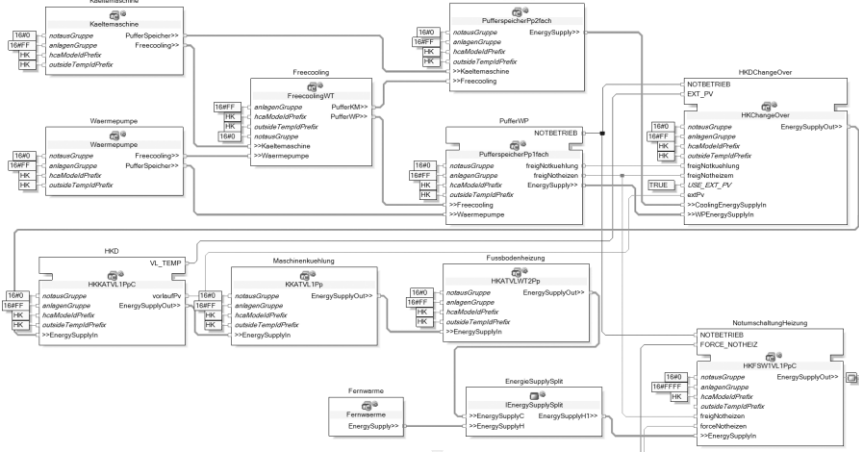


Рисунок 13.8. Системы отопления и охлаждения

Индивидуальные программные компоненты

Наряду с исходными компонентами поставки существуют элементы управления, характерные для этого проекта. Элементы реализуют адаптер

IEnergySupply и используют рассчитанные заданные значения для алгоритмов принятия решений и управления. Адаптеры предоставляют общий интерфейс для реализации пользовательских функций программы, но позволяют использовать их повторно и интегрировать.

13.3.3.1. Архивация исторических данных и аварий

Двумя важными задачами являются архивирование сигналов тревоги и анализ тенденций. Настройка аварийной сигнализации и архивирования может занять много времени.

Структура IEC 61499, охватывающая возможности вложения и атрибутов, помогает инженеру в этой задаче. nxtStudio предоставляет дополнительную поддержку для простого добавления и управления атрибутами в программных компонентах, даже если они вложены в проект. Это называется переопределением атрибута. По сути, создатель библиотечных элементов может определять атрибуты для объекта на уровне типа. Сюда входят, в случае архивирования, атрибуты, позволяющие включить возможность оперативного архивирования и управлять глубиной оперативного буфера, и параметры для фильтрации времени и значений. Аналогичные атрибуты доступны для тревоги (класс тревоги и текст).

Другим заметным атрибутом является *HMIAlias*. Путь к программному компоненту используется для четкой идентификации компонента. Тем не менее, этот путь нелегко распознать в областях тревог и архивирования. Псевдоним HMI дает возможность идентификации компонента человеком. Псевдонимы HMI вложенных элементов автоматически объединяются, чтобы обеспечить разборчивое имя для объекта. Это помогает идентифицировать элементы в представлении списка сигналов тревоги или в представлении списка архивов.

Перепараметризация атрибутов

Атрибуты в основном определяются на уровне типа. Если функциональный блок используется непосредственно на уровне приложения, определенные атрибуты могут быть повторно параметризованы. Это важно для индивидуализации текста тревоги, изменения класса тревоги, изменения глубины онлайн-буфера трендов и изменения других настроек. nxtStudio предоставляет мощный редактор для управления атрибутами с помощью фильтров, функций копирования и вставки и выборочно собранных списков атрибутов.

13.3.4. Аппаратное подключение в рамках проекта

Комнатное управление и оборудование HVAC в проекте здания ставят задачу интеграции широкого спектра аппаратных подсистем с различными протоколами, системами полевых шин и диапазонами сигналов. Технологии, необходимые для получения необходимой информации, включают в себя:

- EtherCAT для цифровых и аналоговых сигналов на основе зажимов срезов и модулей управления затенением;
- Serial Modbus для комнатных устройств управления;
- Serial Modbus для метеостанции;
- Serial Modbus для управления аварийным освещением.

Некоторые из аппаратных сигналов доступны непосредственно на контроллере. Большинство из них являются сигналами на основе зажимов и подключенными к комнатам устройствами управления Modbus. Другие сигналы, например значения от метеостанции, подключаются к одному контроллеру и распространяются через приложение на все остальные контроллеры.

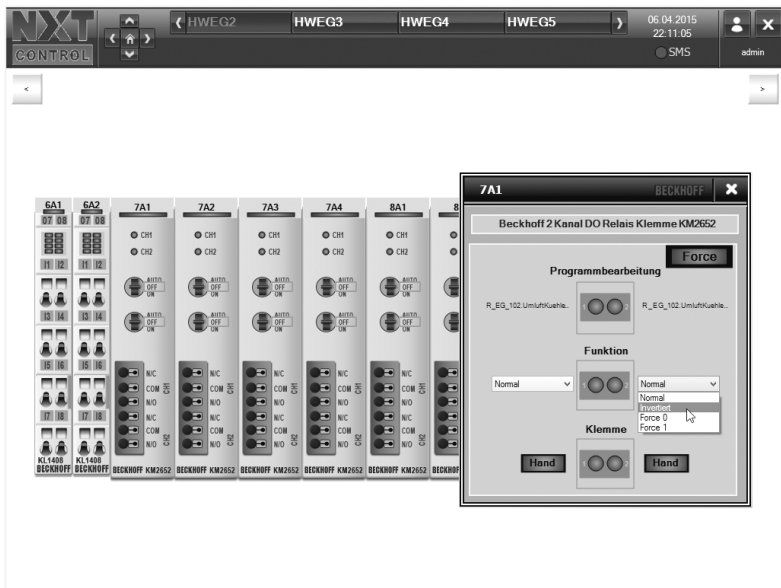


Рисунок 13.9. Аппаратный НМИ

Аппаратные CAT отвечают за нормализацию различных источников сигнала до общего диапазона значений, который совместим со всеми программными компонентами с концепцией SYMLINK. В конфигураторе аппаратного обеспечения определены параметры для подключения подсистемы. Они используются для создания файла конфигурации, который загружается в контроллер на этапе развертывания и используется для настройки соответствующих драйверов.

Аппаратные зажимы, используемые в этом проекте, также имеют возможность вручную управлять выходом канала с помощью переключателей, расположенных на зажимах (Рисунок 13.9). Состояние ручных переключателей считывается и используется элементами управления для непосредственного указания локального переопределенного режима, например, в холсте управления НМИ системы вентиляции. Эти переключатели обычно используются только в

том случае, если в программном обеспечении имеется некоторая неверная конфигурация или для отмены сигнала автоматической программы.

13.4. Выводы

Распределенное управление стало реальностью в автоматизации зданий, особенно благодаря широкому использованию средств управления помещениями. Несколько технологий должны взаимодействовать внутри здания, чтобы обеспечить экономию энергии, необходимую для экономической эксплуатации. Автоматизация зданий – сложное дело. Достижения в области автоматизации зданий (интеллектуальные здания) позволяют добиться ощутимого сокращения выбросов углекислого газа. Проект, описанный в этой главе, ясно показывает, что IEC 61499 хорошо подходит для таких применений.

Легкость распределения логики управления на несколько устройств управления и объектно-ориентированный подход являются ключевыми для простого решения по автоматизации зданий. Архитектура автоматизации зданий для полов, проходов, помещений и сегментов представлена идеально. Кроме того, программные объекты и библиотека, созданные в ходе проекта, можно использовать повторно. Автоматизация резко сократила инженерные усилия в успешных проектах. Готовые к использованию библиотеки и IEC 61499 облегчают простую автоматизацию здания.

Библиография

[1] European Parlament. RICHTLINIE 2012/27/EU DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 25. Oktober 2012 zur Energieeffizienz, zur Änderung der Richtlinien 2009/125/EG und 2010/30/EU und zur Aufhebung der Richtlinien 2004/8/EG und 2006/32/EG. 2012.

[2] VDI Verein Deutscher Ingenieure e.V. Positionspapier Klimaschutz und Energiepolitik: Kriterien für die Bewertung der Nachhaltigkeit von Gebäuden. 2011.

Программное обеспечение для управления станком для резки стекла, созданным с использованием платформы COSME: тематическое исследование

Felix Serna

Университет Сарагосы

Carlos Catalan

Университет Сарагосы

Alfonso Blesa

Университет Сарагосы

Jose Manuel Colom

Университет Сарагосы

Josep Maria Rams

группа TUROMAS

СОДЕРЖАНИЕ

14.1 Введение	330
14.2 Проектирование на основе IEC 61499 в зависимости от предметной области	331
14.3 Моделирование процесса обработки стекла	335
14.3.1 Определенная функциональность	337
14.3.2 Универсальная функциональность	338
14.3.3 FB-дизайн и проектировочные шаблоны	339
14.3.4 Детальное проектирование FBN: сеть EtherCAT	343
14.4 Реализация	344
14.4.1 Платформа COSME	344
14.4.2 IDE COSME	346
14.4.3 Связь с внешними приложениями	349
14.5 Практические проблемы и выводы	352

14.1. Введение

В этой главе представлено тематическое исследование, демонстрирующее конструкцию станка для резки стекла. Это устройство является первой технологической иллюстрацией платформы COSME, ее руководящих принципов и инструментов. Она представляет собой подход, позволяющий упростить разработку управляющего программного обеспечения в производственном контексте за счет сокращения времени выхода на рынок и объединения дополнительных функций. Упомянутая иллюстрация охватывает этапы жизненного цикла программного приложения: (1) построение модели на основе компонентов из шаблонов, связанных с функциями управления, и шаблонов, связанных с профилактическим обслуживанием; (2) внедрение и развертывание модели с использованием функциональных блоков, адаптированных к стандарту IEC 61499; (3) мониторинг и контроль исполнения заявки. Наконец, мы представляем экспериментальные результаты, полученные при создании станков, работающих по реальным промышленным сценариям, которые вместе с опытом, полученным при разработке приложения такого типа, позволяют нам оценить стандарт в типичной отрасли в этом секторе.

Современная обрабатывающая промышленность должна предлагать более быстрые решения на изменения рынка, сохраняя при этом низкие затраты. Гибкие производственные системы (FMS) являются технологическими ответами на этот сценарий. В основном они основаны на стандартизированных станках общего назначения, которые могут производить различные детали и изделия, а также на транспортер

ных линиях для их перемещения продукции. Эти элементы организованы на заводской площадке для согласованной работы. Управление FMS традиционно осуществлялось централизованно, с производственными ячейками и островками, управляемыми программируемыми логическими контроллерами (PLC). Обычно PLC выполняют слабую связь между ними для синхронизации производственных процессов. Правила для систем на основе PLC были установлены стандартом IEC 61131 [13], широко приняты и используются в промышленности. Кроме того, CNC (компьютерное цифровое управление) может представлять собой автономные устройства, связывающиеся с PLC через шины (например, Profibus, CANopen) или интегрированные в качестве функции PLC.

Глобальное экономическое развитие приводит к дальнейшему сокращению жизненных циклов продукции, а покупателям требуются различные характеристики. Чтобы приспособиться к этому новому сценарию и удовлетворить меняющиеся требования, производственные системы должны иметь возможность быстро разрабатывать небольшие серии (даже единичные экземпляры). Эти новые требования привели к появлению нового поколения производственных систем, основанных на технологиях и формах промышленного производства, которые выходят за рамки гибкого производства и определяют новую парадигму: гибкое производство (AM). Новые требования AM были проанализированы несколькими авторами (например, см. [12, 16, 18]),

и они характеризуются использованием перенастраиваемых аппаратных и программных архитектур, которые не останавливают производственный процесс, минимизируют ошибки и их влияние (например, через системы профилактического и прогностического обслуживания и отказоустойчивости) и полностью интегрируются в производство и корпоративные информационные системы с помощью интернет-технологий (электронное производство).

Сектор обработки стекла вовлечен в это технологическое изменение, и в результате производители оборудования для этой отрасли должны учитывать определенные соображения при разработке своей продукции. Наше тематическое исследование касается компании TUROMAS [26], которая разрабатывает и производит оборудование для обработки и хранения листового стекла. TUROMAS хочет, чтобы ее новые производственные системы отвечали требованиям, снижали затраты на разработку и улучшали послепродажное обслуживание на основе интернет-технологий (например, электронная помощь).

Соблюдение вышеуказанных требований делает разработку управляющего программного обеспечения для новых производственных систем гораздо более сложным ПО для текущей FMS. Логично, что эта повышенная сложность влечет за собой увеличение затрат на разработку. Как указано [25], эти расходы могут достигать до 80 % от общих затрат (в настоящее время от 40 до 50 %). Эта повышенная сложность потребовала адаптации методологий и парадигм, ранее использовавшихся в других областях разработки программного обеспечения [10]. Этот сценарий включает стандарт IEC 61499 [14] для разработки приложений для управления.

В этом контексте мы представляем тематическое исследование по управлению линией обработки стекла, разработанной в рамках технологического сотрудничества между TUROMAS и нашей исследовательской группой. Это сотрудничество включало разработку платформы COSME (системы управления и моделирования) для разработки нового механизма управления оборудованием TUROMAS. COSME включает в себя структуру, наряду с инструментами разработки, отладки и контроля для разработки управления станками. COSME основана на урезанном подходе к стандарту IEC 61499, но все же оказывается достаточно мощной для такого рода приложений и сокращает время разработки.

14.2. Пректирование на основе IEC 61499 в зависимости от предметной области

Стандарт IEC 61499 определяет архитектуру и модели, стремящиеся облегчить разработку реконфигурируемых и распределенных систем управления. Наиболее значимой моделью является функциональный блок (FB), определенный в предыдущем стандарте IEC 61131. Приложение на основе IEC 61499 состоит из сети функциональных блоков (FBN). Эта модель FB включает управляемое событиями выполнение, которое более эффективно, чем выполнение сканирования по IEC 61131. Функциональные блоки IEC 61499

выполняют алгоритмы управления только тогда, когда события указывают на наличие новых данных для обработки, тогда как функциональные блоки IEC 61131 выполняют алгоритмы управления циклически, даже без новых данных.

Цель реконфигурируемого программного обеспечения может быть достигнута с использованием парадигмы компонентного программного обеспечения благодаря FB IEC 61499, который считается аналогичным программному компоненту [17]. Переконфигурирование возможно посредством изменений в FBN посредством создания, удаления, замены и присоединения FB, а также развертывания и установки новых типов FB «на лету» (то есть без остановки выполнения). С этой целью стандарт предоставляет модель управления для поддержки проблем реконфигурации. Обзор реконфигурации промышленной автоматики в соответствии с IEC 61499 см. [2].

Распределенная система управления состоит из нескольких контроллеров, соединенных между собой сетями, без центрального контроллера, в котором вопросы связи и синхронизации играют главную роль. Модели и архитектура IEC 61499 естественным образом используются в таких системах, и FBN могут быть развернуты на разных ресурсах, которые размещены на устройствах (то есть контроллерах). Каналы событий и данных между контроллерами абстрагируются функциональными блоками интерфейса службы связи (CSIFB) с моделями PUBLISH–SUBSCRIBE и КЛИЕНТ–СЕРВЕР.

В отличие от IEC 61131, отрасль еще не приняла этот новый стандарт. Некоторые ранние исследования [9] и [31] анализируют возможные причины, в основном незрелость стандарта. Вяткин указывает [28] действия, предпринятые, чтобы способствовать его принятию (следующее появление новой версии, представление первого реального опыта и т. д.).

Есть несколько инициатив, нацеленных на то, чтобы предложить инструменты и платформы. FBDK и его среда выполнения FBRT [11], реализованная в Java, не могут соответствовать ограничениям реального времени, но во многих научных работах они использовались в качестве эталона для стандарта. Важной инициативой с открытым исходным кодом является 4DIAC [1] с инструментом, основанным на платформе Eclipse и средах выполнения для ПК и ARM. Другие предложения можно увидеть в [27], и эта работа также указывает на то, что трудно разрабатывать приложения с помощью этих научных инструментов (например, из-за отсутствия инструментов отладки). Тем не менее важно, чтобы академический мир разрабатывал инструменты и платформы, которые могли бы иметь практическое применение инженерами систем управления вне исследовательских лабораторий. Наконец, два коммерческих предложения появились от производителей оборудования для автоматизации: а) ISaGRAF [15] на основе предыдущей структуры для IEC 61131 и б) NxtStudio [20].

С точки зрения разработки, распределенный дизайн FBN не прост. То есть это не просто монолитный дизайн, разделенный на части, реализованные как FB и распределенные по контроллерам. Распределенные системы должны быть тщательно спроектированы, чтобы их можно было повторно использовать в различных контекстах (то есть взаимодействовать с другими FB) с

использованием методологий разработки программного обеспечения. IEC 61499 не устанавливает методологии или руководств, которые указывают, как идентифицировать типы FB или определять взаимодействия FB [27]. Подход заключается в том, чтобы использовать шаблоны проектирования FBN с первыми предложениями, представленными в [6], или более поздними предложениями, представленными в [5, 22, 23].

Управляющие инженеры могут быть не знакомы с методологиями разработки программного обеспечения и парадигмами IEC 61499 (то есть компонентами, управляемыми событиями, и компонентами программного обеспечения). Обследование промышленных систем управления [7] определяет характеристики и анализирует различные способы внедрения систем автоматизации промышленными партнерами, участвующими в европейском проекте MEDEIA. Результаты показывают, что управляющие приложения все еще работают вручную. Перевод спецификации проекта в реализацию обычно оставляется на усмотрение и компетенцию инженеров. Эти реализации основаны на классическом PLC на языках, которые более или менее соответствуют стандарту IEC 61131, или на промышленных ПК на языке C. Подход COSME к этим недостаткам заключается в использовании моделей, адаптированных к стандарту IEC 61499 («предварительно приготовленных») для конкретных, но универсальных приложений. Этого достаточно, чтобы облегчить распределенный дизайн FBN и избежать ручной работы. Эти модели поддерживаются платформой и инструментами, описанными в последующих разделах.

Данная работа сосредоточена на станках как важных элементах любой производственной системы. С общей точки зрения, станок выполняет операции – резку, сверление, фрезерование и т. д. Эти операции состоят из определенного количества движений деталей, обрабатывающих инструментов и готовых изделий, поэтому программное обеспечение для управления должно учитывать: а) точное позиционирование деталей и обрабатывающих инструментов, б) особенности загрузки и выгрузки деталей и готовых изделий. В качестве поддержки разработки станков для новых производственных систем различные инициативы международных консорциумов касаются использования открытых систем для управления открытой архитектурой (OAC), например OSACA в Европе, OMAC в США и JOP в Японии (см. [21] для обзора этих и других инициатив).

Это управление открытой архитектурой должно основываться на концепции платформы, которая включает в себя конкретные характеристики аппаратного обеспечения, операционной системы и связи, чтобы упростить перенос приложения и его совместимость в гетерогенных распределенных средах. Наряду с платформой, прикладное программное обеспечение подразделяется на модули в независимых блоках, которые содержат как структурную, так и функциональную информацию (например, функциональные блоки IEC 61499).

Коммуникационные аспекты являются ключевыми в рассматриваемых станках. Они должны быть интегрированы на разных уровнях заводских сетей связи. Чтобы выполнить эту интеграцию, программное обеспечение системы

управления должна учитывать связь: а) с другими контроллерами на одном и том же станке (то есть с распределенным управлением), б) с другими элементами того же рабочего процесса на заводе (то есть с производственной ячейкой или линией) и с) с внешними приложениями (например, диспетчерский контроль и сбор данных, управление производством).

Еще одной важной особенностью этих станков является управление отказами для обработки всех сбоев в одном рабочем процессе. Политика ответных действий должна учитывать следующие режимы работы.

1. Нормальный режим работы, то есть машина работает в нормальных условиях.

2. Удовлетворительный режимы работы, связанная с работой некоторых компонентов ниже номинальных значений (например, скорость или ускорение осей будут ограничены в случае перегрева).

3. Сбой, которая должна быть обнаружена в частях машин и в других машинах рабочего процесса.

Кроме того, следует учитывать функции профилактического и прогностического обслуживания как средства предотвращения незапланированных остановок из-за сбоев.

Реальный станок может иметь десятки осей позиционирования и автоматических последовательностей, поэтому для управления на основе IEC 61499 требуются большие FBN с сотнями FB. В первом приближении дизайн FBN может быть выполнен путем разделения управления на основные части (например, управление осями), которые появляются многократно и могут быть реализованы с одним sub-FBN. Затем управление может быть реализовано путем составления этих подчиненных FBN, плюс добавление некоторой бизнес-логики. С этой точки зрения масштабируемость FBN является важной целью для поддержания хорошей производительности и сокращения времени проектирования. Эти проблемы не должны зависеть от количества осей или последовательностей. С точки зрения распределенного управления, результирующая FBN, состоящая из sub-FBN частей управления, может быть распределена между разным количеством контроллеров (от одного до десятков) в зависимости от степени детализации распределения и связей между контроллерами и CSIFB.

Следующий пример может проиллюстрировать это. Грубая гранулярность может означать наличие контроллера для каждой машины в ячейке или системе производственной линии, в то время как мелкое «зерно» будет иметь контроллер для каждой оси или автоматическую последовательность. В последнем случае контроллеры могут быть проще (так как будет меньше задач), чем первом случае. Разработчик должен определить количество контроллеров, установив компромисс между задержками связи (которые ставят в невыгодное положение мелкозернистые модели) и вычислительными ресурсами, необходимыми для выполнения задач, назначенных каждому контроллеру (которые ставят в невыгодное положение грубые модели зернистости). Изменяемая конфигурация также может определять количество контроллеров в том смысле, что

реконфигурируемые части станка могут требовать независимого управления (мехатронный подход).

Более того, из-за характеристик производственных систем (например, реконфигурируемость, отказоустойчивость) должен быть главный контроллер, который управляет другими контроллерами. Работа распределенного приложения может координироваться, устанавливая бизнес-логику, реализованную посредством сообщений между контроллерами. Семантика (или значение) этих сообщений связана с гранулярностью распределения: чем ниже гранулярность, тем выше семантический уровень сообщений из-за повышенной сложности функциональности, реализованной на каждом контроллере. В этом смысле подходом к проектированию распределенного управления и других типов архитектуры является многослойный дизайн. Предложение, которое следует этому подходу для приложений на основе IEC 61499, представлено в [29]. Данный подход устанавливает три уровня: интерфейсный процесс (доступ к датчикам и исполнительным механизмам), операции (последовательности и позиции) и применение (бизнес-логика). Другой подход состоит в том, чтобы идентифицировать и охарактеризовать типы коммуникации на основе предметной области [4]. Характеристика включает в себя: а) типичные задержки, б) периодичность или событие модели связи, в) надежность и г) пропускную способность.

Существует несколько функций с аппаратными платформами для станков на гибких производственных системах. Структура и методология для адаптивных сборочных операций с использованием FB представлены в [8], где функциональные блоки позволяют динамические цеховые операции. В [19] FB используются для реализации моделей состояния управления открытой модульной архитектурой (OMAC) для упаковочных машин. Архитектура для открытой и распространяемой системы CNC на основе STEP-NC и FB [30] была протестирована в лабораторном прототипе с FBDK и FBRT. Реконфигурируемое управление для PLC и контроллера CNC рассматривается в [30], но результаты эксперимента не представлены, за исключением краткого описания тематического исследования. Общим для этих предложений является то, что они не были опробованы в реальных приложениях или не рассматривают упомянутые аспекты.

14.3. Моделирование процесса обработки стекла

Линия обработки для ламинарного и/или монолитного стекла (см. рисунок 14.1) является четко определенным продуктом. Ее функциональные возможности определяются типом обрабатываемого материала – листами монолитного стекла или листами многослойного стекла (то есть двумя листами стекла с промежуточным масляным слоем). Основные требования касаются: а) видов операций обработки: нагревание, резка, разрушение и т. д., б) типов и мер листового стекла, в) расчетных скоростей производства, г) типов перемещений

листов и деталей из стекла: загрузка, выгрузка, квадрат, резка и т. д., и е) количества осей на основе предыдущей функции.



Рисунок 14.1. Машина для резки ламинированного стекла. (Фото любезно предоставил TUROMAS.)

Этот тип машины ограничен различными рыночными соображениями, которые должны быть соблюдены (характеристики рынков, на которые следует ориентироваться: местные нормативы безопасности, ожидаемая цена, сегментация семейства продуктов и т. д.). Помимо элементов обработки, в эти линии входят такие элементы, как загрузчики и разгрузчики, хранилища и транспортные средства. Также необходимо учитывать и другие вопросы: человеко-машинный интерфейс (HMI), интеграция с внешними MES (система управления производством) и ERP-системами (планирование ресурсов предприятия) и прочие нефункциональные системы, которые повышают ценность продукта. Механические, электронные и системные конструкции (например, типы движения, типы привода, количество осей) устанавливают требования к программному обеспечению, которые относятся к различным функциям, подробно описанным ниже.

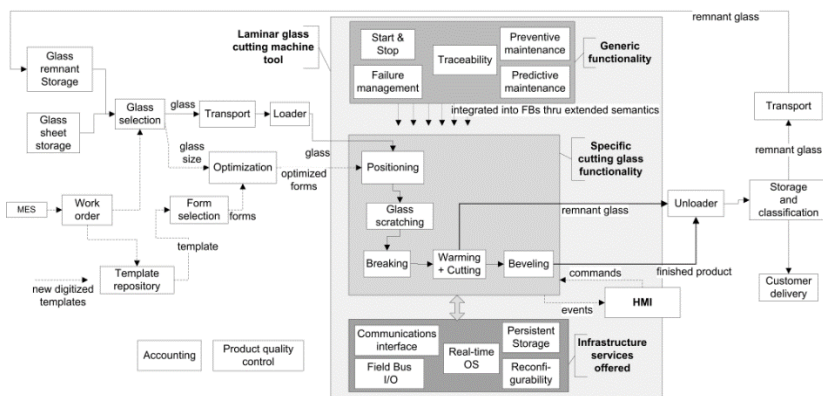


Рисунок 14.2. Станок для резки стекла в зависимости от окружающей среды

На рисунке 14.2 элементы, принадлежащие станку, показаны в рамке и соответственным образом упорядочены

14.3.1. Определенная функциональность

Большинство функциональных элементов, присутствующих в задачах, упомянутых выше, зависит от технологии. Например, ламинарное стекло имеет масляный слой, который должен быть нагрет до резки стекла. Это нагревание может быть достигнуто различными способами, в зависимости от доступной технологии, например осуществлено инфракрасными лампами, лазерными лучами и т. д. В этом случае функциональность (нагревание) должна быть предусмотрена в станках независимо от того, как выполняется процесс.

Эта функциональность может быть реализована с помощью функций PLC и CNC. PLC связан с режимами работы автоматики движения. CNC имеет дело как с инструментами, так и с точным позиционированием деталей. Укажем элементы, связанные с функциональностью PLC.

1. Загрузчик. Он взаимодействует со стеклянными листами, которые будут обрабатываться. Требуются некоторые заранее установленные последовательности, соответствующие перемещениям листа от склада к столу для резки (например, переворачивание, перемещение и т. д.). Определенная «клеякая логика» также необходима для организации последовательностей (то есть конечных автоматов).

2. Механическая обработка. Применяется та же схема, что и выше. Предварительно установленные последовательности соответствуют движениям листа стекла и инструментов (например, выравнивание стекла, предварительное позиционирование инструмента). «Клеякая логика» опять же будет координировать различные последовательности. Есть два процесса обработки. Первый включает в себя двумерное царапание (могут быть нарисованы сложные

формы), в то время как второй допускает только царапание по прямой линии с последующим масляным нагреванием и разбитием стекла.

3. Разгрузчик аналогичен загрузчику, но стеклянные изделия извлекаются и транспортируются на промежуточное хранение до следующих этапов изготовления.

Функциональность CNC соответствует точному позиционированию. Различные элементы образуют иерархию абстракций.

1. Ось. Позволяет точное одномерное позиционирование. Оси состоят из контура управления PID (пропорциональной интегральной производной), коробки передач (элемент программного обеспечения, который масштабирует и выполняет преобразование единиц измерения, например, преобразование импульсов датчика в метры) и генератора заданного значения.

2. Группа осей содержит две или три оси, позволяющие позиционирование в двух или трех измерениях (например, в форме круга).

3. Калькулятор траектории получает желаемые формы, которые необходимо разрезать, генерирует промежуточные точки траектории и рассчитывает скорости и ускорения для каждой промежуточной точки.

Относительно интерфейса процесса (датчики и исполнительные механизмы) используется полевая шина (например, CANopen). Сервомоторы управляются через серводвигатели CANopen. Электроклапаны управляются с помощью островка клапана CANopen. Наконец, различные цифровые входы и выходы обрабатываются с помощью модулей цифрового I/O CANopen.

14.3.2. Универсальная функциональность

Общие функциональные элементы имеют первостепенное значение. Они появляются в станках, не используемых на рынке стекла. Выбор общего подхода к этим темам облегчит взаимодействие между машинами. Данная функциональность может быть сгруппирована в следующем образом.

1. Режимы работы включают холодный и теплый пуск, плановую и внеплановую остановку, эксплуатацию и т. д. в соответствии с указаниями, такими как GEMMA (Пособие о режимах работы и простоев).

2. Должно быть установлено управление отказами, чтобы минимизировать незапланированные остановки из-за неисправностей.

3. Профилактическое обслуживание анализирует время и количество циклов элементов машины, чтобы определить время замены.

4. Местный HMI. Локальный пользовательский интерфейс необходим для выбора режимов работы машины, определения специфических форм, которые необходимо придать, создания сообщений о техобслуживании и предоставления обслуживающему персоналу возможности совершать определенные действия.

5. Контроль управления данными приложения. Определенные данные, полученные во время выполнения управляющего приложения, необходимо будет контролировать. В частности, информация о профилактическом

обслуживании или устранении неисправностей должна постоянно храниться и извлекаться.

6. Управление производственными данными. Станки получают рабочие заказы с указанием форм, которые необходимо придать, размера, типа стеклянного листа, количества изделий и идентификатора клиента. Всякий раз, когда будут приняты различные рабочие заказы, внешние приложения (например, системы MES и ERP) должны быть уведомлены.

7. Связь. Машины должны взаимодействовать с другими приложениями, такими как системы SCADA, MES и ERP, а также с разработанными на заказ решениями.

14.3.3. FB-дизайн и проектировочные шаблоны

С точки зрения IEC 61199, все предыдущие функциональные возможности должны быть реализованы с использованием функциональных блоков (FB). FB-дизайн является очень важной частью процесса проектирования. FB должны быть разработаны квалифицированными инженерами систем управления. Кроме того, дизайн FB должен учитывать все специфические и общие аспекты, перечисленные выше. FB, сгруппированные по следующим категориям, должны быть идентифицированы.

1. FB, сопряженные с процессом. Станок управляет процессом через физические устройства (то есть датчики и исполнительные механизмы). Каждое устройство должно моделироваться компонентом. Доступ к этим элементам осуществляется через полевую шину (например, CANopen, AS-I и т. д.), которая также должна моделироваться одним или несколькими компонентами. Таким образом, устанавливаются два уровня абстракции. Чтобы облегчить реконфигурируемость, между ними может быть промежуточное звено третьего уровня – компонент логического соединителя. Все эти компоненты интерфейса должны быть многоразовыми для любого типа машины.

2. FB, связанные с управлением и автоматизацией. FB, связанные с управлением, соответствуют точному позиционированию оси станка (например, движения листа и инструмента) и основаны на PID-управлении. Обычно эти компоненты должны быть многоразовыми. Компоненты автоматизации соответствуют последовательности операций машины (например, этапы загрузки и разгрузки). Эти компоненты менее пригодны для повторного использования, чем предыдущие, потому что они зависят от конкретной логики.

3. FB, связанные с общей функциональностью (например, управление отказами). Несмотря на то, что компонент включает управление сбоями, необходимо учитывать конкретные компоненты, способствующие этому. Первый – это компонент зоны отказа, который делит управление системой на деноминированные зоны, состоящие из нескольких компонентов. Каждая зона сбоев получает уведомления о неисправностях своих компонентов, а уведомления распространяются на другие зоны, чтобы можно было выработать возможные ответные действия. Каждая зона сбоев должна устанавливать свою

собственную политику распространения сбоев. Задача компонента Supervisor состоит в том, чтобы обнаруживать ошибки, возникающие из нескольких компонентов. Данные о сбоях передаются в соответствующую зону отказов. Наконец, компонент супервизора профилактического и прогностического обслуживания решает связанные с этим проблемы. Компоненты могут быть параметризованы правилами, чтобы сделать их многократно используемыми.

Следующим шагом является создание шаблонов дизайна. Это компоненты подсети, они могут быть сгруппированы в те же категории, что и предыдущие компоненты.

1. Разработка шаблонов, сопряженных с процессом. Структура полевой шины (например, CANopen) состоит из конкретного аппаратного драйвера, универсального главного драйвера и нескольких типов конкретных устройств (например, цифрового и аналогового I/O, драйвера серводвигателя, электроклапанов).

2. Разработка шаблонов, связанных с управлением и автоматизацией. Схема управления (например, PID-управление по оси) состоит из PID-регулятора, генератора заданных значений и коробки передач. Последняя адаптирует уставку и сигнал обратной связи. Чтобы установить шаблоны для автоматизации, компоненты (то есть бизнес-логика) могут быть разделены на несколько частей, каждая из которых соответствует шаблону проектирования (например, накладная и механическая обработка). Это позволяет использовать шаблоны в других машинах.

3. Шаблоны проектирования, связанные с общей функциональностью:

(а) Управление отказами: шаблон проектирования управления отказами имеет несколько компонентов (например, интерфейс, управление, автоматизация, диспетчер отказов). Предложение, основанное на шаблоне проектирования, было представлено в предыдущей работе [23]. Это предложение идентифицирует, характеризует и классифицирует неисправные состояния и их соответствующие ответные действия. Топологический критерий используется для группировки неисправностей, исходящих от элементов машины, в зонах неисправностей, что позволяет создавать иерархическое дерево. Когда происходит сбой, уязвимый элемент уведомляет свою зону сбоя, которая может распространить уведомление на остальные элементы и, возможно, также на другие зоны сбоя. Когда элемент получает такое уведомление, он выбирает соответствующее поведение. Все это препятствует тому, чтобы из-за одного отказа не работала вся машина, если только это не неизбежность. В данной конструкции было создано пять зон отказов, соответствующих четырем физическим частям: загрузчик, разгрузчик, монолитная обработка и многослойная обработка, а также общая зона отказов, выступающая в качестве корня в иерархическом управлении отказами. Аварийные выключатели обрабатываются элементом аварийных ситуаций, поэтому, когда оператор нажимает на любой выключатель, общая зона отказа будет уведомлена напрямую.

(б) Профилактическое обслуживание: шаблон проектирования, представленный авторами [24], применяется к двум элементам станка для резки

стекла: вентиляционным вентиляторам и движущимся частям для позиционирования подрезного колеса. К первому относится одно действие по обслуживанию: проверка выравнивания вентилятора и величина его использования – рабочее время. У последнего есть два связанных действия обслуживания: смазка и замена кабеля. Их связанная величина использования – расстояние, пройденное колесом оценки.

С идентифицированными типами FB и установленными шаблонами проектирования детальный дизайн является инкрементным и итеративным процессом, который генерирует набор компонентов и шаблонов проектирования многократного использования.

Следующим вопросом является разработка конкретного механизма управления машиной после определения требуемых характеристик с учетом: а) типов операций обработки, таких как нагревание, резка и ломка; б) видов и размеров листов стекла; в) предполагаемых темпов производства; г) перемещения стеклянных листов и расходных материалов (например погрузка, разгрузка и резка); е) количества управляемых осей на основе движений, перечисленных в д); и ф) характеристики и классификации неисправностей для выбора ответных действий.

Неисправности классифицируются по потенциальному ущербу. *Предупреждение* означает, что произошел инцидент, но функционирование не нарушено. *Пониженная* производительность означает, что произошел сбой и нарушена функциональность. *Отказ* означает, что произошла ошибка, и система не может работать на назначенном ей уровне функциональности.

Конструкция конкретного устройства управления должна учитывать следующие этапы.

1. Дизайн бизнес-логики. Как указывалось ранее, автоматизация машины может быть разделена на функциональные части. На этом этапе уже разработанные ранее детали должны достигать определенных функциональных возможностей станка. Дизайн бизнес-логики можно рассматривать с точки зрения распределенных систем. Каждый контроллер может обрабатывать одну или несколько из этих функциональных частей. Эти части должны быть свободно связаны со ссылками на высоком семантическом уровне для синхронизации рабочих процессов. Например, загрузчик может сообщить механическому инструменту, что доступен новый лист, и указать его тип. Другой аспект распределенного контроля – управление осями. Связи с этой частью должны быть командами позиционирования (например, перейти в положение XY, перейти в ноль). Кроме того, каждая часть также управляет режимами запуска и остановки из команд HMI или SCADA. Связи между функциональными частями и интерфейсами их процесса основаны на описанных схемах проектирования полевой шины.

2. Проектирование управления отказами и профилактического обслуживания. Хотя шаблоны проектирования управления отказами устанавливают зоны, управляемые компонентом зоны сбоя, для конкретной машины их необходимо соединить вместе. Это означает соединение зон, чтобы любая зона могла быть уведомлена о сбоях в других зонах и распространить

информацию о сбое на ее компоненты. Таким образом, иерархическое управление отказами может быть установлено. Действия по реагированию на неисправности выполняются компонентами и должны учитываться при их разработке. Другая проблема на этом шаге – определить правила для компонентов мониторинга для профилактического обслуживания. Эти правила должны гарантировать, что возможные ошибки выводятся и сообщаются зонам.

3. Разработка аспектов связи с приложениями HMI и SCADA. Программное обеспечение системы управления должна иметь соответствующие локальные и удаленные HMI для управления контролируемыми процессами. Обычно приложение HMI выполняется независимо от управляющего программного обеспечения в многозадачной среде. В представленном подходе платформа позволяет приложениям иметь ограниченный доступ к компонентам. Таким образом, на этом этапе определяется доступ для реализации связи с HMI. Этот доступ также возможен для удаленных интерфейсов.

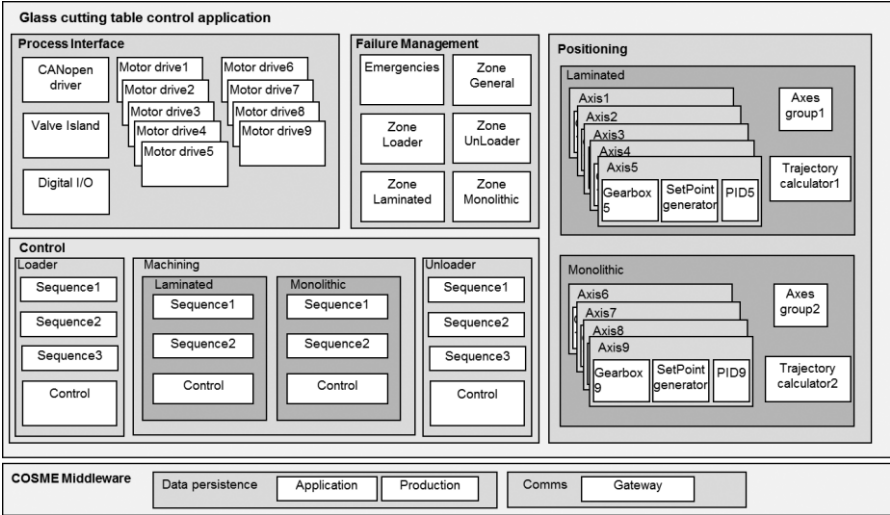


Рисунок 14.3. Высокоуровневая модель стола для резки стекла. Она будет реализована с использованием взаимосвязанных сетей FB (FBN)

Конечный проект представляет собой сеть компонентов, состоящих из различных слабосвязанных частей, в которых бизнес-логика, управление отказами и связь HMI распределены на несколько контроллеров. Это облегчает реконфигурируемость, поскольку изменения в деталях требуют меньших усилий и/или времени из-за меньшей зависимости между ними. Кроме того, общие характеристики машин реализованы как аспекты в компонентной модели.

На рисунке 14.3 архитектурные элементы иерархически сгруппированы в соответствии с типами их функциональности.

14.3.4. Детальное проектирование FBN: сеть EtherCAT

EtherCAT-шина (ethernet для технологии автоматизации управления) – это высокопроизводительная коммуникационная шина на основе Ethernet. Основная цель EtherCAT – возможность использовать определенные уровни Ethernet в приложениях для автоматизации и управления, требующих небольшого времени для обновления данных (также называемого временем цикла), с низкими значениями джиттера (для целей синхронизации) и недорогим оборудованием. Подходит для использования в области применения станков.

Предлагаемый вариант использования основан на контроле ПК, который обслуживает привод и модуль I/O. Необходимо разработать сеть конкретных FB, реализующих полевою шину EtherCAT. Это решение включает в себя сложность, связанную с управляющими и подчиненными FB. Разработчик должен сосредоточиться на функциональности, определенной для каждого приложения.

Чтобы интегрировать EtherCAT полевой шины в проект, экземпляры в сетевых компонентах должны быть созданы из определенных типов FB. Два компонента, созданные для EtherCAT, должны иметь следующие входы и выходы.

Главный FB: а) *Входы:* переменная зоны сбоя получает условие ошибки от других сетевых компонентов (подробности см. в [23]). б) *Выходы:* ошибка указывает на состояние ошибки. Enable_write позволяет подчиненным FB отправлять новый SDO. Hands: тип переменной рукопожатия, используемый для обмена информацией между ведущим и всеми подчиненными. Slaves_init указывает подчиненным, что процесс инициализации завершен и начнется циклический обмен данными.

Подчиненный FB: а) *Входы:* переменная зоны сбоя получает условие ошибки от других сетевых компонентов. Переменная Enable_write изменяется своим управляющими FB и позволяет ведомому устройству отправлять новый SDO. Hands: тип переменной рукопожатия, используемый для обмена информацией между ведущим и всеми подчиненными. Slaves_init указывает подчиненным, что процесс инициализации завершен и начнется циклический обмен данными. б) *Выходы:* ошибка указывает на состояние ошибки. DI, DO, NI, NO и оси: через эти структуры данных подчиненный FB связывается с числовыми вычислениями. Смысл двунаправленный; подчиненный FB изменяет эти переменные с помощью значений PDO и SDO и передает их для численного расчета. В противном случае числовые вычисления FB определяют эти значения и передают их подчиненному FB для передачи с использованием SDO или PDO на шинные устройства.

На рисунке 14.4 показана упрощенная схема. Описанные FB выделены (MST_ETHERCAT для главного FB и ETHERCAT_SLAVE FB для подчиненного привода двигателя и модулей I/O.), и показаны основные подключения данных к другим FB приложения. Существуют цифровые и аналоговые I/O, коробка передач (для изменения масштаба сигнала), управляющие числовые и PID-

вычисления и т. д. Данные I/O FB, связанные с фоновым состоянием, внешним событием, холодным и теплым инициированием, для простоты скрыты.

Обратите внимание, что событийные соединения не отображаются. Как будет объяснено ниже, из-за модели выполнения COSME (основанной на цепочках событий) эти части реализации предопределены COSME.

14.4. Реализация

14.4.1. Платформа COSME

Стандарт IEC 61499 указывает, что FB должны учитывать множество алгоритмов, выбираемых по событиям или условиям. Модель COSME FB [3] основана на предварительно определенных алгоритмах, учитывающих в приложениях для управления общие ситуации, в частности инициализацию и работу при сбое. Входные и выходные события предопределены, а входные и выходные данные – нет, и они зависят от функциональности типа FB. Алгоритмы генерируют событие, когда они закончены. Это позволяет составлять FBN для создания топологии гирляндной цепочки, в которой связаны события одного и того же типа алгоритмов. Расширение распределенных FBN может быть просто создано путем объединения CSIFB в цепочках межконтроллеров. Эти CSIFB должны иметь тип PUBLISH и SUBSCRIBE, потому что они подходят для распространения событий вдоль контроллеров.

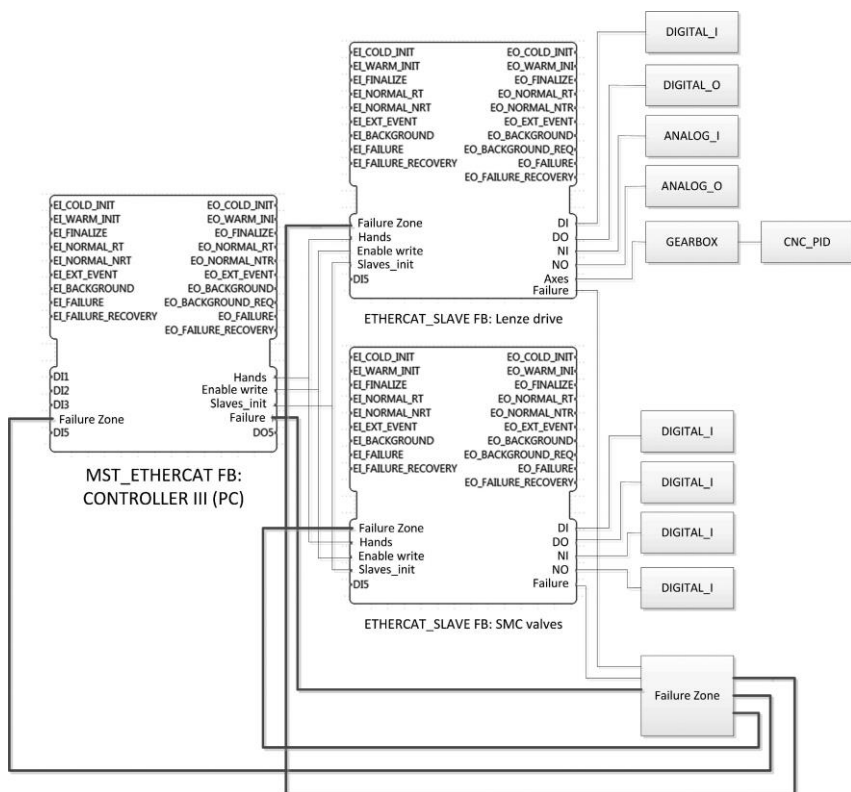


Рисунок 14.4. Шаблон для сетевой схемы EtherCAT FB

«Гирлянда из ромашек» COSME – это промежуточная модель между событиями и сканированием. Разработчики должны написать код ответа для ситуаций, например управляемых событиями, но здесь ситуации предопределены. COSME был разработан для программного обеспечения для управления станками, хотя его можно использовать и в других приложениях. Правила составления просты и могут применяться инженерами, получившими образование в области управления.

Платформа COSME – это слой между приложениями для управления и базовой системой (то есть аппаратным обеспечением, операционной системой и связью). Что касается выполнения, то платформа следует концепции цепочки событий (ЕС). Реализация осуществляется путем связывания этих ЕС с различными приоритетными потоками в соответствии с моделью, называемой «путь события для задачи, приоритет на основе пути события». Алгоритмы управления должны быть выполнены в короткие сроки, потому что они блокируют систему для ЕС с более низкими приоритетами. Из-за ограниченного количества ЕС и потоков реализация легко масштабируется. Таким образом, эта

модель особенно подходит для приложений с большим количеством FB, а также облегчает определение поведения системы во время разработки.

Элементами COSME, которые осуществляют выполнение, являются сервер выполнения, сервер имен и загрузчик приложений. Загрузчик загружает типы FB, создает FB и устанавливает внутренние и межконтроллерные соединения, а также дает команду на загрузку выполнения. Платформа имеет постоянный элемент хранения, готовый для запуска приложений.

Хотя текущая версия COSME сфокусирована на аспектах разработки и распространения приложений, она позволяет просто реконфигурировать FBN во время выполнения с помощью команд (создание экземпляров, переподключение и т. д.). В новых версиях COSME эти команды будут отправлены внешним менеджером реконфигурации, который будет получать данные от управления и MES для реализации динамической реконфигурации. Элементами COSME, которые выполняют эту функцию, являются сервер компонентов, сервер имен и шлюз. Межконтроллерные каналы связи основаны на режиме публикации и подписки. Сообщения содержат номер, имена издателя и подписчика, отметку времени и данные (при необходимости). Связь между контроллерами использует проводную выделенную сеть и коммутируемое устройство Ethernet. Эти устройства гарантируют детерминированные задержки без специального аппаратного или программного обеспечения. Политика аутентификации гарантирует безопасность.

14.4.2 IDE COSME

Domiciano – это инструмент COSME IDE, созданный для облегчения разработки управляющего программного обеспечения. Характеристики Domiciano определяются используемой компонентной моделью (FB) IEC 61499 и областью, для которой она была создана (то есть разработка приложений управления для станков). Его цель – обращаться к конечным доменам приложений. Знание этих характеристик данной предметной области позволяет Domiciano скрывать некоторые аспекты для дизайнеров, поэтому им не нужно заниматься вопросами, связанными со стандартом [9]. Domiciano позволяет выполнять следующие задачи.

1. Создание и редактирование компонентов на основе используемой модели.
2. Управление репозиториями компонентов (то есть библиотек).
3. Редактирование приложений путем создания экземпляров компонентов и их соединения и использования предопределенных шаблонов сетей.
4. Генерация нативного кода приложений различными генераторами кода.
5. Развертывание приложений с платформы удаленной разработки.
6. Отладка приложений путем просмотра внутренних переменных.
7. Управление приложениями путем запуска и остановки действий, загрузки приложений и типов компонентов.

Domiciano предоставляет редакторы, генераторы кода, средства развертывания и отладки. На рисунке 14.5 показан рабочий процесс для

разработки приложения, который, по сути, можно обобщить следующим образом.

1. Создайте компоненты с помощью редактора компонентов и добавьте их в библиотеку компонентов.

2. Генератор кода создает необходимые исходные файлы для выбранной платформы из определения соответствующего файла компонента. Локальная компиляция позволяет избежать синтаксических ошибок.

3. Создается новое приложение, в котором создаются необходимые компоненты. Эти экземпляры параметризуются, и сеть соединений устанавливается с помощью редактора приложений. Когда количество экземпляров и соединений велико, Domícioano организует их, используя представления. Каждое представление показывает только часть сети. Таким образом, дизайнеры могут сосредоточиться на различных аспектах приложения. Наконец, редактор приложения создает файл XML, который включает в себя сетевое представление компонентов.

4. Файлы собственного кода и файл приложения развертываются (локально или удаленно) в целевом средстве с помощью инструмента развертывания.

5. Целевой компилятор вызывается для генерации двоичных файлов.

6. Приложение создается и запускается загрузчиком перед загрузкой типов компонентов.

7. Приложение можно отслеживать на предмет отладки. Domícioano сообщает с промежуточным программным обеспечением через шлюз.

Теперь перечислим элементы Domícioano, обеспечивающие его функциональность.

Редактор типов компонентов: Domícioano создает новые типы компонентов, которые сохраняются в определенном формате в файлах XML. Для этого разработчики должны назначить входные данные и выходные данные внутренних переменных компонентов и определить их типы (рисунок 14.6). Среда IDE предоставляет набор предопределенных типов, но также позволяет определять новые настраиваемые типы. Внутренние переменные компонента могут быть объявлены как постоянные, что означает, что их значения периодически сохраняются во время выполнения и непосредственно перед остановкой приложения с использованием определенной службы промежуточного программного обеспечения. Они также восстанавливаются при запуске приложения. После определения внутренних переменных и входов и выходов компонентов разработчики должны написать исходный код функциональных стадий. Этот исходный код может быть написан на любом из доступных языков, установленных в Domícioano с помощью плагина. Также возможно иметь функции, написанные на разных языках. В настоящее время существует плагин для языка C, а также создается плагин для языка FBD IEC 61131-3. Первый предоставляет интегрированный редактор C. Чтобы структурировать этот код функций на C, разработчики могут определить пользовательские функции, которые вызываются из предыдущих. Плагин для FBD IEC 61131-3 предоставляет графический редактор и библиотеку стандартных функций (числовые и логические функции, выбор, сравнение,

таймеры, счетчики и т. д). Выполнение графического кода на FBD реализуется службой промежуточного программного обеспечения. Редактор типов компонентов также может определять SFC IEC 61131-3 (также обозначенные графы) с помощью графического редактора. Компонент может иметь несколько SFC. Этот редактор SFC представляет собой инструмент для разработки и представления диаграмм, предоставляя набор инструментов для вставки начальных шагов, последующих шагов, переходов, схождений и расхождений. Редактор SCF проверяет, правильно ли сформирована диаграмма. Выполнение SFC также реализуется службой промежуточного программного обеспечения. Типы компонентов сгруппированы как специфические и общие. Первыми являются те, которые используются только в определенных приложениях, например для определения бизнес-логики приложения, и они связаны с приложением. Вторые повторно используются в приложениях, например PID-регулятор. Они составляют библиотеки компонентов.

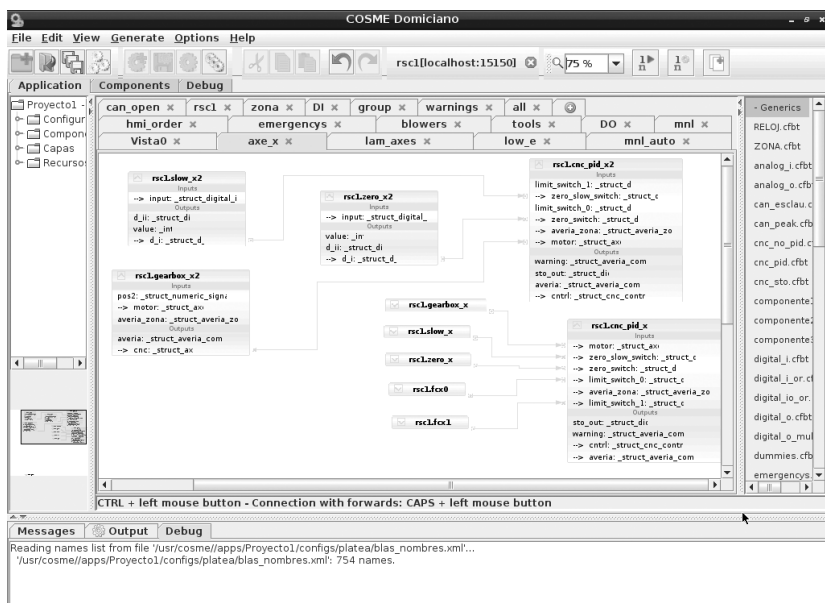


Рисунок 14.6. Компонент и сетевой редактор FB

Редактор приложений: Domiciano включает графический редактор для компонентов сети. Чтобы упростить редактирование, можно определить виды и слои. Представление – это графическое представление одного или нескольких слоев, а слой – это логическая структура, которая содержит подсеть. Компонент и соединения могут отображаться в одном или нескольких слоях. Эта структура позволяет разделить различные аспекты приложения и облегчает просмотр сети. Разработчики должны выбрать компоненты для создания экземпляров, а затем соединить входные и выходные данные экземпляров. Редактор допускает

соединение входов и выходов только между совместимыми типами. Разработчики могут настраивать инициализацию экземпляров, назначая конкретные значения внутренним переменным. Внутренние переменные экземпляра имеют значения по умолчанию, определенные при создании компонентов. Приложения сохраняются в определенном формате в файлах XML. Файл конфигурации XML позволяет Domiciano загружать предопределенные шаблоны подсетей (например, для задач управления отказами) и типы компонентов при создании нового приложения. Файл конфигурации XML указывает доступные генераторы кода и профили связи развертывания; это также экономит рабочее пространство Domiciano (то есть состояние рабочего стола).

Генератор кода: этот элемент автоматически генерирует файлы исходного кода для определенной среды выполнения. Он может генерировать код для любой среды выполнения, для которой установлен плагин. Доступный плагин времени выполнения в Domiciano – RTAI. Он генерирует собственный код для систем Linux с расширениями в реальном времени.

Инструмент развертывания: этот элемент развертывает файлы исходного кода и вызывает целевой компилятор для генерации исполняемых файлов двоичного кода. Если среда выполнения находится на другом контроллере, эти файлы исходного кода передаются по сети и загружаются на постоянный носитель, а компилятор вызывается удаленно. Этот инструмент также вызывает загрузчик приложения, который загружает типы компонентов, создает и запускает приложение. Есть два режима работы загрузки. Первый загружает приложение без остановки среды выполнения, а второй останавливает среду выполнения. Затем загрузчик запускает среду выполнения и загружает приложение. Если среда выполнения находится на другом компьютере, эти действия запускаются удаленно.

Инструмент отладки: этот элемент позволяет читать и записывать внутренние переменные компонентов, входы и выходы для отладки приложения. Он связывается с промежуточным ПО через шлюз. Чтобы облегчить отладку, дизайнеры могут наблюдать за развитием графически. Этот инструмент предоставляет другой тип визуализации, используя таблицы с текущими значениями. Также инструмент может выделять значения вне предопределенного диапазона. Дизайнеры могут изменять значения переменных, объявленных изменяемыми. Свойство компонента устанавливается при его редактировании. На рисунке 14.7 показаны графические и табличные представления.

14.4.3. Связь с внешними приложениями

Связь с внешними приложениями осуществляется с помощью моделей обмена сообщениями КЛИЕНТ–СЕРВЕР или PUBLISH–SUBSCRIBE. В обоих случаях сообщения имеют тот же формат, что и в случае межконтроллера. Библиотека Arcadio осуществляет связь во внешних приложениях, скрывая

эхо-телеграмм). 3) Внешнее приложение явно читает значения из Arcadio, взятые из буфера, в котором хранятся полученные значения.

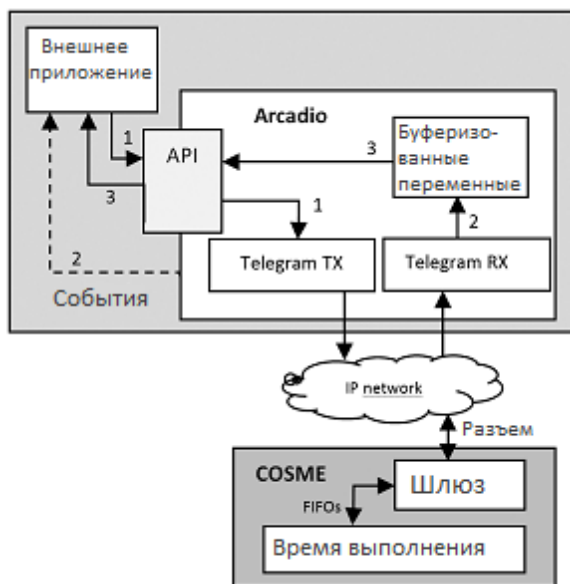


Рисунок 14.8. Внешние приложения могут общаться с COSME с помощью библиотеки Arcadio. Arcadio отдает приоритет модели асинхронного программирования, которая подходит для приложений интеграции HMI, SCADA и MES/ERP; синхронные операции также возможны.

Может потребоваться заблокировать выполнение внешнего приложения до получения значения или до тех пор, пока не будет гарантировано, что определенное значение было записано в приложении COSME. Для этих случаев доступна синхронная модель. Обычно внешние приложения будут заинтересованы в чтении выходных данных компонента и записи входных данных компонента, но Arcadio также предоставляет доступ к внутренним переменным компонента (если они были помечены как открытые или доступные для записи во время разработки). Хотя это явно нарушает принцип инкапсуляции, это было решено с учетом средств отладки и диагностики. Интерфейс HMI, разработанный для машины для изучения конкретных ситуаций, построен на основе библиотеки Arcadio. Учитывая сетевой характер Arcadio, приложение HMI может находиться на одном компьютере, на котором запущены COSME и система управления (что является решением), или на разных компьютерах. Большая часть HMI построена с использованием концепции «мешка», получая от COSME около 5 КБ данных каждые 200 мс (это считается достаточным для частоты обновления HMI). Испытания были успешно проведены с периодами до 5 мс с использованием обычных локальных сетей.

14.5. Практические проблемы и выводы

Используемая версия COSME работает на ПК с Linux RTAI 3.7.1, а некоторые части промежуточного программного обеспечения используют Java SE6. RTAI модифицирует ядро Linux, чтобы иметь возможность выполнять задачи в реальном времени в пространстве пользователя или ядра. В пространстве ядра приложения RTAI реализуются с помощью модулей ядра Linux, которые могут загружаться или выгружаться из памяти во время выполнения. Это решение было принято в текущей версии COSME, в которой элементы времени выполнения и типы компонентов являются модулями ядра. COSME связывает программное обеспечение системы управления и внешние приложения посредством элемента промежуточного программного обеспечения шлюза. При этом используются RTAI FIFO для связи со средой выполнения и программные гнезда для внешних приложений.

Аппаратными элементами, используемыми в этом станке для резки стекла, являются один B&R Automation PC 620 с двумя сетевыми интерфейсами 100 Мбит/с; две мастер-карты PCI-7841Adlink CANopen, один клапан S0700 SMC, один удаленный I/O X20 B&R, девять стандартных приводов Sdrive Power Stages и один CISCO SLM200 Gigabit Smart Switch.

Как упоминалось ранее, модель выполнения компонента основана на последовательной цепи. Укажем predetermined типы цепочек (в порядке убывания): инициализация, финализация, внешнее событие, нормальное реальное время, нормальное нереальное время и фон. Каждая цепочка последовательно выполняет сценические функции соответствующего компонента. Функциональные возможности CNC и PLC выполняются в обычной цепочке реального времени, которая последовательно выполняется каждые 1 мс в соответствии с порядком компонентов, предварительно определенным во время разработки. Функциональные возможности, связанные с профилактическим обслуживанием, внешней связью и управлением данными, выполняются в рамках обычной цепочки не в реальном времени, которая последовательно выполняется каждые 20 мс. Кроме того, вычисления траекторий CNC выполняются в фоновой цепочке, которая последовательно выполняется по требованию. Наконец, обработка CANopen выполняется в цепочке внешних событий, когда происходит внешнее событие.

Прототип станка для резки стекла показан на рисунке 14.1. Программное обеспечение системы управления имеет следующие характеристики.

1. Использовались 35 различных типов FB, например цифровой I/O, шина CAN, коробка передач (для изменения масштаба переменных), PID, аварийные ситуации.

2. Для реализации сети FB необходимо около 200 экземпляров этих типов FB. Таким образом, модель, определенная на рисунке 14.3, может быть реализована.

3. Семьдесят SFC определены.

4. Программное управление работает с 9 осями в цикле 2 мс с помощью ПК Celeron.

COSME предоставляет разработчикам систем управления упрощенную модель, адаптированную к конкретным станкам, что упрощает понимание и обслуживание приложений. Данная модель продвигает более сложные продукты, добавляя такие функции, как встроенное профилактическое и прогностическое обслуживание, интеграция с системами MES и ERP, дистанционная диагностика, повышение надежности программного обеспечения и сокращение времени выхода на рынок.

В этой главе показано промышленное применение с реальными требованиями. Применение хорошо зарекомендовавших себя технологий в программной инженерии (например, компонентное программирование) представляет собой преимущество для компании за счет сокращения времени проектирования, отладки и маркетинга продуктов.

Библиография

- [1] 4DIAC. www.fordiac.org/.
- [2] R. W. Brennan, P. Vrba, P. Tichy, A. Zoitl, C. Sünder, T. Strasser, and V. Marik. Developments in dynamic and intelligent reconfiguration of industrial automation. *Computers in Industry*, Vol. 59, , 533–547, 2008.
- [3] C. Catalán, F. Serna, A. Blesa, and J. Rams. Iec 61499 execution model based on life cycle of function blocks. In *Emerging Technologies and Factory Automation, IEEE Conference on*, pages 1–8, 2010.
- [4] C. Catalán, F. Serna, A. Blesa, J. Rams, and J. Colom. Communication types for manufacturing systems: A proposal to distributed control system based on IEC 61499. In *Automation Science and Engineering, IEEE Conference on*, pages 767–772, 2011.
- [5] C. Catalán, F. Serna, T. Civera, A. Blesa, and J. Rams. Control design for machine tools using domiciano, an IDE based on software components. In *Emerging Technologies Factory Automation, IEEE 16th Conference on*, pages 1–4, 2011.
- [6] J. H. Christensen. Design patterns for systems engineering with IEC 61499. *Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung*, 2000.
- [7] M. Colla, T. Leidi, and M. Semo. Design and implementation of industrial automation control systems: A survey. In *Industrial Informatics, 7th IEEE International Conference on*, pages 570–575, 2009.
- [8] N. Hage and B. Wagner. Implementation alternatives for the OMAC state machines using IEC 61499. In *IEEE Conference on Emerging Technologies and Factory Automation*, 2008.
- [9] K. Hall, R. Staron, and A. Zoitl. Challenges to industry adoption of the IEC 61499 standard on event-based function blocks. In *IEEE Conference on Industrial Informatics (INDIN)*, 2007.
- [10] D. Hästbacka, T. Vepsäläinen, and K. S. Model-driven development of industrial process control applications. *Journal of Systems and Software*, Vol. 84, 1100–1113, 2011.

- [11] Holobloc. <http://www.holobloc.com/doc/fbdk/index.htm>, 2012.
- [12] Iacocca Institute. 21st century manufacturing enterprise strategy: An industry-led view. 1991.
- [13] International Electrotechnical Commission. *Programmable Controllers. Part 3: Programming Languages, IEC 61131-3*, 2.0 edition, 2003.
- [14] International Electrotechnical Commission. *Function Blocks. Part 1: Architecture IEC 61499-1*, 2.0 edition, 2005.
- [15] ISaGRAF. www.isagraf.com.
- [16] Y. Koren, U. Heisel, F. Jovane, T. Moriwak, G. Pritsshow, G. Ulsoy, and H. van Brussel. Reconfigurable manufacturing systems. *Annals of the International Institution for Production Engineering Research (CIRP)*, Vol. 48, No. 2, 527–539, 1999.
- [17] R. Lewis. *Modelling Control Systems Using IEC 61499 Applying Function Blocks to Distributed Systems*. Number 59. Institution of Engineering and Technology, 2001.
- [18] S. Mekid, P. Pruschek, and J. Hernández. Beyond intelligent manufacturing: A new generation of flexible intelligent NC machines. *Mechanism and Machine Theory*, Vol. 44, No. 2, 466–476, 2009.
- [19] M. Minhat, V. Vyatkin, X. Xu, S. Wong, and Z. Al-Bayaa. A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks. *Robotics and Computer-Integrated Manufacturing*, Vol. 25, No. 3, 560–569, June 2009.
- [20] nxtControl. <http://www.nxtcontrol.com>.
- [21] G. Pritschow, Y. Altintas, F. Jovane, and Y. Koren. Open controller architecture: Past, present and future. In *Annals of the CIRP*, volume 50, pages 463–470, 2001.
- [22] F. Serna, C. Catalán, A. Blesa, J. Colom, and J. Rams. Predictive maintenance surveyor design pattern for machine tools control software applications. In *Emerging Technologies Factory Automation, IEEE 16th Conference on*, pages 1–7, 2011.
- [23] F. Serna, C. Catalán, A. Blesa, and J. Rams. Design patterns for failure management in iec 61499 function blocks. In *Emerging Technologies and Factory Automation, IEEE Conference on*, pages 1–7, 2010.
- [24] F. Serna, C. Catalán, A. Blesa, J. Rams, and J. Colom. Preventive maintenance manager design pattern for component-based machine tools. In *Industrial Informatics, 9th IEEE International Conference on*, pages 615–620, 2011.
- [25] T. Strasser, M. Rooker, I. Hegny, M. Wenger, A. Zoitl, L. Ferrarini, A. Dede, and M. Colla. A research roadmap for model-driven design of embedded systems for automation components. In *Industrial Informatics, 7th IEEE International Conference on*, pages 564–569, 2009.
- [26] Tuomas Group. www.tuomas.com.
- [27] V. Vyatkin. *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. Instrumentation Society of Automation (ISA), 2011.
- [28] V. Vyatkin and J. Chouinard. On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations. In *IEEE Conf. on Industrial Informatics*, 2008.

- [29] V. Vyatkin, M. Hirsch, and H.-M. Hanisch. Systematic design and implementation of distributed controllers in industrial automation. In *IEEE Conference on Emerging Technologies and Factory Automation*, pages 633–640, 2006.
- [30] H. Xuemei. Distributed and reconfigurable control of lower level machines in automatic production line. In *8th World Congress on Intelligent Control and Automation*, pages 2339–2344, 2010.
- [31] A. Zoitl, T. Strasser, and A. Valentini. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study. In *Industrial Electronics, IEEE International Symposium on*, pages 3817–3819, 2010.

Распределенное интеллектуальное зондирование и управление для автоматизации производства

Robert W. Brennan

Университет Калгари

СОДЕРЖАНИЕ

15.1 Введение	357
15.2 Похожие проблемы	359
15.2.1 Реконfigurирование распределенных систем	359
15.2.2 Интеллектуальное промежуточное программное обеспечение	360
15.3 Архитектура DISCS	363
15.4 Реализация функционального блока	366
15.5 Пример: мобильное объектное отслеживание	367
15.6 Дальнейшая работа	369

15.1. Введение

Чтобы оставаться конкурентоспособными на современном мировом рынке, производителям требуются системы, способные быстро реагировать на изменения, поддерживая стабильную и эффективную работу. Система управления производством все чаще рассматривается в качестве ключевой для достижения этой цели. Однако основные препятствия для успеха в этой области являются результатом сочетания все более строгих требований клиентов (например, высококачественные, настраиваемые, недорогие продукты, которые можно быстро доставить) и внутренней сложности производственной системы (то есть эти системы имеют распределенный, параллельный и стохастический характер). Хотя технологии производства становятся все более усовершенствованными для решения этих проблем (например, с помощью передовой робототехники и компьютерного числового управления), без адекватного контроля результатом часто является набор «островков автоматизации», в которых отсутствует необходимая интеграция для правильного функционирования.

В этой главе описывается подход распределенных интеллектуальных систем зондирования и управления (DISCS), который отвечает основным потребностям современных производственных систем (то есть обработке нарушений, доступности, гибкости и надежности). Наш подход фокусируется на стратегиях проектирования для реконфигурации, которые поддерживаются соответствующими методами системного анализа и управления безопасностью. Мотивация для этой работы состоит в том, чтобы позволить производственным системам быстро реагировать на изменения, поддерживая стабильную работу системы и эффективно используя доступные ресурсы.

Видя распространение повсеместно используемых вычислений, интеллектуальных датчиков и сертифицированных встроенных компьютеров в самолетах, медицинском оборудовании и других устройствах с высокой степенью защиты, в последние годы мы отмечаем значительный интерес к моделированию и анализу промышленных вычислительных систем [14]. В область производства основные инструменты и методы для поддержки автоматизации и управления пришли из областей агентских систем [34] и языков программируемого логического контроллера (PLC) [27]. Стандарт IEC 61499 [28] связывает эти подходы, используя конструкцию на основе компонентов, которая поддерживает конфигурируемость, совместимость и переносимость. В последние годы это привело к значительному прогрессу в моделировании распределенных интеллектуальных систем автоматизации [31].

Общесистемное развертывание чувствительных и управляющих устройств, особенно беспроводных устройств, становится все более распространенным в промышленных приложениях. Эти устройства пространственно распределены и в основном подключены беспроводным способом для сбора, мониторинга и контроля состояния больших, дорогих или опасных объектов и инфраструктур. В этой главе мы предлагаем применять беспроводные сенсорные сети (WSN) к этой проблеме в форме DISCS. В DISCS чувствительные и управляющие устройства пространственно распределены по узлам инфраструктуры, и принятие решений в приложениях распределено, а не централизовано. Как для повышения производительности и эффективности, так и для повышения безопасности распределенное зондирование и контроль демонстрируют беспрецедентный потенциал.

Для предоставления базовых услуг, необходимых для поддержки распределенного интеллектуального зондирования и управления, мы предлагаем ориентированную на приложение архитектуру промежуточного программного обеспечения, которая учитывает как требования приложений, так и ограничения сетевых ресурсов. Целью нашей работы над этой архитектурой системы является создание основы для поддержки «интеллектуального» поведения наших DISCS (в данном контексте под «интеллектом» понимается способность системы автоматически перенастраиваться в ответ на изменения). Более конкретно мы планируем использовать нашу прикладную архитектуру промежуточного программного обеспечения [5] в качестве основы для многоуровневой модели DISCS, которая объединяет программные агенты высокого уровня с функциональными

блоками низкого уровня IEC 61499 для достижения динамической реконфигурации. Агенты более высокого уровня будут использоваться для управления процессом реконфигурации, который будет поддерживаться низкоуровневыми службами в реальном времени нашей архитектуры промежуточного программного обеспечения.

В следующем разделе мы представим обзор недавней работы над распределенными системами зондирования и управления с акцентом на реконфигурирование системы и поддержку промежуточного программного обеспечения. Далее следует описание предложенного нами DISCS. Сначала описывается общая архитектура DISCS в разделе 15.3, затем реализация функционального блока в разделе 15.4 и, наконец, пример реализации для отслеживания мобильного узла в разделе 15.5. Глава завершается кратким изложением о нашей текущей и будущей работе в этой области.

15.2. Похожие проблемы

В этом разделе мы даем краткий обзор последних работ о распределенных системах зондирования и управления. Учитывая, что одним из ключевых мотивов нашей работы в этой области является разработка систем, способных быстро реагировать на изменения, мы начнем с обзора работы по реконфигурации распределенных систем. Затем следует обзор соответствующей работы по интеллектуальному промежуточному программному обеспечению для поддержки беспроводных сенсорных сетей.

15.2.1. Реконфигурирование распределенных систем

Предыдущие исследования о динамической реконфигурации подпадают под действие того, что Guler и соавторы [9] называет «управлением переходами»: то есть снижение влияния перенастройки программного обеспечения во время выполнения и поддержка согласованности и стабильности системы. Рассмотрение этих вопросов началось с работы Крамера и Маги о динамическом управлении изменениями в распределенных системах [12], которая была сосредоточена на том, чтобы позволить изменениям быть заданными на высоком уровне, затем происходило выполнение «управлением конфигурацией» на уровне задач.

Концепция «динамической реконфигурации» была введена Сетчи и Лагосом [24] и относилась к системам, которые «способны многократно изменять и переставлять [свои компоненты] экономически эффективным способом». Уолш и соавторы [32] классифицировали динамическую реконфигурацию, выделяя различные типы изменений, которые основывались на управлении целостностью системы. Этот подход включал построение модели предметной области, включающей различные режимы отказоустойчивости.

Предыдущие исследования по реконфигурации систем на основе функциональных блоков IEC 61499 [28] были выполнены исследователями из Университета Калгари [2] и Технического университета Вены, PROFACTOR и Университета прикладных наук в Вельсе [25] и были посвящены IEC 61499, архитектуре и концепции безвременных систем [23]. Эти работы были сосредоточены на реконфигурации на уровне компонентов и идентификации ключевых сервисов для поддержки интеллектуальной реконфигурации (то есть сервисов для поддержки контроля выполнения, взаимодействия состояний, запросов, библиотек компонентов). Rockwell Automation применила другой подход к интеллектуальной реконфигурации с помощью своей автономной кооперативной системы (ACS) [20], которая отделяет низкоуровневую часть системы управления в реальном времени от высокоуровневой части системы интеллектуального управления. Затем реконфигурация управляется и выполняется исключительно на высоком уровне (программный агент).

Несмотря на объем работы в этой области, механизмы для создания динамических и возникающих структур, необходимых для интеллектуальной реконфигурации на уровне устройств, нуждаются в лучшем рассмотрении [31]. Компромисс на этом уровне находится между совещательным и реактивным поведением, то есть накладные расходы на обработку, необходимые для поддержки интеллектуального принятия решений, могут препятствовать способности агента удовлетворять временным ограничениям физического устройства.

15.2.2. Интеллектуальное промежуточное программное обеспечение

Промежуточное программное обеспечение распределенных систем зондирования и управления относится к программному обеспечению и инструментам, которые могут помочь скрыть сложность и неоднородность аппаратных систем более низкого уровня, а также упростить обработку и управление информацией на уровне распределенных приложений. Проблемы распределенного программного обеспечения для зондирования и управления многочисленны, поскольку часто меняются требования со стороны приложений, а также ограниченные сетевые ресурсы, включая пропускную способность и время автономной работы, динамическая топология сети и доступность данных, равно как и знания по обработке данных для конкретных приложений. Кроме того, отсутствие инструментов проектирования в этой области является в данный момент еще одной большой проблемой.

По сравнению с промежуточным программным обеспечением обычной сенсорной сети, промежуточное программное обеспечение для промышленной распределенной системы зондирования и управления должно уделять больше внимания требованиям приложений, управлению выполнением и способам упреждающей оптимизации обработки информации в соответствии с QoS (качеством обслуживания) и ограничениями ресурсов. За последние два

десятилетия исследования в этой области привлекли широкий интерес, что обусловлено потенциалом WSN в промышленных приложениях. Из-за уникальных характеристик и технических проблем как со стороны приложения, так и со стороны основной сети, исследования и разработки в области промежуточного программного обеспечения распределенных датчиков и управления были сосредоточены на общей методологии, а также на проблемах, специфичных для приложения.

В раннем обзоре исследований промежуточного программного обеспечения по распределенному зондированию и управлению, Romer и соавторы [22] определили проблемы, связанные с успешным промежуточным программным обеспечением WSN как с точки зрения концепции проектирования, так и реализации системы. Во время их обзора большинство исследований было на ранней стадии, и результаты были основаны на моделировании или небольших экспериментах в лабораторных условиях.

Liu и Martonosi [16] предложили систему промежуточного программного обеспечения для управления автономной параллельной сенсорной системой под названием Impala. Impala обеспечивает модульность, адаптивность и возможность восстановления приложений в WSN. Это также позволяет получать обновления программного обеспечения через беспроводной приемопередатчик узла и динамически применять к работающей системе. Кроме того, Impala также предоставляет интерфейс для адаптации приложений «на лету» для повышения производительности, энергоэффективности и надежности программной системы.

В [11] авторы описали различные типы приложений сенсорных сетей и обсудили существующие методы управления этими типами сетей. Они также рассмотрели различные связанные промежуточные программы и утверждают, что ни один из существующих подходов не обеспечивает все инструменты управления, необходимые для приложений сенсорной сети. Чтобы удовлетворить эту потребность, авторы разработали новое промежуточное программное обеспечение, называемое связующим программным обеспечением и сетями приложений (MiLAN), предназначенное для наблюдения за средой, обеспечения безопасности дома и в офисе и медицинского мониторинга.

Архитектуру сенсорной сети EPC (ESN) предложили Wang и соавторы [33] в качестве системы интеграции RFID и WSN. Ядром ESN была промежуточная часть. В предлагаемом промежуточном программном обеспечении использовалась сложная технология обработки событий, которая могла обрабатывать большие объемы вентиляционных отверстий от распределенных RFID и считывателей датчиков в режиме реального времени. Посредством фильтрации, группировки, агрегирования и построения сложных событий промежуточное ПО ESN предоставило клиентам полезные отчеты и улучшило автоматизацию системы.

Гунгор и Ханке [10] проанализировали проблемы, принципы проектирования и технические подходы в промышленных беспроводных сенсорных сетях. Их работа позволила выявить ряд проблем, которые

необходимо решить с помощью промежуточного программного обеспечения, такие как аналитические модели для эффективного развертывания IWSN в реальном мире, разнообразные требования к промышленным приложениям, крупномасштабные сети, оптимальное развертывание сенсорных узлов, локализация, безопасность и совместимость между различными производителями IWSN. ,

Распределенное зондирование для улучшения качества и производительности обсуждалось в [6]. В документе также обсуждались современные технологии, исследовательские задачи и будущие направления, связанные с распределенным зондированием. Обсуждение включало в себя оптимальный дизайн распределенных сенсорных систем, информационные критерии и обработку для распределенного зондирования и принятия оптимальных решений в распределенном зондировании.

Для решения проблемы совместимости различных типов датчиков и исполнительных механизмов, использующих различные технологии беспроводной связи (WiFi, Bluetooth или RFID), Ramamurthy и соавторы [21] разработали беспроводную интеллектуальную сенсорную платформу, которая обладает функцией «подключи и работай» для поддержки аппаратного интерфейса, потребностей в полезной нагрузке и связи, а также средств для обновления рабочих и контрольных параметров, а также для датчиков и радиочастотных линий и модулей прошивки «по воздуху». Исследование было нацелено на применение измерительных систем и систем прогнозного обслуживания и было сосредоточено на уровнях аппаратного и встроенного программного обеспечения.

Сервис-ориентированная архитектура для конфигурации QoS и управления WSN была предложена Anastasi и соавторами [1]. Предлагаемое промежуточное ПО пыталось скрыть сложность низкоуровневых физических устройств и поддерживать управление разнородными данными в реальном времени на уровне предприятия.

В [35] предлагается быстрое и одновременное решение для агрегации данных для расширенного анализа данных в нескольких регионах. Новая распределенная структура данных, называемая распределенным кубом данных, используется в обработке запросов для расширения приложений WSN.

Мартинес и соавторы [18] представили метод проектирования и проверки для беспроводных сетей датчиков и исполнительных механизмов, позволяющий устранить сложность недетерминированного и одновременного поведения распределенных систем и улучшить результаты анализа и проектирования в сценариях моделирования и тестирования. Вместо того чтобы сосредотачиваться на исследовании общей методологии, подходы WSN для конкретных приложений исследуются в [17] и [13]. Авторы первой из этих работ мотивировались реальными бизнес-кейсами нефтегазовой отрасли и были нацелены на взаимодействие между различными системами микропрограмм. Они представили трехслойную сервис-ориентированную архитектуру, которая вмещала различные сенсорные платформы и унифицировала их функциональность для бизнес-приложений.

Таким образом, развертывание WSN в промышленности будет происходить постепенно, и промежуточное программное обеспечение будет иметь решающее значение для формирования общих беспроводных сенсорных устройств и появления реальных промышленных решений для распределенных прикладных систем. Несмотря на значительные усилия, приложенные к исследованиям промежуточного программного обеспечения, большая часть работы была сосредоточена на проблемах сети или приложениях только для обнаружения.

По-прежнему не существует промежуточного программного обеспечения с эффективными инструментами, обладающими требуемыми функциями для промышленного промежуточного программного обеспечения для распределенного зондирования и управления; то есть интеллект узла и низкоуровневая поддержка управления особенно слабы. В оставшейся части этой главы мы опишем интеллектуальную конструкцию промежуточного программного обеспечения, основанную на международном стандарте IEC 61499 [30] для моделирования распределенных систем управления и интеллектуальных агентов.

15.3. Архитектура DISCS

Предложенная нами прикладная интеллектуальная архитектура промежуточного программного обеспечения для DISCS показана на рисунке 15.1. Предлагаемое промежуточное ПО имеет двухуровневую структуру: агент облегчает управление на верхнем уровне и реализацию на нижнем уровне на основе функциональных блоков.

На верхнем уровне QoS приложения принимается в качестве входных данных для диспетчера устройств и диспетчера задач, которые работают совместно для удовлетворения потребностей приложения путем динамической передачи требуемой конфигурации сети и информации о данных в сеть более низкого уровня. Информация о конфигурации и данных отправляется и принимается через универсальный шлюз на основе технологии «подключи и работай» (UPnP) и базовые протоколы обработки сети и событий.

На более низком уровне основанный на приложениях интеллект встроен в соответствующие узлы датчика (то есть узлы приемника). Зависящие от устройства необработанные данные собираются из определенного узла или кластера узлов и обрабатываются локально. В результате только прикладные ключевые параметры, выводимые из встроенных алгоритмов, передаются в диспетчер задач верхнего уровня. Ключевыми связями между различными модулями в системе являются двунаправленные события, которые запускают, реконфигурируют или переводят узлы в спящий режим в соответствии с потребностями верхнего уровня. Между тем двунаправленные события также используются для обнаружения, удаления или реорганизации топологии узла на основе доступности физического узла и сетевого подключения. Благодаря предложенной архитектуре промежуточного программного обеспечения

информация более высокого уровня и данные более низкого уровня обрабатываются локально, что делает связь между уровнями проще и эффективнее. Кроме того, QoS может быть гарантировано этой гибкой структурой промежуточного программного обеспечения посредством быстрой реконфигурации сети. В предлагаемом промежуточном программном обеспечении есть четыре уникальных компонента: диспетчер устройств, диспетчер задач, узлы приемника и двунаправленные события.

1. Диспетчер устройств заботится о восходящей и нисходящей конфигурации физических сетевых устройств. Восходящая конфигурация включает в себя настройку сети для отражения изменений физического соединения на более низком уровне. Нисходящая конфигурация включает в себя настройку сети в соответствии с потребностями приложения, что может включать кластеризацию узлов и повторную кластеризацию. Как правило, диспетчер устройств хранит информацию о текущем состоянии сетевого подключения, описаниях узлов и рабочем состоянии каждого узла. Что еще более важно, диспетчер устройств сопоставляет сетевые подключения с потребностями приложений.

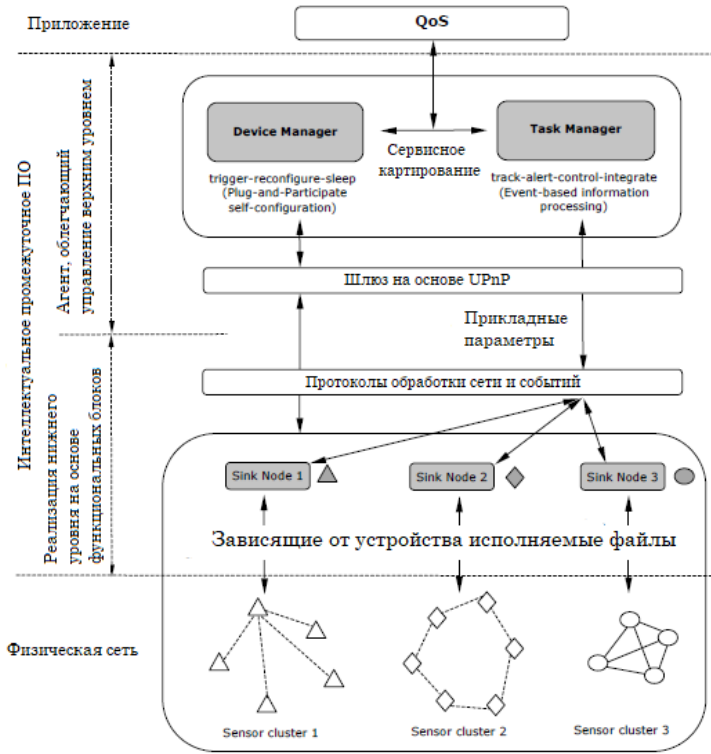


Рисунок 15.1. Архитектура распределенного интеллектуального зондирования и управления

2. Диспетчер задач занимается тем, чтобы удовлетворить требования приложения. Например, типичные задачи приложения включают отслеживание движущихся объектов, оповещение о состоянии, мягкое управление в реальном времени и интеграцию всех этих задач. Чтобы получить необходимую информацию для каждой прикладной задачи на уровне устройства, ключевым моментом является сотрудничество с диспетчером устройств.

3. Узлы приемника функционируют как агрегаторы данных. Когда задача обнаружения или управления очень сложна, собирается большое количество необработанных данных, а встроенные алгоритмы обработки данных требуют более эффективных вычислений; узел приемника будет развернут для отделения обработки данных от сбора основных данных. Ввод узла приемника – это необработанные данные от датчиков узла; выходы узла приемника являются параметрами, требуемыми приложением.

4. Двухнаправленный информационный поток событий внутри нашего интеллектуального промежуточного программного обеспечения управляется двухнаправленными событиями, которые обозначены как двухнаправленные стрелки на рисунке 15.1. Управляющие события перетекают с верхнего уровня на нижний, а воспринимающие события – в противоположном направлении. Это приводит к созданию промежуточного программного обеспечения на основе событий, в котором двухнаправленные события принимают потребности приложений верхнего уровня в качестве входных данных для более эффективного использования ресурсов более низкого уровня. Они также формируют путь от верхнего уровня к нижнему для контроля выполнения, а не просто путь от нижнего уровня к верхнему для восприятия.

Как показано на рисунке 15.1, для нашего интеллектуального промежуточного программного обеспечения приняты две технологии для распределенных приложений. Это интеллектуальные агенты и функциональные блоки IEC 61499. На верхнем уровне интеллектуальные агенты облегчают функции управления предлагаемому промежуточному программному обеспечению благодаря своим возможностям ведения переговоров и принятия решений. Требуемые функции управления включают в себя динамическую конфигурацию устройства, декомпозицию задач приложения и сопоставление услуг между диспетчером устройств и диспетчером приложений. На нижнем уровне функциональные блоки будут использоваться в качестве инструментов моделирования для обработки информации на основе событий, встраивания интеллектуальных узлов и контроля выполнения на уровне устройства. В отличие от программных агентов, функциональные блоки IEC 61499 управляются событиями и могут быть встроены во внутренний алгоритм. Они предназначены для выполнения сложных низкоуровневых устройств и управления ими. Более подробную информацию об общей архитектуре DISCS можно найти в [4] и [5]; реализация многоагентной системы верхнего уровня подробно описана в [7] и [8]. В

следующем разделе мы сосредоточимся на реализации функционального блока нижнего уровня предлагаемого нами DISCS.

15.4. Реализация функционального блока

Функциональные блоки IEC 61499 представляют собой модульные управляемые событиями функциональные блоки, которые являются исполняемыми, масштабируемыми и распространяемыми. Поэтому IEC 61499 хорошо подходит в качестве инструмента моделирования для предлагаемых нами DISCS, учитывая его способность достигать:

- 1) исполнения, управляемого событием;
- 2) инкапсуляции разнообразной прошивки;
- 3) прямого доступа к оборудованию.

Низкоуровневое промежуточное ПО требует явного и эффективного выполнения, учитывая его близость к физическим устройствам. Поскольку операционная система для встроенных WSN де-факто использует управляемое событиями выполнение [15], модель функционального блока на основе событий IEC 61499 является логическим выбором для этого приложения. Например, в нашей модели двунаправленные события верхнего уровня, связанные с диспетчером задач и диспетчером устройств, являются входами и выходами сети функциональных блоков (FBN) на стороне приложения, а триггеры и I/O оборудования являются входами и выходами FBN в сторону физического устройства. Как определено в IEC 61499, выполнение FBN управляется диаграммой внутри каждого функционального блока, которая связана с соответствующими входами и выходами данных.

Промышленные приложения распределенного зондирования и управления обычно включают в себя комбинацию различных типов аппаратных средств и связанных с ними встроенных программ. Чтобы скрыть сложность и неоднородность аппаратных систем более низкого уровня, можно использовать разные модели и типы функциональных блоков для инкапсуляции различных систем встроенного программного обеспечения в различные компоненты функциональных блоков уровня, каждый с унифицированными интерфейсами и четкими входами и выходами. Чтобы приспособиться к различным уровням интеллекта, которыми обладают различные устройства и прошивки, также могут быть разработаны необходимые встроенные интеллектуальные возможности с использованием функциональных блоков, чтобы использовать эти различия. Возможно даже полное промежуточное ПО на основе функциональных блоков с прямым доступом к оборудованию.

Наконец, учитывая неоднородность аппаратных систем низкого уровня, прямой доступ к оборудованию, когда это необходимо, является одной из наиболее важных функций промежуточного программного обеспечения. IEC 61499 основан на широко принятом стандарте IEC 61131-3 [29], который является инструментом прямого управления на уровне устройства. В результате он уже определяет архитектурные элементы, необходимые для

аппаратного доступа. В среде исполнения ведется значительная работа над технологией функциональных блоков для низкоуровневого управления в реальном времени [3, 26, 36].

На рисунке 15.2 показана архитектура подсистемы нашей низкоуровневой реализации на основе функциональных блоков. Ядром этой реализации является сопоставление задач приложений верхнего уровня и сетевых требований с приложениями функциональных блоков более низкого уровня посредством двунаправленных событий. Конечная цель этой подсистемы – поддерживать оптимизацию верхнего уровня и динамическую конфигурацию сети нижнего уровня. Принимая во внимание утечку энергии из-за увеличения связи между междуузлиями, большинство приложений функциональных блоков (см. FBA 1 и FBA 3 на рисунке 15.2) будут привязаны к отдельным узлам датчиков, отвечающим за мониторинг состояния и контроль работы отдельных машин.

15.5. Пример: мобильное объектное отслеживание

Чтобы проиллюстрировать способность модели IEC 61499 отображать задачи приложения между несколькими узлами (см. FBA 2 на рисунке 15.2), в этом разделе мы приведем пример отслеживания мобильных объектов. Отслеживание движущихся объектов является типичной проблемой локализации мобильного датчика, когда несколько статических узлов используются в качестве якорей. Якорные узлы работают вместе с движущимся узлом, оборудованным для отслеживания этого объекта. Связь на разных уровнях приложений функциональных блоков достигается посредством событий и связанных с ними данных через сетевые протоколы.

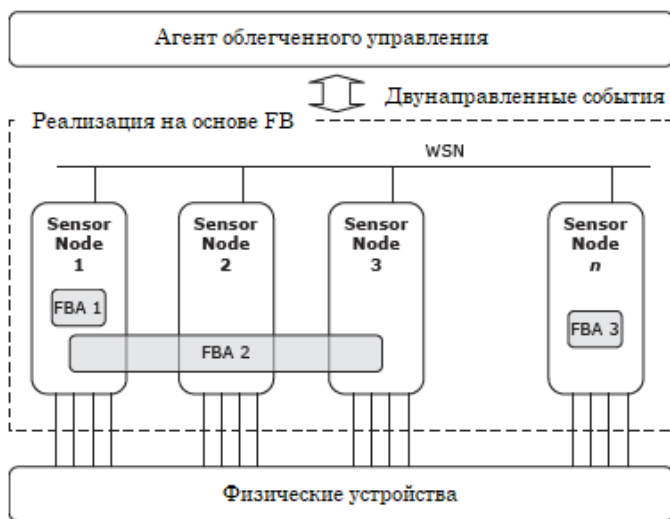


Рисунок 15.2. Архитектура подсистемы на основе функциональных блоков

Для этого приложения мы идентифицируем три типа беспроводных сенсорных узлов и пять программных агентов, как показано на рисунке 15.3. Базовая модель агента построена на архитектуре посредника [19] и опирается на узлы приемника в качестве узлов управления и локализации, учитывая их более высокую вычислительную способность. Мобильные узлы и узлы привязки представлены агентами мобильных узлов и агентами узлов привязки соответственно, и для целей моделирования фабричной среды модель включает в себя агент среды.

Для простоты на рисунке 15.3 показан только один экземпляр каждого из типов узлов. Однако в производственной среде различные опорные узлы распределены по географической области производственного цеха, в то время как один или несколько мобильных узлов перемещаются по среде по установленным путям между оборудованием. Узлы-приемники управляют процессом отслеживания мобильных узлов, привлекая соседние опорные узлы, которые получают оценки расстояния, к мобильным узлам и используя агентов локатора для триангуляции данных о расстоянии для определения местоположения мобильных узлов.

На рисунке 15.4 показано, как полученная беспроводная специализированная сеть отслеживает мобильный узел, формируя, переформируя и распуская специальные кластеры «на лету», когда мобильный узел пересекает свой путь. FBA IEC 61499 показана на рисунке 15.5. Как и на диаграмме классов агентов (см. на рисунке 15.3), мы показываем только один экземпляр узлов. Чтобы упростить рисунок, мы не даем мобильный узел, который включает публикацию сигналов радиомаяка, используемых узлом привязки для оценки расстояния с использованием разницы во времени прибытия (TDoA).

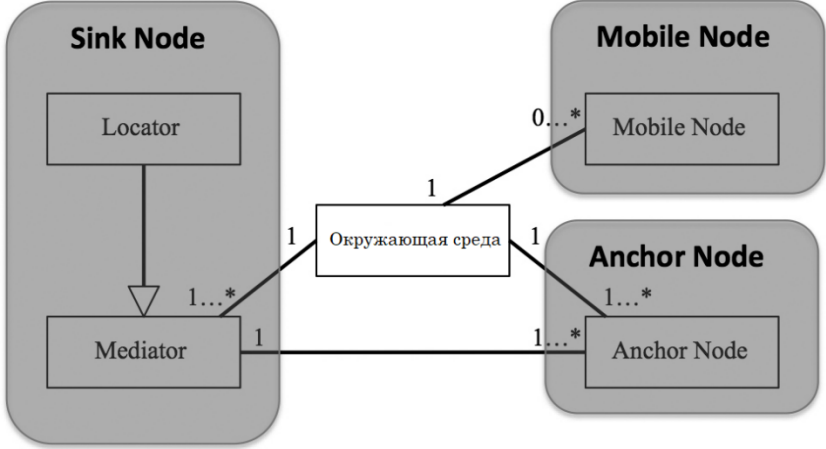


Рисунок 15.3. Диаграмма классов агента отслеживания мобильных объектов

Пример отслеживания движущегося объекта с нашей имитацией DISCS показан на рисунке 15.6. В этом примере отслеживание мобильных объектов управляется FBA, распределенными по нескольким узлам датчика (то есть узлам приемника и кластерам узлов привязки на рисунке 15.6). Группа сенсорных узлов, которые совместно используют задачу отслеживания мобильных объектов, образует кластер беспроводных сенсорных узлов в нашей модели. Чтобы сбалансировать нагрузку на ресурсы, мы передаем задачу отслеживания из одного кластера в другой, когда мобильный объект пересекает поле датчика. Более подробно об этом подходе можно найти в [7] и [8].

Поскольку функциональные блоки являются инструментами модульного проектирования, наша сетевая реализация более низкого уровня может использовать библиотеку предопределенных шаблонов функциональных блоков. Для конкретного приложения эти шаблоны могут быть быстро настроены в соответствии с требованиями задачи и конфигурациями узлов и встроены в FBN для конкретного приложения. Инкапсулируя интеллект узла и конфигурации узла в FBN, реализация более низкого уровня устраняет разрыв между требованиями приложений высокого уровня и аппаратными устройствами низкого уровня. Посредством двунаправленных событий совместное принятие решений агентами и реализация на основе событий, основанная на функциональных блоках, работают вместе, что делает все промежуточное ПО для DISCS интеллектуальным и адаптивным к требованиям приложений и сетевым ограничениям.

15.6. Дальнейшая работа

В этой главе мы описали подход распределенных интеллектуальных систем зондирования и управления (DISCS), предназначенный для того, чтобы позволить производственным системам быстро реагировать на изменения, поддерживая стабильную работу и эффективное использование ресурсов. В центре внимания этой работы была необходимость распределенного зондирования и управления на основе встроенных технологий WSN. Хотя количество исследований и разработок общих сетевых протоколов беспроводных датчиков значительно выросло в последние годы и технология становится все более и более зрелой, внедрение технологии в промышленность происходит намного медленнее, чем бы всем хотелось. Промежуточное программное обеспечение является одной из ключевых причин. Эффективное промежуточное программное обеспечение должно учитывать знания предметной области и ограничения сетевых ресурсов, а также объединять сложные прикладные знания о сети с помощью соответствующих инструментов.

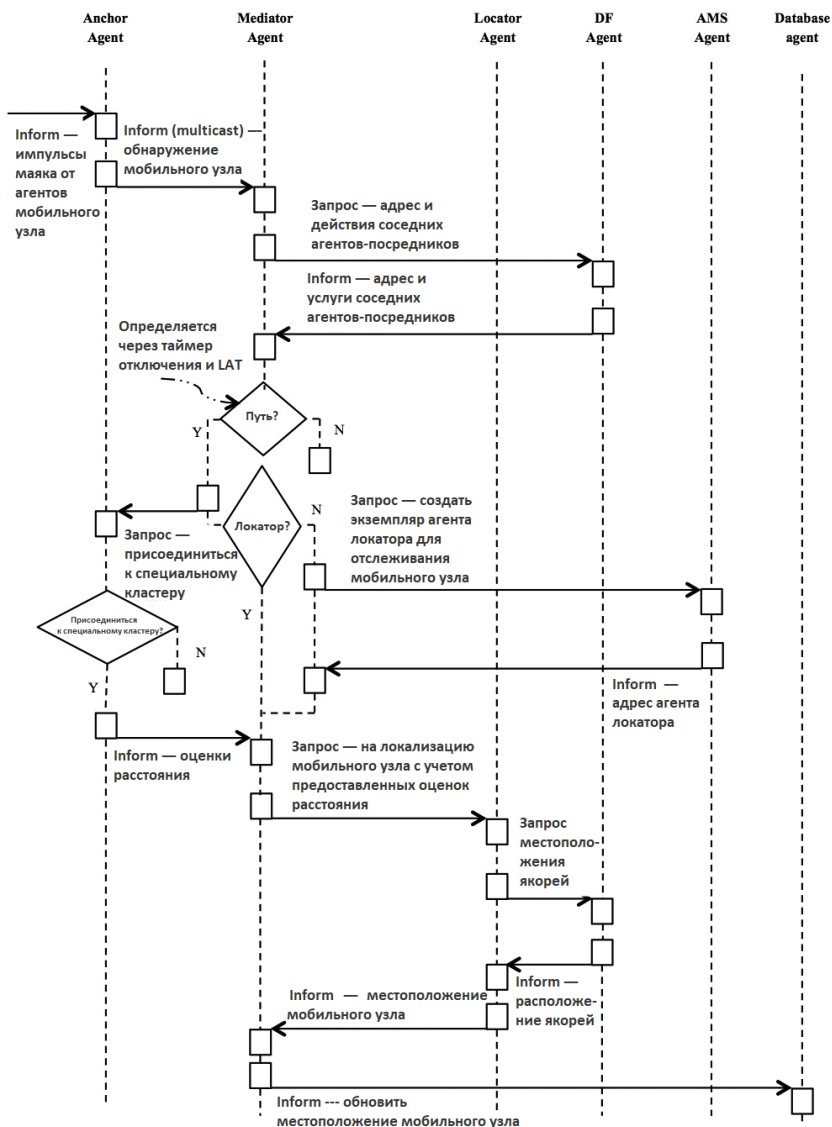


Рисунок 15.4. Схема зависимости агента отслеживания мобильных объектов

В этой главе для решения этих проблем предлагается подход интеллектуального промежуточного программного обеспечения, ориентированный на приложения. Он включает двухуровневую структуру: упрощенное агентское управление верхнего уровня и реализация нижнего уровня на основе функциональных блоков. Первый уровень отвечает за

оптимальное сопоставление между задачей приложения и топологией сети, учитывая как эффективное использование сетевых ресурсов, так и QoS приложения. Второй уровень фокусируется на управляемом событиями исполнении на уровне устройства, а функциональные блоки принимаются в качестве инструментов распределенного моделирования.

Рисунок 15.5. Приложение функционального блока для отслеживания мобильных объектов

Кластер глобальной сети, сформированный по специальной технологии для отслеживания мобильного узла, движущегося по круговому движению

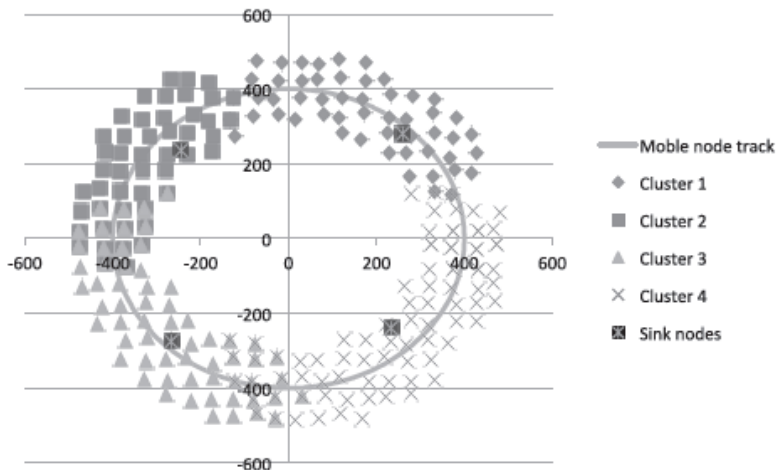


Рисунок 15.6. Отслеживание мобильных объектов с использованием симуляции DISCS

Библиография

- [1] G. Anastasi, E. Bini, A. Romano, and G. Lipari. A service-oriented architecture for qos configuration and management of wireless sensor networks. In *Proceeding of IEEE Conference on Emerging Technologies for Factory Automation*, 2010.
- [2] R. Brennan, M. Fletcher, and D. Norrie. An agent-based approach to reconfiguration of real-time distributed control systems. *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 4, 444–451, 2002.
- [3] R. Brennan, P. Vrba, P. Tichy, A. Zoitl, C. Sünder, T. Strasser, and V. Marik. Developments in dynamic and intelligent reconfiguration of industrial automation. *Computers in Industry*, Vol. 59, , 533–547, 2008.
- [4] N. Cai and R. Brennan. Distributed sensing and control architecture for automotive factory automation. In *Holonic and Multi-Agent Systems for Manufacturing*, volume 5696 of *Lecture Notes in Computer Science*, pages 165–174. Springer, 2009.
- [5] N. Cai, M. Gholami, L. Yang, and R. Brennan. Application-oriented intelligent middleware for distributed sensing and control. *IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, Vol. 42, No. 6, 947–956, 2012.
- [6] Y. Ding, E. Elsayed, S. Kumara, J. Lu, F. Niu, and J. Shi. Distributed sensing for quality and productivity improvements. *IEEE Transactions on Automation Science and Engineering*, Vol. 3, No. 4, 344–359, 2006.

- [7] M. Gholami and R. Brennan. Dynamic cluster formation and management in a factory wireless sensor network. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 2013.
- [8] M. Gholami, M. Taboun, and R. Brennan. Comparing alternative cluster management approaches for mobile node tracking in a factory wireless sensor network. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 2014.
- [9] M. Guler, S. Clements, L. Wills, B. Heck, and G. Vachtsevanos. Transition management for reconfigurable hybrid control systems. *IEEE Control Systems Magazine*, Vol. 23, , 36–49, 2003.
- [10] V. Gungor and G. Hancke. Industrial wireless sensor networks: Challenges, design principles, and technical approaches. *IEEE Transactions on Industrial Electronics*, Vol. 56, No. 10, 4258–4265, 2009.
- [11] B. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo. Middleware to support sensor network applications. *IEEE Networks*, Vol. 18, No. 1, 6–14, 2004.
- [12] J. Kramer and J. Magee. Dynamic configuration for distributed systems. *IEEE Transactions on Software Engineering*, Vol. 11, , 424–436, 1985.
- [13] S. Lee, T. Jeon, H.-S. Hwang, and C.-S. Kim. Design and implementation of wireless sensor based-monitoring system for smart factory. In *Lecture Notes in Computer Science*, volume 4706, pages 584–592, 2007.
- [14] N. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011.
- [15] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: an operating system for sensor networks. In E. A. Werner Weber, Jan M. Rabaey, editor, *Ambient Intelligence*, pages 115–148, 2006.
- [16] T. Liu and M. Martonosi. Impala: A middleware system for managing autonomic parallel sensor systems. In *Proceedings of 9th ACM SIGPLAN Symposium on Principles Practice Parallel Program*, pages 107–118, 2003.
- [17] M. Marin-Perianu, N. Meratnia, P. Havinga, L. DeSouza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker, and G. Stromberg. Decentralized enterprise systems: A multiplatform wireless sensor network approach. *IEEE Wireless Communications*, Vol. 14, No. 6, 57–66, 2007.
- [18] D. Martinez, A. Gonzalez, F. Blanes, R. Aquino, J. Simo, and A. Crespo. Formal specification and design techniques for wireless sensor and actuator networks. *Sensors*, Vol. 11, No. 1, 1059–1077, 2011.
- [19] F. Maturana and D. Norrie. Multi-agent mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing*, Vol. 7, No. 4, 257–270, 1996.
- [20] F. Maturana, R. Staron, and K. Hall. Methodologies and tools for intelligent agents in distributed control. *IEEE Intelligent Systems*, Vol. 20, 42–49, 2005.
- [21] H. Ramamurthy, B. Prabhu, R. Gadh, and A. Madni. Wireless industrial monitoring and control using a smart sensor platform. *IEEE Sensors Journal*, Vol. 7, No. 5, 611–618, 2007.

- [22] K. Romer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 6, No. 4, 59–61, 2002.
- [23] M. Rooker, C. Sünder, T. Strasser, A. Zoitl, O. Hummer, and G. Ebenhofer. Zero downtime reconfiguration of distributed automation systems: The _cedac approach. In *Proceedings of 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pages 326–337, 2007.
- [24] R. Setchi and N. Lagos. Reconfigurability and reconfigurable manufacturing systems: state-of-the-art review. In *Proceedings of 2nd International Conference on Industrial Informatics*, 2004.
- [25] T. Strasser and R. Froschauer. Autonomous application recovery in distributed intelligent automation and control systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 42, No. 6, 1054–1070, 2012.
- [26] T. Strasser, A. Zoitl, J. Christensen, and C. Sünder. Design and execution issues in iec 61499 distributed automation and control systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 39, No. 5, 534–546, 2009.
- [27] TC 65/SC 65B. IEC 61131: Programmable controllers – Part 3: Programming languages. International Electrotechnical Commission, Geneva, 1993.
- [28] TC 65/SC 65B. IEC 61499: Function blocks – Part 1: Architecture. International Electrotechnical Commission, Geneva, 2005.
- [29] TC 65/SC 65B. IEC 61131: Programmable controllers – Part 3: Programming languages. International Electrotechnical Commission, Geneva, 2nd ed., 2003.
- [30] V. Vyatkin. *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. Instrumentation, Systems, and Automation Society, 2007.
- [31] V. Vyatkin. IEC 61499 as enabler of distributed and intelligent automation: state-of-the-art review. *IEEE Transactions on Industrial Informatics*, Vol. 7, No. 4, 768–771, 2011.
- [32] J. Walsh, F. Bordeleau, and B. Selic. Domain analysis of dynamic system reconfiguration. *Software and Systems Modeling*, Vol. 6, No. 4, 355–380, 2007.
- [33] W. Wang, J. Sung, and D. Kim. Complex event processing in EPC sensor network middleware for both RFID and WSN. In *Proceedings of 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing*, pages 165–169, 2008.
- [34] G. Weiss. *Multi-Agent Systems*. MIT Press, 1999.
- [35] D. Wu and M. Wong. Fast and simultaneous data aggregation over multiple regions in wireless sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 41, No. 3, 333–343, 2011.
- [36] A. Zoitl. *Basic real-time reconfiguration services for zero down-time automation systems*. PhD thesis, Vienna University of Technology, 2007.

Управляемая моделями конструкция кардиостимулятора с использованием функциональных блоков IEC 61499

Yu Zhao

Оклендский университет

Partha S. Roop

Оклендский университет

СОДЕРЖАНИЕ

16.1 Введение	375
16.2 Коротко о системе стимуляции	376
16.2.1 Основы электрофизиологии и ее моделирования	377
16.2.2 Требования к времени и устройству кардиостимулятора	379
16.2.3 Связанная с темой работа	381
16.3 Обзор предложенного подхода	382
16.4 Моделирование с использованием функциональных блоков IEC 61499	384
16.4.1 Модель логики стимуляции в режиме DDD	384
16.4.2 Иерархическая структура модели контроллера кардиостимулятора	390
16.5 Создание высококачественной модели	392
16.5.1 Временной анализ синхронных функциональных блоков	393
16.5.2 Высокочастотные модели контроллера кардиостимулятора	394
16.5.3 Создание модели среды для системы замкнутого цикла	398
16.6 Анализ времени отклика с использованием высокочастотных моделей	399
16.7 Выводы	402

16.1. Введение

Использование устройств для контроля сердечного ритма, таких как имплантируемые дефибрилляторы и кардиостимуляторы, за последние десятилетия быстро возросло. Повысилось и качество исполнения таких

устройств, а также их сложность. Однако проблемы безопасности, вызванные программным обеспечением, все еще бросают вызов разработчикам. На основании данных Управления по контролю за продуктами и лекарствами США (FDA) в 2010 году, по крайней мере 6 из 23 отзывов неисправных устройств были связаны с ошибками программного обеспечения [13].

Анализ времени отклика [15] имеет решающее значение для программного обеспечения кардиостимулятора. В существующих методах для проведения такого анализа используются модели с замкнутым циклом на основе временных автоматов (ТА) [3]. Однако эти модели делают идеализированные предположения относительно базовой платформы выполнения, что может повлиять на точность проверки. В этой главе мы предлагаем подход, основанный на моделировании, начиная с моделирования программного обеспечения с использованием функциональных блоков IEC 61499. Мы выполняем анализ синхронизации низкого уровня, чтобы вычислить время выполнения наихудшего случая (WCET) [20] сгенерированного кода C из этих моделей высокого уровня. Полученные значения WCET используются для создания высокоточных моделей ТА, необходимых для анализа времени отклика. Более ранние структуры для анализа времени отклика [11, 13] выполняют анализ без учета низкоуровневой синхронизации базовой архитектуры. В отличие от них, предлагаемый подход позволяет проводить анализ времени точного ответа имплантируемых устройств, таких как кардиостимуляторы.

16.2. Коротко о системе стимуляции

Пациентам, страдающим от *брадикардии* (частота сердечных сокращений ниже нормы), устанавливаются кардиостимуляторы для регуляции сердечного ритма. Кардиостимулятор воспринимает внутренние импульсы от сердца и решает, следует ли стимулировать сердце, подчиняясь основным временным последовательностям. Кардиостимулятор не должен ускорять кардиостимуляцию у пациентов с *тахикардией* (частота сердечных сокращений выше нормальной). На рисунке 16.1 изображена замкнутая система, состоящая из сердца и ритма. Мы называем эту систему кардиостимулятором. Наблюдаются четыре взаимодействия.

- H_APulse обозначает предсердный пульс, генерируемый сердцем, который будет измеряться кардиостимулятором.
- APace обозначает предсердный стимул, генерируемый кардиостимулятором для регулирования частоты сердечных сокращений.
- H_VPulse обозначает вентрикулярный пульс, генерируемый сердцем, который будет измеряться кардиостимулятором.
- VPace обозначает желудочковый стимул, генерируемый кардиостимулятором для регулирования частоты сердечных сокращений.

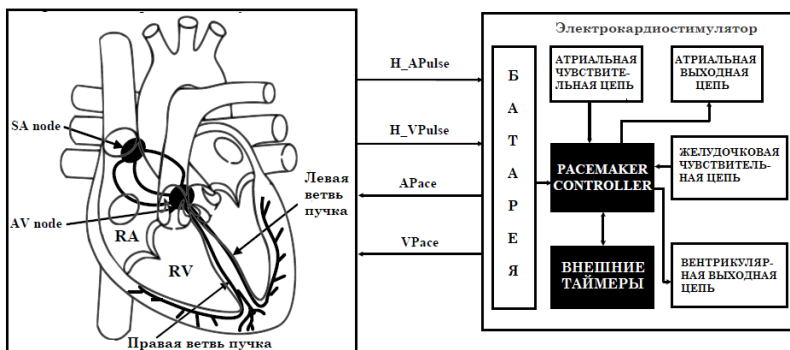


Рисунок 16.1. Замкнутая система сердца и кардиостимулятор

H_APulse и APace – предсердные события, а H_VPulse и VPace – желудочковые события.

В этой главе мы сконцентрируемся на разработке контроллера кардиостимулятора на основе модельно-ориентированного подхода, а также представим модели сердца и таймеры. Этот раздел начинается со знакомства с моделями электрической системы в сердце, после чего следует подробное представление кардиостимулятора в режиме DDD. В главе также обобщены тематические работы.

16.2.1. Электрофизиология сердца и его моделирование

На рисунке 16.2 показаны структура электрической системы в сердце и сигнал электрокардиограммы (ЭКГ), создаваемый электрической проводимостью в этой системе. В электрической системе все импульсы начинаются с узла SinoAtrial (SA) в правом предсердии и спонтанно и периодически генерируются электрическими сигналами. Такие сигналы распространяются вдоль пути проводимости до тех пор, пока они не достигают узла AtrioVentricular (AV). Этот процесс приводит к сокращению правого предсердия, которое выталкивает кровь в правый желудочек. Левый атриум выполняет идентичный процесс почти одновременно. В ЭКГ зубец Р означает такой процесс проводимости. Роль AV узла заключается в синхронизации между предсердием и желудочком, где распространение электрических сигналов будет отложено. Физиологически в течение отложенного времени кровь, вытолкнутая из предсердия, целиком попадает в правый желудочек. После задержки сигналы проходят через волоконную систему Пуркинье, которая вызывает сокращение желудочков. В ЭКГ это представлено комплексом QRS. Следующая волна Т подразумевает расслабление желудочковой ткани, эта волна известна как рефрактерный период. Регулярно бьющееся сердце обычно поддерживает частоту от 60 до 100 ударов в минуту. На ЭКГ будет периодически наблюдаться фиксированная последовательность зубца Р, комплекса QRS и зубца Т.

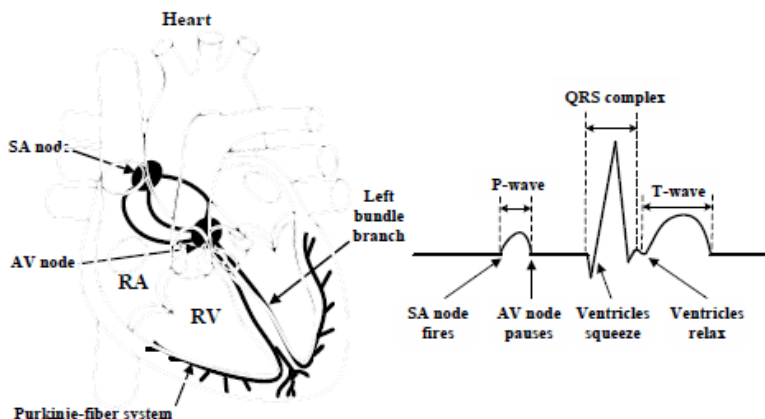


Рисунок 16.2. Электрическая система сердца и ЭКГ

Цзян и соавторы и Чен и соавторы [7, 12] представляют модели электрической системы, где [12] основана на временных автоматах, а гибридные автоматы использованы [7]. Модели сердца с синхронизированными автоматами фиксируют временные характеристики электрической системы и замыкают цикл с моделями кардиостимулятора. Затем проверяются временные свойства на основе моделей такой замкнутой системы. Как предполагается в [12], для модели сердца могут быть достигнуты различные уровни абстракции, основанные на требованиях реальности. Это, однако, может потребовать много времени для проверки, если будут созданы более реалистичные модели сердца. Работа, представленная в [7], также включает исследования электрического поведения сигнала ЭКГ. Используя гибридные автоматы I/O, модель сердца может определить, способно ли накопленное электрическое напряжение вызвать действие.

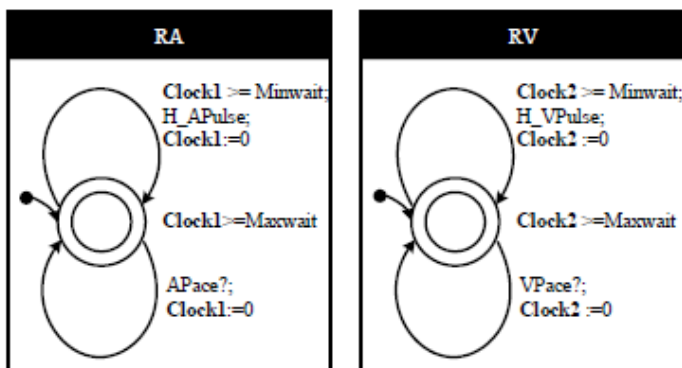


Рисунок 16.3. Пример модели сердца с использованием синхронизированных автоматов [13]

На рисунке 16.3 представлена простая модель сердца, состоящая из модели правого предсердия (RA) и модели правого желудочка (RV) с использованием синхронизированных автоматов [3]. Синхронизированные автоматы облегчают набор вещественных часов (например, Clock1 и Clock2). Все значения часов увеличиваются с одинаковой скоростью и могут быть рассмотрены, эти модели случайным образом генерируют H_APulse и H_VPulse для кардиостимулятора в пределах нижней границы Minwait и верхней границы Maxwait соответственно. Когда значение часов больше или равно нижней границе времени, включается переход, в течение которого импульс (то есть H_APulse или H_VPulse) генерируется для кардиостимулятора, и значение часов сбрасывается. Более того, модель также способна синхронизироваться с кардиостимулятором. APase или VPase, созданные кардиостимулятором, могут инициировать переход и тем временем сбрасывать значение часов. Поскольку в моделях RA и RV существует только одно состояние, все переходы приводят к этому состоянию. Далее значение часов начинает мгновенно увеличиваться.

16.2.2. Временные ограничения и диаграмма кардиостимулятора

Современные производители кардиостимуляторов внедрили в эти устройства различные режимы работы. В этой главе мы выбрали один из часто используемых режимов, называемый DDD [5]. Такой режим способен понимать и стимулировать как предсердие, так и желудочек. Кроме того, для предотвращения слишком быстрой стимуляции желудочка, режим DDD облегчает верхнюю границу и нижнюю границу скорости. Как только частота желудочков достигает верхней границы, стимулы прекращаются; и наоборот, стимулы создаются, если скорость не достигает нижней границы. Кроме того, кардиостимулятор представляет собой жесткое в реальном времени и критически важное для безопасности устройство, в котором должны соблюдаться требования к времени. Такие требования к кардиостимулятору в режиме DDD подробно описаны ниже. Мы используем пример трассировки для объяснения различных временных ограничений, связанных с кардиостимулятором в режиме DDD.

- AEI: интервал предсердного побего. Он описывает максимальный интервал времени между желудочковым событием и соответствующим предсердным событием.

- AVI: предсердный желудочковый интервал. Он описывает максимальный интервал времени между предсердным событием и соответствующим желудочковым событием.

- VRP: желудочковый рефрактерный период. Он определяет период покоя для желудочка после каждого желудочкового события. Физиологически желудочковые события не обнаруживаются кардиостимулятором во время периода отдыха.

- PVARP: послежелудочковый рефрактерный период предсердия. Он определяет период отдыха для предсердия после каждого желудочкового события. Физиологически предсердные события не обнаруживаются кардиостимулятором во время периода отдыха.

- URI: верхний интервал скорости. Он указывает на минимальный интервал времени между двумя последовательными желудочковыми событиями. Такой интервал используется в двухкамерных режимах (таких как режим DDD) и служит верхней границей для частоты сердечных сокращений.

- LRI: нижний интервал скорости. Он указывает на максимальный интервал времени между двумя последовательными желудочковыми событиями. В двухкамерном режиме временной интервал LRI равен суммированию AVI и AEI. Он служит нижней границей для частоты сердечных сокращений, и, следовательно, значение LRI больше, чем URI.

Кроме того, мы используем три периода блокировки в режиме DDD на основе спецификации в [2]. Такие периоды предотвращают перекрестные помехи во время ощущения, когда происходит каждое желудочковое событие или APace. Они обычно короче рефрактерных периодов.

- PVPB: блокировка после желудочкового ритма. Это означает период блокировки после каждого сгенерированного VPace, в течение которого H_APulse будет считаться недействительным событием.

- PVSB: блокировка послежелудочкового ощущения. Это означает период блокировки после каждого желудочкового события, в течение которого H_APulse будет считаться недействительным событием.

- PAPB: блокировка постсердного ритма. Это указывает период блокировки после каждого APace, в течение которого H_VPulse будет считаться недействительным событием.

Пять кривых ЭКГ, представляющих электрическую проводимость сердца, показаны в качестве примеров на рисунке 16.4. Форма волны «1» указывает на нормальное поведение сердца. Мы могли бы заметить, что соответствующие таймеры AVI останавливаются до истечения срока, что показывает нормальную синхронизацию между предсердным событием и последующим желудочковым событием. Аналогично, поведение AEI показывает нормальную синхронность между желудочковым событием и наступающим предсердным событием.

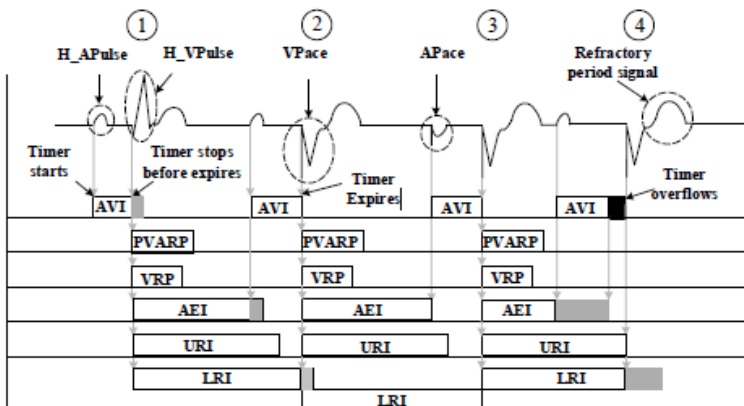


Рисунок 16.4. Временная диаграмма DDD в режиме кардиостимулятора. Адаптировано из [5]

Форма волны «2» представляет ситуацию, в которой искусственный импульс (VPace) будет генерироваться кардиостимулятором, если внутренний желудочковый импульс не обнаружен в пределах интервала AVI. Кроме того, требование LRI не нарушается.

Форма волны «З» имеет ситуацию, когда AEI и AVI истекли. Следовательно, для достижения требований LRI создаются как APace, так и VPace.

Форма волны «4» представляет предсердное событие, которое происходит раньше, чем обычно. Это может вызвать временную быструю желудочковую частоту, если нет верхнего ограничения. Существование URI задерживает создание VPace, расширяя интервал AVI до длины интервала URI.

16.2.3. Родственная работа

Электрические модели сердца бывают двух типов. Существует много вычислительных моделей сердца [8, 10, 18, 22], функционирующих как «виртуальные органы» [19], которые воспроизводят поведение сердца в масштабах от клетки до органа. Эти высокоточные модели используются для компьютерного моделирования, но не подходят для математической проверки (проверки моделей) [4] или моделирования кардиостимуляторов в реальном времени. Напротив, абстрактные модели электрического поведения сердца были разработаны с использованием временных автоматов [3] и гибридных автоматов [9]. Эти модели имеют меньшую вычислительную сложность и, следовательно, идеально подходят для замкнутого цикла, моделирования в реальном времени и анализа для правильности проектирования кардиостимулятора. Гибридные модели на основе автоматов в основном использовались для проверки на основе тестирования [7]. Нас интересуют

модели автоматов на основе времени, поскольку они могут использоваться для формального анализа времени отклика.

В [11, 16] были разработаны два подхода к верификации времени реакции кардиостимуляторов с использованием абстрактной модели сердца с синхронизированными автоматами. Этот подход делает идеализированное предположение относительно временных автоматных переходов, которые считаются мгновенными (то есть занимают нулевое время). Чтобы преодолеть это ограничение, мы используем модельно-ориентированный подход. Мы объединяем информацию из низкоуровневого анализа BCET и WCET [20] для точного вычисления границ задержки при переходах. Результаты затем используются для создания временных моделей автоматов для анализа времени отклика.

16.3. Обзор предлагаемого подхода

Как показано на рисунке 16.5, предлагаемый подход начинается с создания модели с использованием функциональных блоков IEC 61499 (шаг 1). Реализация моделей функциональных блоков основана на синхронной семантике [21]. Для простоты мы называем это синхронными функциональными блоками. Используя FBC [21], мы компилируем каждый функциональный блок в свою собственную функцию C. Шаг 2 статически анализирует выполнение сгенерированного кода и предоставляет BCET и WCET для следующего шага. На шаге 3, чтобы преодолеть нереалистичное предположение, что переходы не требуют времени, для определенных переходов вводятся задержки. Мы называем такие модели с задержкой переходов моделями высокой точности. Подход заканчивается этапом 4, на котором проводится анализ времени отклика.

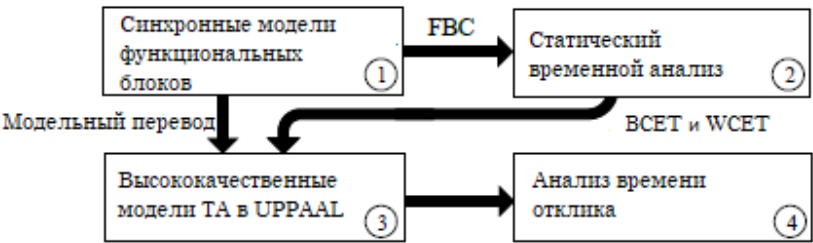


Рисунок 16.5. Обзор предлагаемого подхода

Синхронные функциональные блоки выполняются в пошаговом режиме с использованием глобальных логических тактов. Во время такта все входные события дискретизируются один раз, после чего следует реакция каждого ЕСС в сети функциональных блоков на основе входных событий. Такт заканчивается выпуском всех выходных событий после выполнения ЕСС.

Реакции ECC могут быть следующими: (1) ECC остается в том же состоянии, если не происходит никакого перехода; (2) ECC переходит к следующему состоянию, если входное событие инициирует переход. Во время реакции выполняется алгоритм, связанный с состоянием, и выходное событие генерируется в конце такта. Все ECC в сети выполняются логически параллельно, так как все зависимости производителя и потребителя зависят от задержки устройства.

PTARM [14] выбран в качестве базовой платформы контроллера кардиостимулятора. Архитектура высокоточной синхронизации ARM (PTARM) обеспечивает четыре конвейера с чередованием нитей и локальную память с нуля. На рисунке 16.6 показана временная прогрессия конвейера с чередованием нитей от t0 до t11. Следующая ситуация показывает, что в PTARM отсутствуют зависимости от управления и данных. Временной шаг t4 показывает, что все другие этапы заполнены инструкциями из других потоков, когда инструкция из аппаратного потока 3 находится на этапе выполнения (E). Следовательно, нет никакого контроля и никакой зависимости, связанной с данными, между инструкциями, что устраняет соответствующие риски и упрощает анализ времени.

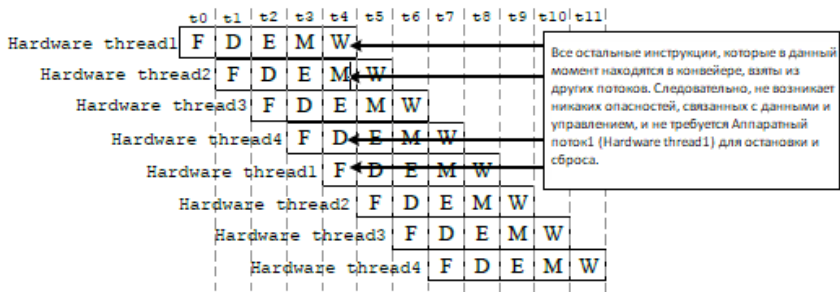


Рисунок 16.6. Распределение задач на PTARM

Синхронизированные автоматы создают высокоточные модели контроллера кардиостимулятора. UPPAAL [6] является инструментом, который поддерживает структуру ТА, которая также облегчает выполнение полезных функций. Мы используем фиксированные местоположения для моделирования состояний, в которых время не может истечь, а выходное событие генерируется мгновенно. Также широкоэвентральные каналы используются для синхронизации между синхронизированными автоматами в сети ТА.

16.4. Моделирование с использованием функциональных блоков стандарта IEC 61499

На основе требований синхронизации (см. раздел 16.2.2) и схемы кардиостимулятора режима DDD, мы создали модели IEC 61499 функционального блока для контроллера кардиостимулятора. Начнем с подробного представления основных функциональных блоков (BFB) для каждого требования синхронизации, после чего перейдем к описанию сети основных функциональных блоков. Такая сеть инкапсулируется в блоке сложной функции (*DDD_Mode_PacingLogic*). Иерархическое расположение моделей представлено в конце. Расположение включает сеть, которая интегрирует желудочковый детектор импульсов, предсердный детектор импульсов, функцию журнала событий и логику следующего режима DDD. С инкапсуляцией сети блок сложной функции создается как модель контроллера кардиостимулятора и готов быть представленным среде, состоящей из основы и внешних таймеров (см. рисунок 16.1).

16.4.1 Логическая модель режима DDD

Семь базовых функциональных блоков созданы для моделирования вышеупомянутых требований к синхронизации, каждый из которых имеет логический суффикс. На рисунке 16.7 мы проиллюстрируем, как разные базовые функциональные блоки используются для захвата требований синхронизации. Например, синхронизация AVI моделируется базовым функциональным блоком *AVI_Logic*. Такие основные функциональные блоки могут взаимодействовать с внешними таймерами, посылая сигналы остановки или запуска или считывая сигналы истечения срока действия. Чтобы поддерживать требования к времени, создаются семь таймеров, и им присваиваются значения временных интервалов для каждого требования к времени.

URI_Logic показан на рисунке 16.8. Этот функциональный блок связывается с остальными функциональными блоками в сети. Конечный автомат начинается с начального состояния, называемого *S0*, и переходит в состояние *S1*, когда происходит событие *VSense* или *VPace*. Во время этого состояния алгоритм (а именно *aSetValue*) устанавливает значение обратного отсчета для внешнего таймера, соответствующего интервалу *URI*, за которым следует выходное событие (*URI_TimerStart*), которое запускает таймер. Затем конечный автомат переходит в состояние *S2* из-за истинного утверждения. Находясь в этом состоянии, конечный автомат может перейти в состояние *S3* или вернуться в исходное состояние на основе сигналов *VSense* или *URI_TimerExpired*. При получении *VSense* таймер *URI* принудительно останавливается с использованием вывода *URI_TimerStop*.

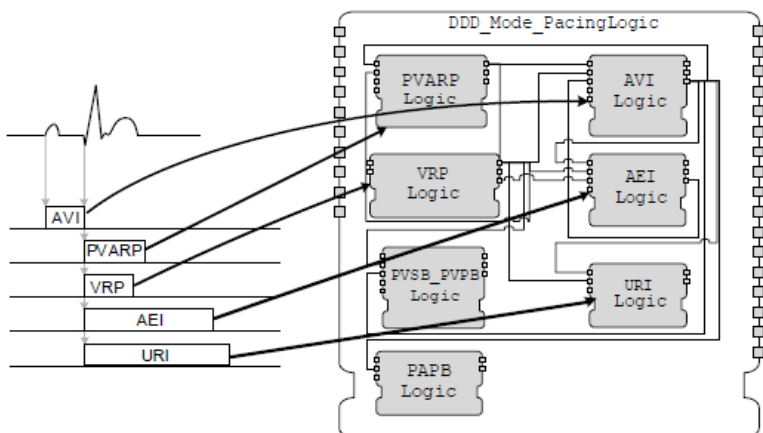


Рисунок 16.7. Обзор диаграммы времени для моделей функциональных блоков

AEI_Logic и AVI_Logic показаны на рисунке 16.9. В качестве важного требования к времени для синхронизации между предсердными импульсами и желудочковыми импульсами AEI_Logic ECC получает желудочковые события для перехода из исходного состояния и генерирует APace_Out, когда время для синхронизации истекло. Это достигается чтением сигнала AEI_TimerExpired. Также, как показано на диаграмме времени, таймер AEI может быть остановлен сигналами ASense. Это интерпретируется в состоянии S1, где создается выход AEI_TimerStop.

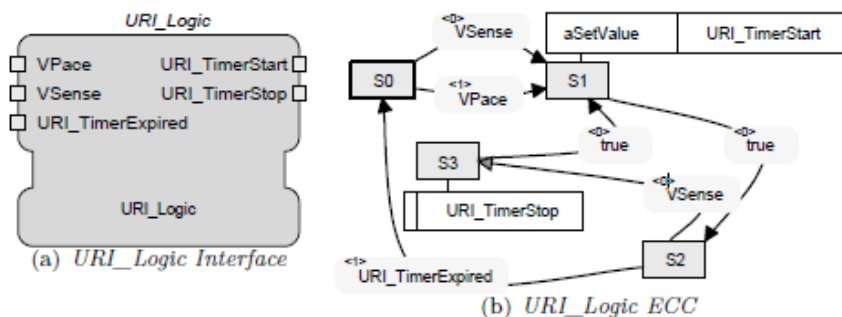


Рисунок 16.8. Основные функциональные блоки для URI_Logic

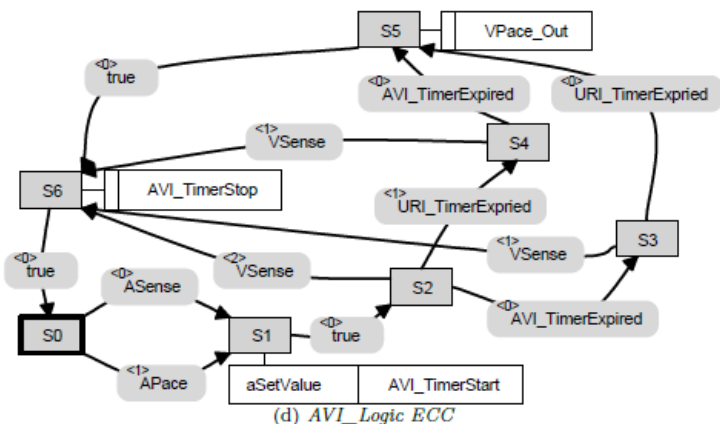
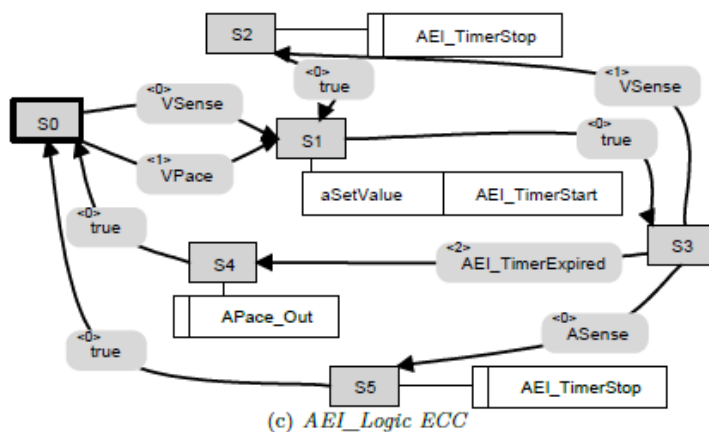
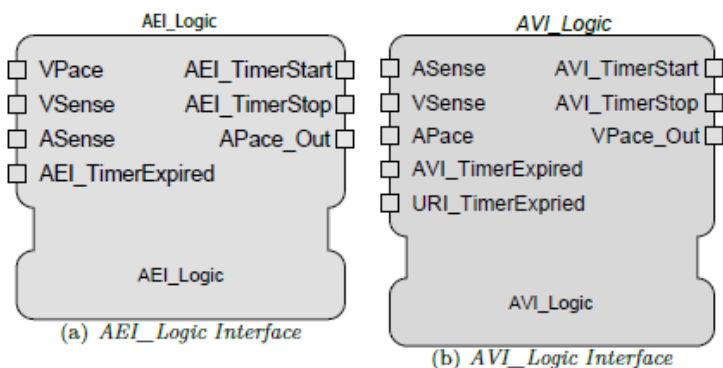


Рисунок 16.9. Основные функциональные блоки для *AEI_Logic* и *AVI_Logic*

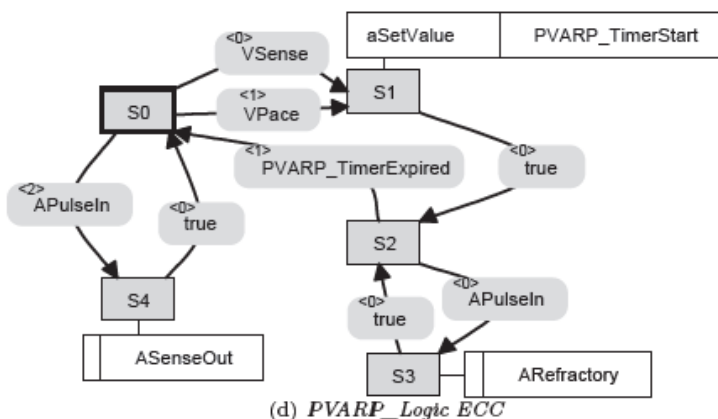
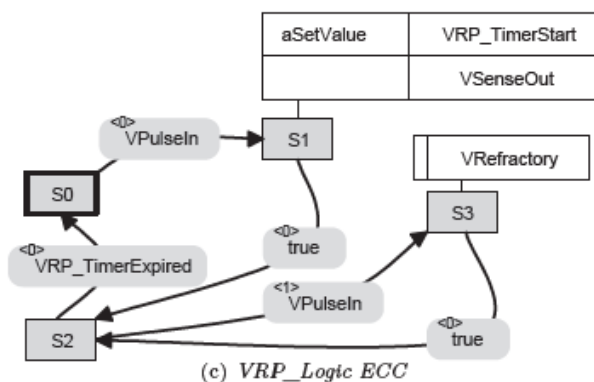
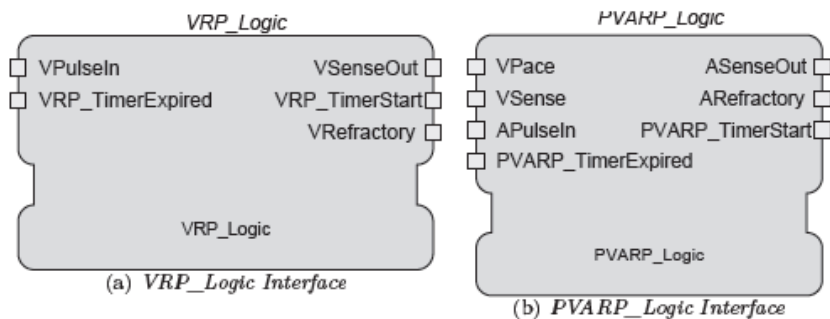


Рисунок 16.10. Основные функциональные блоки для *VRP_Logic* и *PVARP_Logic*

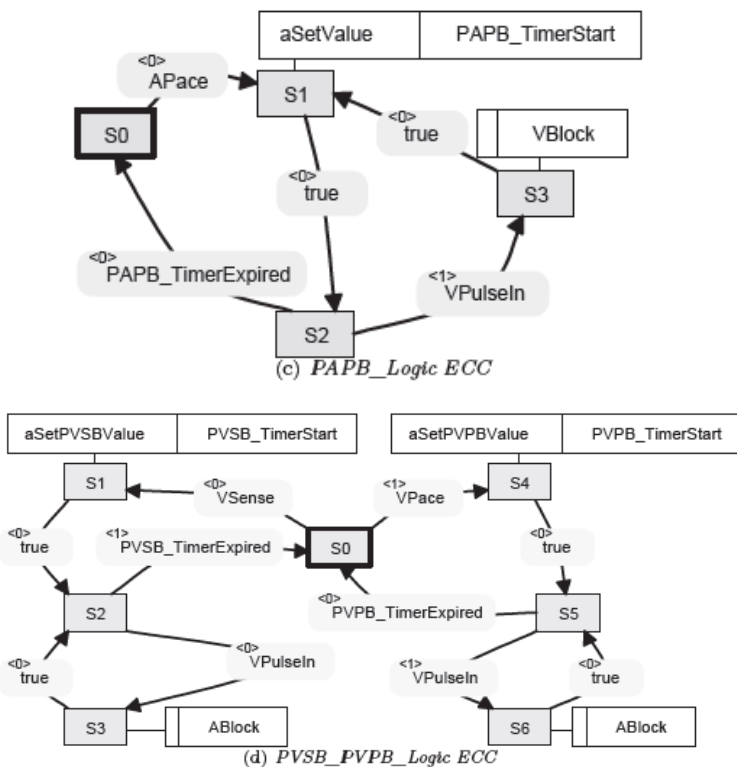
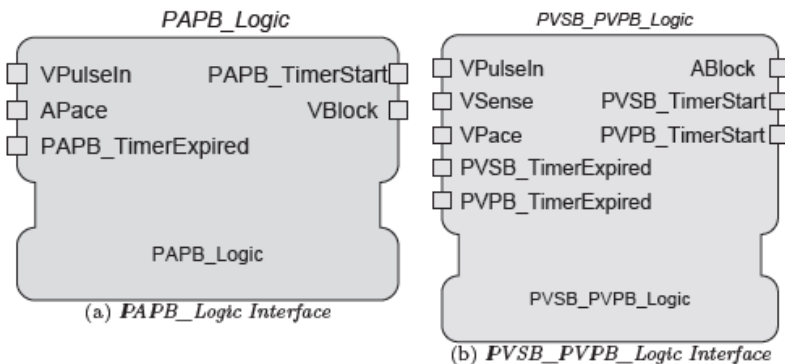


Рисунок 16.11. Основные функциональные блоки для *PAPB_Logic* и *PVSB_PVPB_Logic*

AVI_Logic ECC начинается с ожидания предсердных триггеров и в ответ генерирует VPace_Out. Кроме того, AVI не может создавать VPace, пока не

истечет интервал URI. Следовательно, состояния S3 и S4 создаются, чтобы гарантировать, что оба сигнала истекли до генерации VPace.

VRP_Logíc и PVARP_Logíc показаны на рисунке 16.10. Функциональный блок VRP_Logíc отвечает за определение того, являются ли желудочковые импульсы действительными в качестве входных событий для URI_Logíc и AEI_Logíc. VRP ECC запускает внешний таймер с выхода VRP_TimerStart, когда наблюдается начальный сигнал VPulseIn. Все сигналы VPulseIn, возникающие при подсчете таймера, считаются недействительными. Они называются VRrefactory сигналами. Выход VSenseOut может быть сгенерирован только после достижения состояния S1. PVARP_Logíc ECC следует аналогичному процессу, за исключением того, что начальными триггерами являются VSense и VPace.

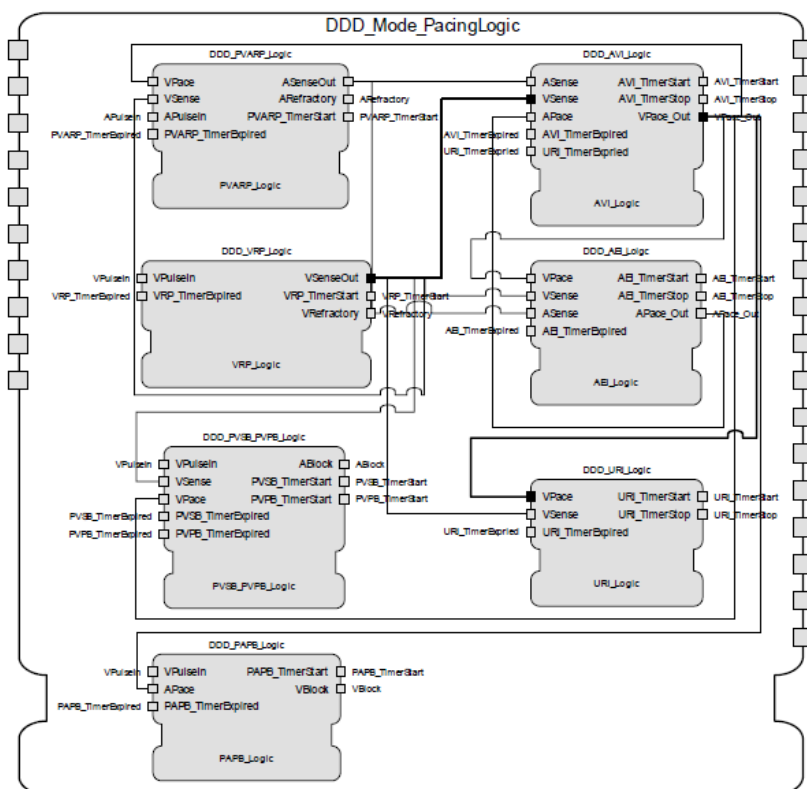


Рисунок 16.12. Сеть основных функциональных блоков требований синхронизации

PAPB_Logíc и PVSB_Logíc показаны на рисунке 16.11. Мы объединяем требования синхронизации PVSB и PVPB в один функциональный блок. ECC этого функционального блока начинается с состояния S0 и переходит либо в

состояние S1, либо в состояние S4 на основе сигналов VSense или VPace. Как только такое состояние достигнуто, соответствующий внешний таймер запускается после назначения значения обратного отсчета. Кроме того, ECC может записывать заблокированный сигнал (например, ABlock), когда VPulseIn происходит в течение периодов обратного отсчета. Аналогично PAPB_Logic ECC обрабатывает сигналы VPulseIn, которые происходят в течение периодов обратного отсчета, как сигналы VBlock. Разница между этими двумя функциональными блоками заключается в начальном триггере.

Сеть основных функциональных блоков показана на рисунке 16.12. После создания базовых функциональных блоков они затем составляются для построения сети. Сеть показывает внутренние соединения этих функциональных блоков и внешние события, потребляемые или производимые извне или снаружи. Например, обнаруженный сигнал ASense (то есть ASenseOut) генерируется функциональным блоком PVARP_Logic и может наблюдаться только подключенными функциональными блоками, такими как AEI_Logic. Кроме того, выходной сигнал можно наблюдать внутри сети или выводить за пределы сети. Например, VPace (то есть VPace_Out) является входным событием для URI_Logic ECC. Он также является внешним выходом для желудочковой схемы кардиостимуляции (см. рисунок 16.1). Сеть затем инкапсулируется в составной функциональный блок, а именно DDD_Mode_PacingLogic, для создания модели контроллера кардиостимулятора.

16.4.2. Иерархическая структура модели контроллера кардиостимулятора

Мы завершим разработку контроллера кардиостимулятора, построив сеть функциональных блоков (представлена на рисунке 16.13). Во-первых, VPulseDetector BFB отвечает за обработку желудочковых импульсов, измеряемых желудочковой чувствительной цепью (показано на рисунке 16.1). Если обнаружено действительное желудочковое событие, этот функциональный блок создает выход для входа функционального блока DDD_Mode_PacingLogic, называемого VPulseIn. Аналогично Атриальпульсометр имеет дело с предсердными импульсами из предсердной чувствительной схемы и способен генерировать выходные события в интерфейс APulseIn DDD_Mode_PacingLogic. CFD DDD_Mode_PacingLogic содержит логику всех требований синхронизации. Наконец, основные функциональные блоки Event_Log передают все записанные сигналы (включая APace, VPace, ABlock и ARefractory) на физическое устройство. Это помогло бы лицам, осуществляющим уход, наблюдать за состоянием кардиостимулятора с замкнутым контуром. Как и сеть на рисунке 16.12, сеть контроллера кардиостимулятора содержит внешние входы и выходы. Эти входы и выходы обычно поступают от таких устройств, как схемы стимуляции и внешние таймеры.

Подробные ECC для BFB Вентрикулярпульсометр, Event_Log и Атриальпульсометр показаны на рисунке 16.14. В ECC Атриальпульсометр, когда наблюдается H_APulseIn, ECC переходит в состояние S1, в котором первоначально выполняется алгоритм обработки сигнала (а именно aPulseDetection). Выход (то есть PulseDetectionOut) указывает, что был обнаружен действительный предсердный пульс. Кроме того, такой процесс обнаружения может быть остановлен, как только DDD_Mode_PacingLogic запускает периоды блокировки (PVSБ и PVPB). Он ожидает истечения таких периодов для обработки новых поступающих импульсов. Аналогично entricularPulseDetector ECC отвечает за обнаружение желудочкового импульса, при котором PAPB может отключить процесс. Наконец, Event_Log ECC просто читает входные события и возвращается к исходному состоянию после генерации TransOut.

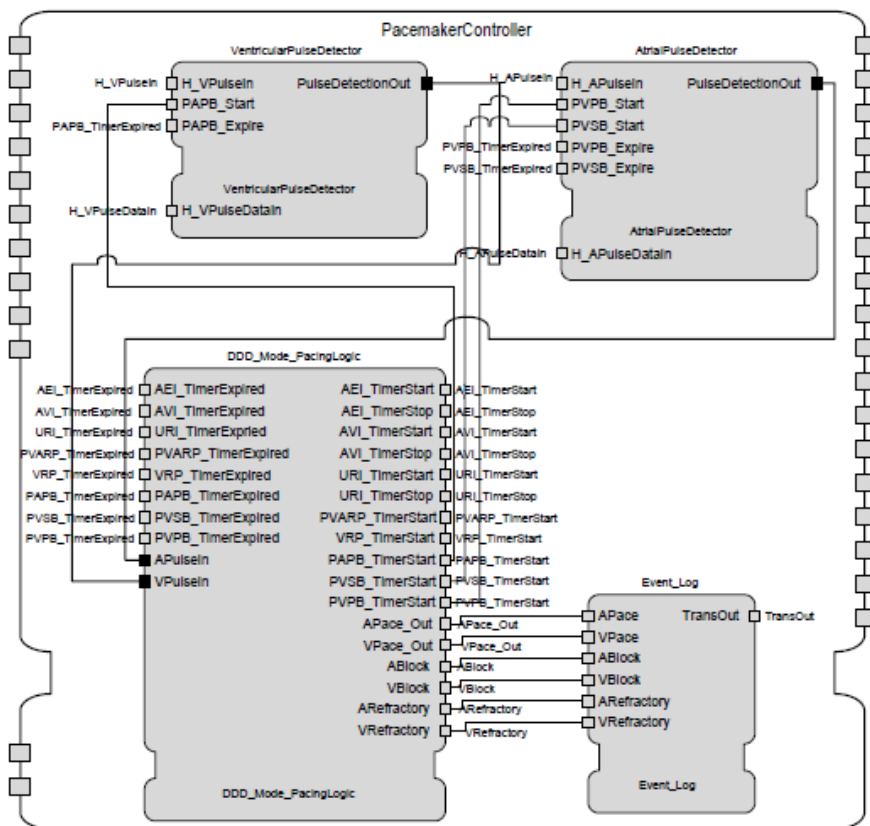


Рисунок 16.13. Функциональный блок сети контроллера кардиостимулятора

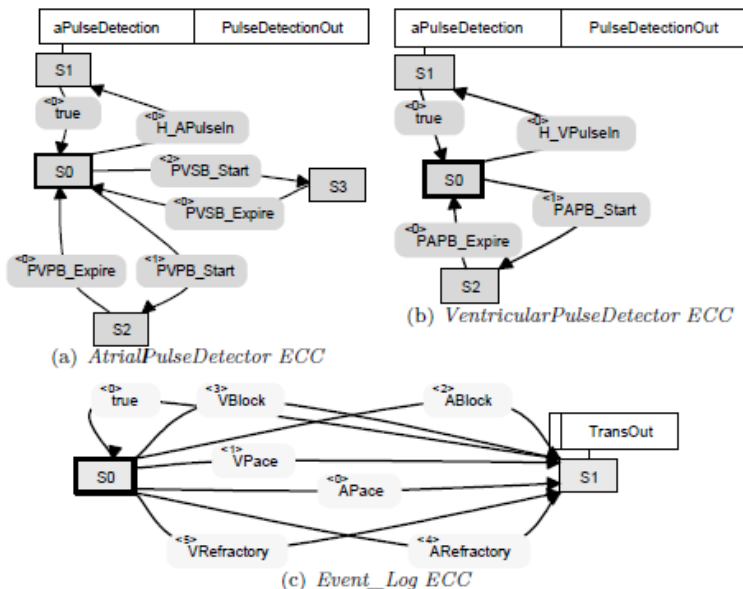


Рисунок 16.14. ECC базовых функциональных блоков Атриальпульсометр, Вентрикулярпульсометр и Event_Log

16.5. Создание высокоточной модели

UPPAAL предоставляет платформу для проведения анализа времени отклика путем создания временных моделей автоматов. Чтобы воспользоваться преимуществами структуры ТА, мы разрабатываем правила перевода для преобразования функциональных блоков IEC 61499 в модели UPPAAL. Чтобы отвергнуть нереалистичное предположение о мгновенных переходах (см. раздел 1.1.3) в моделях UPPAAL, мы используем результаты анализа синхронизации низкоуровневого кода, это позволит обеспечить реалистичные значения синхронизации переходов. Следовательно, модели ТА с высокой точностью создаются путем вставки таких границ в переходы. На основе таких высокоточных моделей проводится анализ времени отклика для обеспечения достоверных результатов. Этот раздел начинается с введения в статический анализ синхронизации сгенерированного кода из функциональных блоков. Затем мы представляем модель сердца и модель внешнего таймера с использованием синхронизированных автоматов, которые являются заводскими моделями. Наконец, модель контроллера кардиостимулятора с использованием ТА представлена с правилами трансляции модели.

16.5.1. Временной анализ синхронных функциональных блоков

Мы вручную распределяем функциональные блоки по четырем аппаратным потокам PTARM. Код, сгенерированный из функционального блока Атриальпульсометр, выделяется аппаратному потоку 1. Аппаратный поток 2 содержит код, сгенерированный из Вентрикулярпульсометр. Аппаратный поток 3 содержит DDD_Mode_PacingLogic, а код Event_Log реализован в аппаратном потоке 4. Такты (см. раздел 16.3) могут быть проиллюстрированы с использованием рисунка 16.15 на основе выполнения PTARM.

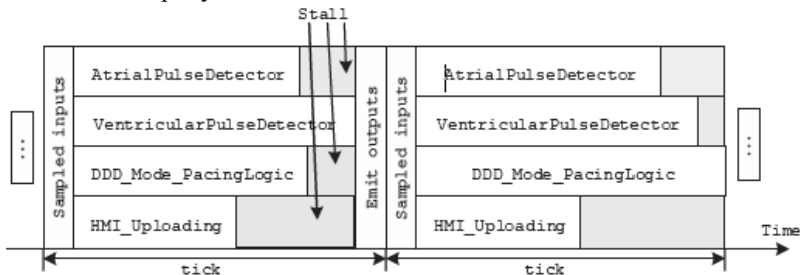


Рисунок 16.15. Запуск контроллера кардиостимулятора на базе PTARM

На рисунке 16.16 представлен обзор подхода анализа времени. Сначала контроллер кардиостимулятора (реализованный в функциональном блоке IEC 61499) компилируется в исполняемый код на языке C компилятором функционального блока (см. раздел 16.3). На рисунке 16.17 представлена структура сгенерированного кода. Для каждого аппаратного потока PTARM реализовано циклическое расписание для выполнения функциональных блоков, выделенных этому потоку. Например, Атриальпульсометр _run представляет функцию C для функционального блока Атриальпульсометр, и эта функция будет реализована в аппаратном потоке 1 PTARM. Кроме того, DDD_Mode_PacingLogic, состоящий из семи основных функциональных блоков, реализован в потоке 3.

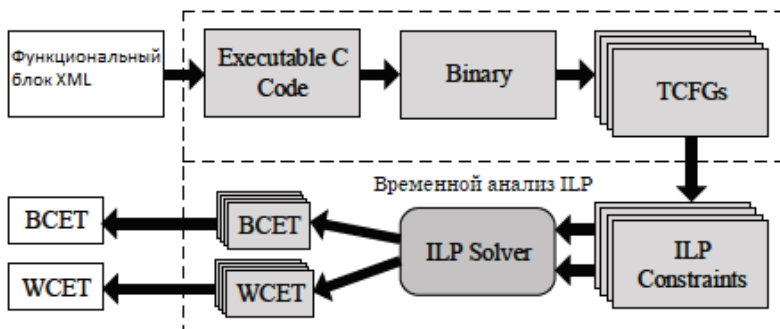


Рисунок 16.16. Обзор статического анализа синхронизации

<pre> for(;;) { [sampled inputs] AtrialPulseDetector_run(); VentricularPulseDetector_run(); DDD_Mode_PacingLogic_run(); Event_Log_run(); [emit outputs] } </pre>	<pre> for(;;) { [sampled inputs] AEI_Logic_run(); AVI_Logic_run(); PAPB_Logic_run(); PVARP_Logic_run(); PVSb_PVPB_Logic_run(); URI_Logic_run(); VRP_Logic_run(); [emit outputs] } </pre>

Рисунок 16.17. Код контроллера кардиостимулятора (слева) и логический код стимуляции в режиме DDD (справа)

Затем код на C компилируется в двоичный файл с использованием компилятора ARM-GCC. Далее для анализа синхронизации двоичный файл преобразуется в промежуточный формат, называемый временным графиком управления (TCFG). TCFG является графиком потока управления со временем выполнения, аннотированным на каждом узле, и генерируется для каждого аппаратного потока. Процесс генерирования TCFG из двоичного файла является адаптацией подхода, представленного в [17].

Чтобы провести анализ синхронизации на основе ILP, мы сначала определяем BCET и WCET системы, которые являются максимальными значениями BCET и WCET среди всех аппаратных потоков. BCET и WCET одного аппаратного потока – это соответственно самое короткое и самое длинное время, необходимое для выполнения итерации циклического расписания (то есть такт). Следовательно, целевой функцией ILP для определения BCET и WCET одного аппаратного потока является нахождение минимального и максимального времени выполнения соответственно. Затем мы создаем объективные формулы и ограничения и используем решатель ILP, такой как Groubi [1], для решения соответствующих целевых функций. Наконец, результаты предоставлены для создания моделей высокой точности.

16.5.2. Высокочастотные модели контроллера кардиостимулятора

Из-за синхронного выполнения мы получаем два типа задержек. Во-первых, алгоритм, связанный с состояниями, откладывает либо генерацию выходных событий, либо получение входных событий от входящего такта. Во-вторых, из-за единичной задержки [21] в сети функциональных блоков

необходимо добавить задержку к внутренним переходам в сети. На рисунке 16.18 представлены задержки в сети `DDD_Mode_PacingLogic`. Оба из вышеупомянутых типов переходов потребляют время такта. Как было показано ранее с использованием временного анализа, значения `BCET` и `WCET` могут быть рассчитаны для ограничения длины такта. Следовательно, задержки переходов также могут быть ограничены, чтобы происходить в закрытом интервале $[BCET, WCET]$. После рассмотрения задержек мы определим некоторые правила для перевода моделей функциональных блоков модели `TA` в `UPPAAL`. Правила перечислены ниже.

Правило 1. *Каждый ECC переводится в синхронизированный автомат для формирования сети.* Поскольку инкапсуляция в сети `TA` не допускается, инкапсуляции в сети функциональных блоков больше не существуют в моделях `UPPAAL`. Например, `DDD_Mode_PacingLogic` инкапсулирует семь основных функциональных блоков и не моделируется в сети `TA`. В частности, сеть `TA` контроллера кардиостимулятора состоит из десяти синхронизированных автоматов благодаря десяти `ECC` в моделях функциональных блоков.

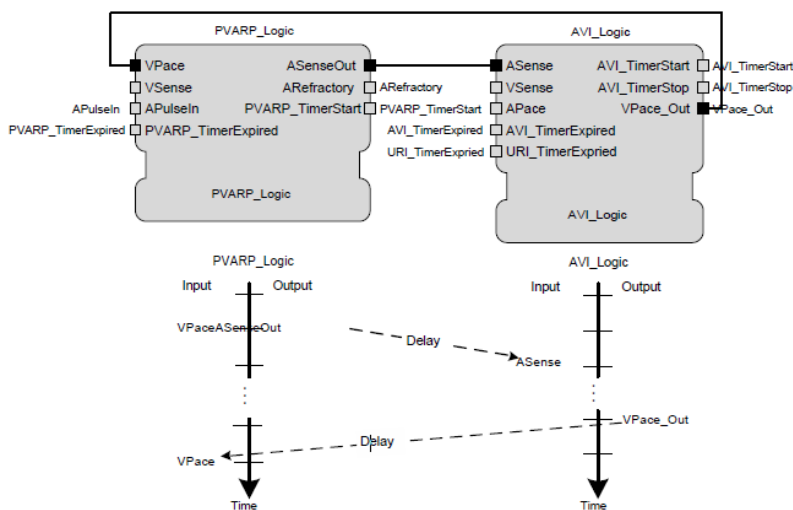


Рисунок 16.18. Блок задержки в режиме `DDD` шаговой логической сети

Правило 2. *Каждое состояние в `ECC` переводится в местоположение `TA` в `UPPAAL`.* В частности, каждое начальное состояние сопоставляется с начальным местоположением, а другие состояния сопоставляются с обычными местоположениями. Это показано с использованием № 2 на рисунке 16.19.

Правило 3. *Каждое входное событие и выходное событие транслируется в канал вещания.* В частности, каждый вход отображается на `channel_name?`, что указывает на потребителя. Каждый выход сопоставляется с `channel_name!`, указывая производителя. Можно моделировать синхронизацию между одним

производителем и несколькими потребителями, используя широковещательные каналы. Это показано с использованием № 3 на рисунке 16.19.

Правило 4. Каждое выходное событие генерируется из зафиксированного местоположения. Для простоты добавления задержек зафиксированное местоположение реализуется до генерации связанного выходного события. Преданные местоположения не дают времени истечь. Это показано с использованием № 4 на рисунке 16.19.

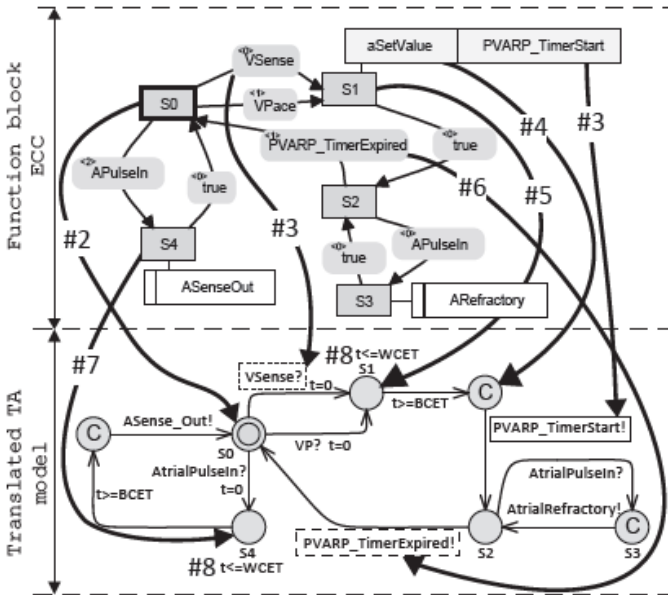


Рисунок 16.19. Пример перевода модели *PVARP_Logics ECC*

Правило 5. Каждое состояние, связанное с алгоритмом, должно вводить ограниченное время задержки в соответствующее местоположение в *TA*. В частности, для представления границы [BCET, WCET] в моделях UPPAAL мы создаем часы для такого состояния, используя инвариант, который присваивается значению WCET, и переход, который запускается, когда значение часов больше, чем BCET. Мы проиллюстрировали это правило, используя № 5 на рисунке 16.19.

Правило 6. Задержки не вводятся для каждого входного события, которое является интерфейсом со средой. Из-за гипотезы синхронизации контроллер кардиостимулятора может захватывать события из окружающей среды (то есть сердечный и внешний таймеры) всякий раз, когда среда производит такие события. Мы проиллюстрировали это правило, используя № 6 на рисунке 16.19.

Правило 7. Каждое состояние, которое генерирует выходное событие для внутренней связи, должно вводить ограниченное время задержки. Это

определяет место для добавления единичной задержки. Как и в пятом правиле, для времени задержки создаются часы. Это показано с использованием № 7 на рисунке 16.19.

Правило 8. *Каждый синхронизируемый автомат реализует максимум один такт для ограничения времени задержки.* Это упрощает процесс сброса часов, а также может сократить время проверки. В частности, все часы должны быть сброшены перед входом или выходом из состояния с ограниченной задержкой. Это правило иллюстрируется с использованием № 8 в модели ТА на рисунке 16.19.

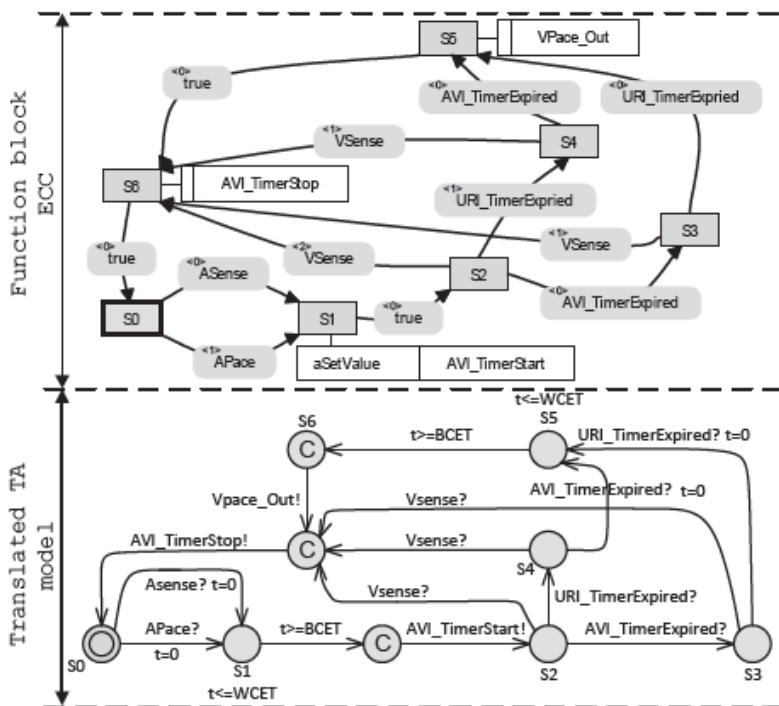


Рисунок 16.20. Переведенная модель AVI_Logics ECC

Мы применяем вышеупомянутые правила к моделям контроллера кардиостимулятора и иллюстрируем переведенные модели ТА PVARP_Logics и AVI_Logics, используя рисунок 16.19. Как заявлено в правиле 8, определен только один локальный тактовый сигнал t . Мы также представили переведенный AVI_Logics ECC на рисунке 16.20. Задержка, добавленная в местоположение S1, основана на алгоритме, связанном с этим состоянием в ECC. Значение задержки не удваивается, потому что единичная задержка вставляется временным автоматом производителя PVARP_Logics. Это наблюдается на рисунке 16.19, где применяется правило 6.

16.5.3. Создание модели среды для системы замкнутого цикла

Для замкнутой системы необходимо создать модели среды. Устанавливая инвариант и защиту часов для часов в UPPAAL, модель внешнего таймера может генерировать сигнал истечения (используя широковещательные каналы), если защита часов обнаруживает, что условие перехода достигнуто. Кроме того, она может быть запущена или остановлена с помощью сигналов от контроллера кардиостимулятора. Пример внешнего таймера представлен на рисунке 16.21 (а). Что касается модели сердца, часы также могут быть использованы. Сердце может генерировать сигналы случайным образом, используя широковещательные каналы в течение определенного промежутка времени, что может быть реализовано с помощью умной настройки инвариантов и защиты часов.

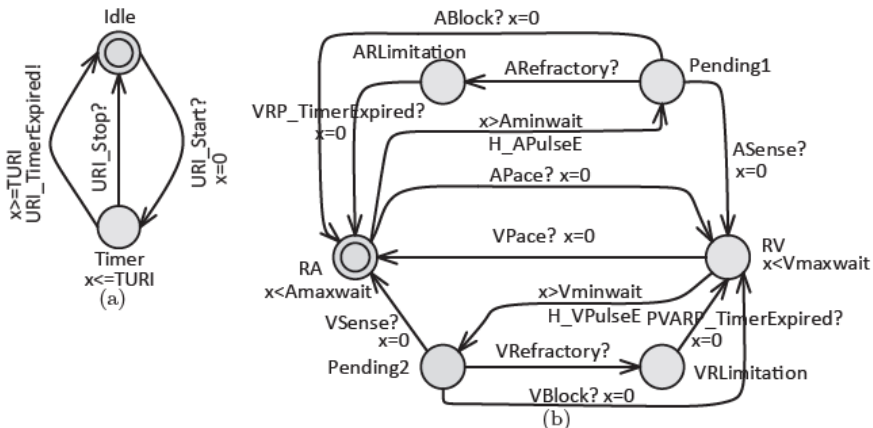


Рисунок 16.21. Модели ТА для сердца (слева) и внешнего таймера (справа)

Чтобы отвергнуть нереалистичные следы модели на рисунке 16.3, оставляя замкнутую систему без блокировки, мы строим связи между **RA** и **RV** и представляем модель сердца на рисунке 16.21 (b). В частности, $APace?$ вызывает переход из **RA** в **RV**, в то время как $VP?$ оказывает переход в противоположном направлении. Поскольку в этом автомате объявлены локальные часы t , каждый переход сбрасывает часы для следующего генерирования случайного события. Преимущество таких синхронизаций заключается в том, что пространство состояний по отношению к модели сердца сокращается во время проверки, что может обеспечить более масштабируемое моделирование кардиостимулятора. Те же самые переходы применяются к спонтанным импульсам (то есть H_VPulse и H_APulse). Однако такие спонтанные импульсы должны блокироваться во время периодов блокировки. Следовательно, переход к местоположению **RV** не может происходить напрямую, если достаточно защиты (то есть x больше, чем $Aminwait$). Вместо этого вводится местоположение (то есть **Pending2**) для

обеспечения синхронизации с PVARP, поскольку PVARP решает, является ли спонтанный импульс действительным. Если ABlock присутствует, автомат вернется в исходное состояние (то есть RA). Кроме того, импульсы, возникающие во время рефрактерного периода, также должны регистрироваться в модели сердца. Чтобы имитировать такое поведение, мы сначала предполагаем, что в течение рефрактерного периода разрешен только один импульс. Это показано, поскольку только один VR запускает переход к местоположению VRLimitation. Несколько импульсов могут быть установлены в соответствии с потребностями.

16.6. Анализ времени отклика с использованием высокочастотных моделей

Мы используем верификатор UPPAAL для проведения анализа времени ответа. Перед анализом нам нужно указать значения для всех требований к срокам. Нам также необходимо построить мониторинговые модели в ТА для соответствующих свойств TCTL. Такие значения могут быть указаны в декларациях UPPAAL. На рисунке 16.22 показаны такие декларации. В частности, значения в деталях требований к синхронизации служат жесткими крайними сроками для каждого требования к синхронизации, используемого в свойствах.

```
// Place global declarations here.
/* BCET & WCET calculation */
const int clock_period = 200; // time unit: ns.
const int scalar = 1000; // 1 us = 1000 ns.
const int BCET_Cycle = 244; // unit: clock cycles.
const int WCET_Cycle = 6373; // unit: clock cycles.
const int BCET = BCET_Cycle * clock_period / scalar; //unit: us.
const int WCET = WCET_Cycle * clock_period / scalar; //unit: us.
/* details of the timing requirements */
const int TAEI = 800000; // 800 ms = 800000 us.
const int TVRP = 150000; // 150 ms = 150000 us.
const int TAVI = 150000; // 150 ms = 150000 us.
const int TPVARP = 200000; // 200 ms = 200000 us.
const int TPVPB = 125000; //For atrial blocking,125ms=125000us.
const int TPVSB = 45000; //For atrial blocking,45ms=45000us.
const int TPAPB = 65000; //For ventricular blocking,65ms=65000us.
const int TURI = 900000; // URI should be greater than AEI and
// less than LRI. 900 ms = 900000 us.
const int TLRI = TAEI + TAVI;
/* parameters for heart pulses */
const int Vmaxwait = 1200000; // 1.2s = 1200 ms = 1200000 us.
const int Vminwait = 1000; // 1 ms = 1000 us.
```

```
const int Amaxwait = 1200000; // 1.2s = 1200 ms = 1200000 us.
const int Aminwait = 1000; // 1 ms = 1000 us.
```

Рисунок 16.22. Объявления в UPPAAL

Для проверки свойств времени отклика необходимо построить мониторинговые модели. Основываясь на методе, представленном в [13], мы создаем такие модели и демонстрируем два примера, используя рисунок 16.23. На рисунке 16.23 (а) монитор проверяет, истек ли интервал PVARP. Создавая часы в автомате, он измеряет время, начиная с желудочкового события до момента получения сигнала об истечении срока действия. Такое измеренное время записывается в местоположение Ex_Get. Данное значение используется для сравнения с объявленным значением. Сравнение проводится мгновенно, и, следовательно, фиксируется местоположение. Аналогично монитор LRI (показанный на рисунке 16.23 (b)) использует зафиксированное местоположение, обнаруженное для записи интервала между двумя последовательными желудочковыми событиями. Он отслеживает такой интервал, так как местоположение Wait2 всегда достигается после обнаруженного местоположения.

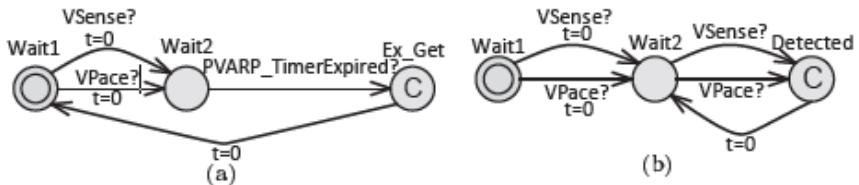


Рисунок 16.23. Мониторы ТА для PVARP (слева) и LRI (справа)

Как следствие, связанные свойства могут быть созданы как:

- $A[](\text{Monitor\_PVARP.Ex\_Get} \text{ imply } \text{Monitor\_PVARP.x} \geq \text{TPVARP}).$
- $A[](\text{Monitor\_LRI.Detected} \text{ imply } \text{Monitor\_LRI.x} \leq \text{TLRI}).$

Для проверки всех случаев, когда такие свойства должны быть выполнены, используется $A[]$. Слово «imply», означает, что если предыдущее условие удовлетворяется, следующее утверждение также должно быть выполнено. Мы распространяем такой метод на все связанные требования к времени и создаем дополнительные свойства, как показано ниже:

- $A[] \text{ not deadlock}.$
- $A[](\text{Monitor\_URI.interval} \text{ imply } \text{Monitor\_URI.x} \geq \text{TURI}).$
- $A[](\text{Monitor\_VRP.Ex\_Get} \text{ imply } \text{Monitor\_VRP.x} \geq \text{TVRP}).$
- $A[](\text{Monitor\_PVPB.Ex\_Get} \text{ imply } \text{Monitor\_PVPB.x} \geq \text{TPVPB}).$
- $A[](\text{Monitor\_PVSb.Ex\_Get} \text{ imply } \text{Monitor\_PVSb.x} \geq \text{TPVSb}).$
- $A[](\text{Monitor\_PAPB.Ex\_Get} \text{ imply } \text{Monitor\_PAPB.x} \geq \text{TPAPB}).$

Нет необходимости создавать свойства для AVI и AEI, поскольку LRI отслеживает более значительный интервал, который включает в себя интервалы как AVI, так и AEI. Результат анализа времени отклика показывает, что свойства, связанные с LRI, VRP, PVARP и PVPB, не выполняются. Поскольку в моделях высокой точности вводятся отложенные переходы, указанные декларации для требований к времени больше не могут служить для анализа. Мы устанавливаем новые крайние сроки для деклараций, добавляя допуски [2], показанные на рисунке 16.24. Соответственно, мы изменяем свойства, добавляя эти допуски. Ниже перечислим новые свойства:

- A[] not deadlock.
- A[](Monitor_LRI.Detected imply Monitor_LRI.x <= (TLRI+TLRI_tolerance)).
- A[](Monitor_VRP.Ex_Get imply Monitor_VRP.x >= (TVRP+TVRP_tolerance)).
- A[](Monitor_PVARP.Ex_Get imply Monitor_PVARP.x >= (TPVARP+TPVARP_tolerance)).
- A[](Monitor_PVPB.Ex_Get imply Monitor_PVPB.x >= (TPVPB+TPVPB_tolerance)).
- A[](Monitor_PAPB.Ex_Get imply Monitor_PAPB.x >= TPAPB).
- A[](Monitor_PVSB.Ex_Get imply Monitor_PVSB.x >= (TPVSB+TPVSB_tolerance)).
- A[](Monitor_PAPB.Ex_Get imply Monitor_PAPB.x >= (TPAPB+TPAPB_tolerance)).

Затем проводится анализ времени отклика с использованием измененных свойств. Результат показывает, что все свойства выполнены. После проведения анализа времени отклика на высокоточных моделях выясняется, что этот подход приносит разработчикам два преимущества. Во-первых, он помогает разработчику определить местонахождение проблем у модели по диагностическим следам. Во-вторых, он информирует разработчика о том, что сгенерированный код не может уложиться в жесткие сроки до его реализации. Разработчик должен либо перепроектировать модели, либо снизить тактовую частоту платформы, чтобы та прошла анализ.

```
/* tolerance values for the timing requirements */
const int TLRI_tolerance = 5000; // 5ms = 5000 us.
const int TVRP_tolerance = 5000; // 5ms = 5000 us.
const int TPVARP_tolerance = 5000; // 5ms = 5000 us.
const int TAVI_tolerance = 5000; // 5ms = 5000 us.
const int TPVSB_tolerance = 5000; // 5ms = 5000 us.
const int TPAPB_tolerance = 5000; // 5ms = 5000 us.
const int TPVPB_tolerance = 5000; // 5ms = 5000 us.
```

Рисунок 16.24. Допустимые значения в декларациях

16.7. Выводы

В этой главе мы представили структуру для анализа времени отклика кардиостимуляторов, которая сочетает статический анализ синхронизации на уровне кода с высокоуровневым моделированием. Мы выбрали функциональные блоки IEC 61499 для достижения высокого уровня моделирования и синхронной семантики для выполнения таких моделей. На уровне кода статический анализ синхронизации проводится с использованием целочисленного линейного программирования. Рассчитанное время выполнения в наилучшем и худшем случаях используется для ограничения выполнения определенных переходов в моделях ТА, в которые преобразуются модели функциональных блоков. Соответствующие правила перевода предназначены для создания высокоточных моделей с использованием синхронизированных автоматов. Затем проводится анализ времени отклика на этих моделях. Насколько нам известно, предложенный подход впервые разрабатывает унифицированную структуру для анализа времени отклика, которая учитывает время выполнения на уровне кода при получении результатов высокоуровневого анализа времени отклика.

Есть несколько ограничений предлагаемого подхода. Во-первых, наша модель сердца основана на абстрактном дизайне, называемом случайной сердечной системой [13]. Хотя модель хороша для масштабируемости нашего анализа, ей не хватает точности по сравнению с недавними моделями, такими как [7]. Наша работа улучшает точность модели кардиостимулятора, и мы также должны рассмотреть модель сердца с высокой точностью, поскольку обе модели состоят из замкнутых систем. Во-вторых, такие вопросы, как энергопотребление [7], следует учитывать в дополнение к времени отклика при проведении анализа, поскольку кардиостимулятор должен работать непрерывно в течение нескольких лет после его имплантации.

Библиография

- [1] Gurobi. <http://www.gurobi.com/>.
- [2] Boston scientific. ADVANTIO pacing system specifications, 2012. <http://www.bostonscientific.com/>.
- [3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, Vol. 126, No. 2, 183–235, 1994.
- [4] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.
- [5] S. S. Barold, R. X. Stroobandt, and A. F. Sinnaeve. *Cardiac Pacemakers Step by Step*. Futura Publishing, 2004.
- [6] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on Uppaal. Department of Computer Science, Aalborg University, Denmark, 2004.
- [7] T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative verification of implantable cardiac pacemakers over hybrid heart models. *Information and Computation*, Vol. 236, No. 0, 87–101, 2014.

- [8] G. D. Clifford, S. Nemati, and R. Sameni. An artificial vector model for generating abnormal electrocardiographic rhythms. *Physiological Measurement*, Vol. 31, No. 5, 595, 2010.
- [9] T. A. Henzinger. The theory of hybrid automata. In M. K. Inan and R. P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170, pages 265–292. Springer, Heidelberg, 2000.
- [10] J. R. Hussan, P. J. Hunter, and M. L. Trew. A clustering method for calculating membrane currents in cardiac electrical models. *Cardiovascular Engineering and Technology*, Vol. 3, No. 1, 3–16, 2012.
- [11] E. Jee, S. Wang, J. K. Kim, J. Lee, O. Sokolsky, and I. Lee. A safetyassured development approach for real-time software. In *Embedded and Real-Time Computing Systems and Applications, IEEE 16th International Conference on*, pages 133–142, 2010.
- [12] Z. Jiang, M. Pajic, R. Alur, and R. Mangharam. Closed-loop verification of medical devices with model abstraction and refinement. *International Journal on Software Tools for Technology Transfer*, Vol. 16, No. 2, 191–213, 2014.
- [13] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In C. Flanagan and B. König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214, pages 188–203. Springer, Heidelberg, 2012.
- [14] I. Liu. *Precision Timed Machines*. PhD thesis, University of California, Berkeley, May 2012.
- [15] S. Malik, M. Martonosi, and Y.-T. S. Li. Static timing analysis of embedded software. In *Proceedings of 34th Annual Design Automation Conference*, pages 147–152, New York, 1997. ACM.
- [16] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In *18th Real-Time and Embedded Technology and Applications Symposium, IEEE*, pages 173–184, 2012.
- [17] P. S. Roop, S. Andalam, R. von Hanxleden, S. Yuan, and C. Traulsen. Tight WCRT analysis of synchronous C programs. In *Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 205–214, New York, 2009. ACM.
- [18] R. F. Schulte, G. B. Sands, F. B. Sachse, O. Dossel, and A. J. Pullan. Creation of a human heart, model and its customisation using ultrasound images. *Biomedizinische Technik/Biomedical Engineering*, Vol. 46, No. s2, 26–28, 2001.
- [19] N. Trayanova. Your personal virtual heart. *Spectrum, IEEE*, Vol. 51, No. 11, 34–59, November 2014.
- [20] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al. The worst-case execution-time problem-overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 7, No. 3, 36, 2008.
- [21] L. Yoong, P. Roop, V. Vyatkin, and Z. Salcic. A synchronous approach for IEC 61499 function block implementation. *Computers, IEEE Transactions on*, Vol. 58, No. 12, 1599–1614, 2009.

[22] J. Zhao, T. D. Butters, H. Zhang, A. J. Pullan, I. J. LeGrice, G. B. Sands, and B. H. Smaill. An image-based model of atrial muscular architecture effects of structural anisotropy on electrical activation. *Circulation: Arrhythmia and Electrophysiology*, Vol. 5, No. 2, 361–370, 2012.

Применение Smart Grid через экономическую диспетчеризацию с использованием IEC 61499

Srikrishnan Jagannathan

Университет штата Пенсильвания

Peter Idowu

Университет штата Пенсильвания

СОДЕРЖАНИЕ

17.1 Введение	406
17.1.1 Функции интеллектуальной сети	407
17.1.2 Мотивация для исследования	408
17.1.3 Вклад	409
17.1.4 Улучшения, достигнутые благодаря внедрению IEC 61499	409
17.2 Ключевые понятия	410
17.2.1 Экономическая часть	410
17.2.2 Прогнозирование нагрузки	411
17.2.3 Ранжируемая стоимость энергии	411
17.2.3.1 Факторы, влияющие на LCOE	412
17.2.3.2 Вычисления LCOE	414
17.3 Программные инструменты	415
17.4 Разработка приложений	415
17.4.1 Описание сети энергосистемы	415
17.4.2 Этап 1 – нормальный режим работы модулятора	417
17.4.3 Этап 2 – экономическая часть единого микромодулятора	417
17.4.3.1 Список функциональных блоков, используемых на этапе 2	418
17.4.4 Этап 3 – Межмикрогридная экономическая диспетчеризация	419
17.4.4.1 Математическая формулировка	420
17.4.4.2 Список функциональных блоков, используемых на этапе 3	423
17.5 Совместное моделирование MATLAB® и FBDC	424
17.6 Результаты моделирования, выводы и дальнейшая работа	427
17.6.1 Результаты симуляции для выходных мощностей	428
17.6.2 Ежедневная стоимость генерации для различных этапов	332

17.1. Введение

Автоматизация интеллектуальной сети в ее предварительном и развивающемся состоянии достигается посредством централизованного управления, где центральная система управления контролирует всю автоматизацию сети и коммуникации. Эти системы более известны как системы диспетчерского управления и сбора данных (SCADA), которые по сути являются централизованными. Системы SCADA реализуются с использованием больших программ, которые разрабатываются на основе приложения и которые трудно использовать для других приложений. Современные системы управления требуют гибкости для продвинутой автоматизации. Для достижения такого высокого уровня гибкости в этих системах требуется новая программная технология, основанная на взаимодействии распределенных объектов и нацеленная на децентрализацию управления.

IEC 61499 – это новый метод разработки программного обеспечения, который позволяет распределять моделирование приложений управления. Стандарт представляет руководящие принципы для протоколов автоматизации в различных приложениях, от промышленной до интеллектуальной сети.

Цель этой главы – подчеркнуть и продемонстрировать преимущества генерации микросетей с помощью инновационного приложения экономической диспетчеризации, реализованного в IEC 61499. Вклад этого исследования заключается в использовании концепции функциональных блоков (FB) для реализации приложения экономической диспетчеризации с учетом уровня стоимости энергии (LCOE), наличия распределенных генераторов (DG) и прогноза нагрузки, чтобы ее сбалансировать. LCOE был рассчитан на основе исследований, проведенных Управлением энергетической информации (EIA). Предыдущая работа в этой области заключалась в том, чтобы выполнить балансировку нагрузки с использованием FB – с упором только на доступность, а не на экономический аспект. Экономическое моделирование систем возобновляемой энергии реализуется в приложении экономической диспетчеризации в сочетании с моделью прогнозирования каждой нагрузки. Ожидается, что экономия затрат на производство энергии будет достигнута за счет применения различных методов тестирования. Следовательно, эффективность каждого действия изображается путем сравнения мощности, поставляемой утилитой под различные приложения, и путем сравнения стоимости производства энергии каждого из генерирующих источников в течение одного дня. Эта стоимость может помочь лицам, определяющим политику, в принятии законов о взаимодействии различных коммунальных предприятий.

17.1.1. Функции интеллектуальной сети

Умная сеть может рассматриваться как электрическая сеть с возможностью принятия решений. Это электрическая сеть, которая может разумно сочетать выработку электроэнергии и потребности потребителя, чтобы обеспечить эффективную, экономичную и надежную доставку устойчивой электроэнергии. Основная идея концепции интеллектуальных сетей заключается в преобразовании электрической сети из централизованной сети, контролируемой производителем, в децентрализованную и интерактивную для потребителей [10]. Технология Smart Grid может использовать датчики и счетчики для отправки и получения данных о сети. Это может помочь в решении проблемы энергетических помех в сети, а также повысить эффективность и безопасность энергоснабжения. Современные микроконтроллеры, реализованные с подходящим ПО, могут обеспечить распределенную генерацию и помочь избежать перегрузки сети, тем самым улучшая реализацию возобновляемых источников энергии. Интеллектуальная сеть разработана с целью преобразования традиционной сети в современную систему, которая идет в ногу с развитием технологий генерации и распределения.

Двусторонняя связь в режиме реального времени доступна в интеллектуальной сети, которая позволяет клиентам взаимодействовать с сетью. Потребители могут экономить энергию и продавать энергию в сеть. Связь осуществляется с помощью передовых измерительных технологий. Установка распределенных генерирующих блоков, таких как бытовые солнечные панели и небольшие ветряные турбины, повысит эффективность интеллектуальной сети за счет выработки электроэнергии вблизи мест нагрузок, избегая потерь при распределении и пиковых нагрузок. «Пиковое бритье» – это процесс уменьшения количества энергии, приобретаемой у коммунальной компании в часы-пик, когда ставки самые высокие. Это позволит мелким отечественным потребителям и предприятиям продавать электроэнергию своим соседям или даже возвращать ее в распределительную сеть. Эта же концепция может быть применена к более крупным коммерческим организациям, имеющим системы возобновляемой энергии, которые могут передавать избыточную мощность обратно в сеть в часы максимальной нагрузки.

С помощью интеллектуальной энергосистемы можно повысить надежность энергосистемы и качество электроэнергии, а также сократить выбросы оксида углерода за счет интеграции возобновляемых источников энергии и использования распределенного хранилища, чтобы обеспечить проникновение этих возобновляемых ресурсов генерации экономически эффективным образом. Предоставляя развитую инфраструктуру, можно разрабатывать новые автоматизированные системы передачи и распределения, способные реализовывать различные функции защиты и управления энергосистемой, такие как управление реактивной мощностью для поддержания напряжения системы, регулирование напряжения, что позволит снизить стоимость

операций на основе предельного производства, линейное изменение и отслеживание нагрузки, а также ее снижение в периоды перегрузок. Таким образом, для реализации концепции интеллектуальной сети, сеть энергосистемы нужно оснастить различными функциями – и это стоит обсудить.

Особенности интеллектуальных сетей включают в себя: (1) интеграцию возобновляемых источников энергии; (2) внедрение различных распределенных энергоресурсов (DER) и концепцию микросети; (3) оптимизированную реализацию распределенной генерации с учетом таких факторов, как стоимость топлива и эксплуатационные расходы; (4) использование надежных протоколов связи в качестве эффективного способа обмена данными между различными интеллектуальными электронными устройствами, разработанными различными поставщиками в режиме «подключи и работай», и (5) распределенное управление и автоматизацию, необходимые для эксплуатации и обслуживания сети.

17.1.2. Мотивация для исследования

При наличии соответствующих стратегий для внедрения интеллектуальной сети на практическом уровне традиционная энергосистема может быть преобразована в интеллектуальную сеть, которая сделает ее эффективной и экономически выгодной для работы; это создаст беспроигрышную ситуацию для производителей и потребителей электроэнергии. Мотивация для этого исследования заключается во внедрении интеллектуального управления интеллектуальными сетками на уровне приложений в интеллектуальных сетях, характеристики которых обсуждались ранее. Традиционные системы управления, такие как PLC и DCS, не обладают гибкостью и соответствующей конструкцией и не могут обеспечить желаемый уровень интеграции [5]. Стандарт автоматизации IEC 61499 рассматривается как новое решение для автоматизации, позволяющее преодолеть эти ограничения [2]. Преимущество этого стандарта состоит в том, что он предоставляет гибкое решение, которое можно использовать для моделирования систем управления, разбивая процесс управления на базовые экземпляры FB. В настоящее время он применяется в различных промышленных приложениях. Диапазон его применения в настоящее время расширяется и включает интеллектуальные сетки. Функциональные возможности, реализованные с использованием этого стандарта, относятся к области координации защиты, изоляции и устранения неисправностей [13, 14].

17.1.3. Вклад

Исследование, описанное в этой работе, предлагает инновационное приложение для экономической диспетчеризации, реализованное с помощью концепции FB в IEC 61499. Предыдущей работой в этой области была балансировка нагрузки с использованием FB с акцентом только на доступности выработки электроэнергии – без учета экономического аспекта. Предыдущая работа также не учитывает способность прогнозировать потребности или нагрузку [7, 12]. Работа, описанная в этой главе реализует приложение экономической отправки с факторами, которые не были включены в предыдущей работе. Экономическое моделирование включает исследования, проведенные EIA и Национальной лабораторией возобновляемой энергии (NREL) для оценки параметров ценовой функции. Входные данные для системы управления, реализующей приложение экономической диспетчеризации, будут включать стоимость генерации для возобновляемых и традиционных энергетических систем, основанную на реальных экономических ограничениях, таких как капитальные затраты, расход топлива, скорость нагрева, наряду с доступностью генерации и прогнозируемой нагрузкой. Конечная цель – добиться экономии затрат на производство энергии за счет реализации каждого необходимого действия. Следовательно, эффективность каждого действия изображается путем сравнения мощности, поставляемой утилитой для различных приложений, и сравнения стоимости производства энергии различными распределенными генераторами за один день.

17.1.4. Улучшения, достигнутые благодаря внедрению IEC 61499

Моделирование объектов реального мира. При разработке приложения полезно разбивать сложные реальные проблемы на базовые программные компоненты. Это снижает сложность проблемы и упрощает процесс разработки приложений. Экономическая диспетчеризация рассматривалась как практическая сфера деятельности. Благодаря внедрению IEC 61499 экономическое распределение было разбито на небольшие этапы путем создания FB, которые реализуют определенные участки всего приложения.

Возможность повторного использования. В приложениях объектно-ориентированного программирования разработчики могут повторно использовать одни и те же классы объектов путем создания экземпляров. Эта функция может быть расширена до FB. Например, как часть этого приложения, FB был создан для сравнения мощности. Как только этот FB был разработан и протестирован, он стал частью базы данных или библиотеки инструмента разработки. Экземпляры этого FB затем использовались в приложении несколько раз.

Понижение сложности. Хотя хорошей практикой является понимание поведения FB, тем не менее, может использоваться без понимания внутреннего

поведения, пока известна связанная с ним задача. Приложение было построено путем создания и связывания FB, предоставляемых в библиотеке используемого программного инструмента. Это облегчало разработку и ведение различных схем управления. Время, необходимое для сборки приложения, было значительно сокращено. Система имеет модульность кода и простую реконфигурацию. Это может быть полезно для инженеров, которые не имеют опыта в области автоматизации, но заинтересованы в разработке приложений для управления.

Простая реконфигурация. IEC 61499 предоставляет метод графического проектирования системы, а также простой способ распределения функций в процессе автоматизации. Также имеется преимущество простой смены существующих приложений.

17.2. Ключевые понятия

Перед описанием приложения обсудим основные понятия, относящиеся к пониманию управленческих действий.

17.2.1. Экономическая часть

Экономическая диспетчеризация определяется как надежная работа генерирующих блоков для производства энергии с наименьшими затратами для потребителей с учетом ограничений, существующих в системе. Классический подход к экономическому распределению, по сути, предполагает экономическое распределение нагрузки на генерацию. Четырьмя основными направлениями в области экономической диспетчеризации являются [1]:

- **оптимальный поток энергии** направлен на минимизацию затрат на генерацию в системе с учетом реальной мощности (P), реактивной мощности (Q) и ограничений напряжения в системе;
- **экономическая диспетчеризация** в отношении автоматического управления генерацией (AGC) обеспечивает оптимальную диспетчеризацию с ограничениями частоты нагрузки;
- **динамическая диспетчеризация** предполагает минимизацию эксплуатационных расходов и обеспечение генерации на основе прогнозов нагрузки;
- **диспетчеризация с использованием возобновляемых источников энергии** – это экономическое решение для диспетчеризации с финансовыми ограничениями затрат на моделирование объектов с использованием возобновляемых источников энергии.

По сути, экономическая диспетчеризация направлена на оптимизацию целевой функции с учетом различных ограничений системы. Этими ограничениями могут быть заданные значения напряжения и частоты, стоимость генерации или параметры прогноза нагрузки. На каждое из этих

ограничений могут влиять различные параметры. Например, переменная прогнозирования нагрузки может быть смоделирована как случайный процесс, или на стоимость генерации могут влиять такие параметры, как местоположение, стоимость топлива и коэффициент мощности.

17.2.2. Прогнозирование нагрузки

Прогнозирование нагрузки является важным аспектом планирования и эксплуатации энергосистемы. Оно как бы подразумевает прогнозирование груза перед доставкой груза. Интервалы прогнозирования – это в основном интервалы времени, на которые прогнозируется нагрузка. Такие интервалы можно классифицировать как очень краткосрочные (от нескольких секунд до нескольких минут), краткосрочные (от получаса до нескольких часов), среднесрочные (от дней до недель) и долгосрочные (от месяцев до лет) [4]. Чтобы максимизировать финансовую выгоду, хороший метод прогнозирования нагрузки должен быть способен отражать текущие и будущие тенденции. На основе исторических данных и статистических методов может быть выполнено краткосрочное прогнозирование нагрузки (STLF).

17.2.3. Ранжируемая стоимость энергии

В этом разделе описывается экономическая модель, используемая для оценки системы. Для каждой развивающейся технологии ключевым аспектом, который изучают исследователи и работники отрасли, является осуществимость технологии. Следовательно, инженерная экономика является ключевым направлением. Осуществимость потенциальных решений, которые разрабатывают инженеры, оценивается наряду с техническими аспектами. В области выработки электроэнергии LCOE – это математическая величина, которую можно использовать для сравнения относительных затрат на производство энергии из генерирующего источника.

LCOE можно считать мерой, обобщающей общую конкурентоспособность различных генерирующих технологий. Это стоимость в долларах за киловатт-часов, а также стоимость строительства, эксплуатации и обслуживания генерирующей установки в течение предполагаемого срока ее службы. Основные исходные данные для расчета LCOE включают капитальные затраты, постоянные и переменные затраты на эксплуатацию и обслуживание (OandM), затраты на финансирование, факторы завода и затраты на топливо. Эти факторы варьируются в зависимости от типа завода. Влияние каждого из факторов на LCOE варьируется для разных генерирующих технологий. Такие технологии, как гидро- и солнечная энергия, не требуют затрат на топливо и, следовательно, имеют относительно небольшие переменные затраты OandM. В таких случаях на LCOE влияют предполагаемые капитальные затраты и генерирующие мощности. Для технологий со значительными расходами на

топливо капитальные затраты и затраты на топливо влияют на LCOE. Наличие различных стимулов, включая налоговые льготы, также может повлиять на расчет LCOE. Каждый из этих факторов и его значения меняются в зависимости от местоположения и времени по мере развития технологий и изменения цен на топливо.

Уравнение LCOE позволяет сравнивать альтернативные технологии для различных масштабов работы, периодов времени эксплуатации или инвестиций [9]. Используя эту мотивацию, LCOE был использован в качестве эталонной модели для экономической оценки потенциальных решений, предложенных в этой работе.

17.2.3.1. Факторы, влияющие на LCOE

Как обсуждалось в предыдущем разделе, существует несколько факторов, влияющих на LCOE. Этот раздел описывает их подробно.

1. Капитальные инвестиционные затраты. Основными составляющими затрат, влияющими на генерирующие технологии, являются:

(а) **Затраты на строительство.** Обычно это основные расходы любого проекта. Расходы на гражданское строительство определяются тенденциями цен в стране, где проект будет разрабатываться. В странах, переживающих переходный период экономики, затраты на гражданское строительство, как правило, ниже, чем в развитых странах. Это связано с размещением местных строительных материалов и местной рабочей силы. Затраты на строительство зависят от внутренних характеристик, топологии, геологических и погодных условий и конструкции проекта. Следовательно, стоимость строительства зависит от конкретной площадки. Это может привести к изменению инвестиционных затрат и, следовательно, LCOE – даже для проектов с одинаковыми возможностями.

(б) **Стоимость оборудования энергосистемы.** Стоимость электромеханического оборудования соответствует мировым рыночным ценам на эти компоненты, а не стоимость строительства. Тип технологии, используемой для генерации, является важным компонентом. Например, на солнечной электростанции тип солнечного модуля влияет на инвестиционные затраты, поскольку требует установки специального оборудования. Кроме того, инвестиционные затраты также включают планирование, лицензирование, анализ воздействия на окружающую среду, смягчение воздействия на дикую природу, исторические и археологические памятники и места отдыха.

Исследования [8] показывают общую тенденцию увеличения инвестиционных затрат по мере увеличения емкости, а также то, что для проектов с такой же мощностью существует широкий диапазон затрат. Для установок с меньшей установленной мощностью, например 5 МВт, затраты на электромеханическое оборудование, как правило, более значительны. Однако с увеличением мощности завода затраты на строительство становятся более

влиятельными. Та же общая генерирующая мощность может быть достигнута за счет комбинации нескольких небольших или больших генерирующих блоков. Установки, в которых используются несколько генерирующих блоков, имеют более высокую стоимость за киловатт (кВт) по сравнению с установками, использующими меньшее количество более крупных блоков. Причина увеличения затрат на кВт, связанных с большим количеством генерирующих блоков, заключается в большей гибкости и эффективности, достигаемой за счет интеграции станций в электрическую сеть. Для заводов, у которых меньше установок с большей генерирующей мощностью, инвестиционные затраты, как правило, снижаются. Например, для более крупного энергоблока проект гидроэнергетики может быть настроен на использование меньшего объемного расхода и, следовательно, меньших гидравлических трубопроводов. Размеры оборудования и связанные с ним расходы также будут ниже. Результаты [8] показали характерное распределение инвестиционных затрат по географическим районам.

2. Стоимость эксплуатации и технического обслуживания. Технологии использования возобновляемых источников энергии после ввода в эксплуатацию, как правило, требуют минимальной эксплуатации и технического обслуживания. Основная причина заключается в том, что заводы, работающие с такими технологиями, не несут непрерывных расходов на топливо. Затраты OandM могут быть рассчитаны как процент от стоимости инвестиций за кВт. В отличие от этого, традиционные генерирующие блоки, такие как базовые топливные блоки, имеют более высокую стоимость OandM из-за периодических расходов на топливо. Это также зависит от местоположения и условий рынка.

3. Коэффициент мощности. Коэффициент мощности электростанции – это отношение ее фактической мощности к выходной мощности, которую она может производить при непрерывной работе на полной заводской мощности или номинальной мощности в течение определенного периода. Параметром, влияющим на коэффициенты мощности, является в основном доступность возобновляемых ресурсов. Это, в свою очередь, зависит от местоположения и типа используемой технологии. Для гидроэнергетики коэффициент мощности обычно рассчитывается на этапах планирования и оптимизации проекта с учетом как рыночного спроса на электроэнергию, так и статистического распределения потока. Резервуары могут быть спроектированы таким образом, чтобы повысить стабильность потока для производства при базовой нагрузке и обеспечить очень изменчивый и надежный поток для электростанции. Для солнечной энергии основным параметром, влияющим на коэффициент мощности, является годовая солнечная радиация и тип используемой панели или модуля. Тип модуля влияет на эффективность преобразования света в электричество, что влияет на фактическую производительность установки и, следовательно, коэффициент мощности. Низкий коэффициент мощности подразумевает низкое производство и более высокий LCOE. В целом коэффициенты мощности технологий использования возобновляемых источников энергии ниже, чем у традиционных энергоблоков. Это связано с

неустойчивостью или недоступностью возобновляемых источников энергии в любое время.

4. Срок службы. Установки, связанные с возобновляемой энергией, такие как плотины, туннели, каналы, гидроэлектростанции или компоненты технологии концентрированной солнечной энергии (CSP), являются основными источниками затрат, поскольку они должны иметь длительный срок службы. Электрическое и механическое оборудование со сравнительно более коротким сроком службы способствует снижению затрат. Следовательно, для гидроэлектростанций характерно использование более продолжительного срока службы по сравнению с другими источниками производства электроэнергии. Типичный срок службы находится в диапазоне от 40 до 80 лет.

5. Ставка дисконтирования. Ставка дисконтирования в основном представляет собой процент, который может влиять на LCOE, основываясь на тенденциях расходов и доходов, которые наблюдаются в течение срока действия инвестиций. Инвесторы, как правило, выбирают ставки дисконтирования на основе характеристик доходности риска имеющихся вариантов инвестирования. Высокая ставка дисконтирования выгодна для технологий, которые имеют низкие начальные капиталовложения и высокие эксплуатационные расходы. С другой стороны, низкая ставка дисконтирования способствует возобновляемым источникам энергии, таким как гидроэнергия и солнечная энергия, которые имеют сравнительно более высокие капитальные вложения и более низкие затраты на техническое обслуживание. Типичные значения для технологий использования возобновляемых источников энергии составляют от 7 до 10.

17.2.3.2. Вычисления LCOE

Уравнение LCOE зависит от стоимости капитальных вложений, ставки дисконтирования, стоимости OandM, коэффициента мощности, стоимости топлива, скорости нагрева и срока службы установки. Учетная ставка может быть реальной или номинальной. Используя значения ставки дисконтирования и срок службы, рассчитывается коэффициент восстановления капитала (CRF). CRF – это отношение постоянного аннуитета к приведенной стоимости получения этого аннуитета за определенный период времени. Используя ставку дисконтирования i и срок службы завода n в годах, CRF рассчитывается, как показано в уравнении (17.1).

$$CRF = \frac{i \cdot [1+i]^n}{[1+i]^n - 1} \quad (17.1)$$

На основании определений в [6] LCOE рассчитывается, как показано в уравнении (17.2):

$$LCODE = \frac{CI * CRF + FOM}{8760 * CF} + (FC * HR) + VOM \quad (17.2)$$

где:

CI: капитальные затраты;

FOM: исправлено О-и-М;

CRF: коэффициент возврата капитала;

CF: коэффициент мощности;

FC: стоимость топлива;

HR: скорость нагрева;

VOM: переменные затраты на эксплуатацию и техническое обслуживание.

17.3. Программные инструменты

Программным средством, используемым для разработки приложения для экономической диспетчеризации, является комплект функциональных блоков (FBDK). Программным обеспечением, используемым для разработки сети энергосистемы, является MATLAB®/Simulink. FBDK может программироваться на JAVA. MATLAB®/Simulink также поддерживает программирование на JAVA. Эта функция используется для достижения синхронизированного моделирования между двумя средами.

17.4. Разработка приложений

В этом разделе описывается реализация двух приложений экономической отправки. Каждое приложение имеет различное поведение управления в среде FB. В этом разделе подробно описывается конфигурация системы и каждое поведение экономической отправки.

17.4.1. Описание сети энергосистемы

Конфигурация и параметры энергосистемы были извлечены из эталонной системы стандарта IEEE 399-1997 с некоторыми изменениями, обеспечивающими автономную работу микросети [3]. Система, показанная на рисунке 17.1, состоит из трехсистемной распределительной подсистемы, работающей при напряжении 13,8 кВ. Эта подсистема подключена к основной сети или сети через радиальную линию 69 кВ. Большая сеть в конце радиальной линии представлена шиной с коротким замыканием 69 кВ, 1000 МВА. Каждый из трех фидеров подключен к большой сети на одном конце и источнику DG и комбинации переменных нагрузок на другом. Поскольку основное внимание уделяется наблюдению за поведением приложений с

балансировкой нагрузки в нормальных условиях, каждый из DG смоделирован для работы в установившемся режиме и обозначен DG 1, DG 2, DG 3, каждый из которых работает при напряжении 13,8 кВ. DG 1 – это обычная установка, работающая на природном газе, способная вырабатывать 4 МВт. DG 2 моделируется как фотоэлектрическая установка мощностью 4 МВт. DG 3 является гидроэлектростанцией, способной вырабатывать 3 МВт. Нагрузки от L1 до L6 подаются через радиальные питатели подсистемы. Нагрузки меняются со временем и создают переменный профиль нагрузки. Время моделирования составляет 12 секунд, что соответствует 24 часам в день. Таким образом, микросеть совместно имеет DG, две нагрузки и соответствующие компоненты распределительной сети.

Каждый DG имеет LCOE, который рассчитывается с использованием экономической модели. Коэффициенты, используемые при расчете значения LCOE, описаны в разделе 17.2.3.1. Значения LCOE приведены в таблице 17.1.

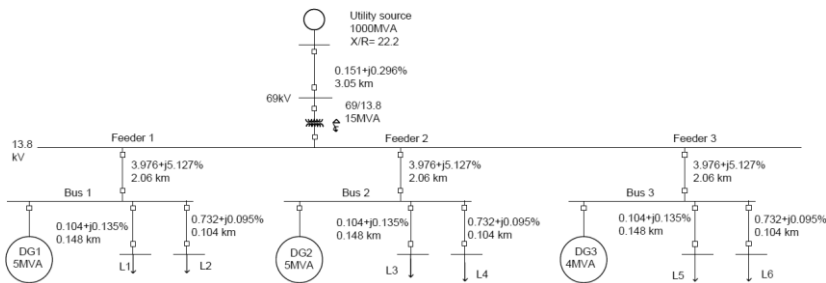


Рисунок 17.1. Однолинейная схема системной сети

Таблица 17.1. LCOE для рассматриваемых генерирующих технологий

Фактор	DG1	DG2	DG3
Период (в годах)	30	30	30
Процентная ставка (в процентах)	10	7	7
Коэффициент возврата капитала	0,10	0,08	0,08
Капитальные затраты (USD / кВт)	1090	2600	2359,3
Коэффициент емкости (в процентах)	75	25	57,5
Фиксированные О-и-М (USD / кВт-год)	11,96	7,56	11,3975
Переменная OandM (USD / кВт-ч)	0,003	0,001	0,0036
Тепловая мощность (БТЕ / кВт-	8185	0	0

ч)			
Стоимость топлива (USD / ММБТЕ)	5,03	0	0
LCOE (центы / кВт)	6,1	10,1	4,4

17.4.2. Этап 1 — нормальный режим работы модулятора

В этом случае нагрузки получают питание от своих соответствующих DG микросетей и коммунального предприятия без учета распределения нагрузки. Мощность, подаваемая на нагрузки, распределяется между генераторами в зависимости от их сопротивления.

17.4.3. Этап 2 — экономическая часть единого микромодулятора

Приложение, разработанное на этом этапе, включает взаимодействие между утилитой и DG, связанным с нагрузками, которые должны быть сбалансированы. Любое увеличение нагрузки выше пороговых пределов в конкретной микросети распределяется только между утилитой и DG в этой микросети. Например, при увеличении нагрузки L1 необходимое управляющее действие выбирается между утилитой или ее комбинацией и DG 1 для балансировки нагрузки. Приложение разработано на одном устройстве с одним ресурсом. Архитектура управления показана на рисунке 17.2.

Критерии для балансировки нагрузки основаны только на доступности DG для согласования мощности и LCOE генерирующих источников. Каждая сетка имеет две связанные с ней нагрузки. Мощность, потребляемая каждой из нагрузок, обозначается P(1), P(2), P(3), P(4), P(5), P(6). Утилита подключается к сетям через СВ, обозначенные СВ(1), СВ(2), СВ(3) соответственно для сетей 1, 2 и 3. Каждая нагрузка имеет предварительно определенное пороговое значение (Pth1, Pth2, ..., Pth6). Отдельные пороговые значения рассчитываются как среднее значение варьирующегося профиля нагрузки, демонстрируемого каждой из нагрузок. Атрибуты нагрузки принимаются в главном центре управления. Значение прогнозируемой нагрузки в сети сравнивается с пороговым значением для этой конкретной сети. Если потребляемая мощность превышает пороговое значение хотя бы для одной из нагрузок в микросети, СВ между энергосистемой и микросетью, содержащей эту нагрузку, закрывается. Это гарантирует, что необходимая мощность подается на нагрузку эффективным образом и, следовательно, нагрузка сбалансирована. Описанный алгоритм выполняется на одном устройстве.

17.4.3.1. Список функциональных блоков, используемых на этапе 2

FB КЛИЕНТА для обмена данными: FB слева на рисунке 17.3 выступает в качестве интерфейса для приема данных от интеллектуального измерительного устройства (SMD) в Simulink. Входом в этот блок является IP-адрес измерительного устройства и условие защиты, которое имеет логическое значение 1. Выполнение FB запускается только тогда, когда он получает триггер входа события в INIT. Значения LREAL, TIME являются типами данных значений, полученных от SMD. По сути, это значения мощности, времени запуска и продолжительности нагрузки соответственно.

FB справа на рисунке 17.3 отправляет сигналы CB обратно в утилиту в Simulink через входы SD_1, SD_2 и SD_3. FB слева получает данные от Simulink. Снова оба SIFB действуют как ссылки для обмена данными из приложения в FBDK с сетью в Simulink.

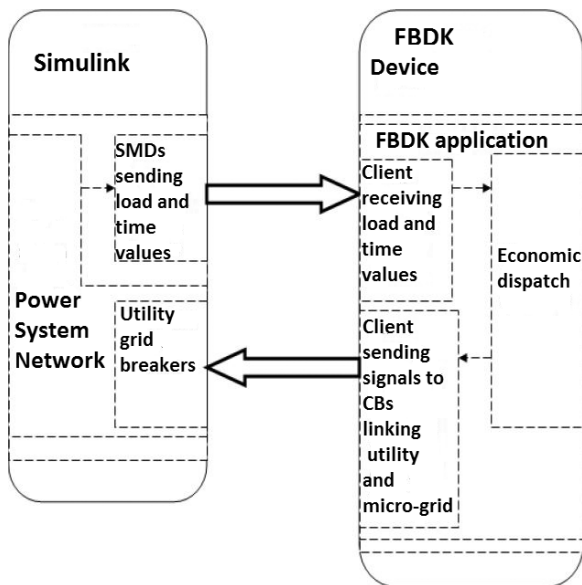


Рисунок 17.2. Эталонная архитектура для этапа 2

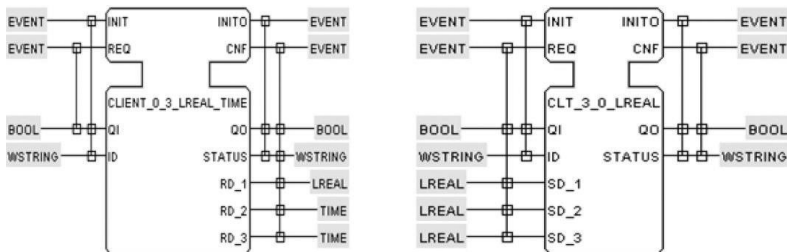


Рисунок 17.3. Функциональный блок для приема и отправки данных между Simulink и FBDC

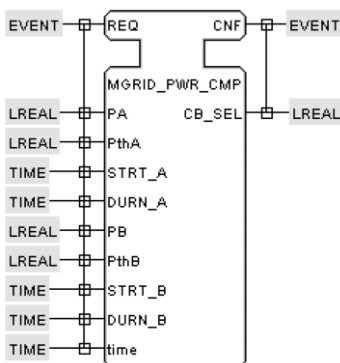


Рисунок 17.4. Функциональный блок для сравнения мощности и синхронизации времени

FB сравнения мощности: этот FB, показанный на рисунке 17.4, по существу выполняет сравнение нагрузок с их порогом для одной сети синхронизированным по времени образом. Выход этого FB – это сигнал, передаваемый в СВ, соединяющий сеть с соответствующей микросетью. Если нагрузки превышают пороговое значение, выход имеет значение 1, а если они меньше порогового значения, выход этого FB имеет значение 0.

17.4.4. Этап 3 — межмикросетевая экономическая диспетчеризация

В этом подразделе основное внимание уделяется применению экономической диспетчеризации и включению бокового взаимодействия между микросетями. Любое увеличение нагрузки в конкретной микросети может быть разделено утилитой и другими микросетями в сети. Приложение было разделено на несколько подприложений и реализовано на нескольких устройствах. Критерии для экономической диспетчеризации основаны на

наличии электроэнергии, стоимости генерации и прогнозируемой нагрузке. Этот этап является примером распределенного приложения, то есть одного приложения, распределенного по нескольким подпрограммам на нескольких устройствах, и, следовательно, может рассматриваться как приложение, реализованное на удаленных устройствах. Блок-схема показана на рисунке 17.5.

17.4.4.1. Математическая формулировка

Рассмотрим математическую формулировку приложения экономической отправки. В основе каждого приложения экономической отправки лежит целевая функция, которая направлена на поставку грузов экономически эффективным способом. Эта целевая функция затем реализуется в среде FB. Целевая функция и ограничения, используемые в этой работе, могут быть сформулированы как в уравнениях с 17.3 по 17.7.

$$\begin{aligned} [CB_{signal}, C_{min}] = f(P_{load}, Time_{start}, Time_{duration}, LCODE, P_{Gi}) \\ \text{при условии, что } L_{(min,i)} \leq P_{Gi} \leq P_{(Gi,max)} \end{aligned} \quad (17.3)$$

$$\sum P_{Gi} = \sum P_{load}$$

i представляет i -ый источник генерации

$$P_{Gi,max} = \min(P_{Gi,rated}, P_{th}) \quad (17.4)$$

$$P_{th} = \sum_{j=1}^K mean(P_{load,j}) \quad (17.5)$$

K : количество нагрузок в микросети;

$P_{Gi,номинальная}$: номинальная мощность генератора i

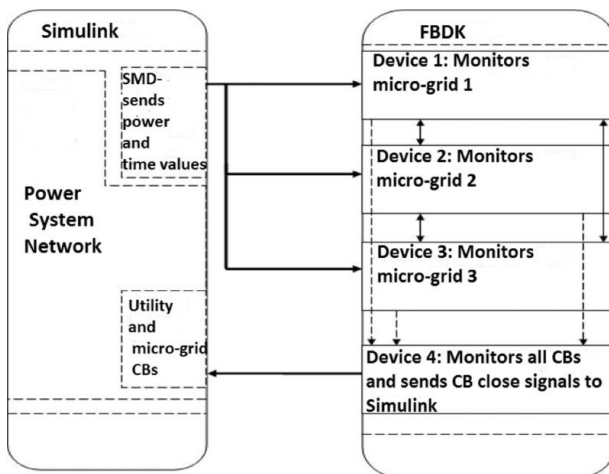


Рисунок 17.5. Эталонная архитектура для этапа 3

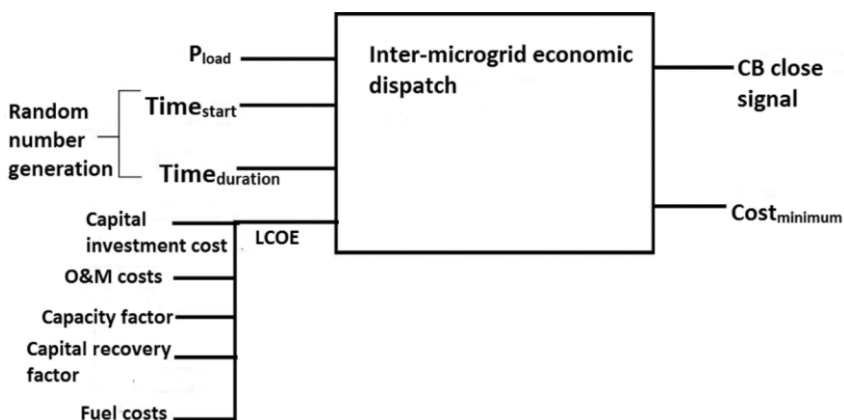


Рисунок 17.6. Приложение распределения активной нагрузки

где

$$L_{min} = \min(P_{load})$$

$$C_{min} = \min \sum_{j=1}^N LCODE * E_{Gi} \quad (17.6)$$

N: общее количество генерирующих источников

E_{Gi} : выходная мощность генератора на основе прогнозируемой P_{load}

$$E_{Gi} = P_{Gi} * 1hr \quad (17.7)$$

P_{load} : нагрузка (МВт)

$Time_{start}$: время начала нагрузки

$Time_{duration}$: продолжительность нагрузки

LCOE: приведенная стоимость энергии (USD / МВт-ч), см. уравнение (17.2)

На рисунке 17.6 показана блок-схема приложения экономической диспетчеризации, реализованного в данной работе. Предложение состоит в том, чтобы минимизировать стоимость взаимодействия микросетей. Исходя из экономических ограничений, ограничений нагрузки и доступности системы, которые используются в качестве входных данных для системы управления, выходные данные системы представляют собой минимальные эксплуатационные расходы и соответствующие управляющие сигналы для автоматических выключателей. Здесь входными данными для системы являются прогнозируемое требование нагрузки (P_{Load}), время и продолжительность начала нагрузки ($Time_{Start}$, $Time_{Duration}$) и LCOE. Время и продолжительность начала нагрузки генерируются генератором случайных чисел после равномерного распределения, чтобы создать профиль нагрузки, который действует как прогноз для системы управления. Функция стоимости для систем возобновляемой энергии основана на LCOE, который учитывает инвестиционные затраты на оборудование, капитальные затраты, затраты О-и-М и коэффициент мощности. Таким образом, экономическое диспетчерское решение сочетает в себе динамическую диспетчеризацию и прогнозирование нагрузки для обеспечения экономически эффективного решения. Значения LCOE основаны на исследованиях статистического анализа, проведенных Управлением энергетической информации, Национальными лабораториями возобновляемой энергии США.

Как и на этапе 2, каждая нагрузка имеет предварительно определенное пороговое значение (P_{th1} , P_{th2} , ..., P_{th6}). Каждая микросетка контролируется локальным центром управления (LCC) и отвечает за выполнение приложения экономической диспетчеризации в соответствующих сетях. Управляющие приложения LCC разработаны в FBDK. Следовательно, есть три локальных центра управления. Эти локальные центры управления затем сообщают о своих решениях по размыканию или замыканию автоматических выключателей в главном центре управления (CCC), который отправляет сигнал отключения соответствующему СВ. Три сетки связаны друг с другом через автоматический выключатель. СВ между сеткой 1 и сеткой 2 – CBgrid1-grid2, СВ между сеткой 2 и сеткой 3 – CBgrid2-grid3, а СВ между сеткой 1 и сеткой 3 – CBgrid1-grid3. В дополнение к этим СВ есть также три СВ, соединяющие каждую из сетей с основной утилитой. Они обозначены как CBUtility1, CBUtility2 и CBUtility3 соответственно для сетей 1, 2 и 3. Эти СВ

контролируются ССС после получения сигналов от ЛСС. Каждая нагрузка имеет определенное пороговое значение. Когда нагрузка в микросети превышает пороговое значение, эта сетка ищет источники, которые предлагают мощность по самой низкой цене. Если какая-либо из микросетей имеет избыточную емкость, она предоставляет ее в сеть с дефицитом. Если ни один DG не может обеспечить необходимое количество энергии целиком, питание подается через комбинацию DG и основной утилиты. Только когда нагрузка не может быть обеспечена DG микросети, дефицит полностью покрывается основной сеткой. В зависимости от того, какой DG планирует подавать питание, соответствующий прерыватель замыкается, обеспечивая тем самым необходимое количество питания для нагрузки.

17.4.4.2. Список функциональных блоков, используемых на этапе 3

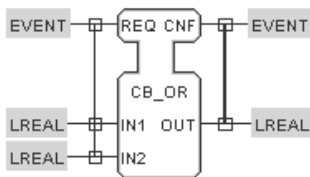
В дополнение к FB, используемым на этапе 2, на этапе 3 реализуются еще два FB.

Сигнал принятия решения о закрытии СВ: этот FB присутствует в устройстве 4 эталонной архитектуры для этапа 3 и показан на рисунке 17.7 (а). Он принимает в качестве входных сигналов СВ от первых трех устройств 2 одновременно. СВ будет закрыт, если хотя бы один из входов перейдет к 1. Алгоритм FB показан на рисунке 17.7 (b).

FB для сравнения затрат: FB на рисунке 17.8 выполняет алгоритм сравнения. Входные данные CostX, CostY и CostUTIL – это LCOE DG, от которых может получить питание соответствующая сетка.

После выполнения алгоритма сравнения мощности FB отправляет три выхода для трех СВ. Например, если мы рассматриваем сетку 1 как энергосистему, требующую мощности, СВ_A_B – это СВ, соединяющая сетку 1 и сетку 2. СВ_B_C – это СВ, соединяющая сетку 1 и сетку 3. СВ_UTIL – это СВ, соединяющая сетку 1 и утилиту. 1 означает закрытие СВ, а 0 – открытие СВ. Другими входами являются мгновенные значения нагрузки, их продолжительность и пороговые пределы.

Реализация показана на рисунке 17.9. Вывод FB, показанный на рисунке 17.4, запускает сравнение стоимости FB. Этот FB отвечает за выполнение приложения экономической отправки на основе затрат и нагрузки. Результаты сравнения стоимости FB в основном сигнализируют наиболее экономически эффективному распределенному генератору об увеличении его генерации для компенсации нагрузки.



(a)

```

if (IN1.value == 1 &&
    IN2.value == 1){
    OUT.value = 1;
} else if (IN1.value == 1 &&
    IN2.value == 2){
    OUT.value = 1;
} else if (IN1.value == 2 &&
    IN2.value == 1){
    OUT.value = 1;
} else if (IN1.value == 2 &&
    IN2.value == 2){
    OUT.value = 1;
}

```

(b)

Рисунок 17.7. Функциональный блок принятия решения (слева). Алгоритм функционального блока принятия решения (справа)

Действия, реализованные на этом этапе, являются примером распределенного приложения. Для успешного достижения распределенного поведения необходимо обеспечить надлежащую передачу соответствующих данных между различными устройствами. В FBDK это было реализовано с помощью блоков публикации и подписки – PUBL1 и SUBL1 соответственно. Данные, которыми обмениваются различные устройства с помощью блоков публикации и подписки, являются значениями нагрузки. Функциональные блоки PUBLISH и SUBSCRIBE вместе с СЕРВЕРОМ и КЛИЕНТОМ, которые будут обсуждаться в последующих разделах, вместе называются FB интерфейса службы, реализованными в этом приложении. Они составляют основу разработки распределенного приложения.

17.5. Совместное моделирование MATLAB® и FBDK

Приложение было реализовано посредством модели совместного моделирования между MATLAB® и FBDK. Чтобы добиться успешного совместного моделирования, оба программных инструмента должны иметь возможность обмениваться данными на языке, понятном обеим сторонам. Перед отправкой данных из MATLAB® в FBDK данные и переменные должны быть закодированы в типы данных, поддерживаемые и понимаемые FBDK. Это было основным принципом совместного моделирования, которому следуют в этой работе. Совместное моделирование было выполнено с помощью простого подхода с использованием файлов классов, используемых для создания FB в FBDK. Совместное моделирование было выполнено с использованием функции FBDK, основанной на Java. Следовательно, каждый FB состоит из файла класса. Эти файлы классов могут быть сжаты в файлы jar. Уникальная особенность MATLAB® заключается в том, что он позволяет

добавлять файлы библиотеки Java, когда они сжимаются в файл jar. Файлы jar могут быть добавлены в MATLAB® через командное окно. Добавление файлов jar подразумевает добавление файлов классов в MATLAB®. Следовательно, экземпляры или объекты этих классов могут быть включены в программы в MATLAB® и Simulink. Совместное моделирование выполняется с использованием TCP/IP. Поскольку MATLAB® является стороной, инициирующей связь, это – СЕРВЕР, а FBDK – КЛИЕНТ. Следовательно, файлы классов СЕРВЕР включены в MATLAB®.

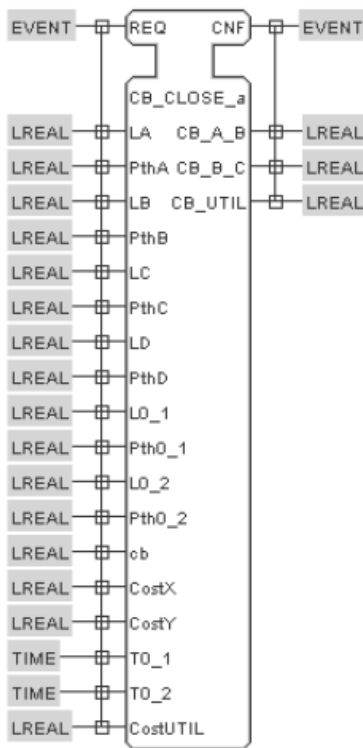


Рисунок 17.8. Функциональный блок сравнения затрат

Процедура включения файла класса СЕРВЕР для обмена значениями времени выглядит следующим образом. В папке или пакете, где хранится файл, откройте командную строку. Компиляция и запуск рассматриваемой Java-программы выполняются путем ввода следующих команд в командном окне:

```
javac $-$cp fbrt.jar $-$d . TIMESENDER.java
java $-$cp fbrt.jar;. time.server.TIMESENDER
```

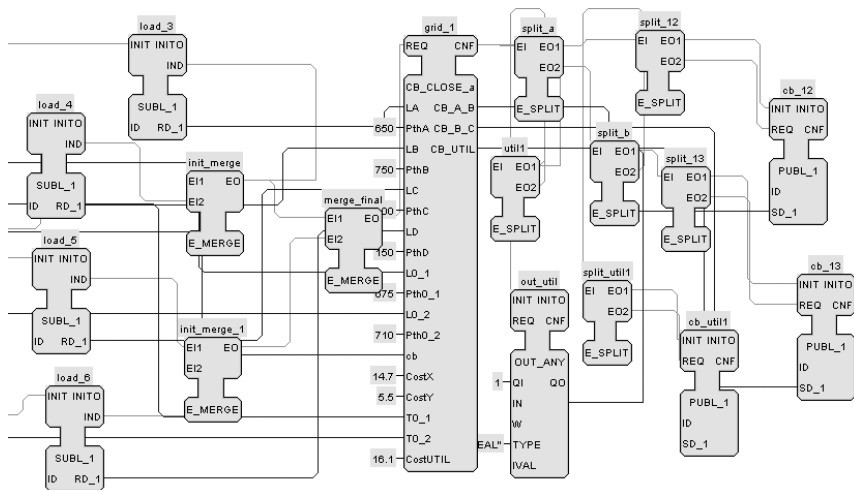


Рисунок 17.9. Реализация функционального блока сравнения затрат

Здесь имя файла Java – TIMESENDER. При выполнении вышеупомянутых двух команд файл TIMESENDER компилируется, затем мы создаем файл JAR с файлами классов и включаем их в каталог MATLAB®.

TIME SENDER – это, по сути, FB типа SERVER_1_0, который отправляет значения времени в FBDK. Файлы базовых классов унаследованы от fbrt.jar. Следовательно, TIME SENDER можно рассматривать как виртуальный FB, созданный в конце MATLAB® для включения совместного моделирования. На принимающей стороне FB будет иметь тип КЛИЕНТ. В этом приложении он был назван CLT_0_1_TIME. Составной FB инкапсулирует КЛИЕНТА, который обменивается значениями времени. Составной FB выглядит так, как показано на рисунке 17.10 (а), а SIFB выглядит как рисунок 17.10 (б). В этом случае значение данных RD_1 имеет тип данных TIME.

Для успешного достижения совместного моделирования между Simulink и FBDK необходима синхронизация времени между двумя платформами. Альтернативно: временной шаг в Simulink должен совпадать с временным шагом в FBDK. Метод, реализованный в этой заявке, является адаптированным методом, описанным в [11]. Блок MGRID_PWR_CMP является FB, который выполняет сравнение времени и, следовательно, синхронизацию. Для удобства это снова показано на рисунке 17.11 (а) вместе с FB. Входы с метками STRT_A, DURN_A, STRT_B, DURN_B выполняют синхронизацию времени. Это позволяет выполнить определенное событие в указанный момент времени.

Ввод, помеченный TIME, получает свой вход от СЕРВЕРА, созданного в MATLAB®, используя процедуру, описанную в разделе 17.5. ECC для FB выше выглядит так, как показано на рисунке 17.11 (б). После описания ECC и переходов между состояниями часть приложения, выполняющая синхронизацию времени, показана на рисунке 17.12 для лучшего понимания. Данные из MATLAB® получены с использованием CLIENT_0_3_LREAL_TIME. Полученные значения обозначают требование

нагрузки, время ее начала и ее продолжительность. Это основные параметры, необходимые для реализации приложения экономической отправки. Затем они отправляются в FB с меткой MGRID_PWR_CMP. Синхронизация времени и сравнение нагрузки выполняются с использованием этого FB. Эта часть приложения общая для этапа 2 и этапа 3.

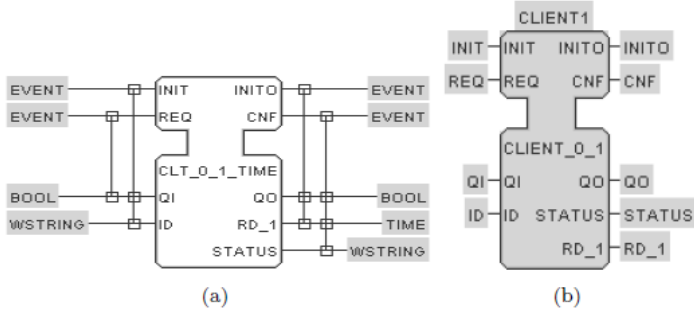


Рисунок 17.10. Составной функциональный блок (слева) и инкапсулированный SIFB (справа) для получения значений времени из MATLAB®/Simulink в FBDK

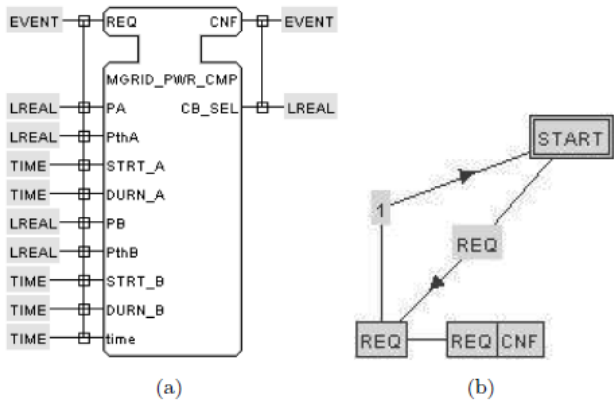


Рисунок 17.11. Функциональный блок для сравнения мощности и синхронизации времени (слева) и ECC (справа)

17.6. Результаты моделирования, выводы и дальнейшая работа

Как обсуждалось в предыдущих разделах, разработка приложений делится на три этапа. На каждом этапе реализуется разное поведение выравнивания нагрузки для одного и того же ее изменения. Изменения во времени для каждой нагрузки разработаны с использованием генератора случайных чисел.

Эта логика обеспечивает случайное изменение профиля нагрузки. Общее моделирование выполняется в течение 12 секунд, чтобы имитировать 24 часа в день. Таким образом, была создана среда для сравнения эффективности каждого поведения. Эффективность каждого приложения определяется путем сравнения мощности, поставляемой DG в каждой из микро-сетей и утилитой. На основе произведенной мощности рассчитывается и сравнивается общая стоимость эксплуатации за один день. Результаты показывают мощность, генерируемую DG и утилитой, а также расчет стоимости производства энергии, генерируемой DG и утилитой.

17.6.1. Результаты симуляции для выходных мощностей

Этап 1

Результаты для этого этапа показаны на рисунке 17.13. Утилита и DG подают нагрузки на основе импедансов генератора. Этот этап представляет собой типичную работу сети электропитания без какого-либо контроля формы. Это создает основу для развития последующих этапов.

Этап 2

На рисунке 17.14 показано питание, подаваемое утилитой и DG в приложении балансировки нагрузки на этапе 2.

По сравнению с мощностью, поставляемой на этапе 1, наблюдается снижение мощности, подаваемой утилитой, и соответствующее увеличение мощности, поставляемой микросетью DG 3. Этот результат основан на сравнении доступности и LCOE DG 3 и утилиты. Для рассматриваемой модели, поскольку стоимость DG 3 меньше стоимости электроэнергии коммунальной сети, DG 3 является основным выбором для удовлетворения избыточного спроса. Снижение энергопотребления можно заметить во временном окне, начиная с 0,5 до 6 секунд. На первом этапе мощность, поставляемая утилитой, составляет приблизительно 1,25 МВт. На этапе 2 для того же временного окна мощность, подаваемая утилитой, уменьшается примерно до 1 МВт. Это снижение мощности, поставляемой утилитой, должно быть компенсировано нагрузкой для постоянного приема энергии. Это уменьшение компенсируется DG 3. Увеличение генерации на ту же величину может наблюдаться для DG 3. Мощность, генерируемая DG 1 и DG 2, практически не изменяется, что указывает на то, что нагрузки, связанные с этими DG, находятся ниже порогового значения. Преимущество приложения на этом этапе заключается в том, что оно снижает стоимость генерации по сравнению со стоимостью генерации на этапе 1 и является простым в реализации.

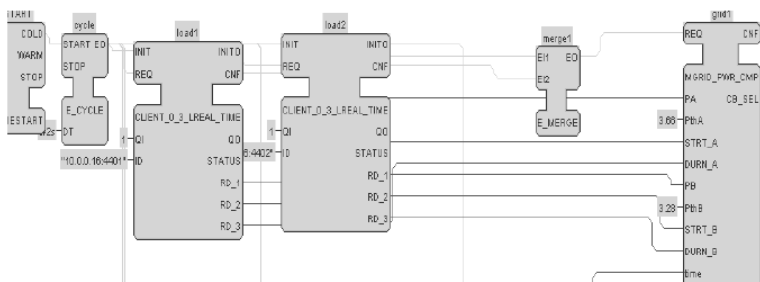


Рисунок 17.12. Приложение для синхронизации времени и сравнения мощности

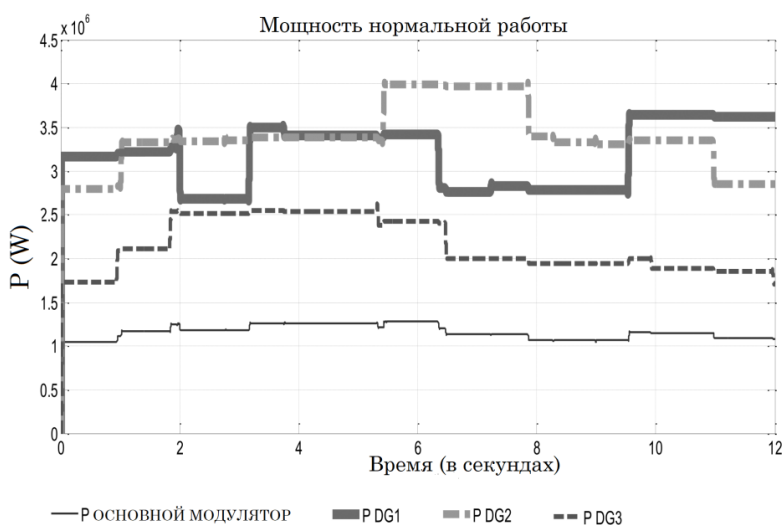


Рисунок 17.13. Питание подается на 24 часа: 1 этап

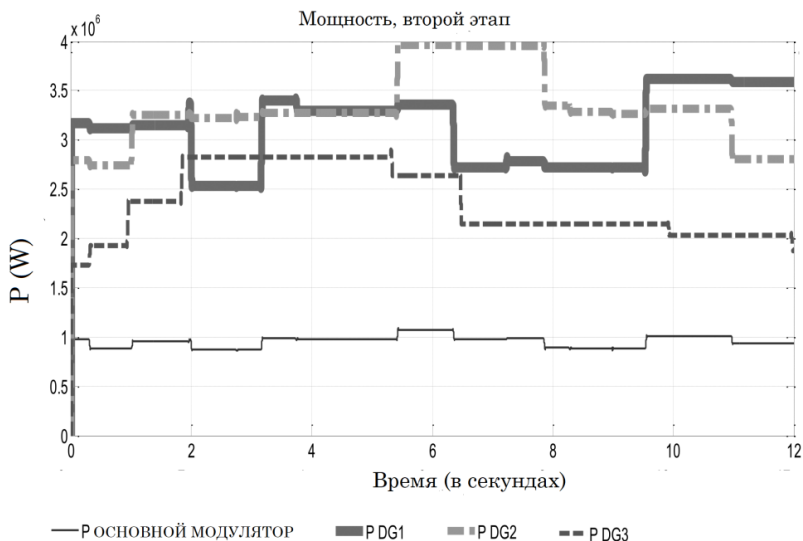


Рисунок 17.14. Питание подается на 24 часа: 2 этап

Этап 3

Приложение для балансировки нагрузки на этапе 3 более эффективно, чем приложение на этапе 2, с точки зрения стоимости эксплуатации. Это связано с тем, что приложение экономической диспетчеризации также учитывает экономику генерации источников в соседних микросетях. Приложение учитывает доступность и LCOE как факторы при расчете выработки электроэнергии для каждого источника на этапе 3. Этот фактор стоимости помогает лицам, определяющим политику, в определении стоимости продажи мощности. На рисунке 17.15 показана мощность, генерируемая DG и сетью инженерных сетей.

Разницу между выходными сигналами мощности на этом этапе и выходными сигналами мощности на этапе 2 можно заметить и для общего пользования, и для других DG. Во время нулевого энергопотребления DG компенсируют, увеличивая их генерацию.

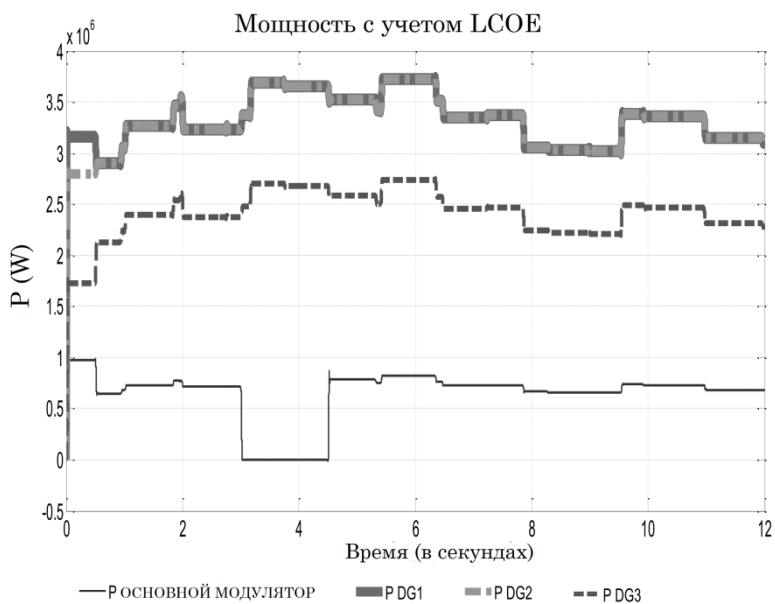


Рисунок 17.15. Питание подается на 24 часа: 3 этап

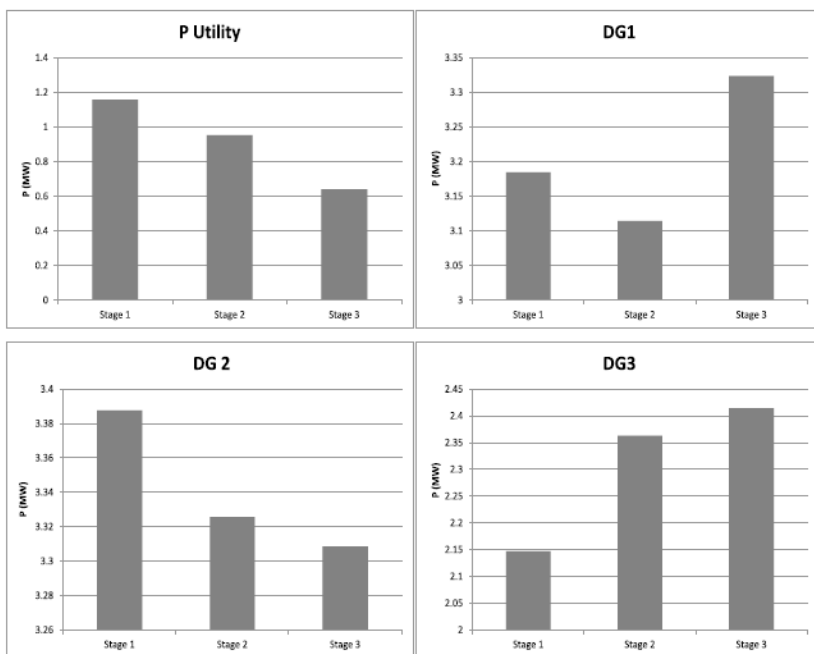


Рисунок 17.16. Средняя мощность, подаваемая каждым из энергоблоков для каждой ступени за 24 часа

На рисунке 17.16 представлена группа из четырех столбчатых диаграмм, показывающих мощность, подаваемую четырьмя генерирующими блоками для каждого этапа. Наблюдая за средней мощностью, поставляемой генерирующими блоками на каждом из этапов, можно сделать вывод, что приложение экономической диспетчеризации направлено на выбор генерирующего блока, способного давать нагрузки по самой низкой цене. Уменьшение мощности, генерируемой одним или несколькими генерирующими блоками, компенсируется другими более экономически целесообразными генерирующими блоками.

17.6.2. Ежедневная стоимость генерации для различных этапов

В этом разделе приведены общие затраты на генерацию каждого из источников за один день. Стоимость производства энергии является хорошей основой для эффективности приложения, разработанного в среде FB.

Таблица 17.2 показывает, что значения в столбце общих затрат уменьшаются с развитием последовательных этапов. Общая сумма представляет собой сумму затрат на энергию отдельных энергоблоков и

инженерных сетей. Это указывает на эффективность каждого приложения балансировки нагрузки. Сокращение в этом приложении между стадиями 3 и 1 составляет приблизительно 7,7 %. Это экономия, достигнутая при рассмотрении этих размеров генерирующих единиц. Для более крупных энергоблоков экономия будет пропорциональной. Эти приложения разрабатываются как глобальные решения. Следовательно, в зависимости от местоположения, индивидуальная стоимость генерации (DG 1, DG 2, DG 3 или утилиты) будет варьироваться. Это видно из графического представления затрат отдельных энергоблоков на рисунках с 17.17 по 17.20, приведенных ниже.

Таблица 17.2. Стоимость генерации за 24 часа в долларах США в день

	Утилита	DG 1	DG 2	DG 3	Общая стоимость
Стадия 1	03741	04441	08630	01938	18750
Стадия 2	03071	04340	08471	02133	18015
Стадия 3	02060	04674	08388	02170	17292

Тем не менее общая цель достижения уменьшенной общей стоимости достигнута. Снижение мощности, подаваемой одним генерирующим блоком, компенсируется увеличением генерации других генераторов или утилиты. Этот аспект также отражен в стоимости производства энергии для этих DG. Для рассматриваемого сценария уровень затрат на генерацию для DG ниже, чем стоимость электроэнергии утилиты. Следовательно, происходит снижение мощности, подаваемой для коммунальной сети. Это, в свою очередь, отражает стоимость электроэнергии, распределяемой коммунальной сетью за один день. Кроме того, на основе LCOE отдельных генерирующих блоков будет происходить увеличение или уменьшение стоимости операции, базирующейся на алгоритме экономической диспетчеризации.

Общая стоимость работы в течение 24 часов для всей сети показана на рисунке 17.21. Наблюдается постепенное снижение общей стоимости производства энергии за день.

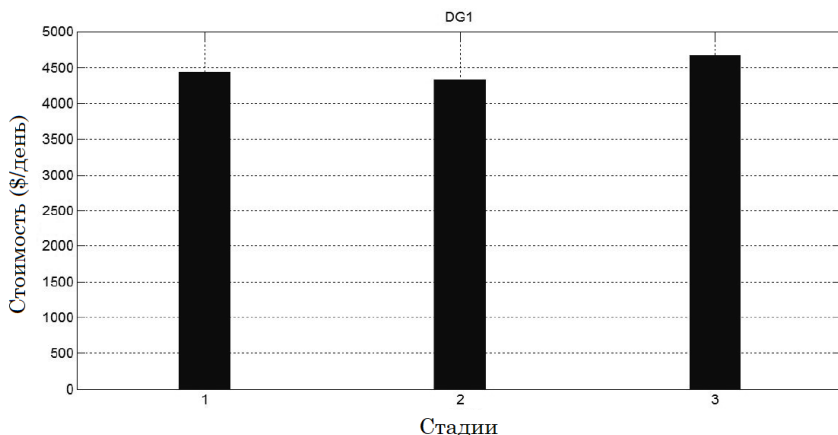


Рисунок 17.17. Стоимость производства энергии DG1 за 24 часа

Эффективность каждой схемы может быть лучше продемонстрирована путем сравнения процентной экономии, достигнутой приложениями балансировки нагрузки на всех этапах, с нормальной работой на этапе 1. Это показано на рисунке 17.22.

17.6.3. Выводы

Наблюдая за результатами для каждого этапа, можно заключить, что при разработке каждого этапа с использованием концепции FB происходит снижение стоимости генерации. Приложение экономической диспетчеризации уникальным образом сочетает в себе функции прогнозирования нагрузки, экономического моделирования систем возобновляемой энергии и концепции LCOE при выполнении приложения экономической диспетчеризации. Преимущество использования LCOE для финансовых расчетов заключается в том, что входные переменные уравнения LCOE моделируются с использованием доступных данных, что позволяет более точно отражать неопределенность затрат, связанную с конкретной технологией генерации. Приложение для экономической диспетчеризации является глобальным решением, и в зависимости от местоположения, мощности и технологии производства результаты могут отличаться. Экономическая диспетчеризация может быть проведена для того, чтобы направлять политиков, регуляторов и инвесторов в принятии обоснованных финансовых решений для оптимальной работы энергосистемы. Это может помочь операторам коммунальных услуг принять решение о политике, касающейся взаимодействия сетей энергосистем. Индекс LCOE обеспечивает платформу для оценки различных возобновляемых источников энергии, обеспечивая систему экономического ранжирования, которая учитывает начальные капитальные затраты, стоимость OandM и воздействие возобновляемых источников энергии на окружающую среду.

Политики постоянно прилагают усилия для создания взаимовыгодной среды для взаимодействующих инженерных сетей. Это может быть достигнуто путем разработки и внедрения интеллектуальных приложений для управления, которые дают регулирующим органам экономическую выгоду. Этот проект сочетает в себе аспекты интеллектуального управления и экономической выгоды за счет внедрения управления FB для разработки приложений балансировки нагрузки, которые оказываются экономически выгодными.

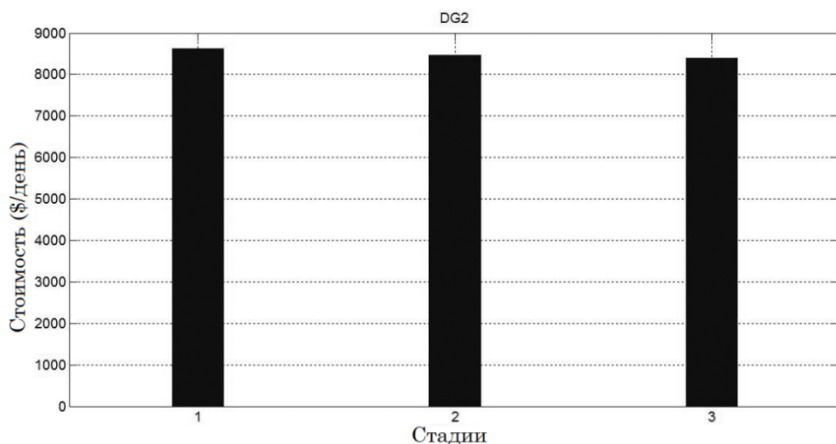


Рисунок 17.18. Стоимость производства энергии DG2 за 24 часа

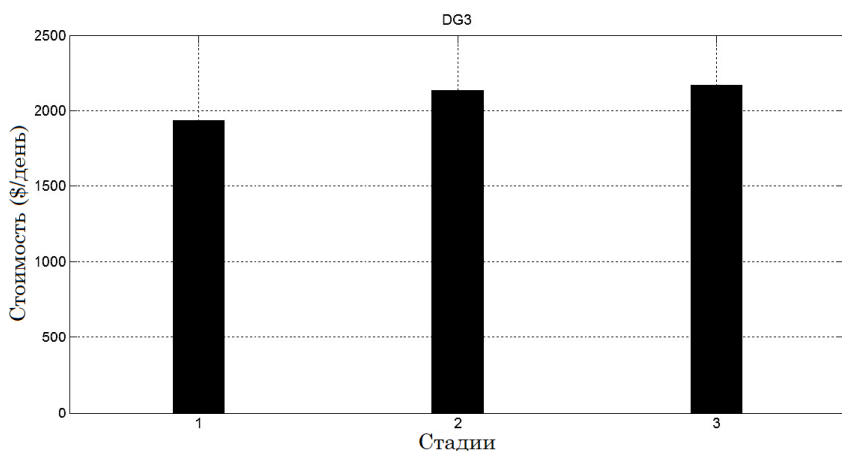


Рисунок 17.19. Стоимость производства энергии DG3 за 24 часа

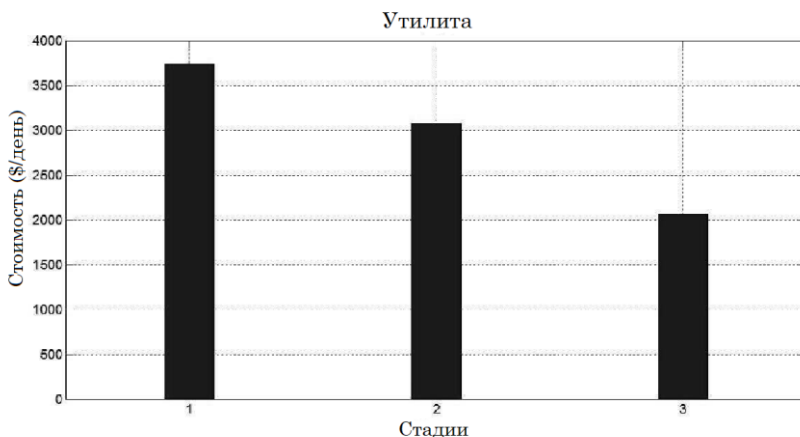


Рисунок 17.20. Затраты на производство энергии общего назначения за 24 часа

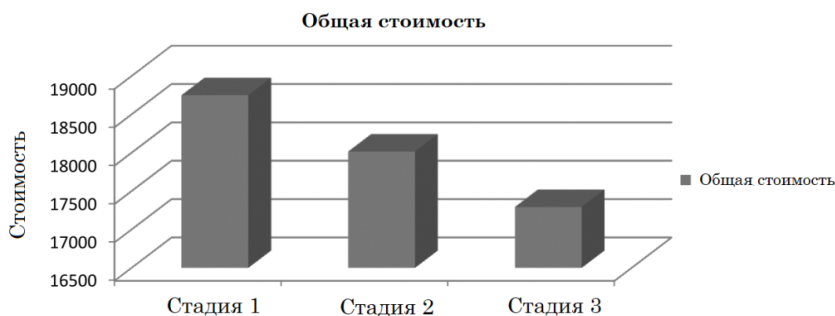


Рисунок 17.21. Общая стоимость производства энергии за 24 часа

В заключение можно сказать, что IES 61499, безусловно, является эффективным и надежным методом для разработки приложений для управления в интеллектуальной сети. Можно также сказать, что любой сложный процесс можно рассматривать как совокупность простых функциональных модулей. Метод совместного моделирования, использованный в этой работе, можно применять, чтобы показать, что FB можно рассматривать как концепцию, которая может быть реализована с использованием языка программирования, поддерживающего объектно-ориентированное программирование. Следует тщательно изучить возможность применения IES 61499 для других управляющих приложений, таких как реакция спроса и управление возбуждением генератора в интеллектуальной сети.

17.6.4. Дальнейшая работа

Основное внимание в этой главе было уделено контролю распределенной генерации сети. Дальнейшая работа может включать в себя контроль нагрузки в сети, также называемый управлением спроса, включая организацию очереди нагрузки. Это расширяет область применения IEC 61499 в ответ на существующую потребность. Также модель учитывала один DG в каждой микросети. Будущее расширение будет заключаться в изменении поведения экономической отправки для размещения нескольких DG в одной и той же микросети.

Список благодарностей

Искренняя благодарность Jim Christensen, Alois Zoitl и другим членам исследовательской группы IEC 61499 за их постоянную поддержку и руководство.

Библиография

- [1] E. Chowdhury and S. Rahman. A review of recent advances in economic dispatch. *IEEE Transactions on Power Systems*, 1990.
- [2] T. Hussain and G. Frey. Migration of a PLC controller to an IEC 61499 compliant distributed control system: Hands-on. In *International Conference on Robotics and Automation*, pages 3984–3989. IEEE, April 2005.
- [3] F. Katiraei. Micro-grid autonomous operation during and subsequent to islanding process. In *Power Engineering Society General Meeting*, volume 2. IEEE, June 2004.
- [4] D. Kothari and I. Nagrath. *Modern Power System Analysis*. Tata Mc-Graw Hill., 2011.
- [5] R. Lewis. *Modeling Control Systems Using IEC 61499: Applying Function Blocks to Distributed Systems*. The Institution of Electrical Engineers, 2001.
- [6] National Renewable Energy Laboratory, Colorado. *A Manual for the Economic Evaluation of Energy Efficiency and Renewable Energy Technologies*. U.S. Department of Energy, Golden, Colorado, March 1995.
- [7] S. Patil, V. Vyatkin, and B. McMillin. Implementation of FREEDM smart grid distributed load balancing using IEC 61499 function blocks. In *Industrial Electronics Society, 39th Annual Conference of IEEE*, pages 8154–8159, Nov 2013.
- [8] R. Pichs-Madruga, O. Edenhofer, and Y. Sokona. Renewable energy sources and climate change mitigation: Special report of the Intergovernmental Panel on Climate Change. Technical report, Cambridge University Press, 2011.

- [9] U.S. Energy Information Administration. Levelized cost and levelized avoided cost of new generation sources. Technical report, Office of Integrated and International Energy Analysis, U.S. Department of Energy, Washington, 2014.
- [10] J. Vasudevan, K. Sekkappan, K. Yashwant, and K. S. Swarup. Research issues in a smart grid-application to automation, distributed energy sources and demand response. *IIT, Madras*, 2012.
- [11] C.-H. Yang, G. Zhabelova, C.-W. Yang, and V. Vyatkin. Cosimulation environment for event-driven distributed controls of smart grid. *IEEE Transactions on Industrial Informatics*, 2013.
- [12] G. Zhabelova, S. Patil, C.-W. Yang, and V. Vyatkin. Smart grid applications with IEC 61499 reference architecture. In *Industrial Informatics 11th IEEE International Conference on*, pages 458–463, July 2013.
- [13] G. Zhabelova and V. Vyatkin. Intelligent logical nodes of IEC 61850 and IEC 61499 for multi-agent smart grid automation. *IEEE Transactions on Industrial Electronics*, 2011.
- [14] G. Zhabelova, V. Vyatkin, and N. Nair. Standards-based intelligent fault management system for FREEDM green hub model. In *IEEE International Conference on Industrial Electronics*, 2011.

IV часть

Примеры лабораторной автоматизации

Роботы для создания общего рабочего пространства: применение IEC 61499

Matthias Plasch

PROFACTOR GmbH

Gerhard Ebenhofer

PROFACTOR GmbH

Michael Hofmann

PROFACTOR GmbH

Martijn Rooker

PROFACTOR GmbH

Sharath Chandra Akkaladevi

PROFACTOR GmbH

Andreas Pichler

PROFACTOR GmbH

СОДЕРЖАНИЕ

18.1 Введение	442
18.2 Аналогичные работы	443
18.2.1 Платформы интеграции	443
18.2.2 Интегрированные технологии	445
18.3 Описание автоматизированной системы	446
18.3.1 Расположение оборудования	446
18.3.2 Эксперименты и демонстрационный вариант использования	447
18.3.3 Необходимые функциональные компоненты системы	448
18.3.3.1 Система 3D Vision	448
18.3.3.2 Планировщик манипуляций: Планирование путей без столкновений	449
18.3.3.3 Модуль для управления роботом	449
18.4 Подход к разработке	449
18.4.1 Интеграция неоднородных компонентов программного обеспечения	450

18.4.2 Поточковые типы функционального блока	451
18.4.2.1 Распределение сервисов	452
18.4.2.2 Сервисный контроллер с ленивым шаблоном подключения	454
18.4.3 Шаблоны проектирования приложений	455
18.4.3.1 Гибкий сервисный шаблон параметризации	455
18.4.3.2 Сценарии LUA FB для быстрого прототипирования алгоритма и координации FB	456
18.5 Полученная архитектура системы	461
18.6 Сводка и выводы	465

18.1. Введение

Будущие производственные системы характеризуются высокораспределенной и модульной архитектурой, которая должна быть легко адаптируемой для удовлетворения потребностей клиентов и рынка. Эти системы обычно состоят из разнородных программных и аппаратных компонентов, основанных на разных платформах, что приводит к высоким уровням сложности при разработке и интеграции системы. Сложные роботизированные сборочные системы обладают многими из ранее упомянутых характеристик. Чтобы справиться с этими трудностями, мы представляем набор концепций, которые помогают системной интеграции и разработке приложений на основе функционально независимых и высокомодульных компонентов программного и аппаратного обеспечения. Цель состоит в том, чтобы сократить время разработки и повысить стабильность приложений, опираясь на проверенные, повторно используемые программные компоненты и подходы к разработке. Эта работа объединяет опыт, полученный в различных промышленных и исследовательских проектах в области робототехники, и основана на предыдущих работах [7, 25, 26].

Представленные концепции были применены к сложной роботизированной системе, состоящей из двух роботов, совместно использующих рабочее пространство. Данная система была разработана для совместного решения задачи сборки. Для системной интеграции и моделирования приложений использовалась структура 4DIAC для распределенной автоматизации и управления [34] в соответствии с промышленным стандартом IEC 61499 [13]. В разделе 18.2 представлен современный обзор системной интеграции, языков, сценариев и технологий, используемых в представленных подходах к разработке. Раздел 18.3 описывает вариант использования приложения и роботизированную систему с общим рабочим пространством, а также ее системные компоненты; подходы к развитию, представленные в разделе 18.4, были также реализованы. В разделе 18.5 шаблоны разработки применяются к примеру приложения. Раздел 18.6 обобщает и завершает работу.

18.2. Аналогичные работы

В этом разделе дается обзор аналогичных работ. Первая часть посвящена платформам, которые поддерживают интеграцию разнородных сервисов в области робототехники и промышленной автоматизации. Во второй части представлены вспомогательные технологии и программные компоненты для концепций, представленных в этой главе.

18.2.1. Платформы интеграции

Различные подходы нацелены на интеграцию сервисов, базирующихся на разных платформах, в систему, основанную на концепциях сервис-ориентированной архитектуры (SOA). Компоненты, разработанные SOA, характеризуются функциональной доступностью и призваны стимулировать повторное использование и развитие существующих функций для улучшения сервисов [8]. Основными преимуществами SOA являются трактовка сервисов как независимых ресурсов и их доступность без необходимости специальных знаний о платформах [37]. В области робототехники Конончук и соавторы [17] разработали протокол взаимосвязи RoboCop, чтобы объединить цепочки и программы робототехнических инструментов в структуру масштабируемой системы управления. Структуры, основанные на SOA-управляемой разработке для интеграции устройств (включая их датчики и исполнительные механизмы) в роботизированную систему, предложены в [9] и [37]. В обоих случаях устройства рассматриваются как повсеместные сервисы. Ян и Ли [37] дополнительно используют иерархический подход для выражения сложных действий (компонентов) в виде композиций простых действий. Сравнение реализованных подходов SOA, адаптированных к роботизированному программированию рабочих ячеек, было выполнено Veiga и соавторами [35]. Основное внимание здесь уделяется методам описания услуг и их действий. Анализ показал, что разработка сервисных компонентов на основе SOA в сочетании с графической средой разработки значительно сокращает усилия по интеграции [35].

В области робототехники Orocos является одной из наиболее зрелых и устоявшихся программных платформ с открытым исходным кодом [21]. Более поздняя работа расширяет эту ориентированную на поток данных инфраструктуру с использованием технологии Java, что приводит к созданию JOrocos, находящемуся под влиянием концепций архитектуры сервисных компонентов (SCA) [31], и дополнительно устраняет разрыв между подходами на основе компонентов (например, используя функциональность управления в реальном времени внутри функционала) и современных технологий информационных систем (например, с использованием самых современных инструментариев Java для графических пользовательских интерфейсов) [4].

YARP (еще одна роботизированная платформа) [19] – это библиотека программного обеспечения с открытым исходным кодом, которая позволяет

разрабатывать программное обеспечение на основе модулей для роботизированных платформ. Модули YARP работают как отдельные исполняемые файлы и обмениваются данными с помощью расширяемого набора протоколов связи, таких как TCP, UDP и MPI. Шаблоны связи, предоставляемые YARP, включают в себя функциональные блоки PUBLISH и SUBSCRIBE, а также архитектуры КЛИЕНТА и СЕРВИСА. YARP был запущен в 2002 году для поддержки разработки программного обеспечения для гуманоидных роботов, таких как iCub [20].

Чтобы упростить крупномасштабную интеграцию программного обеспечения, ROS (операционная система робота) [30] была разработана в 2007 году. Она была разработана не для конкретного целевого приложения (как раньше был широкий спектр сред), а для поддержки философии модульных инструментов [28]. Стремительно растущее сообщество ROS доказывает успех принципов проектирования ROS, хотя оно в основном ориентировано на платформы Linux и вряд ли предоставляет информацию, касающуюся подхода к разработке.

В этом контексте проект BRICS (Best Practices in Robotics) предложил первоначальные руководящие указания по проектированию компонентов, что привело к получению лучших интегрируемых компонентов [5] в соответствии с подходом модельно-ориентированного проектирования (MDE). Недавно были опубликованы данные об ориентированном на повторное использование процессе разработки робототехнических систем на основе компонентов [3] в соответствии с разработанной философией проектирования с использованием интегрированной среды разработки BRICS (BRIDE) [5], представляющей собой цепочку прототипных программных инструментов. Она включает графический интерфейс пользователя для поддержки его подхода к разработке роботизированных систем.

Исследуя область промышленной автоматизации, в которой конкретные встраиваемые устройства все больше продвигают технологию в направлении распределенных систем управления, а также в сторону более ориентированных на обслуживание архитектур [16], можно столкнуться с инициативой 4DIAC, основанной на стандарте IEC 61499 [13], которая с 2007 года предлагает систему с открытым исходным кодом для распределенной промышленной автоматизации и управления [34]. Структура 4DIAC состоит из среды автоматизации и управления, соответствующей стандарту IEC 61499, для распределенных систем.

В настоящее время доступны модульная среда выполнения, интегрированная среда проектирования, библиотека функциональных блоков (FB) и набор примеров проектов. Среда выполнения FORTE поддерживает операционные системы Windows, все совместимые с Posix системы, NET+OS 7, eCos и VxWorks. Кроме того, поддерживаются несколько плат оценки (например, Digi Connect ME, Pandaboard) и системы PLC (например, Bachmann M1, Siemens EC31 с модулями I/O S7).

Система 4DIAC была применена в разных областях. Примером является управление перевернутым маятником, в котором контроллер пространства

состояний был динамически заменен более сложным контроллером пространства состояний [38]. Австрийский технологический институт представил пример применения 4DIAC для приложений интеллектуальных сетей, где IEC 61850 был интегрирован в FB IEC 61499 [33]. Наш подход к разработке, описанный в следующем разделе, основан на стандарте IEC 61499, а не на его предшественнике IEC 61131. Это связано с тем, что парадигма программирования IEC 61499 больше подходит для распространения управляющих данных на автономные устройства, отходящие от централизованных структур управления. Более того, управляемая событиями структура исполнения FB облегчает разработку приложений для координации функционально-независимых компонентов, как того требует предложенная методология интеграции.

18.2.2. Интегрированные технологии

Языки сценариев используются в различных областях, особенно для быстрого создания прототипов и подключения программного обеспечения и веб-приложений. Результирующая функциональная совместимость взаимосвязанных гетерогенных систем или языков программирования может рассматриваться как основное преимущество использования языков сценариев [22]. LUA – это легковесный и встраиваемый язык сценариев на основе C, который в основном используется в качестве языка расширения для программных компонентов [14]. Области применения LUA – видеоигры, мобильные и встроенные устройства и приложения автоматизации. В качестве примера для промышленного применения Girder [27] – программный инструмент для промышленной и домашней автоматизации – использует LUA в качестве базового языка. Интеграция программного обеспечения облегчена предоставленным LUA C-API. Выполнение LUA ускоряется с помощью LuaJIT [23], библиотеки интерпретаторов «точно в срок», которая расширяет пакет базовой библиотеки LUA. Краткий обзор многих известных языков сценариев, включая интересные факты об их истории, представлен в [2].

ZeroMQ (ZMQ) – это высокопроизводительная библиотека асинхронных сообщений, используемая в областях распределенных вычислений и сред параллелизма. Конечные точки связи основаны на общих программных гнездах и поддерживают такие протоколы, как межпроцессный, внутрипроцессный, TCP и многоадресные. Распределение сообщений основано на очереди, которая не требует, чтобы система посредника выполнялась должным образом. На основе базовых шаблонов взаимодействия, таких как PUBLISH–SUBSCRIBE, запрос–ответ и PUSH–PULL, в руководстве пользователя описано огромное разнообразие подходов к реализации среды параллелизма [11]. ZMQ – это проект с открытым исходным кодом, и его интеграция основана на интерфейсе прикладного программирования (API), который доступен на широком диапазоне языков программирования [11].

CANDELOR SDK [10] – это библиотека программного обеспечения для интерпретации трехмерных сцен. Она содержит алгоритм локализации объектов для обнаружения трехмерных объектов CAD в облаке точек. CANDELOR работает на полных трехмерных облаках точек. Конфигурация выполняется автоматически на основе модели CAD. CANDELOR – это масштабируемое решение для локализации объектов. Оно может использоваться для распознавания в реальном времени, а также для высокоточного распознавания в автономном режиме.

18.3. Описание автоматизированной системы

В этом разделе дается обзор структуры оборудования роботизированной системы и ее функциональных программных компонентов. Роботизированная система используется для демонстрации применимости концепций, представленных в разделе 18.4. Эти концепции были разработаны для поддержки автоматизации и координации нескольких роботов, выполняющих задачу сборки.

18.3.1. Расположение оборудования

Испытанная роботизированная система состоит из двух независимых промышленных манипуляторов типа Stäubli RX130B и Stäubli TX90L [29], каждый из которых обеспечивает шесть степеней свободы. Каждый робот снабжен контроллером. Контроллер роботов серии CS8 в среде VAL3 управляет роботами для запуска различных приложений. Для обеспечения возможности манипулирования объектами каждый робот оснащен сервоэлектрическим параллельным захватом с двумя пальцами PG70 от Schunk [32]. Эти захваты обеспечивают высокоточное управление силой захвата и длительный ход (около 34 мм на палец). Также предусмотрена активная обратная связь о расстоянии между захватывающими пальцами, чтобы помочь роботу распознать успешное сцепление. Для обеспечения возможности получения информации о текущей сцене система оснащена недорогим датчиком глубины типа Asus Xtion PRO [1]. Полученные данные сцены представляются в виде изображения глубины на основе трехмерного облака точек. Обзор настройки оборудования приведен на рисунке 18.1.

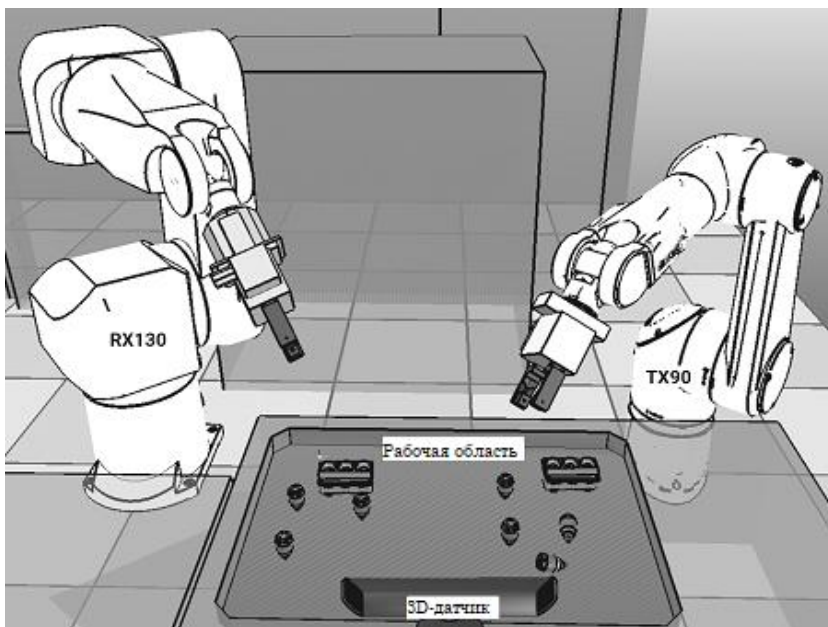


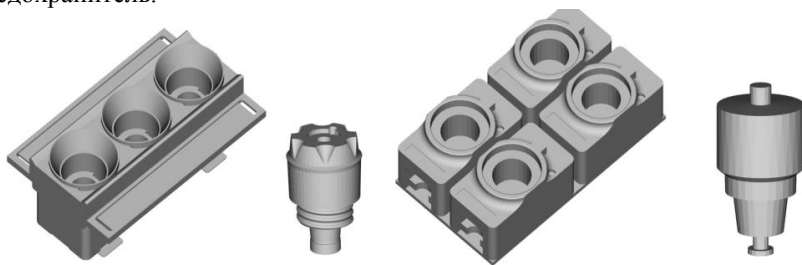
Рисунок 18.1. Аппаратная настройка роботизированной системы

18.3.2. Эксперименты и демонстрационный вариант использования

Настройка системы, описанная в разделе 18.3.1, была выбрана для выполнения задачи сборки в нескольких вариантах. Основная цель – собрать блоки предохранителей, вставив несколько предохранителей в соответствующие блоки. При использовании двух различных типов конфигураций объекта (обозначенных как тип А и тип В), представленных на рисунке 18.2, будет показана гибкость и адаптивность системы. В дополнение к выполнению задачи сборки роботизированная система должна была соответствовать условиям, приведенным ниже.

- Роботы должны действовать параллельно для выполнения задачи, разделяя рабочее пространство и избегая столкновений друг с другом и объектами.
- Предохранители могут быть расположены в рабочей области в произвольных положениях (в перевернутом, лежа или стоя), но их необходимо размещать на лотке рабочей области (см. рисунок 18.1).
- В случае, если робот не может вставить предохранитель из-за своей первоначальной ориентации, роботы должны сотрудничать, чтобы завершить вставку. Например, робот А может выбрать лежащий предохранитель. Для

установки робот А передает предохранитель роботу В, и он может вставить предохранитель.



Блок для предохранителей: Предохранитель: Тип А
Тип А

Блок для предохранителей: Предохранитель: тип В
Тип В

Рисунок 18.2. Типы объектов сборки

18.3.3 Необходимые функциональные компоненты системы

В этом разделе дается обзор функциональных компонентов системы, которые необходимы для выполнения задачи сборки, как описано в разделе 18.3.2.

18.3.3.1 Система 3D Vision

Система трехмерного видения основана на распознавании и локализации трехмерных объектов на основе CAD-моделей объектов, представляющих для нас интерес. Этот процесс основан на сопоставлении на основе признаков, при котором должна быть обнаружена корреляции полученного изображения и предварительно обработанных CAD-моделей объектов. Функциональность реализуется алгоритмом рандомизированной глобальной локализации объектов (RANGO) [10], который является расширением алгоритма согласования случайной выборки (RANSAC) [24]. Процедура RANGO возвращает список матриц преобразования, в то время как каждая матрица описывает местоположение и ориентацию обнаруженного объекта в рабочей области. В дополнение к локализации объекта модуль зрения обеспечивает дополнительные функции, перечисленные ниже.

- Обработка ложноположительных обнаружений путем отклонения, например, плавающих объектов.
- Расчет потенциальных мест размещения и фиксации для сборочных операций.

В качестве предварительного условия датчик глубины должен быть откалиброван относительно рабочей области робота для использования той же системы координат, что и две системы промышленного робота.

18.3.3.2 Планировщик манипуляций: Планирование путей без столкновений

Результаты работы модуля визуального контроля используются для планирования и расчета траекторий манипуляций робота без столкновений, что позволяет обрабатывать обнаруженные объекты. Основываясь на предопределенных захватах, а также точках отложения на CAD-модели объектов, планировщик манипуляций определяет, как можно захватить объект. Алгоритм планирования пути основан на подходе быстро исследуемых случайных деревьев (RRT) [18] [6]. Чтобы обеспечить взаимодействие обоих роботов без столкновений, в рабочую область динамически вводится виртуальная стена для разделения рабочего пространства на две области. Используя ограниченное рабочее пространство, первый робот пытается найти путь без столкновений для захвата и манипулирования объектами. Из двух роботов один считается ведущим и ему разрешено сдвигать стену, пока он не найдет действительный путь без столкновений. Другой робот (ведомый) может использовать оставшееся рабочее пространство для планирования пути.

В зависимости от выбранной точки захвата компонент планировщика манипуляций решает, может ли быть рассчитан путь передачи или необходима передача контроля. Для выполнения передачи объекта запланированы пути без столкновений к предварительно определенным позициям передачи. В зависимости от текущего положения роботов пути реализуются параллельно или последовательно, чтобы избежать столкновений роботов.

18.3.3.3. Модуль для управления роботом

Для управления промышленными роботами устанавливается соединение ModbusTCP для отправки команд и получения информации о состоянии от контроллеров роботов Stäubli. Таким образом, возможна последовательная отправка набора совместных конфигураций, что приводит к траектории движения. Двухкулачковые захваты типа Schunk Power-Cube, используемые для соответствующих операций, управляются через их интерфейс CAN, используя предоставленный API PowerCube. Используя движения захвата на основе силы (тока), можно интерпретировать активную обратную связь от захватов, которая используется для определения успешности операции.

18.4. Подход к разработке

Одной из основных задач разработки системы является создание расширяемой системной архитектуры, которая включает в себя гетерогенные и функционально независимые компоненты аппаратного и программного обеспечения. Эти компоненты основаны на различных технологиях и средах, что приводит к увеличению сложности при системной интеграции и

разработке приложений. Подход к разработке приложений для описанной системы групп роботов, которая выполняет задачи сборки, основан на стандарте IEC 61499. Предыдущая работа [7] показала применимость основных концепций IEC 61499 для координации разнородных компонентов системы для достижения определенного поведения системы. В этом разделе дается обзор концепций, используемых для интеграции компонентов гетерогенной системы. Кроме того, будут представлены шаблоны и методы проектирования, которые значительно помогли в разработке сложных приложений управления. Концепции, обсуждаемые в этом разделе, будут применены к примерной роботизированной сборочной системе.

18.4.1. Интеграция неоднородных компонентов программного обеспечения

Функциональные модули, описанные в разделах 18.3.3.1 и 18.3.3.2, являются внешними высокопроизводительными программными компонентами, которые необходимо интегрировать для разработки приложений с использованием цепочки инструментов 4DIAC. Основанная на C++ среда исполнения FB FORTE может быть расширена внешними программными библиотеками. Это достигается с помощью системы сборки FORTE, основанной на CMake, для включения необходимых файлов библиотеки и файлов исходного кода. Включенные библиотечные функции и алгоритмы могут быть отнесены к типу функционального блока. На рисунке 18.3 изображена схема, поясняющая процесс расширения, который обобщен ниже.

- Используя редактор типов функциональных блоков 4DIAC, создается интерфейс сервисного FB. Кроме того, указаны точки входа для алгоритмов, которые должны вызываться FB. Диаграмма управления выполнением (ECC) может быть определена, чтобы гарантировать, что сервисные функции могут вызываться только в определенной последовательности, например при инициализации перед выполнением алгоритма.

- На следующем шаге генерируются заглушки исходного кода для типа FB и включаются в FORTE путем соответствующей настройки системы сборки. Алгоритмы, которые также вызывают функции из включенных библиотек, теперь реализованы в сгенерированных исходных файлах.

- На последнем шаге FORTE необходимо перекомпилировать, чтобы включить функции внешней библиотеки.

Поскольку эти алгоритмы требуют больших вычислительных ресурсов, требовалось решение, которое позволяет избежать нарушения поведения во время выполнения по умолчанию. Для решения этой проблемы будет представлен тип функционального блока с резьбой, как описано в разделе 18.4.2.

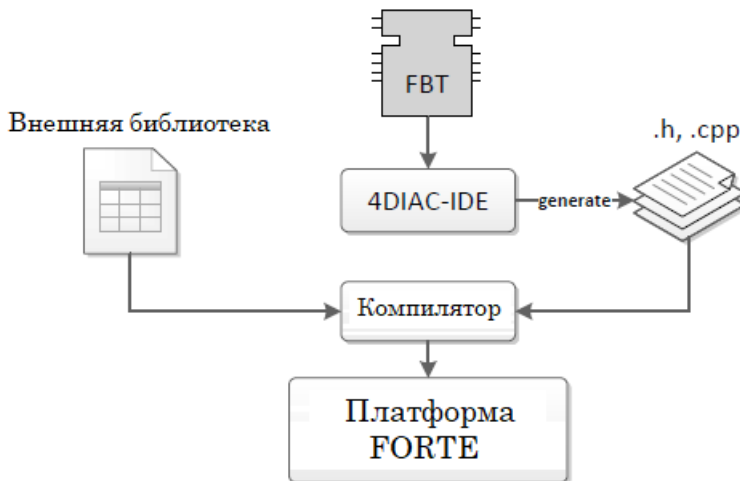


Рисунок 18.3. Расширение исходного кода FB с помощью функций внешней библиотеки

В качестве альтернативы гибкая коммуникационная инфраструктура FORTE может быть расширена для установления соединения с внешним компонентом или упрощения распределенных архитектур управления. В дополнение к протоколу связи ASN.1, как описано в профиле соответствия для демонстрации осуществимости [13], FORTE в настоящее время поддерживает полевую шину и протоколы связи ModbusTCP, Ethernet Powerlink и OPC-DA. Однако, особенно при работе с большими объемами данных (например, данными, полученными с помощью 3D-датчиков), которые необходимо передавать в распределенных системах, требуются эффективные концепции обмена сообщениями. Следовательно, реализована поддержка ZMQ, которая является библиотекой асинхронного обмена сообщениями. Она подходит для использования в качестве масштабируемого протокола обмена сообщениями в архитектурах распределенных систем. Протокол специально разработан для поддержки параллельных вычислений без ограничения топологии сети; при этом он имеет очень низкие накладные расходы на связь.

18.4.2. Потокковые типы функционального блока

Выполнение событий средой исполнения FB FORTE реализуется отдельным потоком выполнения событий. Если событие возникает, оно помещается в очередь событий, которая обрабатывается непрерывно. Поскольку FORTE необходимо реагировать на вызванные события как можно быстрее, основной поток не должен блокироваться во время выполнения алгоритма FB. Особенно в случае алгоритмов, которые требуют больших

вычислительных ресурсов, поток выполнения события, скорее всего, будет заблокирован.

Чтобы предотвратить блокировку или приостановку потока выполнения событий из-за трудоемких алгоритмов, для вызова таких функций следует использовать типы потоковых функциональных блоков. Потоковые FB запускают другой рабочий поток, который отвечает за выполнение алгоритмов FB. После выполнения алгоритма рабочий поток приостанавливается и возобновляется при запуске следующего события выполнения алгоритма. На рисунке 18.4 показан пример интерфейса с винтовым типом функционального блока. Событие ввода REQ_LONG_RUNNING_WORK используется для запуска ресурсоемкого алгоритма. Событие соответственно выводит сигналы STARTED_LONG_RUNNING_WORK и FINISHED_LONG_RUNNING_WORK о том, что алгоритм был запущен и завершил выполнение.

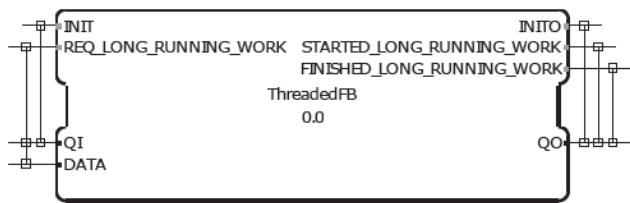


Рисунок 18.4. Пример интерфейса винтового функционального блока

18.4.2.1. Распределение сервисов

Сложные алгоритмы, такие как операции обработки изображений, обычно требуют обширных вычислительных ресурсов. Чтобы обеспечить высокую общую производительность, эти алгоритмы могут быть распределены по сети. На рисунке 18.5 показан пример последовательности ресурсоемких алгоритмов. FB моделируют вызов алгоритма видения и алгоритма планирования пути. Данные датчика служат входом для алгоритма видения.

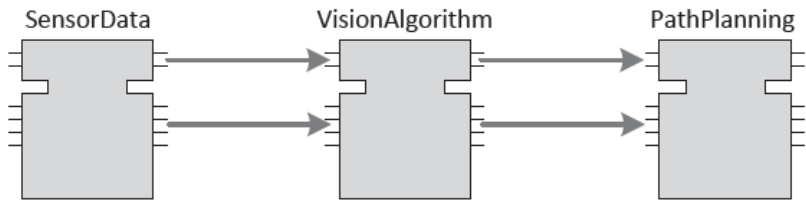


Рисунок 18.5. Приложение с алгоритмами, инкапсулированными как FB

Извлечение алгоритма FB в сервис может быть сделано путем вставки клиентских и серверных FB, которые действуют как удаленный вызов

процедуры. На рисунке 18.6 изображен извлеченный алгоритм как услуга. Это позволяет сопоставить сервис с другим ресурсом. На этом этапе для моделирования распределенного приложения требуются только функции IEC 61499.

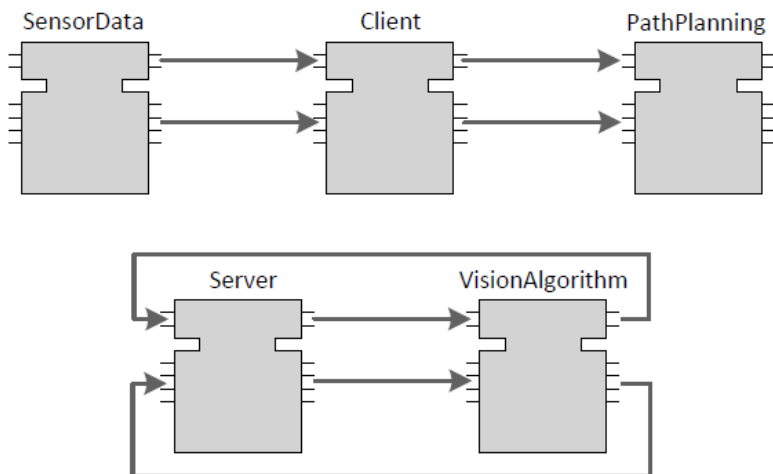


Рисунок 18.6. Инкапсуляция в работу алгоритмов FB

Чтобы использовать преимущество этой концепции, алгоритм (сервис) должен работать на нескольких серверах. Каждый клиент может получить доступ к алгоритму видения. Основная особенность заключается в том, что клиент не знает, какой сервер фактически был вызван. Для этого в среду выполнения IEC 61499 были добавлены клиентский сервер и дилер маршрутизатора с шаблоном планировщика, предоставленным ZMQ [12]. В архитектуре КЛИЕНТ–СЕРВЕР ZMQ автоматически распределяет все запросы на серверы с циклическим планированием. Этот шаблон поддерживает добавление большего количества клиентов, но циклическое планирование не гарантирует равномерного распределения рабочей нагрузки. На рисунке 18.7 показан подход КЛИЕНТ–СЕРВЕР. Более сложный шаблон – это шаблон дилера маршрутизатора, где центральный брокер (посредник), как показано на рисунке 18.8, распределяет все запросы на обслуживание для доступных серверов. Основным преимуществом является то, что КЛИЕНТЫ и СЕРВЕР могут подключаться к брокеру и отключаться от него. Посредник должен иметь дело с прерванными запросами и перепланировать запросы, если сервер отключен. Балансировщик нагрузки может быть изготовлен на заказ в соответствии с требованиями и доступными данными. Основным недостатком шаблона является то, что у брокера есть единая точка отказа.

18.4.2.2. Сервисный контроллер с ленивым шаблоном подключения

Как объяснено в разделе 18.4.2.1, инкапсуляция алгоритмов как сервиса является эффективным шаблоном разработки. Основным недостатком подхода является результирующая цепочка инициализации сервисных и клиентских функциональных блоков. В этой главе будет представлено расширение этого шаблона для предотвращения длинных цепочек инициализации путем добавления функционального блока с функциями для мониторинга и установления соединения.

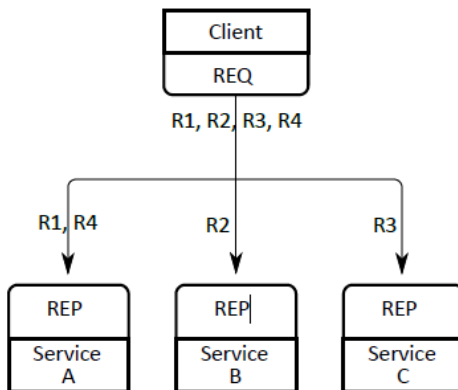


Рисунок 18.7. Архитектура КЛИЕНТ–СЕРВЕР с циклическим распределением

Основная идея заключается в том, чтобы устанавливать соединение со службой только при необходимости, что называется ленивым шаблоном инициализации. Это означает, что соединение установлено, только если запрос запущен. При этом обработка будет осуществляться функциональным блоком *servicecontroller*. Функциональный блок отслеживает текущее состояние соединения и решает, должно ли соединение быть установлено или запрос должен быть отброшен из-за ожидающего запроса. Кроме того, функциональный блок отслеживает текущий запрос. Если время ожидания истекло, соединение будет отключено и восстановлено, если возникнет новый запрос. Логика показана как ECC на рисунке 18.9.

Типичное использование показано на рисунке 18.10, который изображает составной функциональный блок, который инкапсулирует службу, как описано в разделе 18.4.2.1, а также введенный *servicecontroller*. Поскольку *servicecontroller* обрабатывает процедуру инициализации, выделенные события INIT и INITO не нужны. Пример включает в себя дополнительный функциональный блок *E_DELAY* для мониторинга соединения и запросов. Если соединение или запрос превышает указанное время ожидания, соединение закрывается, и для состояния QO будет установлено значение «false».

Шаблон может быть расширен за счет дополнительных функций, таких как автоматическое отключение соединения (если оно не требуется в течение определенного времени) или автоматическая повторная попытка запроса в случае сбоя.

18.4.3. Шаблоны проектирования приложений

В ходе реализации общей логики приложения для управления был применен ряд шаблонов проектирования для сетей функциональных блоков IEC 61499. Кроме того, были реализованы определенные типы FB для оценки логических условий. Этот раздел посвящен обзору и описанию используемых методов и шаблонов.

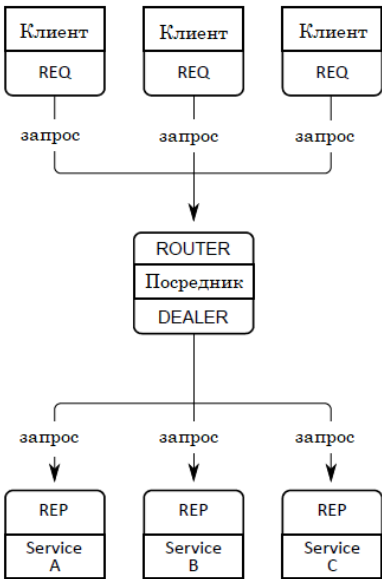


Рисунок 18.8. Запрос посредника

18.4.3.1. Гибкий сервисный шаблон параметризации

Мультиплексоры данных могут эффективно использоваться в сочетании с простыми мультиплексорами событий, чтобы выполнить гибкий шаблон параметризации для услуг, как показано на рисунке 18.11. Мультиплексор событий (*E_MUX*) используется для определения случая параметризации для последующего вызова сервисной функции сервисного FB. Кроме того, мультиплексор данных (*F_MUX*) требуется для выбора значений

параметризации, передаваемых услуге. События подтверждения выполнения разделяются в соответствии с данным случаем параметризации с использованием демультиплексора событий (*E_DEMUX*) для каждого события подтверждения услуги.

Помимо обеспечения гибкой параметризации службы, этот шаблон реализует однозначное сопоставление службы с определенным аппаратным или программным компонентом. Такое отображение требуется, если функциональный компонент может быть доступен только через одно соединение за раз.

18.4.3.2. Сценарии LUA FB для быстрого прототипирования алгоритма и координации FB

Основываясь на работе [25], можно использовать общие типы FB, а также FB, которые выполняют сценарии LUA, для обеспечения быстрого прототипирования алгоритмов и служебных FB.

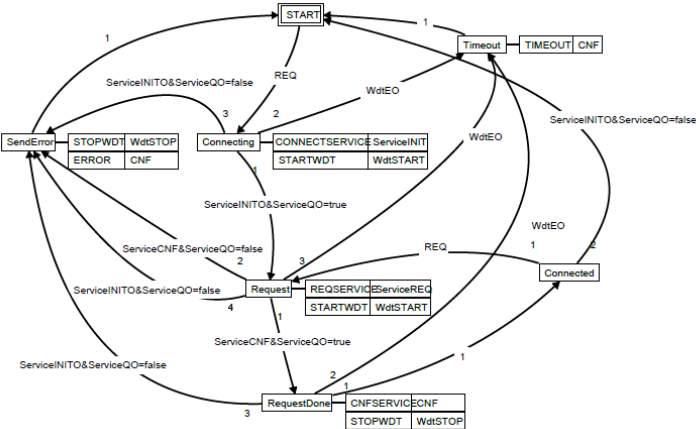


Рисунок 18.9. Логика *ServiceController* как ECC

Базовые типы сценариев LUA FB

В [25] было показано, что сценарии LUA могут эффективно использоваться для реализации интерактивных интерпретируемых ECC, используемых в основных типах FB. Интерпретируемость основных типов FB в режиме онлайн подразумевает следующие требования.

- Защитные условия переходов ECC и выражения алгоритма в действии ECC необходимо интерпретировать при загрузке FB.
- Выполнение ECC должно быть настраиваемым на основе извлеченной информации ECC и интерпретированных алгоритмов и условий.

Чтобы реализовать онлайн-интерпретацию выражений и условий, язык сценариев LUA [14] был интегрирован в FORTE в нашей предыдущей работе [25]. Поскольку базовые алгоритмы и условия FB обычно кодируются на языке структурированного текста (ST), совместимом с IEC 61131 [15], их необходимо экспортировать в выражения LUA с использованием ST-TO-LUA-exporter. На рисунке 18.12 показан подход к прямой настройке ECC на основе выражений условий и алгоритмов LUA. Выражения ST преобразуются в код LUA и возвращаются как C-String. Исполняемый скрипт LUA создается путем помещения полученных выражений LUA в функцию LUA, которую можно поместить в стек LUA и запустить во время выполнения FB. После создания сценарий компилируется, и в случае успеха создается указатель функции, ссылающийся на метод выполнения сценария.

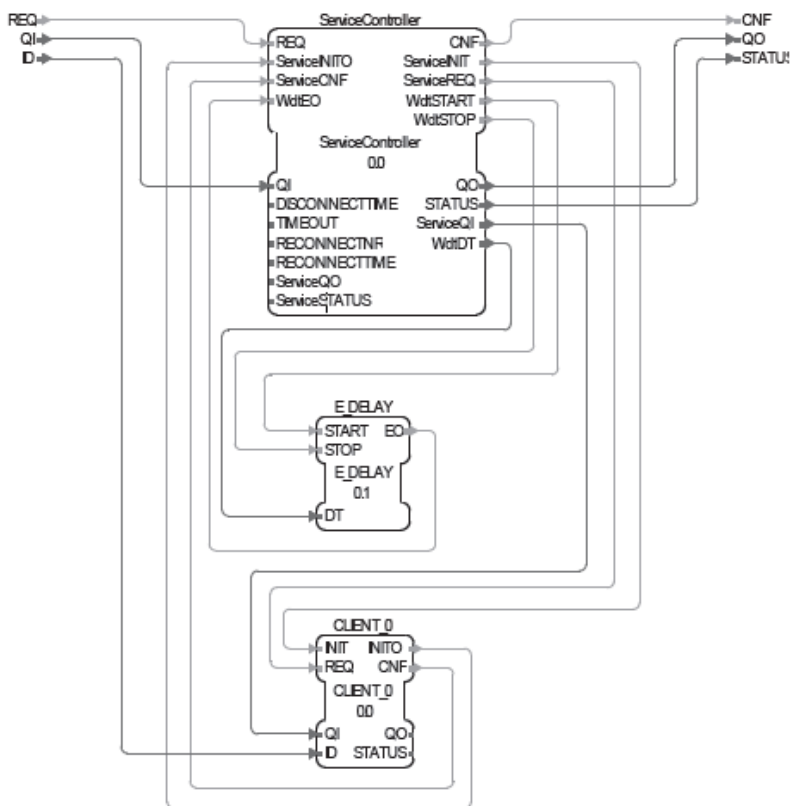


Рисунок 18.10. Типичное использование *ServiceController* с круговой инициализацией

Во время выполнения сценария внутренние переменные, значения ввода и вывода данных регистрируются как глобальные переменные, содержащие их текущие значения, а затем сценарий запускается путем вызова `lua_pcall`. Как

Расширение общих FB с использованием скриптов LUA

The diagram illustrates the ServiceCall model architecture. It consists of several main blocks: **PARAM_SPLIT**, **PARAMETERS**, **CALL_SPLIT**, **ServiceFB**, **SCFBTemplate**, **CNF_CASE_1**, **CNF_CASE_2**, **E_DEMUX_4**, and **E_MUX_4**. The flow is as follows: **EXE_CASE** (E01-E04) provides input to **PARAM_SPLIT** (E01-E04), which outputs to **PARAMETERS** (E01-E04). **PARAMETERS** feeds into **CALL_SPLIT** (E01-E04), which outputs to **ServiceFB** (E01-E04). **ServiceFB** feeds into **SCFBTemplate** (E01-E04). **SCFBTemplate** outputs to **CNF_CASE_1** (E01-E04) and **CNF_CASE_2** (E01-E04). **CNF_CASE_1** outputs to **E_DEMUX_4** (E01-E04). **CNF_CASE_2** outputs to **E_DEMUX_4** (E01-E04). **E_DEMUX_4** outputs to **E_MUX_4** (E01-E04), which then feeds back into **EXE_CASE** (E01-E04). Internal signals within **SCFBTemplate** include **STATUS**, **ERR_CODE**, and **RSP_TIME**. The diagram also shows various control signals like **Q**, **K**, and **QO** connecting the blocks.

Рисунок 18.11. Гибкая параметризация сервисных FB



Рисунок 18.12. Преобразование и компиляция структурированного текста в LUA

Гибкость общих FB может быть объединена со сценариями LUA, чтобы облегчить изменения логики выполнения, например для оценки логических условий или простых выражений. На рисунке 18.13 показан общий тип IF FB, который используется для оценки логических условий, аналогично оператору *if*. ЛЮБЫЕ входы данных используются для передачи значений параметров, которые включены в строку условия (требуется синтаксис LUA), заданную с помощью входа COND. Условие $\#IN\_1 / 2 < IN\_2$, где унарный $\#$ -оператор возвращает длину своего операнда IN_1 , не выполняется, и вывод события ELSE инициируется. Сценарий LUA создается путем включения указанного условия в предопределенную заглушку кода LUA и перекомпилируется, только если условие изменяется во время выполнения.

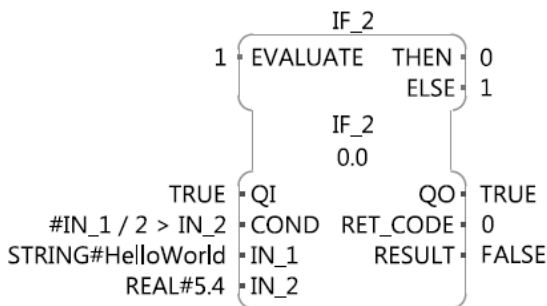


Рисунок 18.13. Общий тип IF FB

В качестве основного преимущества выражение условия может быть изменено во время выполнения, поскольку результирующий скрипт просто

нужно перекомпилировать через интерпретатор LUA. На рисунке 18.14, в сравнении с общим типом IF FB показана реализация того же поведения с использованием дискретной логики приложения FB. С точки зрения скорости выполнения, дискретное приложение FB выполняется примерно в три раза быстрее, чем сценарий, общий IF FB. Это связано с вызовом выполнения сценария LUA, а также с передачей данных в глобальные переменные LUA и из них. Однако дискретной сети FB не хватает гибкости, поскольку логика условия является фиксированной. Изменения в логике условий прямо подразумевают изменения в сети FB.

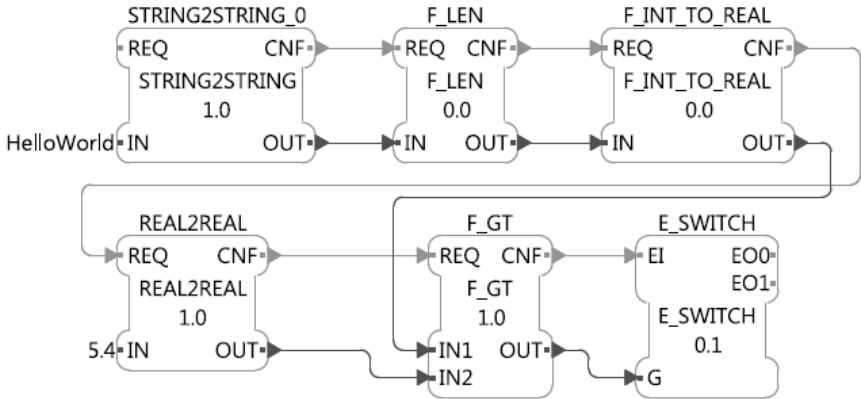


Рисунок 18.14. IF реализована как сеть FB

Преимущества использования скриптовых функциональных блоков

Использование функциональных блоков с возможностью сценариев в сочетании с динамической библиотекой типов, как обсуждалось в [25], позволяет быстро создавать прототипы логических и алгоритмических типов FB, не требуя непрерывных шагов повторной компиляции среды выполнения FORTE. На этапе разработки изменения в логике исполнения FB принимаются путем простого перезапуска FORTE. Поскольку FB LUA динамически загружаются на основе стандартных файлов типа FB, их можно легко экспортировать как файлы C++ и скомпилировать в FORTE после завершения разработки и тестирования.

Как обсуждалось в разделе 18.4.3.2, общие типы FB с возможностью написания сценариев понижение сложности приложений FB, поскольку оценка простых выражений или условий может быть реализована с использованием одного экземпляра FB вместо дискретных логик приложения. Более того, типы FB логических сценариев обеспечивают гибкость в отношении их параметризации, поскольку сценарии могут обновляться во время выполнения.

18.5. Полученная архитектура системы

Путем интеграции функциональных компонентов, представленных в разделе 18.3.3, в соответствии с методологиями, описанными в разделе 18.4, была получена архитектура системы, как показано на рисунке 18.15.

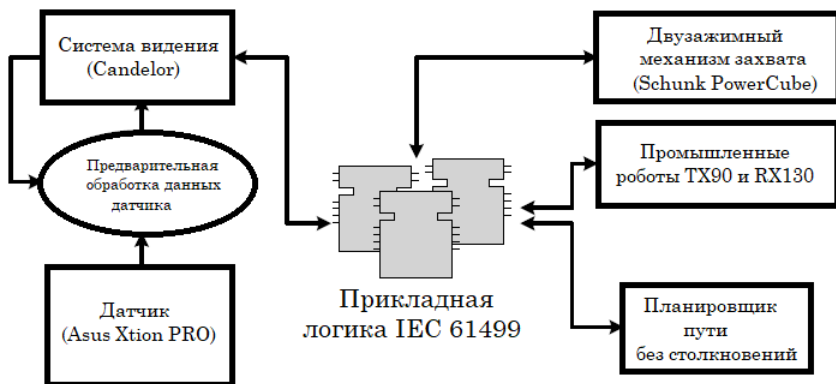


Рисунок 18.15. Результирующая системная архитектура рабочей области, совместно использующей роботизированную систему сборки

Для реализации задачи сборки, как описано в разделе 18.3.2, компоненты системы, изображенные на рисунке 18.15, должны координироваться через логику приложения. Цель состоит в том, чтобы разработать общую прикладную логику, основанную на концепциях IEC 61499, которая позволит роботизированной системе выполнять операции манипуляции и сборки. На рисунке 18.15 показаны коммуникационные отношения между логикой и функциональными компонентами. Последовательность выполнения функций в общей логике приложения обобщена ниже:

- Сбор данных с места и предварительная обработка полученного облака точек (шумоподавление при облучении);
- распознавание и локализация объектов для поиска позиций и ориентаций представляющих интерес объектов;
- планирование путей без столкновений для манипулирования и сборки объектов; возможно взаимодействие с роботом (например, передача захваченного объекта);
- выполнение путей манипуляции и координации захвата.

В зависимости от конкретной задачи сборки различные экземпляры универсальной логики приложения, которые по-разному параметризуются, выполняются и координируются управляющим приложением для конкретной задачи (контроллером). На рисунке 18.16 показан пример, в котором два экземпляра общей логики используются для манипулирования двумя различными типами объектов. Кроме того, логика, используемая для управления простым графическим интерфейсом пользователя (GUI) на основе

WxWidgets [36], позволяет оператору-человеку взаимодействовать с системой. Приложение управления отвечает за координацию логических экземпляров и обмен данными. Основным преимуществом этой архитектуры является ее непрерывная иерархическая структура, что позволяет расширять приложения с разумными затратами времени.

Вариант использования задачи сборки

Со ссылкой на раздел 18.5 в этом разделе на рисунке 18.16 изображен конкретный вариант использования задачи сборки на основе архитектуры приложения.

Создаются два экземпляра универсальной прикладной логики, способной манипулировать объектами: один для управления предохранителями, а второй для управления блоками предохранителей (см. рисунок 18.2). Приложение контроллера зависит от конкретного случая и координирует экземпляры приложения, управляющего объектами, для выполнения задачи сборки. Предполагается следующий сценарий сборки.

- Один блок предохранителей и три предохранителя должны быть собраны.
- Блок предохранителей и один предохранитель расположены рядом с роботом А, а остальные предохранители – рядом с роботом В.
- Поскольку второй и третий предохранители не находятся в пределах досягаемости робота А, а блок предохранителей не может быть достигнут роботом В, для выполнения задачи требуется взаимодействие между обоими роботами.

Необходимая функциональность приложения контроллера представлена на рисунке 18.17.

Во время запуска системы все приложения FB, в соответствии с рисунком 18.16, инициализируются. После того как общие экземпляры приложения для управления предохранителями и блоками предохранителей были инициализированы, процедура сборки запускается путем сигнала *Start* графического интерфейса пользователя. Приложение контроллера получает состояния инициализации, а также сигнал от GUI и запускает приложение-предохранитель для выполнения задачи сборки предохранителя. Сборка предохранителей повторяется до тех пор, пока не произойдет сбой вставки в первый раз по следующим причинам:

- предохранители не установлены;
- нет пустого слота;
- планировщик манипуляций не может рассчитать пути захвата и перемещения без столкновений для роботов.

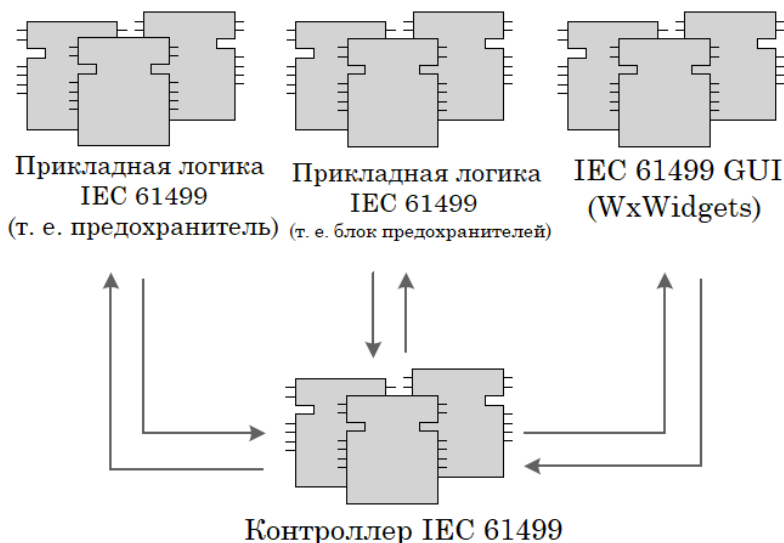


Рисунок 18.16. Несколько экземпляров логики приложения, обрабатывающих разные типы объектов, координируются главным контроллером

Последняя потенциальная причина включает объекты, которые недоступны для робота. Пример такой ситуации объясняется в начале раздела 18.5. В этом случае проблема сборки также не может быть решена, например путем передачи предохранителя от робота А роботу В. Следовательно, согласно рисунку 18.17, приложение блока предохранителей запускается, чтобы переместить блок предохранителей в общую рабочую область, которая является достижимой обоими роботами. Для успешного завершения этой операции процедуру вставки предохранителей повторяют до тех пор, пока все предохранители не будут вставлены и задача сборки не будет завершена. Обратите внимание, что по причинам упрощения на рисунке 18.17 не рассматриваются дополнительные процедуры обработки, например для ошибок локализации объекта.

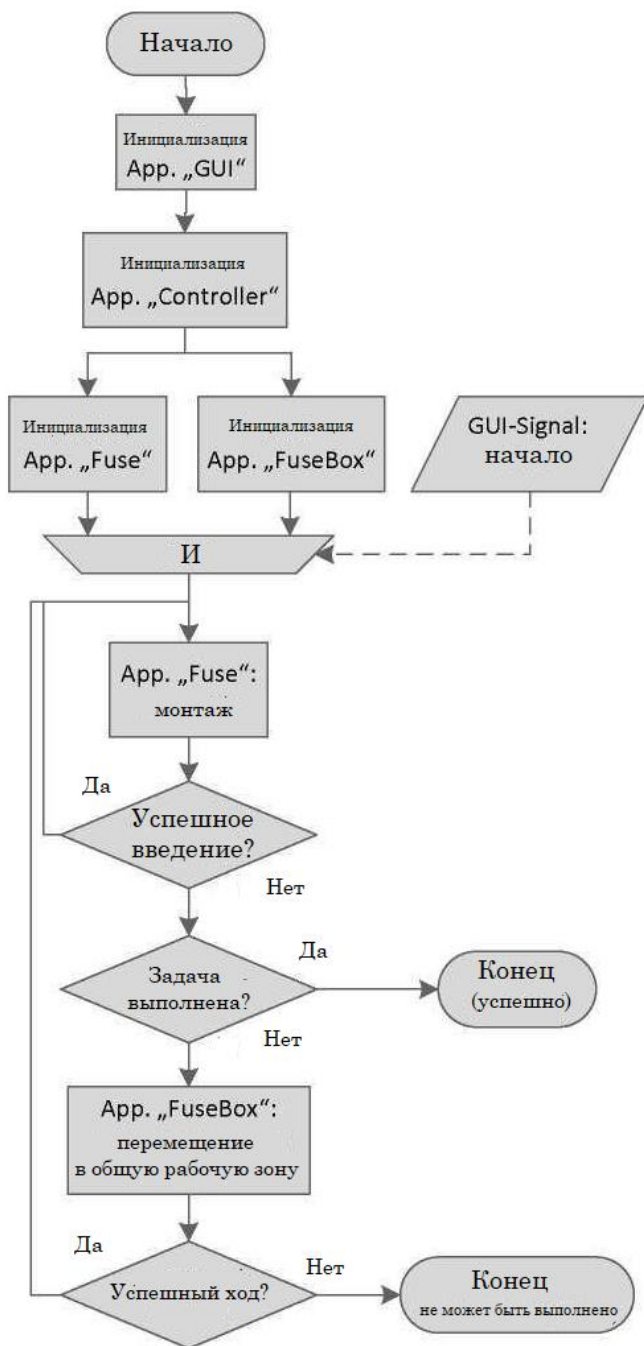


Рисунок 18.17. Процедура сборки

18.6. Сводка и выводы

В этой главе представлен набор концепций системной интеграции и разработки приложений. Эти концепции были применены к конкретному сценарию роботизированного приложения, в котором два робота, совместно использующие общее рабочее пространство, взаимодействуют для выполнения задачи сборки. Чтобы понизить сложность разработки приложений, каждая система управления основывается на службах, которые инкапсулируют функциональность компонента. Следовательно, для обеспечения правильного поведения и стабильности управляющего приложения необходимы тщательные тесты и циклы ревизии для каждого сервиса. Повторное использование проверенных программных компонентов значительно сокращает усилия по разработке и отладке при интеграции и разработке приложений. Высокий уровень гибкости инфраструктуры 4DIAC обеспечивает быструю интеграцию компонентов и разработку приложений на основе представленных шаблонов разработки. В процессе интеграции и разработки авторы определили ряд полезных функций для повышения гибкости разработки и сокращения трудозатрат. Примерами являются функциональные возможности мониторинга для составных типов FB и фильтр экспорта типа FB для потоковых типов FB. Кроме того, требуется функциональность инструмента 4DIAC для динамического расширения интерфейса общих типов FB в приложении. Это может уменьшить количество файлов типов для каждой конфигурации интерфейса общего типа FB в библиотеке типов инструмента и повысить ясность.

Библиография

- [1] ASUSTeK Computer Inc. ASUS Xtion PRO.
http://www.asus.com/Commercial_3D_Sensor/Xtion/.
- [2] N. Bezroukov. Ciptorama: A Slightly Skeptical View on Scripting Languages. <http://www.softpanorama.org/Scripting/index.shtml>.
- [3] D. Brugali, L. Gherardi, A. Biziak, A. Luzzana, and A. Zakharov. A reuse-oriented development process for component-based robotic systems. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 361–374. Springer, 2012.
- [4] D. Brugali, L. Gherardi, M. Klotzbücher, and H. Bruyninckx. *Service Component Architectures in Robotics: The SCA-OROCOS Integration*. Springer, 2012.
- [5] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetschmar, L. Gherardi, and D. Brugali. The BRICS component model: A modelbased development paradigm for complex robotics software systems. In *Proceedings of 28th Annual ACM Symposium on Applied Computing*, pages 1758–1764. ACM, 2013.

- [6] J. Capco, M. Rooker, and A. Pichler. RRT planner for the binpicking problem. In *Robot Motion and Control, 9th Workshop on*, pages 154–160. IEEE, 2013.
- [7] G. Ebenhofer, H. Bauer, M. Plasch, S. Zambal, S. C. Akkaladevi, and A. Pichler. A system integration approach for service-oriented robotics. In *Emerging Technologies & Factory Automation, 18th Conference on*, pages 1–8. IEEE, 2013.
- [8] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, Upper Saddle River, 2005.
- [9] Y.-G. Ha, J.-C. Sohn, Y.-J. Cho, and H. Yoon. A robotic service framework supporting automated integration of ubiquitous sensors and devices. *Information Sciences*, Vol. 177, No. 3, 657–679, 2007.
- [10] P. Hintjens. CANDELOR: Computer vision library for 3D scene interpretation. <http://candelor.com/>.
- [11] P. Hintjens. Zguide: ZeroMQ Guide. <http://zguide.zeromq.org/page:all>.
- [12] P. Hintjens. *Code Connected Volume 1: Learning ZeroMQ*. iMatrix Corporation, 2013.
- [13] IEC 61499-1. *Function Blocks. Part 1: Architecture*. International Electrotechnical Commission, Geneva, 2005.
- [14] R. Ierusalimsky, W. Celes, and L. Figueiredo. What is LUA?
<http://www.lua.org/about.html>.
- [15] International Electrotechnical Commission. IEC 61131-3. *Programmable Controllers. Part*, Vol. 3, , 1993.
- [16] F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 1, 62–70, 2005.
- [17] D. Kononchuk, V. Kandoba, S. Zhigalov, P. Abduramanov, and Y. Okulovsky. Robocop: A protocol for service-oriented robot control systems. In *Research and Education in Robotics*, pages 158–171. Springer, 2011.
- [18] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001. IEEE, 2000.
- [19] G. Metta, P. Fitzpatrick, and L. Natale. Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*, Vol. 3, No. 1, 43–48, 2006.
- [20] L. Natale, F. Nori, G. Metta, M. Fumagalli, S. Ivaldi, U. Pattacini, M. Randazzo, A. Schmitz, and G. Sandini. The ICUB platform: a tool for studying intrinsically motivated learning. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 433–458. Springer, 2013.
- [21] OROCOS. Open Robot Control Software. <http://www.oroocos.org>.
- [22] J. K. Ousterhout. Scripting: Higher level programming for the 21st century. *Computer*, Vol. 31, No. 3, 23–30, 1998.
- [23] M. Pall. The LuaJIT Project. <http://luajit.org/index.html>.
- [24] C. Papazov and D. Burschka. An efficient RANSAC for 3D object recognition in noisy and occluded scenes. In *Computer Vision-ACCV 2010*, pages 135–148. Springer, 2011.

- [25] M. Plasch, M. Hofmann, G. Ebenhofer, and M. Rooker. Reduction of development time by using scriptable IEC 61499 function blocks in a dynamically loadable type library. In *Emerging Technology and Factory Automation*, pages 1–8. IEEE, 2014.
- [26] M. Plasch, A. Pichler, H. Bauer, M. Rooker, and G. Ebenhofer. A plug & produce approach to design robot assistants in a sustainable manufacturing environment. In *22nd International Conference on Flexible Automation and Intelligent Manufacturing, Helsinki*, 2012.
- [27] Promixis. Girder: Home and Industrial Automation Software. <http://www.promixis.com/index.php>.
- [28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3.2, page 5, 2009.
- [29] S. Robotics. Staeubli Robotics. <http://www.staubli.com/en/robotics/>.
- [30] ROS-Project. ROS âAS Robot operating System. <http://www.ros.org/>.
- [31] SCA. Service Component Architecture. <http://www.osoa.org/>.
- [32] Schunk. Schunk: Superior Clamping and Gripping. <http://www.schunk.com/>.
- [33] T. Strasser, F. Andren, and M. Stifter. A test and validation approach for the standard-based implementation of intelligent electronic devices in smart grids. In *Holonic and Multi-Agent Systems for Manufacturing*, pages 50–61. Springer, 2011.
- [34] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, and A. Martel. Framework for distributed industrial automation and control (4DIAC). In *6th IEEE International Conference on Industrial Informatics*, pages 283–288. IEEE, 2008.
- [35] G. Veiga, J. Pires, and K. Nilsson. Experiments with service-oriented architectures for industrial robotic cells programming. *Robotics and Computer-Integrated Manufacturing*, Vol. 25, No. 4, 746–755, 2009.
- [36] WxWidgets. WxWidgets: A Cross-Platform GUI Library. <https://www.wxwidgets.org/>.
- [37] T.-H. Yang and W.-P. Lee. A service-oriented framework for the development of home robots. *International Journal of Advanced Robotic Systems*, Vol. 10, , 2013.
- [38] A. Zoitl. *Real-Time Execution for IEC 61499*. ISA, 2008.

Иерархически структурированная система управления для станции сбора и размещения

Monika Wenger
fortiss GmbH

Milan Vathoopan
fortiss GmbH

Alois Zoitl
fortiss GmbH

Herbert Prahofer
Линцский университет имени Иоганна Кеплера

СОДЕРЖАНИЕ

19.1 Введение	470
19.2 Принципы для иерархически структурированных управляющих приложений	470
19.2.1 Многоуровневая структура	471
19.2.2 Разделение на уровни с IEC 61499	472
19.3 Структура станции сбора и размещения	473
19.4 Приложение иерархического управления	475
19.4.1 Уровень 0: атомарные компоненты	475
19.4.2 Уровень 1: элементарные компоненты	476
19.4.2.1 Компоненты, связанные с манипулятором	477
19.4.2.2 Компоненты, связанные с конвейером	479
19.4.2.3 Компонент индикатора состояния	481
19.4.3 Уровень 2: координирование компонентов	481
19.4.3.1 Слайд	481
19.4.3.2 Манипулятор	482
19.4.3.3 Конвейер	482
19.4.3.4 Управление состоянием	484
19.4.4 Уровень 3: компонент управления процессом	485
19.5 Выводы	486

19.1. Введение

Ключевым моментом для разработки систем автоматизации является сокращение усилий по разработке программного обеспечения для систем управления. IEC 61499, уделяя особое внимание инкапсуляции и повторному использованию, претендует на улучшение ситуации для инженеров систем управления. IEC 61499 определяет только средства и модели для разработки систем управления компонентно-ориентированным способом. Однако ключевой момент, как использовать модели IEC 61499 для достижения цели модульных многоразовых приложений для управления, выходит за рамки настоящего стандарта.

В предыдущих работах [3, 4] мы разработали шаблон структурного проектирования, направляющий инженеров систем управления в сторону структурирования управляющих приложений для мехатронных машин. Основная идея шаблона проектирования – это строгий иерархический подход, при котором более высокие уровни координируют базовые уровни и получают команды от координатора более высокого уровня. Шаблоны проектирования требуют, чтобы иерархия управляющего программного обеспечения строго следовала мехатронной иерархии. Это имеет большое преимущество, заключающееся в том, что каждый мехатронный компонент машины имеет аналог в структуре программного обеспечения, что облегчает навигацию программного обеспечения для управления. Кроме того, изменения в мехатронной настройке машины могут быть легко сделаны заново в управляющем программном обеспечении, и изменения будут изолированы от области мехатронных изменений.

В дополнение к иерархии шаблон проектирования также определяет, что интерфейсы адаптера IEC 61499 должны использоваться для определения интерфейсов взаимодействия компонента. Это дополнительно разъединяет элементы управляющего приложения, что увеличивает возможность повторного использования.

В этой главе мы покажем, как использовать этот иерархический шаблон проектирования, применяя его к учебному стенду сбора и размещения. В разделе 19.2 будет повторен шаблон проектирования. В разделе 19.3 мы опишем результирующую иерархическую структуру нашего управляющего приложения. Далее, в разделе 19.4, следует подробное описание компонентов и их взаимодействия. Выводы завершают эту главу.

19.2. Принципы для иерархических структурированных управляющих приложений

Принципы проектирования иерархически структурированных управляющих приложений [4] суммированы в этом разделе. Эти принципы для многоуровневого проектирования управляющих приложений применяются к

приложениям, установленным Международной электротехнической комиссией (IEC) 61499.

19.2.1. Многоуровневая структура

Многоуровневая структура управляющего приложения размещает компоненты на разных уровнях иерархии, которые предоставляют услуги на разных глубинах проектирования. Один уровень предлагает услуги для уровня выше и получает услуги от уровня ниже. Каждый компонент обеспечивает четко определенный интерфейс и напоминает иерархическую структуру машины. Каждый программный компонент разработан так, чтобы зависеть от механической структуры установки и представлять определенный аппаратный компонент. Первым шагом при проектировании слоистой структуры является изучение механической структуры желаемой установки. Для реализации иерархической архитектуры управления рекомендуется восходящая стратегия, поскольку она предотвращает создание ненужных слоев. Это также помогает использовать модульную архитектуру и облегчает повторное использование компонентов и гибкость системы в целом.

Компоненты нижнего уровня координируются и контролируются компонентами более высокого уровня и выполняют задачи более низкого уровня управления, тогда как компоненты более высокого уровня выполняют задачи более высокого уровня управления. Компоненты для более высокого уровня определяются в зависимости от приложения и его сложности. Взаимодействие между компонентами достигается за счет определенных интерфейсов между компонентами верхнего и нижнего уровней. Команды отправляются от компонентов более высокого уровня компонентам более низкого уровня, и обратная связь осуществляется в обратном направлении. Соединение различных компонентов является иерархическим, но поведение реализуется компонентами самого нижнего уровня. В основном выделяют четыре типа слоев.

- Уровень 0: компоненты этого уровня реализуют интерфейс с оборудованием, таким как датчики или исполнительные механизмы, а также могут реализовывать элементарные непрерывные контуры управления.

- Уровень 1: компоненты этого уровня напрямую управляют одним или несколькими компонентами уровня 0, чтобы объединить элементарные действия в элементарных операциях управления и предоставить их в качестве автономных операций управления следующему более высокому уровню. Эти компоненты устанавливают исполнительные механизмы и включают или выключают соответствующие контуры управления, контролируют значения датчиков от компонентов более низкого уровня, чтобы определить, правильно ли выполняется процесс, и сообщают об ошибках на следующий, более высокий уровень.

- От уровня 2 до уровня n: компоненты этих уровней координируют действия своих подкомпонентов, синхронизируют потенциально параллельные

действия, безопасно завершают любой процесс подкомпонента, проверяют, не возникли ли ошибки, и отчитываются перед подключенными компонентами более высокого уровня.

- Уровень n: компоненты этого слоя обеспечивают общее управление процессом для установки.

19.2.2. Разделение на уровни с IEC 61499

IEC 61499 [2] предоставляет ориентированный на приложения инженерный подход, включающий модель приложения, не зависящую от устройства, и позднюю привязку аппаратного обеспечения через его модель распределения. Zoitl и Prähofer [4] представляют некоторые руководящие принципы и правила проектирования для структурирования компонентно-ориентированных решений иерархической автоматизации, реализованных в соответствии со стандартом IEC 61499. Следующие руководящие принципы основаны на этой работе.

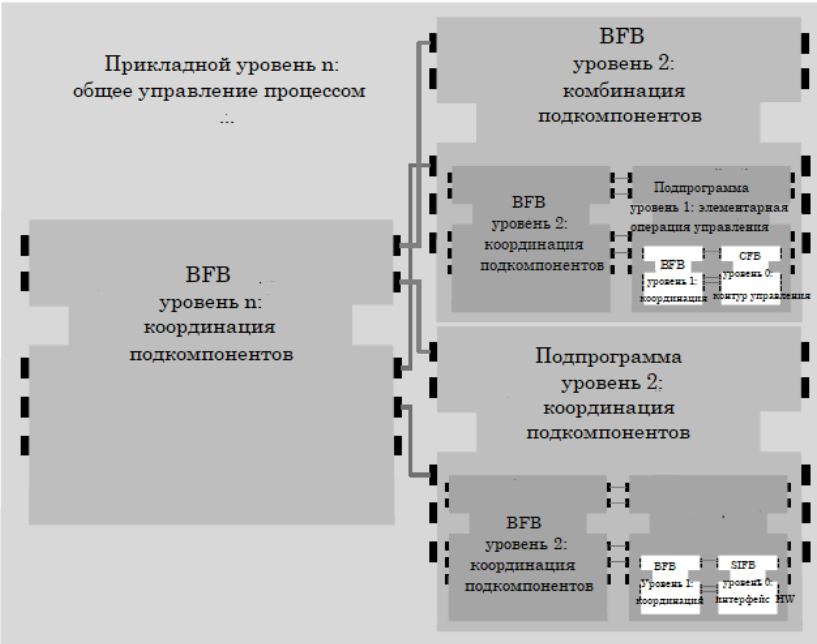


Рисунок 19.1. Иерархические уровни применяются на IEC 61499 и основаны на [4].

Самый верхний уровень n представлен приложением IEC 61499, как показано на рисунке 19.1. Каждый уровень должен предоставить координационный функциональный блок (FB) для своих подкомпонентов.

Такой координационный FB должен быть реализован с помощью контрольной диаграммы исполнения (ЕСС) базового функционального блока (BFB), поскольку ЕСС координирует поведение для различных событий посредством переходов в конкретное состояние, которое выполняет надлежащее действие. При рассмотрении обработки ошибок ЕСС могут стать довольно сложными. Чтобы понизить сложность, можно использовать один переход ошибки из каждого состояния в общее состояние ошибки.

Компоненты более низкого уровня могут быть реализованы любым типом FB IEC 61499, например составными функциональными блоками (CFB), BFB или функциональными блоками интерфейса службы (SIFB), тогда как атомарные компоненты уровня 0, которые обычно обеспечивают аппаратный доступ, должны быть реализованы по SIFB. Поэтому любой компонент реализуется одним или несколькими FB IEC 61499.

Чтобы организовать компоненты иерархически, CFB или подпрограммы предусмотрены IEC 61499. Сети FB внутри CFB не являются распределяемыми и, следовательно, недостаточны для построения иерархических структур. Группирование компонентов или FB выполняется посредством подприложений в IEC 61499. Подпрограммы помогают уменьшить количество FB и, следовательно, сложность приложения в целом. Они моделируют аппаратную абстракцию и скрывают внутреннюю сложность системы. Для иерархической организации требуются SIFB, независимые от платформы для аппаратного доступа на уровне 0, который можно настроить в ресурсе после сопоставления.

Так как интерфейсы компонентов могут стать довольно большими, адаптеры, предоставленные IEC 61499, используются для моделирования взаимодействия между уровнями. Адаптеры группируют полученные и отправленные события и данные для повышения ясности и понятности компонентов. Благодаря функциональности группировки они сокращают количество необходимых соединений. Любой адаптер отражает услуги, предлагаемые или требуемые компонентом. Два типа адаптеров в IEC 61499 – это вилки и розетки. Розетки используются в любом компоненте для управления службой с точки зрения запуска, остановки или обновления. Поскольку ЕСС не поддерживает условия тайм-аута, компоненты первого уровня используют адаптер тайм-аута ATimeout, подключенный к CFB E_TimeOut, который обеспечивает функциональность тайм-аута.

19.3. Структура станции сбора и размещения

Станция сбора и размещения от Festo Didactic – это автоматическое (разборное) устройство, которое позволяет монтировать заготовки (часы, термометр, гигрометр) в корпус заготовки [1]. Эта станция состоит из пяти модулей: конвейерной ленты с пневматическим сепаратором, стека для вкладышей заготовок (до 6 штук), пневматического погрузочно-разгрузочного

устройства и индикатора состояния. Механическая структура станции сбора и размещения показана на рисунке 19.2.

Конвейерная лента отвечает за транспортировку заготовки (заготовок) через станцию. Она оснащена двумя диффузионными датчиками: один на входе на конвейерную ленту, он определяет вход в корпус заготовки, а другой – посередине перед точкой сборки. Барьерный датчик на выходе конвейерной ленты обнаруживает готовый продукт, покидающий ленту. Корпус заготовки входит в станцию, пропуская первый диффузный датчик, что вызывает запуск конвейерной ленты. Конвейерная лента транспортирует заготовку к пневматическому сепаратору, который обнаруживает поступающую деталь вторым диффузным датчиком и останавливает деталь. Пневматический сепаратор представляет собой рычаг, расположенный в станции сборки, который останавливает корпус заготовки в точке сборки. На слайде хранится заготовка для вставки. Пневматический манипулятор представляет собой двуххосное устройство, которое подбирает обрабатываемую деталь из стека и помещает ее на корпус заготовки для формирования готового изделия. Готовый продукт выпускается сепаратором и транспортируется к концу конвейерной ленты, где он обнаруживается датчиком светового барьера. Зеленый, красный и желтый индикаторы состояния указывают на режимы станции. Пустой стек обозначается вакуумным датчиком манипулятора и красным мигающим индикатором состояния. Фаза инициализации обозначается оранжевым мигающим индикатором состояния. Станция сбора расположения и хода обозначена зеленым индикатором состояния.

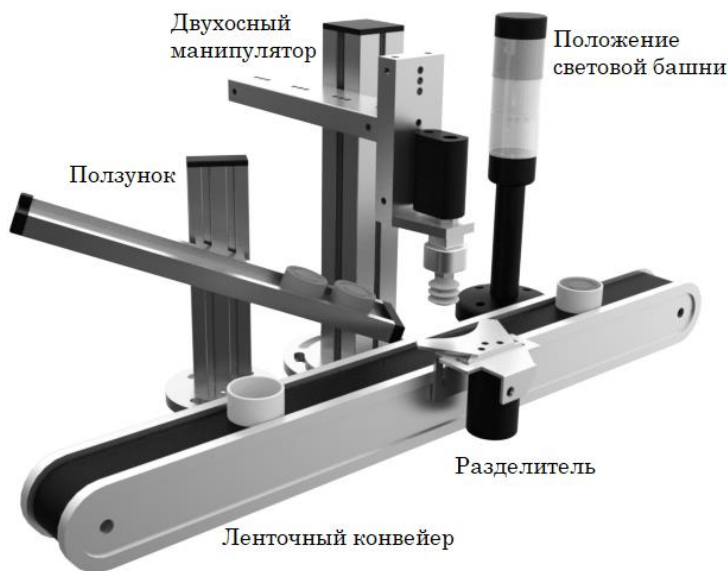


Рисунок 19.2. Структура станции сбора и размещения на основе [1]

19.4. Приложение иерархического управления

Первоначально, следуя принципам проектирования, упомянутым выше, механическая структура станции тщательно анализируется, и на разных уровнях идентифицируются различные компоненты. Концептуальный дизайн управляющего приложения с иерархической структурой затем строится на основе идентифицированных компонентов, напоминающих общую структуру станции сбора и размещения, как показано на рисунке 19.3. Низшие атомарные компоненты, которые представляют цифровые входы и выходы (I/O), размещаются на самом низком уровне 0. В слое 1 элементарные компоненты (индикатор состояния, горизонтальная ось, вертикальная ось, захватчик, счетчик деталей, разделитель и транспортер) размещены для координации функций атомарных компонентов и выполнения элементарных операций управления. Модули управления состоянием, манипулятор и конвейер расположены на уровне 2, координируя базовые элементарные компоненты. Для управления всем процессом компонент под названием picknPlace остается на верхнем уровне 3.

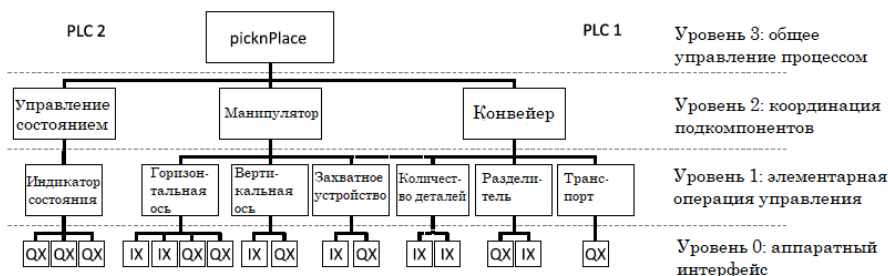


Рисунок 19.3. Иерархическая структура компонентов станции сбора и размещения на основе [4]

Иерархически структурированная система управления для станции сбора и размещения реализовано в рамках системы с открытым исходным кодом для распределенной промышленной автоматизации и управления (4DIAC) [5], которая является реализацией стандарта IEC 61499. Приложение разработано с учетом повторного использования и реконфигурации станции и ее компонентов.

19.4.1. Уровень 0: атомарные компоненты

Атомарные компоненты расположены в самом нижнем слое 0. Эти компоненты взаимодействуют непосредственно с оборудованием. Многие элементы, такие как двигатель, клапаны и датчики переключающего типа, могут быть смоделированы напрямую с помощью цифровых IO FB. Эти FB получают информацию от любого датчика или управляют любым приводом.

Они обеспечивают аппаратный интерфейс и должны быть смоделированы как SIFB в рамках IEC 61499. Эти SIFB должны быть независимыми от платформы, чтобы их можно было использовать на любом программируемом логическом контроллере (PLC), а также на FB верхнего уровня. В 4DIAC IX и QX были разработаны FB, которые являются общими цифровыми FB I/O. Все приводы смоделированы с помощью QX FB, а все датчики – с помощью IX FB, как показано на рисунке 19.4.

Вход PARAMS определяет параметр, который поддерживает спецификацию желаемых чисел входа и выхода соответственно. Настройка параметра зависит от оборудования. Компоненты верхнего уровня, которые используют эти IX и QX FB для аппаратного доступа, зависят от этой спецификации. Прямое указание номера I/O предотвращает многократное создание любого компонента верхнего уровня в одном PLC, поскольку разные датчики или исполнительные механизмы не могут использовать один и тот же I/O. Вот почему вход PARAMS должен быть указан функциональностью службы виртуальных доменных имен (DNS) 4DIAC. Функциональность виртуального DNS помогает создать список символических имен и соответствующих значений. Символические имена заменяются соответствующими значениями во время загрузки.

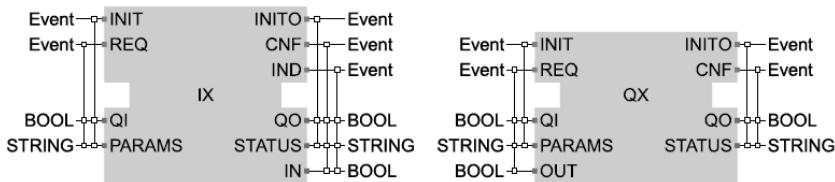


Рисунок 19.4. Платформо-независимые входные и выходные FB, используемые для аппаратного доступа на уровне 0

19.4.2. Уровень 1: элементарные компоненты

Элементарные компоненты соединяют атомарные компоненты, обеспечивают независимое функционирование и могут быть повторно использованы в других приложениях того же домена. Эти компоненты идентифицируются на основе аппаратной структуры станции и обобщаются для лучшего повторного использования. Элементарные компоненты, предназначенные для станции сбора и размещения, поясняются в следующих разделах в соответствии с верхними уровнями.

19.4.2.1. Компоненты, связанные с манипулятором

Рассматривая двухосное пневматическое манипулирующее устройство, нужно отметить, что горизонтальная ось может моделироваться цилиндром двойного действия с информацией о состоянии в обоих направлениях (обратная связь от бесконтактных датчиков с обеих сторон), а вертикальная ось с цилиндром одностороннего действия – с информацией о состоянии только в одном направлении (обратная связь от датчика с одной стороны). Элементарными компонентами для погрузочно-разгрузочного устройства являются цилиндр двойного и одинарного действия и вакуумный захват.

Цилиндр двойного действия выполнен с BFB. Интерфейс для сервисов, предлагаемых компонентом (команда от более высокого уровня и информация о состоянии обратно до более высокого уровня), структурирован с помощью адаптера ACylinder, как показано на рисунке 19.5. Таким образом, адаптер ACylinder является абстракцией функциональности, предлагаемой цилиндром.

Цилиндр имеет два входных события (Retract и Extend) и два события подтверждения (Retracted и Extended). Если после события Extend цилиндр достигает максимальной конечной позиции, он отправляет событие подтверждения Extended; если ему не удастся достичь этой позиции, он отправляет событие ExtendFailed. Это похоже на втягивание. Если цилиндр достигает минимальной конечной позиции после события Retract, он отправляет событие подтверждения Retracted; если ему не удастся достичь этой позиции, он отправляет событие RetractFailed. Помимо адаптера ACylinder цилиндр двойного действия соединен с двумя электромагнитными клапанами для операций выдвижения и втягивания и получает входные сигналы от двух датчиков приближения. На рисунке 19.5 показан цилиндр двойного действия FB для управления горизонтальной осью и его соединение со слоем 0.

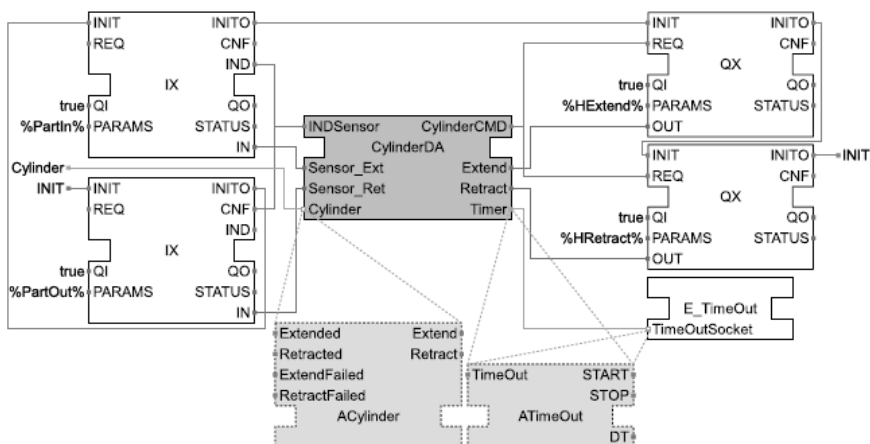


Рисунок 19.5. Уровень 1: Подпрограмма для управления цилиндром двойного действия с цилиндром двойного действия FB для всего элемента

управления (серый цвет), адаптерами, предоставляющими услуги перемещения и тайм-аута (светло-серый цвет), и соединениями с поставщиком услуг тайм-аута и FB уровня 0 для аппаратного доступа (белый цвет)

Цилиндр одностороннего действия также реализован BFB. Функциональность цилиндра одностороннего действия аналогична функциональности цилиндра двойного действия. Услуги, предлагаемые цилиндром одностороннего действия для связи с верхним слоем, также моделируются с помощью адаптера ACylinder. В отличие от цилиндра двойного действия, существует только один электромагнитный клапан для раскрытия цилиндра, и операция втягивания выполняется с помощью пружинного плунжера. Вот почему интерфейс содержит только один выход для привода и получает обратную связь только от одного датчика для обнаружения втянутого цилиндра.

Захват также реализован с помощью BFB, как показано на рисунке 19.6. Услуги, предлагаемые захватом для связи с верхним слоем, моделируются с помощью адаптера AGripperPneumatic. Этот адаптер обеспечивает захват и высвобождение двумя соответствующими событиями. Обратная связь получена от датчика событием Gripped после того, как произошло событие Grip. Если в течение 5 секунд не получено никакой обратной связи, отправляется событие GripFail. Для выпуска аналогичные функциональные возможности реализуются событиями Release, Released и ReleaseFail. Этот FB подает управляющий вход на электромагнитный клапан привода всасывания и получает вход от датчика давления.

Все три компонента используют адаптер ATimeout для моделирования времени ожидания. Все эти компоненты подключены к CFB E_TimeOut, который отвечает за реализацию запрошенной задержки.

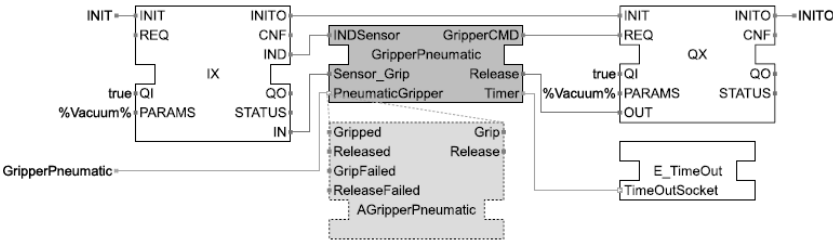


Рисунок 19.6. Уровень 1: Подпрограмма для управления захватом с помощью пневматического захвата FB (серый цвет) для всего элемента управления, адаптером, обеспечивающим службу захвата (светло-серый цвет) и подключением к поставщику услуг тайм-аута и FB уровня 0 для аппаратного доступа (белый)

19.4.2.2. Компоненты, связанные с конвейером

Конвейерная лента может быть разделена на транспортный, сепараторный и противодействующий компоненты. Транспортный компонент реализуется BFB, как показано на рисунке 19.7. Это фактическая транспортная единица для заготовки. Данный компонент представляет собой двигатель, который запускает и останавливает работу конвейерной ленты.

Этот FB полностью работает по команде и не предоставляет по своему статусу обратной связи. Транспортный компонент имеет события MotorStart и MotorStop. Он подает управляющий сигнал на исполнительный блок конвейерной ленты. В интерфейсе установлен таймер для поддержки задержки выключения двигателя в случае, если световой барьер, который обнаруживает законченные детали, не может быть расположен в самом конце конвейера.

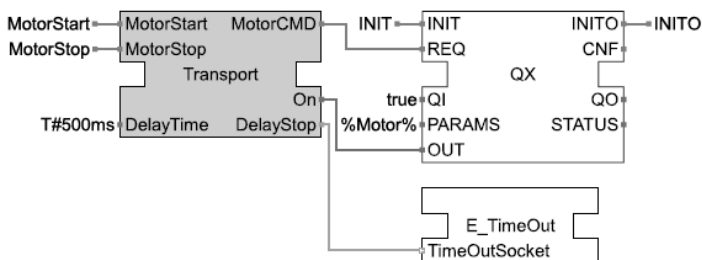


Рисунок 19.7. Уровень 1: Подпрограмма для управления транспортировкой с транспортным FB для всего элемента управления (серый цвет) и его подключением к FB уровня 0 для аппаратного доступа (белый цвет)

Разделитель – это вращающийся рычаг, используемый для отделения отдельных компонентов от группы. Он обнаруживает поступающие детали с помощью диффузного датчика. В положении по умолчанию он не позволяет проходить никаким компонентам. Когда он вращается против часовой стрелки, то может захватывать заготовку и позиционировать эту заготовку для любой обработки. Если же он вращается по часовой стрелке, любая захваченная деталь освобождается. Компонент разделителя также реализован с помощью BFB, который имеет события Release и Confine для управления положением разделителя, как показано на рисунке 19.8. Этот блок подает вход на соленоид сепаратора и получает вход от диффузного датчика перед сепаратором. Внешний таймер подключен к интерфейсу компонента-разделителя для обеспечения задержки во времени после высвобождения. Задержка во времени препятствует вращению сепаратора для захвата заготовки до тех пор, пока только что выпущенная заготовка не окажется вне рабочего пространства сепаратора.

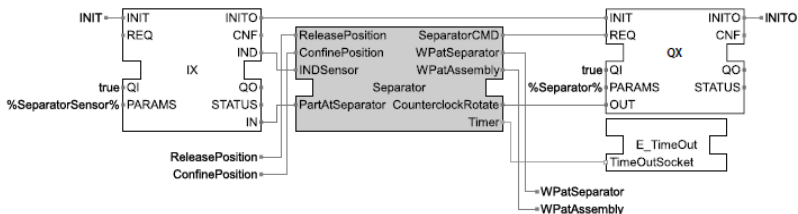


Рисунок 19.8. Уровень 1: Подпрограмма для управления разделителем с разделителем FB для всего элемента управления (серый цвет) и соединениями с поставщиком услуг тайм-аута и FB уровня 0 для аппаратного доступа (белый цвет)

Компонент счетчика подсчитывает количество деталей на конвейерной ленте, глядя на поступающие детали, которые обнаруживаются датчиком приближения в начале конвейерной ленты, и исходящие детали, обнаруживаемые световым барьером на конце конвейерной ленты. Этот компонент также является BFB, как показано на рисунке 19.9. Чтобы предотвратить многократное обнаружение заготовок, для увеличения и уменьшения их количества применяется отладка в 100 мс. Фактическое количество деталей на конвейерной ленте предоставляется как услуги для следующего верхнего слоя адаптером APartCount.

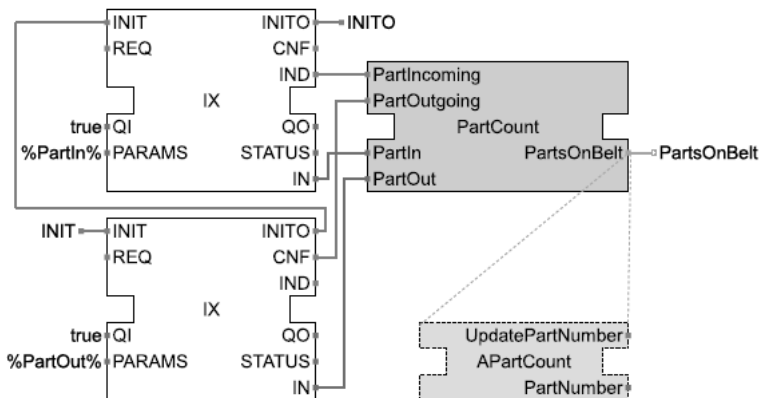


Рисунок 19.9. Уровень 1: Подпрограмма для подсчета деталей на ремне с помощью счетчика деталей FB (серый цвет), адаптера, обеспечивающего службу подсчета (светло-серый цвет), и подключения к FB уровня 0 для аппаратного доступа (белый цвет)

19.4.2.3. Компонент индикатора состояния

Индикатор состояния отображает индикацию станции. Он состоит из башни красного, оранжевого и зеленого огней. Индикатор состояния реализован с помощью CFB, как показано на рисунке 19.10. Сервис, предлагаемый этим компонентом, предоставляется на следующий верхний уровень с помощью адаптера AStatusLight, который запускает события для управления тремя лампочками, которые светятся или мигают красным, оранжевым и зеленым, с регулируемой частотой мигания 2 Гц.

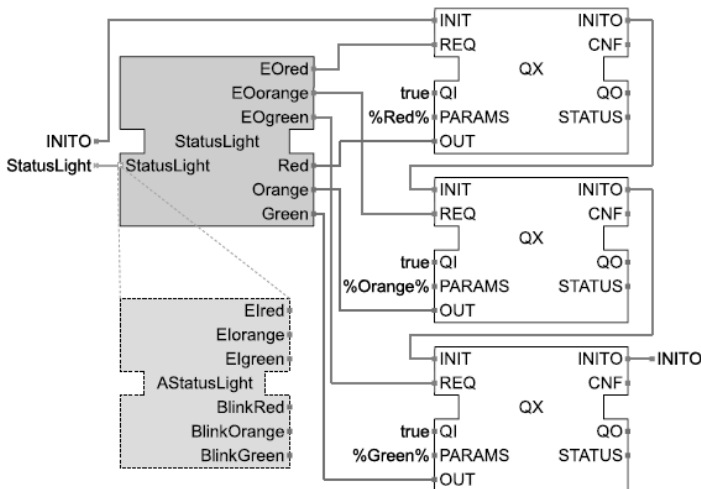


Рисунок 19.10. Уровень 1: Подпрограмма для управления индикатором состояния со световым индикатором состояния FB для всего элемента управления (серый цвет), адаптером, обеспечивающим службу индикации (светло-серый цвет), и его соединениями с FB уровня 0 для аппаратного доступа (белый цвет)

19.4.3. Уровень 2: координирование компонентов

Координирующие компоненты на уровне 2 координируют работу элементарных компонентов на уровне 1. Рассматривая станцию в целом, были идентифицированы четыре аппаратных модуля: предметное стекло, манипулятор, конвейер и индикатор состояния.

19.4.3.1. Слайд

Слайд не нужно моделировать, поскольку он представляет собой просто физическую единицу, и с ним не связан ни один датчик или функция управления. На слайде хранятся вставки заготовок, и они транспортируются к

блоку захвата и размещения. Функция скольжения достигается с помощью наклона и изменения высоты, доступных в физическом дизайне.

19.4.3.2. Манипулятор

Компонент манипулятора отвечает за координацию горизонтальной оси, вертикальной оси и пневматического захвата нижележащего модуля, которые реализуются подпрограммами CylinerDAControl, CylinderSAControl и GripperControl соответственно. Чтобы координировать компоненты нижнего уровня, общие элементы интерфейса обоих уровней объединяются внутри адаптеров, как показано на рисунке 19.11. Соответствующие связи элементарных компонентов координируются адаптерами ACylinder, AGripperPneumatic и APartCount.

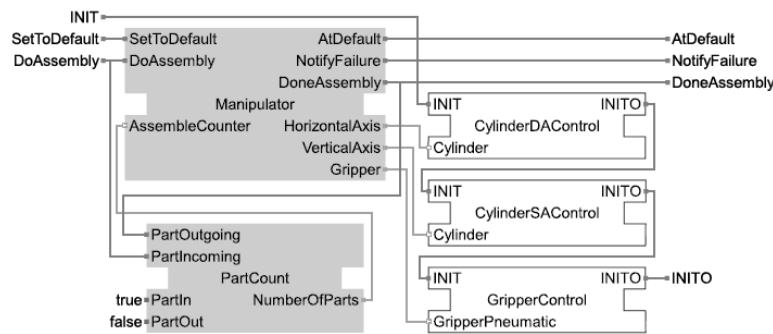


Рисунок 19.11. Уровень 2: Подпрограмма для управления манипулятором с помощью FB-манипулятора для всего элемента управления, сборка команд с подсчетом FB (серый цвет) и подключений к подпрограммам уровня 1 (белый цвет)

Помимо этих адаптеров, координатор содержит дополнительные входные события SetToDefault и DoAssembly, чтобы переместить манипулятор в положение по умолчанию и координировать процесс сборки соответственно. Для связи со следующим верхним уровнем выходные события AtDefault, DoneAssembly и NotifyFailure информируют верхний уровень о достижении позиции по умолчанию, выполнении процесса сборки или возникновении ошибки захвата. Компонент координатора подключен к экземпляру BFB PartCount, который подсчитывает поступающие команды сборки верхнего уровня, чтобы повторить процесс сборки.

19.4.3.3. Конвейер

Компонент конвейера отвечает за координацию базовых модулей для транспортировки и разделения, которые реализуются подпрограммами

TransportControl и SeparatorControl, как показано на рисунке 19.12. Поступающие части подсчитываются подпрограммами PartCountControl и передаются на компонент конвейера через соединение адаптера APartCount. Транспортный компонент активируется в любой входящей части путем отправки события MoveWPtoAssembly. Он останавливается, когда на ремне нет деталей, отправляя событие ConveyorEmpty. Компонент разделителя информирует компонент конвейера о деталях, которые поступают в разделитель и позицию сборки соответственно, отправляя событие WPatSeparator или WPatAssembly. Компонент конвейера решает, следует ли повернуть сепаратор в положение выпуска или ограничения. У компонента конвейера также есть выходное событие ReadyforAssembly, которое информирует верхний слой о том, что заготовка прибыла в положение сборки и готова к обработке.

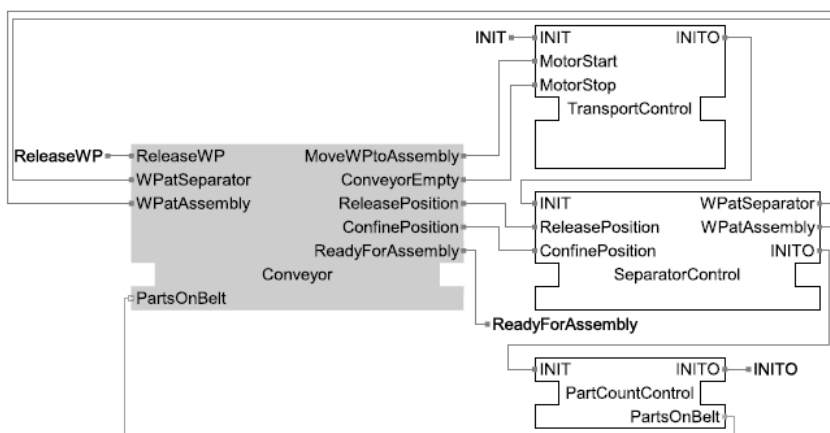


Рисунок 19.12. Уровень 2: Подпрограмма для управления конвейером с конвейером FB для всего элемента управления (серый цвет) и его соединения с подпрограммами слоя 1 (белый цвет)

Координация FB для конвейера реализована как BFB. Его ECC показан на рисунке 19.12. ECC состоит из двух переходных контуров. Состояния с двумя исходящими переходами имеют приоритет, как указано маленькими числами на переходах, где меньшие номера получают более высокий приоритет.

Небольшая петля слева на рисунке 19.12 останавливает ремень, когда на нем нет деталей. Этот переходный цикл вводится, когда адаптер APartCount отправляет событие обновления с номером детали, меньшим или равным нулю. Второй переходный цикл управляет процессом транспортировки и вводится, когда адаптер APartCount отправляет событие обновления с номером детали больше нуля. После достижения позиции сборки и отправки запроса на сборку на следующий верхний уровень он ожидает, пока запрос на выпуск не будет возвращен с верхнего уровня. Пока количество деталей на ленте больше

единицы, процесс транспортировки повторяется. Следовательно, координация конвейера BFB зависит от значений, предоставляемых PartCountControl FB.

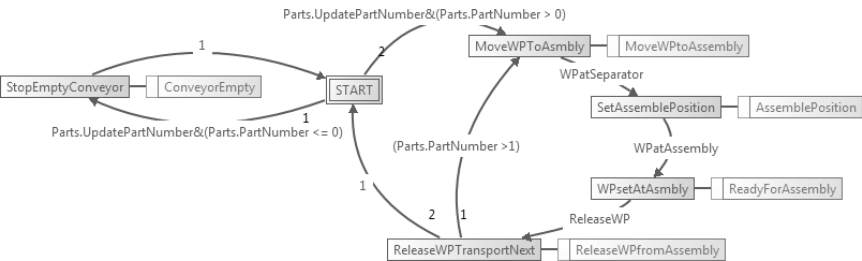


Рисунок 19.13. Уровень 2: ECC для координации подкомпонентов конвейера

19.4.3.4. Управление состоянием

Компонент контроля состояния координирует работу осветительной вышки в качестве индикации для станции, как показано на рисунке 19.14. Для координации красного, оранжевого и зеленого индикаторов был разработан FB StatusManagement. Во время инициализации всех аппаратных модулей мигает оранжевый индикатор.

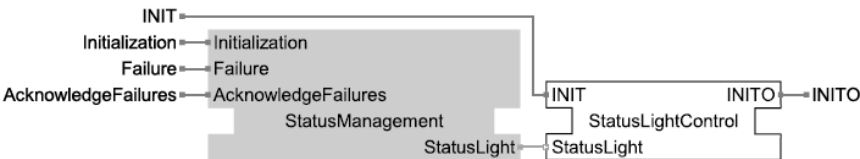


Рисунок 19.14. Уровень 2: управление индикатором FB (серый цвет) и его подключением к слою 1 FB (белый цвет)

После установки манипулятора в положение по умолчанию оранжевый индикатор сбрасывается, а зеленый светится, что указывает на то, что станция работает и может обрабатывать любую поступающую деталь. Если возникает ошибка, красный индикатор начинает мигать, пока она не будет подтверждена. Команды освещения отправляются на следующий нижний уровень через адаптер AStatusLight.

19.4.4. Уровень 3: компонент управления процессом

Общее управление процессом станции сбора и размещения осуществляется на уровне 3. FB PickNPlace координирует этот верхний слой и функционально напоминает станцию сбора и размещения, как показано на рисунке 19.15.

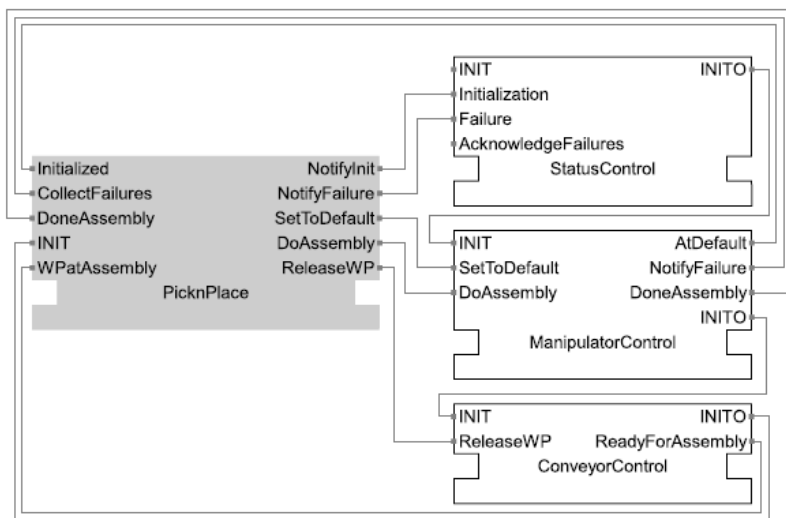


Рисунок 19.15. Слой 3: общий контроль станции сбора и размещения (серый цвет) и ее соединений с подпрограммами уровня 2 (белый цвет)

Компонент сбора и размещения реализован в виде BFB и координирует подпрограммы StatusControl, ManipulatorControl и ConveyorControl. Во время инициализации этот компонент координации инициализирует свои аппаратные модули, отправляя событие SetToDefault. Эта команда инициирует перемещение манипулятора в его положение по умолчанию и информирует компонент сбора и размещения после завершения, отправляя событие AtDefault. После инициализации подпрограмма ConveyorControl может автономно обнаруживать входящие детали, запускать конвейерную ленту и управлять сепаратором. Когда какая-либо заготовка достигла позиции сборки, вложенное приложение ConveyorControl информирует компонент координации, отправляя событие ReadyForAssembly. Компонент координации затем решает, должна ли заготовка быть освобождена или должна быть собрана, отправляя событие ReleaseWP или DoAssembly на конвейер или элемент управления манипулятора. В случае, если вставка заготовки не может быть извлечена из стека, элемент управления манипулятора информирует координирующий компонент, отправляя событие NotifyFailure. Координирующий компонент передает данные о сбое компоненту контроля состояния; сбой будет сигнализироваться красным мигающим светом, пока оператор не подтвердит его. Каждый компонент соответствует мехатронному

блоку и может функционировать самостоятельно с функциональной точки зрения.

19.5. Выводы

Путем иерархической структуризации управляющего приложения можно назначить каждому аппаратному компоненту один программный компонент для управления этим компонентом. Представленная система управления с иерархической структурой облегчает повторное использование компонентов и облегчает реконфигурацию. Внедренные программные компоненты напоминают аппаратные компоненты, и это, в принципе, позволяет осуществлять прямое аппаратно-программное сопоставление. Среди FB, разработанных для конкретного управляющего приложения, все компоненты вплоть до универсального модуля будут полностью повторно используемыми. Атомарные компоненты, такие как двигатели и электромагнитные клапаны, разработанные с помощью IX, QX FB, не зависят от применяемого аппаратного обеспечения и могут использоваться с любым типом конфигурации оборудования и в любом приложении независимо от области. Элементарные компоненты координируют атомарные компоненты и могут повторно использоваться в одной и той же области для аналогичных аппаратных компонентов. Например, цилиндр двойного действия FB может быть применен к любому типу цилиндра двойного действия для телескопических функций. Все элементарные компоненты могут рассматриваться одинаково. Предлагаемая структура также обеспечивает реконфигурируемость от атомарного уровня до уровня управления процессом. Проектирование иерархической структуры программного обеспечения иногда затруднительно, поскольку информация, необходимая для одного компонента, может потребоваться для другого компонента другой ветви в дереве иерархии. PartCount FB является типичным примером информации, требуемой в разных ветвях дерева иерархии, что привело к двум экземплярам этого FB: один для конвейера – для подсчета деталей на ленте – и один для манипулятора – для подсчета входящих команд сборки.

Требование для создания иерархически структурированных приложений – аппаратная независимость. Без аппаратной независимости компоненты уровня 0 не могут быть повторно использованы, и это влияет на интерфейсы компонентов более высокого уровня. Поэтому IX и QX FB были введены, чтобы обеспечить общий интерфейс для любого типа устройства и сделать аппаратное обеспечение приложения независимым. Назначение I/O для конкретного компонента иногда неоднозначно, так как один I/O может потребоваться в более, чем одном компоненте. Это касается датчиков присутствия на ленточном конвейере, которые представлены IX FB и могут быть назначены сепаратору и элементу транспортировки. Введение PartCount FB позволило нам инкапсулировать операции I/O, отслеживающие детали, и перенести функциональность координации на верхний уровень иерархии.

Компонент отслеживания деталей также улучшил иерархическое структурирование, так как теперь I/O PartIn и PartOut могут быть назначены точно одному FB, в то время как ECC конвейерного FB может быть упрощен из-за уменьшенного количества переходов.

Конфигурация FB IX и QX вызывает аппаратное соотношение, поскольку их значение параметра определяет вывод на PLC. Там, где есть несколько PLC, есть также несколько одинаковых контактов. FB IX и QX с одинаковыми номерами выводов не могут быть отображены на одном и том же PLC. Следовательно, нумерация выводов влияет на отображение на конкретном ресурсе. Расширенное описание устройства в терминах I/O и сетевых интерфейсов позволило бы дать однозначные символические имена для каждого I/O, которые, в свою очередь, могли бы быть выбраны в раскрывающемся списке в FB IX и QX после отображения.

Распределенные приложения требуют скоординированной инициализации. Порядок инициализации распределенных FB важен для их правильной инициализации. Это особенно касается коммуникационных FB, таких как PUBLISH и SUBSCRIBE. SUBSCRIBE необходимо инициализировать в первую очередь для получения желаемых значений инициализации от соответствующих блоков PUBLISH. Следовательно, порядок инициализации FB определяет порядок загрузки ресурсов. В приложении для сбора и размещения на индикатор состояния влияет порядок инициализации. Из-за ограниченного числа операций I/O в компактном PLC управление индикатором состояния выполняется в другом PLC, отличном от остальной части приложения. Блоки SUBSCRIBE, работающие на PLC управления индикаторами состояния, должны получить значения инициализации от блоков PUBLISH, работающих на другом PLC.

Для совместной разработки, а также для обслуживания приложений IEC 61499, соглашения об именах были бы полезны. Правильное именование FB и интерфейсных элементов помогает прояснить назначение элемента и указать, как разные FB должны быть подключены. Результатами полезных соглашений об именах для приложений IEC 61499 могут быть:

- именование FB в соответствии с их функциональностью или соответствующими аппаратными компонентами;
- именование любого логического элемента в соответствии с его назначением (например, как-нибудь назовите выход для извлечения цилиндра, но не InOut);
- использование эквивалентных имен для событий разных FB, которые должны быть подключены.

Поддержка отладки в инструментах разработки в области автоматизации очень ограничена. Внедрение IEC 61499 4DIAC обеспечивает мониторинг событий, а также изменений значений данных. Инженерные инструменты IEC 61131, такие как logi.CAD и CoDeSys V2.3, обеспечивают поддержку трассировки для отслеживания изменений значений во времени. Для эффективной отладки приложения, управляемого событиями, в соответствии с IEC 61499, ни мониторинга, ни отслеживания недостаточно. Приложения,

которые содержат BFB, требуют механизма отладки, который следует за шагами выполнения с точки зрения произошедших событий и введенных состояний.

Библиография

[1] F. Ebel and M. Pany. Pick&Place Station Manual. Technical report, Festo Didactic GmbH & Co. KG, 2006.

[2] I. SC65B. IEC 61499-1: Function blocks for industrial process measurement and control systems. Part 1: Architecture, 2005. Geneva.

[3] C. Sünder, A. Zoitl, and C. Dutzler. Functional structure-based modeling of automation systems. *International Journal of Manufacturing Research*, Vol. 1, No. 4, 405–420, 2006.

[4] A. Zoitl and H. Prähofer. Guidelines and patterns for building hierarchical automation solutions in the iec 61499 modeling language. *IEEE Transactions on Industrial Informatics*, Vol. 9, No. 4, 2387–2396, 2013.

[5] A. Zoitl, T. Strasser, and A. Valentini. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study. In *IEEE International Symposium on Industrial Electronics, Bari*, pages 3817–3819, 2010.

На пути к пакетной обработке с IEC 61499

Wilfried Lepuschitz

Практический институт робототехники Австрии

Alois Zoitl

fortiss GmbH

СОДЕРЖАНИЕ

20.1 Введение	489
20.2 Пакетное управление ANSI/ISA-88	490
20.2.1 Структурные модели	491
20.2.2 Конечный автомат	492
20.2.3 Поправки и область интересов	493
20.3 Подход 1: иерархическая структура на основе автоматизации компонентов	494
20.3.1 Концепция подхода	494
20.3.2 Реализация с IEC 61499	496
20.4 Подход 2: реализация конечного автомата S88	499
20.4.1 Концепция подхода	499
20.4.2 Реализация с IEC 61499	500
20.5 Подход 3: общая структура фаз S88 и соединение с промышленной системой для управления партиями	502
20.5.1 Концепция подхода	503
20.5.2 Реализация с IEC 61499	505
20.6 Выводы	506

20.1. Введение

Промышленные отрасли, в которых применяются повторяющиеся процессы, существует определенная степень гибкости благодаря использованию модифицируемых наборов команд. Однако применяемые системы управления обычно основаны на централизованных концепциях, что неизбежно усложняет модификации предоставляемых функций оборудования. Большая гибкость может быть достигнута путем применения подхода распределенного управления со структурами, подобными физическому

оборудованию. Таким образом, могут быть достигнуты объекты, которые инкапсулируют физическое оборудование с различными функциональными возможностями и информацией. Эти объекты затем могут быть объединены для реализации функциональности на более сложном уровне, и дополнительные функции могут быть добавлены путем интеграции дополнительных объектов [1].

IEC 61499 – это подходящий стандарт для реализации программных аспектов таких объектов. Для применения подходов, основанных на этом стандарте, в промышленности необходимо соблюдение других соответствующих стандартов. Что касается автоматизации циклического процесса, то важен стандарт ANSI/ISA-88 Batch Control (S88) и его аналог IEC 61512 [4].

В этой главе обобщены работы, выполненные в отношении применения стандарта IEC 61499 для лабораторной установки периодического действия [6, 7, 8]. Три подхода на основе IEC 61499 для автоматизации циклического процесса были разработаны и внедрены на лабораторной технологической установке. Эти подходы представляют собой эволюционную работу по приближению IEC 61499 к промышленной практике в области периодического процесса. Прежде чем объяснить реализованные подходы, в разделе 20.2 дается обзор S88 и его использования на практике. Первый подход описан в разделе 20.3 и применяет стандартизированную архитектуру для компонентов на разных иерархических уровнях в области пакетного процесса. Для большей совместимости с промышленными системами управления партиями второй подход, представленный в разделе 20.4, основан на конечном автомате, как определено в S88 внутри функциональных блоков. В разделе 20.5 представлен третий подход, который преодолевает ограничения второго подхода, представляя более общий дизайн для конечного автомата и его применения, что приводит к интеграции с промышленной системой управления партиями. Наконец, эта глава заканчивается краткими выводами.

20.2. Пакетное управление ANSI/ISA-88

Применяемые в настоящее время системы управления партиями используют S88 в качестве основы для своих прикладных структурных моделей и, таким образом, достигают определенной степени сопоставимости и функциональной совместимости [5]. В S88 описана «структура для спецификации требований к управлению пакетными процессами и их последующей трансляции в прикладное программное обеспечение» [9]. Модификации и расширения введенных моделей явно разрешены, если обеспечивается согласованность определенных моделей.



Рисунок 20.1. Структурные модели S88 и их взаимосвязи [4]

20.2.1. Структурные модели

Стандарт вводит набор структурных моделей для организации процессов, физического оборудования и управляющего программного обеспечения. Как правило, все эти структурные модели содержат четыре значимых уровня:

- модель процесса: процесс, стадия процесса, операция процесса, действие процесса;
- физическая модель: технологическая ячейка, блок, модуль оборудования, модуль управления;
- процедурная модель управления: процедура, процедура установки, операция, фаза.

Отображение элементов модели процедурного управления на физическую модель обеспечивает объекты оборудования, которые реализуют функциональные возможности процесса (см. рисунок 20.1). Например, отображение фазы на модуль оборудования реализует объект оборудования, который обеспечивает действие процесса. Более того, концепция наборов команд описана в S88 как средство для выполнения процесса. Различают четыре типа наборов команд: общий набор команд, набор команд сайта, главный набор команд и контрольный набор команд. Общий набор команд и

набор команд сайта составлены независимо от информации об оборудовании и структурированы в соответствии с моделью процесса. И наоборот, главный набор команд и контрольный набор структурированы в соответствии с процедурной моделью управления. Главный набор команд определяет типы необходимых объектов оборудования, в то время как контрольный набор определяет отдельные объекты оборудования, которые будут использоваться для производства. На основе этой информации в наборе команд для управления система управления партиями может активировать соответствующее программное обеспечение системы управления устройств, обычно расположенных на программируемых логических контроллерах (PLC). Такое связывание системы управления партиями с управляющим программным обеспечением может быть выполнено на любом уровне, но большинство применяемых в промышленности систем осуществляют это связывание на фазовом уровне [9]. Для этой взаимосвязи используется так называемый фазовый логический интерфейс, который преобразует последовательность команд для управления в команды для управляющего программного обеспечения. Информация о состоянии из управляющего программного обеспечения возвращается в систему управления партиями с использованием этого интерфейса.

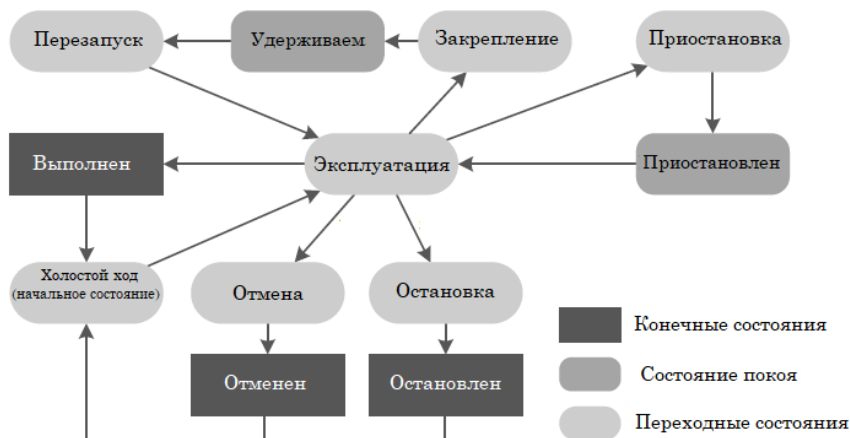


Рисунок 20.2. Состояния и переходы конечного автомата S88 [4]

20.2.2. Конечный автомат

S88 предоставляет набор рабочих состояний и их переходов для объектов оборудования и процедурных элементов (см. рисунок 20.2). Также в стандарт введены команды для изменения состояния в этом автомате. Этот конечный автомат может быть уменьшен или расширен в соответствии с требованиями

приложения и применен на любом иерархическом уровне. Промышленные системы управления партиями, такие как FactoryTalk Batch [10] и Zenon Batch Control [3], связывают наборы команд для управления на уровне фазы с программным обеспечением для управления. Как следствие, конечный автомат должен быть реализован только на этом уровне в PLC. Более высокие уровни управляющего программного обеспечения не требуются, так как эти уровни контролируются системой управления партиями.

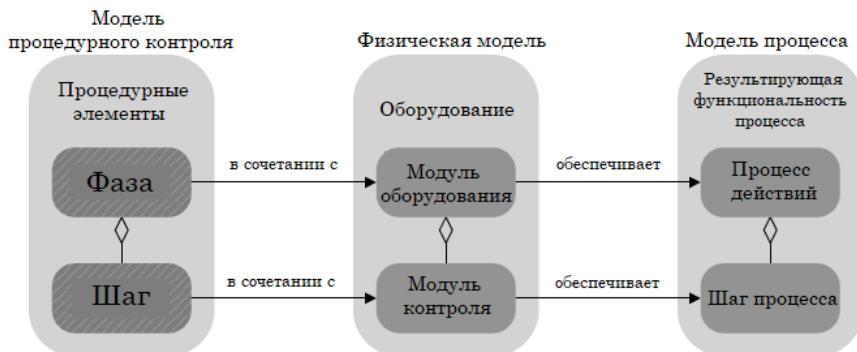


Рисунок 20.3. Область интереса в отношении управляющего программного обеспечения представленных подходов

20.2.3. Поправки и область интересов

Как видно на рисунке 20.1, элементы различных уровней модели процедурного управления отображаются на физическую модель для достижения элементов уровня процесса. Однако уровни трех моделей не полностью равны, что можно увидеть на уровне единиц физической модели. Очевидно, что процедурные элементы трех разных уровней могут быть отображены на блоки для достижения разных уровней функциональности процесса. Также фаза процедурной модели управления может быть отображена на оба блока или модули оборудования. Тем не менее модуль управления не появляется в отношениях между моделями.

В стандарте указано, что структурные модели могут быть уменьшены или расширены при условии обеспечения согласованности моделей. Кроме того, согласно стандарту, фаза может быть разделена на этапы и переходы. Эта дальнейшая агрегация подчеркивается в представленных подходах, и этот шаг представлен как самый низкий уровень модели процедурного контроля в оставшейся части этой главы. Такой шаг может быть отображен на модуль управления для достижения функциональности процесса на очень низком уровне. Таким образом, модель процесса должна быть расширена на шаг процесса как самый низкий уровень для повышения согласованности во всех структурных моделях.

Как упоминалось ранее, промышленные системы управления партиями связывают наборы команд для управления с управляющим программным обеспечением на уровне фазы. В результате область интересов для подходов, касающихся управляющего программного обеспечения, охватывает только фазовый уровень и недавно введенный уровень шага (см. рисунок 20.3).

20.3. Подход 1: иерархическая структура на основе автоматизации компонентов

Естественно, техническое оснащение объекта может быть структурировано. Отдельные машины можно подразделить на датчики и исполнительные механизмы или объединить в группы. Такое объединение может быть расширено до полного производственного объекта. Преобразование такой структуры из физической модели в управляющую структуру облегчает разработку последней, обеспечивая естественную структуру для инкапсуляции функциональности. Это подчеркивает возможность многократного использования программных компонентов и делает их заменяемыми, если их замены предлагают требуемую функциональность и интерфейсы для соответствия необходимым спецификациям.

20.3.1. Концепция подхода

Подход, описываемый в этом разделе, представляет собой иерархическую систему управления, состоящую из устройств со стандартизированной архитектурой. Наличие одинаковой архитектуры для всех компонентов независимо от их положения в системе облегчает использование этого подхода. В принципе, эти компоненты автоматизации (АС) состоят из трех типов подкомпонентов [12]:

- датчики и исполнительные механизмы для сопряжения процесса;
- устройство управления с интерфейсами к датчикам, исполнительным механизмам и сети;
- программные компоненты, размещенные на устройстве управления с данными и логикой управления для выполнения функций автоматизации.

Согласно Sünder и соавторам, каждый программный компонент может быть далее разделен на следующие элементы [11].

Логика: эта программная часть включает в себя основные функциональные возможности управления АС, объединяющего входы и состояния внутреннего устройства для генерации соответствующих выходов..

Диагностика: диагностические алгоритмы обнаруживают ошибки.

Универсальный компонентный интерфейс (UCI): UCI обеспечивает связь с высокоуровневыми и внешними компонентами.

Человеко-машинный интерфейс (HMI): HMI используется для взаимодействия между операторами и АС и представляет собой внешний

компонент, требующий доступа к UCI. Это не центральная часть подхода, и поэтому о ней не будем говорить подробно.

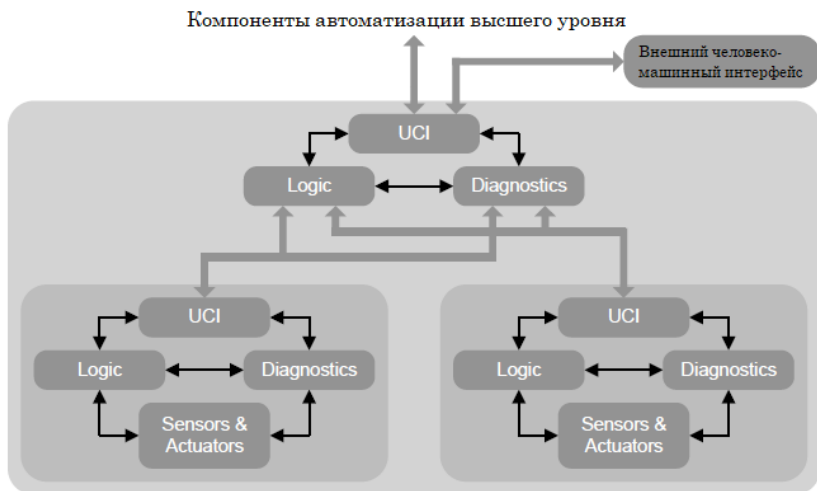


Рисунок 20.4. Архитектура агрегированного компонента автоматизации, включающего два основных компонента автоматизации [11]

Согласно Dutzler, можно выделить два типа АС [2]: базовые АС и агрегированные АС (см. рисунок 20.4). Базовые АС, обозначенные как интеллектуальные сенсорные или исполнительные устройства, напрямую взаимодействуют с процессом через свои логические и диагностические компоненты, которые имеют доступ к физическому оборудованию компонента. Некоторые диагностические алгоритмы выполняются независимо от компонентов более высокого уровня. Например, комбинация шарового клапана с двумя датчиками для проверки положения клапана обеспечивает базовый АС с диагностическими возможностями. Шаровой клапан управляется логической частью для обеспечения функциональности компонента включения или предотвращения потока жидкости. Когда шаровому крану требуется изменить его состояние, диагностическая часть выполняет проверку положения клапана с помощью датчиков. Если датчики не показывают изменение положения клапана, как требуется по истечении заданного промежутка времени, событие ошибки выдается объекту более высокого уровня через UCI. Что касается S88, базовые АС представляют объекты оборудования на уровне модуля управления, которые представляют функциональные возможности на уровне этапа процесса.

Комбинация АС поставляет агрегированные АС. Инкапсулируя АС более низкого уровня, которые могут быть базовыми или агрегированными АС, можно добиться более абстрактной и сложной функциональности. Части логики и диагностики такого агрегированного АС связываются с UCI инкапсулированных АС и предоставляют UCI для других объектов более

высокого уровня. Кроме того, агрегированный АС может включать дополнительные функции в свои программные компоненты. Например, объект управления потоком, реализованный как агрегированный АС, содержит два основных АС: насос и датчик потока. Алгоритм управления в логике управления потоком получает требуемую уставку через UCI от объекта более высокого уровня и комбинирует ее с информацией от инкапсулированного датчика АС переменного тока о текущем значении. Следовательно, алгоритм управления создает значение управления для герметичного насоса АС. Также АС управления потоком может быть агрегирован с другими АС, чтобы сформировать агрегированный АС более высокого уровня. Наконец, даже целый завод можно рассматривать как агрегированный АС. Что касается S88, агрегированные АС представляют объекты оборудования на уровне оборудования или выше, реализуя функциональность на уровне действия процесса или выше.

Для реализации этого подхода необходимо создать библиотеку компонентов, содержащую программные компоненты АС для всех необходимых иерархических уровней. Для максимизации возможности повторного использования и гибкости программные компоненты должны быть разработаны как можно более независимыми от оборудования. Затем их можно использовать для преобразования набора команд для управления в исполняемое приложение, в результате чего получится система управления, состоящая из АС, которые образуют иерархическую структуру.

20.3.2. Реализация с IEC 61499

Реализация программных компонентов АС в IEC 61499 означает реализацию логики, диагностики и UCI элементов программного обеспечения с помощью функциональных блоков (FB). Компоненты программного обеспечения должны быть сопоставлены с физическими объектами для достижения полных АС.

В представленном подходе каждый из элементов программного обеспечения реализован с помощью составного FB. Принципиальное наличие только трех FB обеспечивает четкую компоновку сети FB каждого компонента (см. рисунок 20.5). Следует отметить, что объединение АС в АС более высокого уровня выполняется только на функциональном уровне. Это очевидно, поскольку к UCI АС более низкого уровня необходимо обращаться как с помощью логики, так и диагностики АС более высокого уровня, что приведет интерфейсы в составных FB логики и диагностики к UCI более низкого уровня (см. рисунок 20.6). Кроме того, эта конструкция предлагает возможность замены АС более низкого уровня без необходимости адаптации превосходящего его АС, если не требуется модифицировать соответствующие интерфейсные FB в пределах вышестоящего АС. Следовательно, замена, например, магнитного клапана шаровым клапаном и, соответственно, его

программных компонентов не требует адаптации программного компонента превосходящего АС, так как интерфейс аналогичен для обоих типов клапанов.

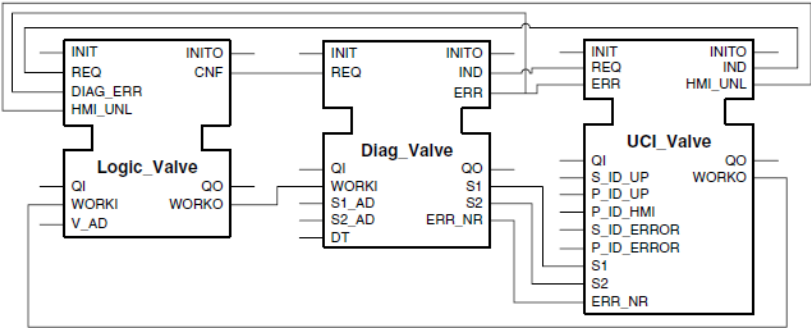


Рисунок 20.5. Функциональный блок сети ступени для управления клапаном

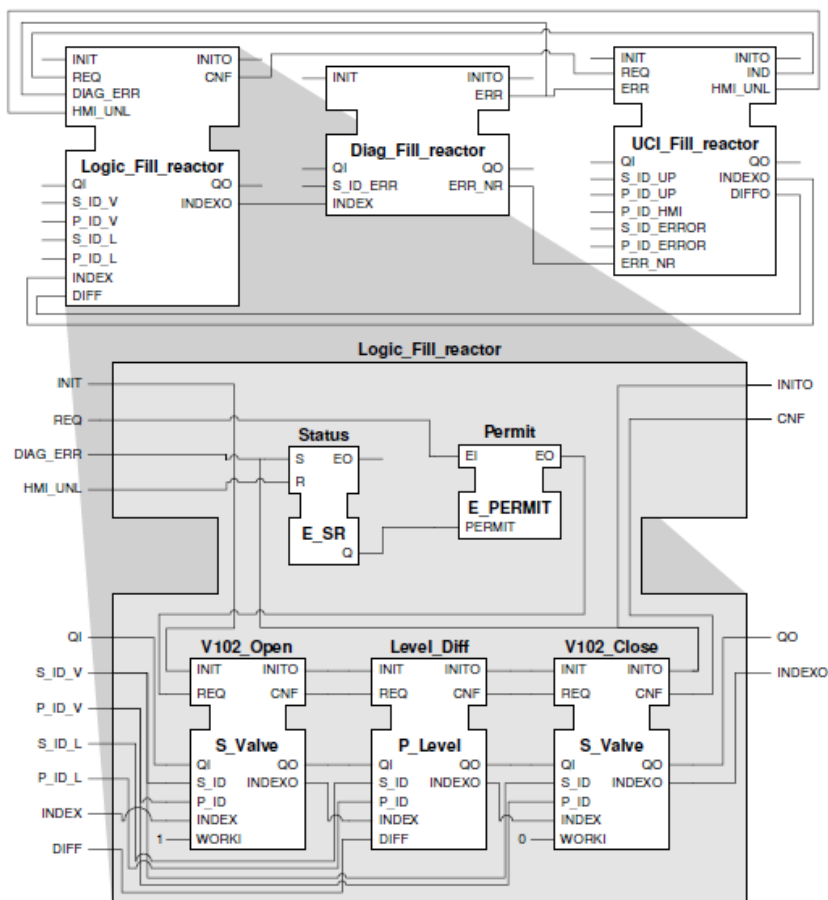


Рисунок 20.6. Сеть функциональных блоков фазы для заполнения реактора и подробное представление его логики с функциональными блоками интерфейса для шагового уровня UCI

Преобразование контрольного набора команд в приложение осуществляется путем определения требуемых АС в соответствии с процедурными элементами набора. Приложение формируется путем упорядочения их деятельности, что теоретически может быть сделано на любом уровне иерархических структур. Однако создание приложения на самом низком уровне требует набора команд для управления, включающего каждый отдельный шаг, который требует значительных усилий со стороны инженера-технолога. Напротив, если преобразование выполняется на самом высоком уровне, набор команд для управления будет вызывать только одну конкретную процедуру. Следовательно, необходимо предусмотреть процедурные элементы

на всех уровнях, которые включают процедуры для всех мыслимых наборов команд. Это, очевидно, невыполнимая задача. Следовательно, преобразование набора команд для управления в приложение является разумным на фазовом уровне и на рабочем уровне. Следовательно, требуются элементы до уровня фазы или операции.

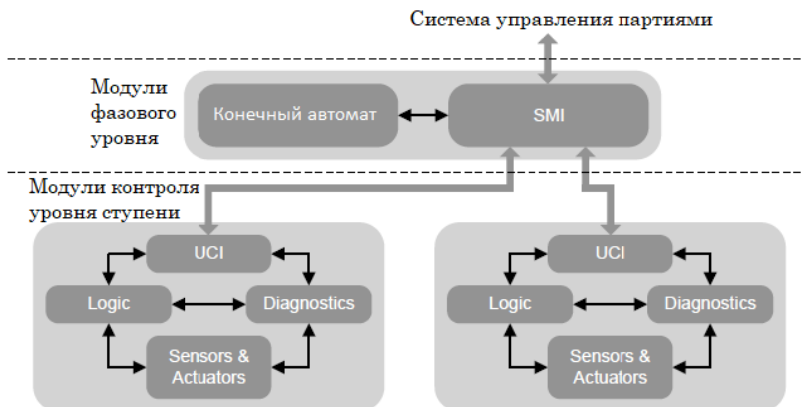


Рисунок 20.7. Программное обеспечение для управления модулем оборудования, включающим модули управления

20.4. Подход 2: реализация конечного автомата S88

Ранее упомянутый подход годится для преобразования набора команд для управления в исполняемое приложение. Однако он не учитывает соответствие промышленных систем управления партиями. FactoryTalk Batch [10] и Zenon Batch Control [3] предлагают протоколы для создания интерфейса фазовой логики. В соответствии с названием этого интерфейса, система управления партиями и программное обеспечение фактического контроля подключены на фазовом уровне. Для установления соединения на фазовом уровне требуется программное обеспечение системы управления для включения конечного автомата S88. Это явно отсутствует в подходе, представленном в предыдущем разделе.

20.4.1. Концепция подхода

На рисунке 20.7 показана структура управляющего программного обеспечения модуля оборудования. Уровень шага организован аналогично иерархическому подходу, описанному выше. Программное обеспечение системы управления на фазовом уровне состоит из следующих двух элементов:

- конечный автомат: этот конечный автомат S88 включает в себя рабочие состояния, их переходы и алгоритмы для обеспечения функциональности модуля оборудования;

- интерфейс конечного автомата: этот интерфейс используется для связи с системой управления партиями и с соседними модулями управления.

Программное обеспечение системы управления на этом уровне отвечает за управление модулями оборудования, которые включают в себя один или несколько модулей управления. Функциональность на этом уровне представляет собой сочетание функциональности встроенных модулей управления с дополнительными функциями. Интеграция конечного автомата S88 для каждого модуля оборудования обеспечивает соответствие стандарту S88.

20.4.2. Реализация с IEC 61499

Программные элементы на уровне шага реализуются в соответствии с ранее описанным подходом с одним отдельным FB (для логики, диагностики и UCI), также элементы на фазовом уровне реализуются с одним FB. Подход объясняется на примере модуля оборудования для контроля температуры в реакторной емкости. Задача этого модуля состоит в том, чтобы нагреть жидкость в реакторе до определенной температуры и поддерживать ее до тех пор, пока не произойдет событие остановки.

Как упоминалось ранее, программное обеспечение контроля уровня фазы включает в себя конечный автомат S88. Он реализован в базовом FB (см. рисунок 20.8) и содержит дополнительные алгоритмы для модуля оборудования. В приведенном примере алгоритмы охватывают контроль температуры и обработку ошибок. Ошибки могут указываться либо внутренними алгоритмами конечного автомата, либо смежными модулями управления. Конечный автомат реализован в виде состояний в диаграмме управления выполнением (ЕСС) FB и содержит следующие рабочие состояния и алгоритмы для данного примера (см. рисунок 20.9).

- Initializing (инициализация): это состояние устанавливает для внутренних переменных начальные значения.

- Idle (холостую): внутренняя переменная сбрасывается.

- RunInit: для инициализации контура управления заданное значение сохраняется и запрашивается первое текущее значение из модуля датчика температуры.

- Running (эксплуатация): на основе заданного значения и текущего значения температуры выдается событие для включения или выключения нагревателя.

- GetTemp: текущее значение температуры собирается и сравнивается с ранее собранным значением. Если нагреватель активен, а новое значение ниже, чем старое, должен быть обнаружен дефект нагревателя и установлен флаг ошибки.

- Paused (приостановлено): нагреватель выключен до возобновления процесса.
- Stopped (остановлен): нагреватель выключен, а контур регулирования температуры остановлен.
- E1: это состояние ошибки вводится, если модуль нагрева сообщает об обнаружении недостаточного количества жидкости в реакторе для безопасного процесса нагрева.
- E2: это состояние ошибки вводится, если в реакторе достигается критически высокая температура.
- E3: это состояние ошибки вводится, если нагреватель поврежден.

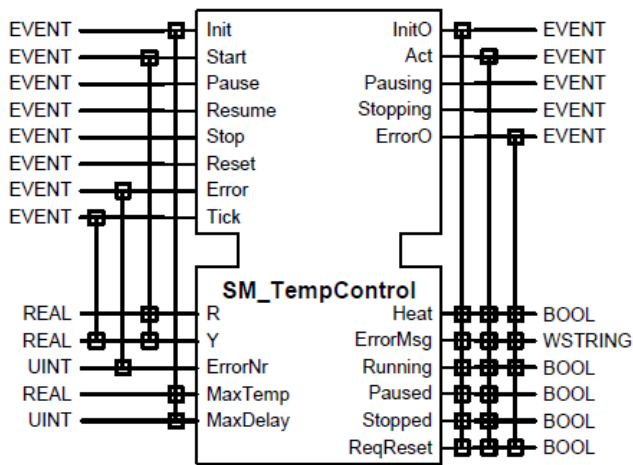


Рисунок 20.8. Основной функциональный блок, представляющий конечный автомат модуля оборудования для контроля температуры

Переходы в состояния ошибки инициируются внутренними флагами ошибок или событиями ошибок из соседних модулей управления. Для устранения ошибок требуется событие сброса, которое может выдать человек-оператор или система управления партиями. Выход из состояния «Остановлено» требует события сброса для перевода конечного автомата в состояние «Ожидание».

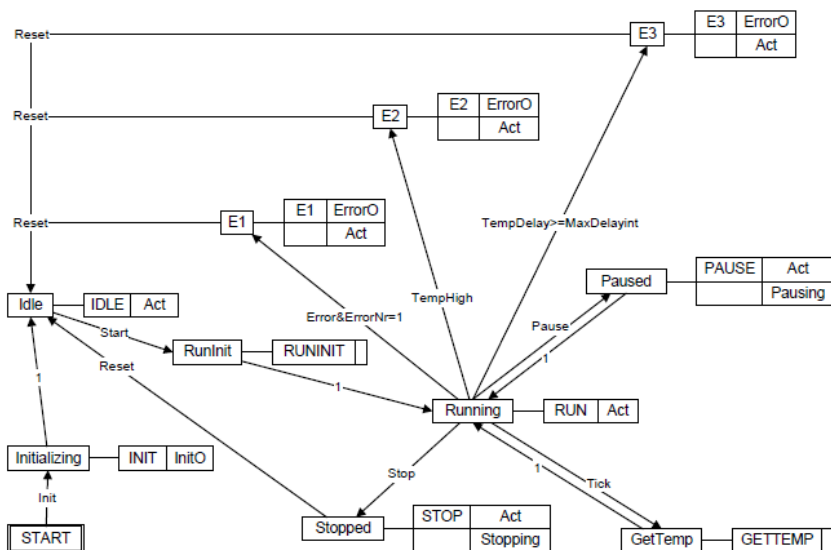


Рисунок 20.9. ЕСС реализует конечный автомат модуля оборудования для контроля температуры

Описанный конечный автомат не содержит всех состояний, как определено в стандарте S88, что является разумным, поскольку различные задачи и компоненты могут требовать разных реализованных состояний. В качестве примера: состояние «Остановка», определенное на этапе S88 для выполнения процедуры безопасной остановки оборудования, не реализовано для модуля контроля температуры. Выключение обогревателя достигается простым отключением соответствующего цифрового выхода контроллера.

Как следствие, остановка означает прямой переход из состояния «Работа» в состояние «Остановлено» путем выключения нагревателя.

20.5. Подход 3: общая структура фаз S88 и соединение с промышленной системой для управления партиями

Предыдущий подход показывает осуществимость IEC 61499 для реализации управляющего программного обеспечения на фазовом уровне с интеграцией конечного автомата S88. Однако в этом подходе не учитываются протоколы связи систем управления партиями разных поставщиков. Большинство систем управления партиями связываются с управляющим программным обеспечением с помощью тегов в PLC. В FactoryTalk Batch существует протокол командного рукопожатия, и на каждой фазе есть тег команды для передачи команд с пакетного сервера на эту фазу. Команды представлены числами и инициируют изменение состояния конечного

автомата фазы. Кроме того, протокол сбоя используется для сбора уведомлений о сбоях, а сторожевой протокол проверяет жизнеспособность связи между пакетным сервером и фазами.

Подход, который должен быть интегрирован с промышленной системой управления партиями, должен обрабатывать такие протоколы соответствующим образом. Этот подход должен обеспечить средства для интеграции IEC 61499 в промышленную систему управления партиями.

20.5.1. Концепция подхода

Программное обеспечение управления на уровне шага организовано в соответствии с ранее упомянутыми подходами, включающими логику, диагностику и UCI. Однако программное обеспечение управления фазовым уровнем должно быть структурировано для выполнения следующих задач:

- связь с системой управления партиями;
- расшифровка и интерпретация команд из системы управления партиями;
- включение конечного автомата S88 для соответствия системе управления партиями;
- управление смежными управляющими модулями на уровне шага для реализации фактической функциональности процесса;
- обработка ошибок.

Эти задачи могут быть легко решены с использованием принципа «разделения интересов», что приводит к четкой структуре прикладной программы управления, как показано на рисунке 20.10. Каждое задание выполняется одним отдельным программным элементом.

Доступ к тегам в PLC обычно достигается драйверами в системе управления партиями. Тем не менее общая таблица тегов не существует в концепции распределенного управления, основанной на IEC 61499, и доступно несколько подходящих драйверов для связи на основе сообщений. Как следствие, связь осуществляется с использованием сервера OPC (см. рисунок 20.11). Функциональность для доступа к серверу OPC обеспечивается универсальным фазовым интерфейсом (UPI). Он может читать и записывать теги на сервере OPC и, следовательно, может связываться с системами управления партиями.

Системы управления партиями разных производителей полагаются на различные методы кодирования информации, такие как команды от системы управления партиями к этапам. Обработчик команд отвечает за декодирование команд, полученных UPI, проверяет их достоверность и, таким образом, включает в себя протокол командного рукопожатия для обеспечения правильной связи. В случае действительной команды инициируется соответствующее действие, такое как изменение состояния фазы. В случае неверной команды обработчик ошибки уведомляется.



Рисунок 20.10. Структура программного обеспечения управления фазой

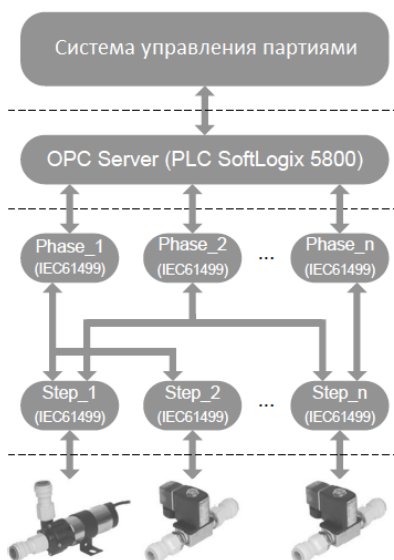


Рисунок 20.11. Архитектура реализации с использованием OPC-сервера для коммуникации

Конечный автомат реализует конечный автомат S88, аналогичный упомянутому ранее подходу, но в более общем виде и, таким образом,

полностью соответствует стандарту. Команды для изменения состояния принимаются от обработчика команд и подтверждаются отправкой нового значения состояния в UPI для пересылки в систему управления пакетами. Изменение состояния обычно требует, чтобы действующие модули управления также работали. Следовательно, управление фазой уведомляется для выдачи команд на уровень шага.

Связь с уровнем шага осуществляется посредством управления фазой. Операции контроля фазы запускаются конечным автоматом на основе текущего состояния, и подтверждение их завершения отправляется обратно. Команды отправляются в модули управления для побуждения действий физического оборудования, и получают уведомления об их успешном завершении. Сообщения об ошибках также могут быть получены от модулей управления.

В случае ошибки обработчик сбоя генерирует соответствующее сообщение об ошибке, которое отправляется в UPI для пересылки в систему управления пакетами. Ошибки могут возникать из-за недопустимых команд или сбоев физического оборудования, обнаруженного модулями управления.

20.5.2. Реализация с IEC 61499

Каждый из элементов программного обеспечения для управления фазовым уровнем реализован с одним FB (см. рисунок 20.12). UPI – это составной FB, содержащий набор FB интерфейса службы для реализации связи с сервером OPC. После инициализации в качестве параметров необходимо указать имя сервера и адреса элементов для доступа к тегам на сервере. Когда новая команда считывается с сервера, ее значение отправляется в обработчик команд вместе с соответствующим событием. Путем обмена реализацией UPI с другим могут быть реализованы другие средства связи, чем OPC.

Обработчик команд – это базовый FB, включающий два основных алгоритма. Они выполняются последовательно каждый раз, когда от UPI поступает новая команда. Команда проверяется на достоверность и на основании ее значения генерируется соответствующее событие, которое отправляется конечному автомату для запуска изменения состояния. В случае неверной команды обработчик ошибки уведомляется. Впоследствии UPI определит значение для записи на сервер для уведомления системы управления пакетами об успешном считывании команды. Таким образом, протокол командного рукопожатия реализован. Этот FB представляет драйвер для обработки кодирования состояний и команд отдельной системы управления партиями. Как следствие, различные реализации обработчика команд позволяют соединяться с системами управления партиями других поставщиков.

Еще один базовый FB представляет конечный автомат. Его ECC содержит состояния и переходы в соответствии с конечным автоматом S88. Переходы инициируются событиями, полученными от обработчика команд, или с

помощью управления фазой, когда модуль управления завершает действие. После достижения нового состояния в ECC событие управления фазой выдается, и значение состояния отправляется в UPI для записи на сервер OPC.

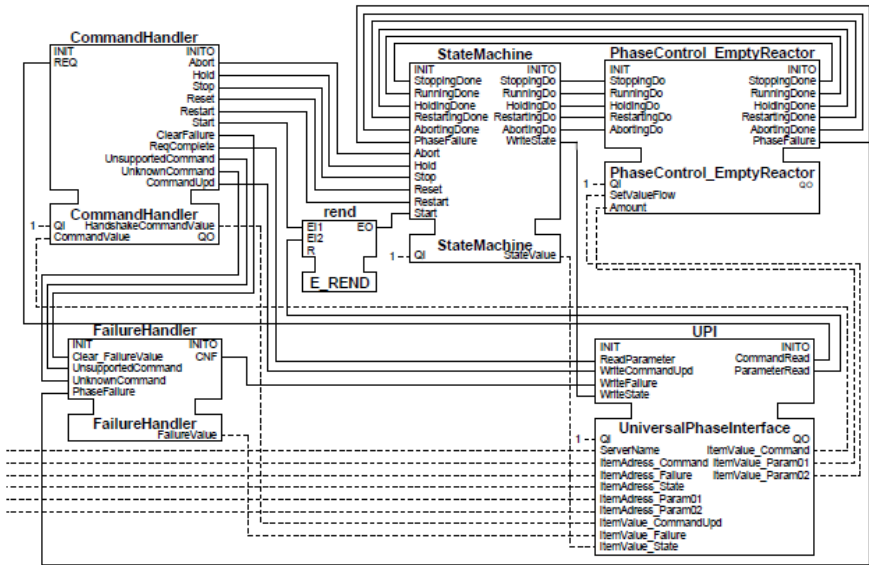


Рисунок 20.12. Сеть функциональных блоков для фазы (строки события инициализации не отображаются для большей наглядности)

Управление фазой представляет собой составной FB, содержащий FB интерфейса службы для связи с соседними модулями управления и дополнительные FB для включения алгоритмов координации и поддержки (например, алгоритм для замкнутого контура управления). Таким образом, фактическая функциональность модуля оборудования реализована. Команды на уровне шага инициируются событиями, полученными от конечного автомата, и соответствующие события возвращаются после уведомления о завершении действия. Если модуль управления сообщает об ошибке, событие генерируется и отправляется обработчику ошибок. Обработчик ошибок – это базовый FB, который получает уведомления об ошибках и генерирует соответствующее значение ошибки. Затем он отправляется в UCI для записи на сервер OPC.

20.6. Заключение

В этой главе была представлена эволюционная работа по интеграции IEC 61499 для области пакетного процесса. Стандарт S88 имеет большое значение и должен быть принят во внимание для приближения IEC 61499 к

промышленной практике. В этом контексте уместно показать возможность интеграции IEC 61499 с системами более высокого уровня, такими как управление партиями.

В этой главе были описаны три подхода. Первый фокусировался на использовании стандартизированной архитектуры для компонентов на различных уровнях в области процессов. Несмотря на то, что этого недостаточно для интеграции с системой управления партиями, данная архитектура компонентов может использоваться для модулей управления, то есть для самых низких физических объектов в установке для обработки партий.

Уровень фазы был определен как уровень для связывания контрольного набора команд и управляющего программного обеспечения. S88 представляет конечный автомат, и эта концепция необходима на уровне фаз для обеспечения соответствия системам управления партиями. Как следствие, дальнейший подход представил конечный автомат на этом уровне в форме базового FB. В этом подходе отсутствовала реализация необходимых протоколов для связи с такой системой.

Наконец, третий подход представил концепцию управляющего программного обеспечения на фазовом уровне с большей гибкостью его программных элементов. Однако подход может быть адаптирован в соответствии со средствами связи, а также с кодированием состояний и переходов системы управления партиями. В качестве примера была представлена интеграция IEC 61499 с FactoryTalk Batch от Rockwell Automation. Для связи с этой системой управления партиями был использован сервер OPC.

Последний подход показал осуществимость IEC 61499 для интеграции с промышленной системой управления партиями. Подход является общим на фазовом уровне при сохранении гибкости на шаговом уровне, поскольку он основан на объектах с инкапсулированной функциональностью.

Дальнейшие исследования могут улучшить интеграцию с другими системами управления партиями. Следовательно, связывание на основе тегов существующих систем управления партиями с фазами можно было бы избежать путем реализации соответствующих драйверов для связи на основе сообщений.

Библиография

[1] R. W. Brennan. Toward real-time distributed intelligent control: A survey of research themes and applications. *IEEE Transactions on Systems Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 37, No. 5, 744–765, 2007.

[2] C. Dutzler. *A Modular Distributed Control Architecture for Industrial Automation Systems – Designed for Holonic Systems and Vertical Integration*. PhD thesis, Vienna University of Technology, Austria, 2003.

[3] Ing. Punzenberger COPA-DATA GmbH. zenon Batch Control.

<http://www.copadata.com/en/products/food-beverage/isa-88-compatible-batch-control-in-zenon.html>.

[4] International Electrotechnical Commission. *IEC 61512 Batch Control. Parts 1-4*. Geneva, 2000–2010.

[5] S. Kuikka. *A Batch Process Management Framework: Domain-Specific, Design Pattern and Software Component Based Approach*. PhD thesis, Helsinki University of Technology, Finland, 1999.

[6] W. Lepuschitz, F. Königseder, and A. Zoitl. Conjunction of a distributed control system based on IEC 61499 with a commercial batch management system. In *Proceedings of 14th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1–8, 2009.

[7] W. Lepuschitz and A. Zoitl. An engineering method for batch process automation using a component oriented design based on IEC 61499. In *Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 207–214, 2008.

[8] W. Lepuschitz and A. Zoitl. Integration of a DCS based on IEC 61499 with industrial batch management systems. In *Proceedings of 13th IFAC Symposium on Information Control Problems in Manufacturing*, pages 690–695, 2009.

[9] M. Parker and J. Rawtani. *Practical Batch Process Management*. Newnes, The Netherlands, 2005.

[10] Rockwell Automation. FactoryTalk Batch.

<http://www.rockwellautomation.com/rockwellsoftware/products/factorytalk-batch.page>.

[11] C. Sünder, A. Zoitl, and C. Dutzler. Functional structure-based modeling of automation systems. *International Journal of Manufacturing Research*, Vol. 1, No. 4, 405–420, 2006.

[12] V. Vyatkin. Intelligent mechatronic components: control system engineering using an open distributed architecture. In *Proceedings of 10th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 277–284, 2003.

Лабораторный подход к автоматизации интеллектуальной сети с использованием IEC 61499

Filip Andrén

AIT GmbH

Georg Lauss

AIT GmbH

Roland Bründlinger

AIT GmbH

Philipp Svec

AIT GmbH

Christian Seidl

AIT GmbH

Thomas Strasser

AIT GmbH

СОДЕРЖАНИЕ

21.1 Введение и мотивация	510
21.2 Лаборатории интеллектуальной сети: потребности и требования	511
21.3 Краткий обзор лабораторной среды SmartEST	512
21.3.1 Цель лаборатории и ее использование	512
21.3.2 Варианты и сценарии лабораторного использования	514
21.3.3 Требования для лабораторной системы автоматизации	515
21.4 Лабораторная система автоматизации на базе IEC 61499	516
21.4.1 Архитектура и концепция	516
21.4.2 Прикладная модель автоматизации на основе IEC 61499	518
21.4.2.1 Конфигурация системы	518
21.4.2.2 Система управления	519
21.4.3 Среда автоматизации с открытым исходным кодом	525
21.5 Сводка и выводы	526

21.1. Введение и мотивация

Возобновляемые источники энергии рассматриваются как ключевые факторы, способствующие сокращению выбросов парниковых газов и преодолению тенденции антропогенного глобального потепления. Их нестабильное поведение и ограниченные возможности хранения представляют новые проблемы для операторов энергосистем в поддержании качества и надежности электропитания [18].

Однако доступность информационных и коммуникационных технологий (ICT), передовых подходов к автоматизации и интеллектуальных устройств обеспечивает новые интеллектуальные решения этих проблем. В результате разработок традиционная энергетическая система превращается в киберфизическую систему – интеллектуальную сеть [8, 10, 15]. Такое видение интеграции энергосистем с поддержкой современных технологий ICT может внести существенный вклад в более стабильное и безопасное электроснабжение [10, 11, 22, 26].

Наряду с техническим, организационным и образовательным прогрессом [24], требуется надлежащая проверка инфраструктуры для систем и компонентов интеллектуальных сетей. Лаборатория интеллектуальных электрических систем и технологий (лаборатория SmartEST), созданная австрийским технологическим институтом AIT, предоставляет инфраструктуру для исследований и испытаний, для анализа взаимодействия компонентов интеллектуальной сети и энергосистемы в реальных условиях. Кроме того, она также используется для разработки компонентов сети. Кандидаты для тестирования в лаборатории SmartEST имеют подключенные к сети и автономные инверторные системы для распределенных энергоресурсов (DER), системы хранения энергии, алгоритмы и стратегии для управления сетью, инфраструктуру связи и соответствующие системы управления энергией, а также инфраструктуру для зарядки электромобилей [6].

Для работы такой многофункциональной инфраструктуры для исследований, разработок и испытаний в области интеллектуальных сетей требуется гибкая и масштабируемая среда автоматизации. Целью этой главы является рассмотрение систем автоматизации и управления для лаборатории SmartEST на основе эталонной модели IEC 61499 [25] с использованием инструментов с открытым исходным кодом. Такой подход к автоматизации обеспечивает необходимую основу для реализации гибкой и настраиваемой лабораторной автоматизации, поддерживающей проектирование, разработку и проверку новых подходов, методов и концепций в области интеллектуальных сетей.

В этой главе обобщены предыдущие работы, выполненные с использованием стандарта IEC 61499 для автоматизации лаборатории интеллектуальных сетей [2, 6, 17]. Основное внимание в этой главе уделяется разработанной архитектуре автоматизации и моделированию приложения автоматизации для лабораторной среды.

В разделе 21.2 кратко рассматриваются потребности лабораторий с интеллектуальными сетями, после чего в разделе 21.3 представлена многофункциональная лаборатория SmartEST. Основная часть этой главы посвящена системам автоматизации на основе IEC 61499 и соответствующим приложениям автоматизации (раздел 21.4). Раздел 21.5 завершает эту главу основными выводами и сведениями о запланированной будущей работе.

21.2. Лаборатории интеллектуальной сети: потребности и требования

Проектирование, разработка и внедрение технологий и компонентов интеллектуальных сетей и их установка в полевых условиях продолжают усложняться. Одной из основных причин является киберфизическая природа концепций и компонентов интеллектуальной сети. Кроме того, проверка и тестирование различных конфигураций (то есть аппаратного и программного обеспечения) будут играть важную роль в будущих исследованиях и разработке технологий, чтобы гарантировать установку надежных и безошибочных компонентов и приложений.

В последние годы несколько демонстрационных и полевых тестовых проектов в этой области оказались успешными в разных странах. В ближайшем будущем ожидается широкомасштабное внедрение новых и улучшенных продуктов, услуг и решений. Государственные органы власти разработали интеллектуальную инфраструктуру для исследований и разработок в качестве основных приоритетов [5, 9, 16, 19].

Чтобы проверять и тестировать различные конфигурации интеллектуальной сети интегрированным (кибер-физическим) образом, система питания и соответствующие информационные, коммуникационные и управляющие компоненты должны быть охвачены в равной степени [24]. Сложность такой среды валидации для будущих интеллектуальных энергосистем требует виртуальных и реальных методов, основанных на следующих пунктах [2, 13, 14, 17, 24].

- Необходим мультидоменный, киберфизический подход для анализа, проверки и тестирования конфигураций интеллектуальных сетей на системном уровне.

- Требуются методы, концепции и процедуры для анализа, разработки и тестирования компонентов интеллектуальной сети (энергосистема, ICT, автоматизация и управление).

- Исследователи и инженеры, занимающиеся интеллектуальными конфигурациями сети в кибер-физической форме, должны быть хорошо обучены.

Кроме того, требуется надлежащая интегрированная исследовательская инфраструктура, учитывающая моделируемые и основанные на лабораторных условиях методы проектирования, разработки, валидации и тестирования для интеллектуальных сетей и связанных компонентов (включая комбинацию

обоих методов). Испытание новых подходов и компонентов в широком масштабе в полевых условиях часто не является подходящим вариантом, поскольку необходимо гарантировать доступность и надежность энергосистемы. Следовательно, исследовательская инфраструктура, использующая моделируемые системы с реальным лабораторным оборудованием, обеспечивает разумную среду.

21.3. Краткий обзор лабораторной среды SmartEST

В следующих разделах дан краткий обзор многофункциональной лабораторной среды SmartEST и соответствующих тематических исследований. Кроме того, вводятся важные требования и потребности в гибкой и масштабируемой системе автоматизации лабораторий [2, 17].

21.3.1. Цель лаборатории и ее использование

Лаборатория SmartEST – это инфраструктура для тестирования компонентов интеллектуальной сети и их интеграции в существующую систему электропитания. Она также предлагает гибкую и многофункциональную среду для проверки различных конфигураций интеллектуальной сети (то есть аппаратного и программного обеспечения). Она играет центральную роль в австрийских исследованиях и разработках в области интеллектуальных сетей. Лаборатория специализируется на тестировании инверторов и (под-)систем, системных тестах с несколькими компонентами и экологических тестах. SmartEST предлагает расширенные возможности для совмещения моделирования и экспериментальной проверки. Область ее применения варьируется от исследований, проектирования и валидации для интеллектуальных энергосистем и разработки соответствующих компонентов до анализа концепций автоматизации и коммуникаций [6].

Инфраструктура лаборатории включает в себя компоненты DER, системы накопления энергии, регуляторы напряжения и другое необходимое электрооборудование. На рисунке 21.1 представлен обзор топологии и основных элементов лаборатории SmartEST. Крупномасштабные, свободно программируемые источники переменного и постоянного тока с высокой пропускной способностью позволяют проводить тестирование на полной мощности до 1 МВА (AC), включая высокопроизводительное моделирование PV Array (DC) до 1 МВт (DC). Оборудование для моделирования интерфейсов управления и связи и возможность эксплуатации оборудования в определенных условиях экстремальных температур и влажности предлагает дополнительные возможности тестирования. Кроме того, лаборатория обладает очень гибкими возможностями различных конфигураций низковольтных сетей, которые оказывают огромное влияние на исследование сложных проблем электросетей [6].

Кроме того, лаборатория SmartEST предлагает возможность в реальном времени моделировать сложные энергосистемы и подключать их к лабораторному оборудованию в режиме работы с обратной связью. Эта так называемая конфигурация аппаратного обеспечения контроллера в контуре (CHIL) и/или аппаратного обеспечения питания в контуре (PHIL) позволяет интегрировать реальные компоненты в среду виртуальной электросети и тестировать в реальных условиях (то есть происходит симуляция в реальном времени с замкнутым контуром с временными шагами до 10 мкс) в зависимости от эмулируемой сети и вычислительной мощности в реальном времени [21]. Она также позволяет подключать автономное совместное моделирование решений в области питания и ICT [11].

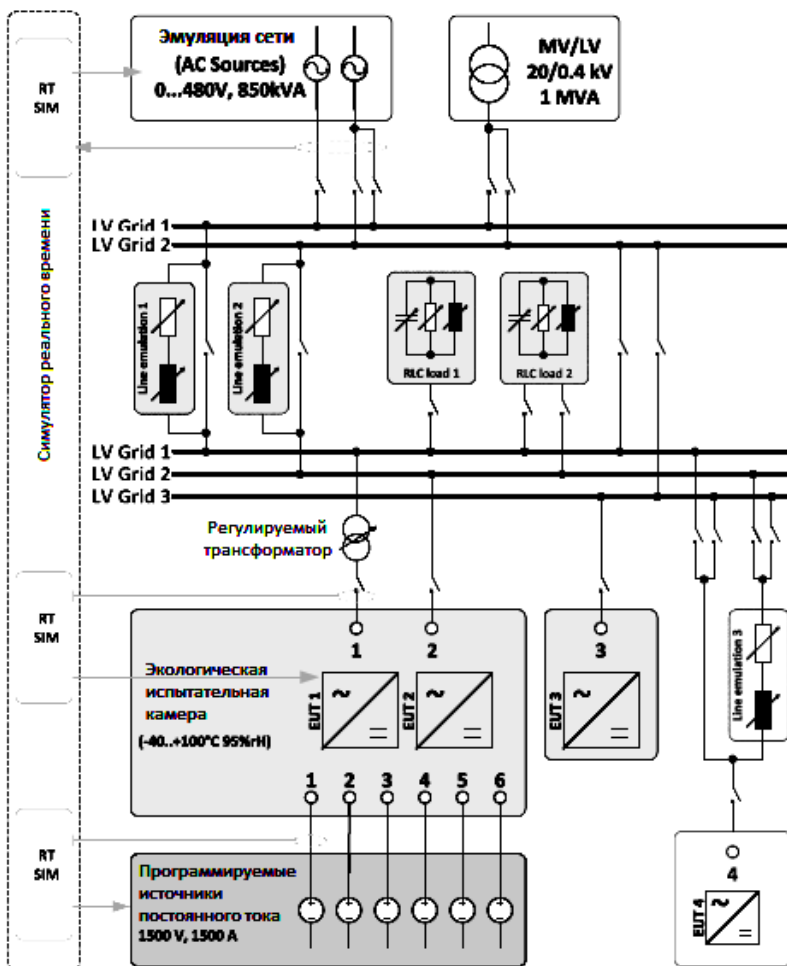


Рисунок 21.1. Обзор лабораторной среды SmartEST [2, 6, 17].

Условные обозначения:

MV/LV: среднее напряжение / низкое напряжение трансформатора

LV Grid: Низковольтная сеть

RT SIM: цифровой симулятор реального времени

EUT: проверяемое оборудование

21.3.2. Варианты и сценарии лабораторного использования

Среда SmartEST была разработана для поддержки различных вариантов использования и сценариев и может рассматриваться как многофункциональная лабораторная среда. Может быть выполнен широкий спектр действий – от тестирования DER и компонентов энергосистемы до экспериментов по интеграции системы. В таблице 21.1 кратко изложены сценарии валидации и тестирования, выполненные в этой лаборатории [2].

Таблица 21.1. Примеры использования и сценарии, выполненные в SmartEST [2]

Тема исследования	Лабораторная деятельность
Проверка компонента DER	Инверторные тесты (компонент, интеграция). Тесты зарядных устройств (компонент, интеграция).
Разработка и валидация ICT / контроллеров	Валидация реализации контроллера. Валидация протоколов связи для DER. Тестирование разработок системы SCADA.
Разработка новых сетевых компонентов (силовая электроника)	Тест новых топологий. Испытание материалов для силовых электронных устройств. Валидация метода контроля компонентов.
Проверка на основе совместного моделирования	Контроль энергосистемы и тестирование коммуникационного поведения. Контроль компонентов и тестирование связи.
Моделирование в реальном времени и HIL	Интеграционный тест для DER. Валидация новых силовых электронных компонентов.

Помимо вышеупомянутых сценариев, в лабораторной среде проводятся учебные и образовательные мероприятия. Это включает обучение обслуживающего персонала новым (дистанционным) функциям управления, предоставляемым компонентами DER на основе инвертора (например, управление напряжением и частотой). Кроме того, лаборатория также используется для обучения студентов университетов в области систем интеллектуальных сетей и автоматизации.

21.3.3. Требования для лабораторной системы автоматизации

Лаборатория SmartEST предоставляет гибкую и настраиваемую многофункциональную среду проектирования, валидации и тестирования для интеллектуальных сетей. Следовательно, соответствующая система автоматизации должна быть максимально гибкой. Для разработки, тестирования и проверки новых концепций автоматизации, связи и безопасности для приложений на интеллектуальных энергосистемах, особенно в лаборатории SmartEST, из собранных выше вариантов использования могут быть выведены следующие требования [2].

Аппаратная среда

- Гибкость: среда лабораторных оценок и испытаний должна обеспечивать достаточную гибкость для включения различных протоколов и интерфейсов связи, а также платформ контроллеров. Поддержка различных сред моделирования для систем распределения электроэнергии, ICT и концепций автоматизации должна быть возможной.
- Масштабируемость: должно быть возможно легкое расширение лабораторной оценки и среды тестирования. Если пределы масштабируемости достигнуты, необходимо обеспечить и разработать методы моделирования для достижения масштабируемости по мощности и гибкости (например, моделирование в реальном времени, CHIL/PHIL).

Программная среда приложения

- Конфигурируемость: должна быть возможна конфигурация гетерогенных контроллеров с помощью различных программных инструментов.
- Портативность: протестированные программы автоматизации и управления должны легко переноситься на реальные контроллеры.
- Аппаратная независимость: управляющие приложения должны быть реализованы аппаратно-независимым образом, чтобы их можно было выполнять на разных аппаратных ресурсах.
- Распределение: следует поддерживать распространение программ на оборудование контроллера, чтобы оно соответствовало распределенной природе приложений интеллектуальных сетей.

Среда моделирования

- Автономное моделирование: необходимо поддерживать автономное моделирование моделей распределения мощности (то есть физических и управляющих) и концепций ICT.
- Моделирование в реальном времени: должно быть возможным моделирование в реальном времени систем распределения электроэнергии и их соответствующих компонентов (то есть физического оборудования и эмулируемых моделей в программном обеспечении) вместе с концепциями ICT.

Открытая системная архитектура и соответствие стандартам

- Функциональная совместимость: должна поддерживаться функциональная совместимость имитационных моделей и сред, реальных сетевых компонентов, а также аппаратного и программного обеспечения контроллера.

- Соответствие стандарту: лабораторная среда должна основываться на общеизвестных стандартах связи (то есть промышленного Ethernet, Ethernet, полевых шин), автоматизации и управлении (то есть IEC 61499, OPC UA) и области систем электропитания доменов (то есть IEC 61850, IEC 60870) [20, 23].

- Открытые коммуникационные интерфейсы: различные открытые коммуникационные сети и интерфейсы (например, IEC 61850, OPC UA, Modbus RTU/TCP, Industrial Ethernet) должны позволять тестировать и проверять широкий спектр приложений и компонентов в области интеллектуальных сетей [20, 23].

- Подходы с открытым исходным кодом: использование компонентов и модулей программного обеспечения с открытым исходным кодом должно помочь улучшить лабораторную среду оценки и тестирования, что не всегда возможно при использовании подходов с закрытым исходным кодом.

21.4. Лабораторная система автоматизации на базе IEC 61499

Чтобы удовлетворить потребности в гибкой и масштабируемой лаборатории интеллектуальных сетей, была разработана система автоматизации, использующая эталонную модель IEC 61499 [25] для распределенной автоматизации и управления. В следующих разделах представлены архитектура автоматизации, систем автоматизации и полученная в результате среда автоматизации с открытым исходным кодом.

21.4.1. Архитектура и концепция

Основным требованием к системе автоматизации SmartEST является гибкая и быстро адаптируемая системная архитектура, которая учитывает масштабируемость. Быстрая адаптация к новому лабораторному оборудованию и устройствам (то есть аппаратному и программному обеспечению) без изменения основных элементов приложения автоматизации является ключевым моментом для его реализации.

Общая концепция – четырехуровневый подход – основана на модели архитектуры интеллектуальных сетей (SGAM), предоставленной Международной электротехнической комиссией (IEC) [20]. Для реализации логики автоматизации и управления используется стандарт IEC 61499,

поскольку он является многообещающим подходом, предложенным DKE¹ и IEC [7, 20]. Различные уровни имеют следующие значения [3].

- Уровень контроля, визуализации и взаимодействия: взаимодействие с пользователем (то есть с оператором лаборатории) и визуализация охватываются этим уровнем. Как правило, интерфейс человек–машина (HMI) предоставляется для отображения состояния лабораторной конфигурации путем представления результатов измерений и мониторинга в удобочитаемом формате. Этот уровень можно использовать для подбора лабораторных настроек. Он также называется уровнем диспетчерского контроля и сбора данных (SCADA).

- Уровень управления и контроля: на этом уровне выполняются функции управления и автоматизации, которые обычно критичны ко времени или не зависят от взаимодействия с пользователем. Измерения датчиков, полученные на уровне компонентов системы, обрабатываются и фильтруются на этом уровне, прежде чем они распространяются на уровень SCADA. Кроме того, связанные с безопасностью функции (например, блокировка, аварийные сигналы) защищают лабораторию от сбоев и неправильных конфигураций. Из-за критической природы функций на этом уровне, с точки зрения производительности компьютера, они обычно выполняются на встроенных устройствах.

- Уровень компонентов системы: этот уровень содержит электрические компоненты, используемые для взаимодействия с техническим процессом. Примерами являются датчики (то есть измерительные устройства, датчики обратной связи), исполнительные механизмы (то есть силовые выключатели и прерыватели) и входы и выходы (цифровые и аналоговые распределенные I/O). Эти компоненты используются для влияния на технический процесс, то есть компоненты системы могут контролироваться функциями на уровне управления и контроля.

- Уровень технического процесса: лабораторные сети и соответствующие компоненты (силовые выключатели, симуляторы переменного и постоянного тока, трансформаторы и т. д.), как показано на рисунке 21.1, за исключением компонентов системы с предыдущего уровня, расположены на этом уровне.

Результирующая архитектура показана на рисунке 21.2. Уровень контроля и мониторинга реализован с использованием эталонной модели IEC 61499 для распределенной автоматизации и управления (см. раздел 21.4.2). С этой концепцией вышеупомянутые требования и потребности в гибкой лабораторной среде могут быть выполнены в значительной степени.

¹ Немецкая комиссия по электрическим, электронным и информационным технологиям DIN (Deutsches Institut für Normung e. V. и VDE Verband der Elektrotechnik Elektronik Informationstechnik e.V.)

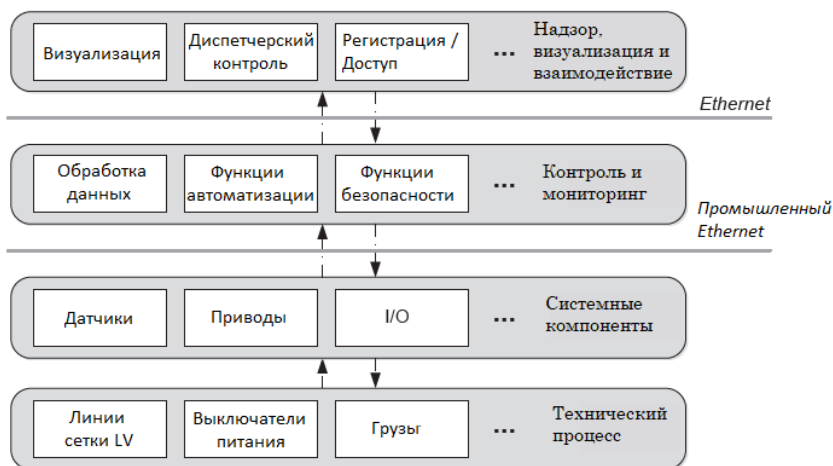


Рисунок 21.2. Архитектура автоматизации, позволяющая гибко настраивать интеллектуальные лабораторные проекты

21.4.2. Прикладная модель автоматизации на основе IEC 61499

Модель приложения автоматизации, описанная ниже, содержит (1) конфигурацию системы и (2) систем автоматизации. Вся установка смоделирована и реализована с использованием эталонной модели IEC 61499 [25] для распределенной автоматизации и управления.

21.4.2.1. Конфигурация системы

Основными задачами системы автоматизации SmartEST являются безопасное управление лабораторными конфигурациями (то есть различными настройками лабораторных сетей и компонентов) и их визуализация для взаимодействия с оператором. Вся система должна быть гибкой и адаптируемой к новому оборудованию без изменения программного обеспечения для автоматизации и управления ядром.

На рисунке 21.3 показана конфигурация системы IEC 61499, которая позволила нам достичь этой сложной цели. Она содержит два встроенных управляющих устройства, соответствующих IEC 61499, для выполнения лабораторных задач автоматизации и управления. Оба управляющих устройства подключаются через интерфейс связи к сети связи и, наконец, к лаборатории SCADA. Кроме того, интерфейс процесса соединяет эти устройства с лабораторным оборудованием (то есть лабораторными сетями, переключателями, нагрузками, симулятором переменного и постоянного тока, как показано на рисунке 21.1). Возможно взаимодействие с имитируемыми

компонентами в автономном режиме и/или в реальном времени. Управляющие устройства отвечают за выполнение различных функций управления и контроля (обозначены как приложения автоматизации на рисунке 21.3).

В результате требования, описанные в разделе 21.3.3, могут быть выполнены с этой предложенной архитектурой.

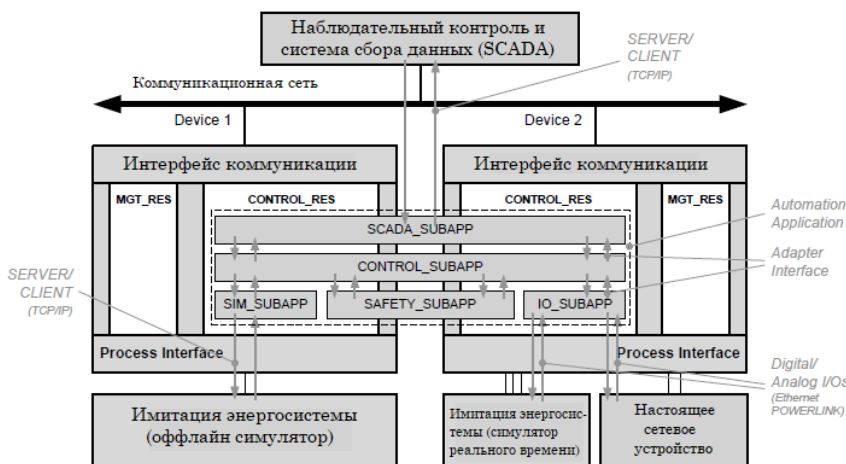


Рисунок 21.3. Конфигурация системы IEC 61499 для лаборатории SmartEST [2]

21.4.2.2. Система управления

Для разработки приложения автоматизации основное внимание было уделено возможности повторного использования компонентов программного обеспечения (то есть функциональных блоков IEC 61499) и ясности кода для управления, поскольку лаборатория SmartEST содержит большое количество различных устройств. Такая проблема уже обсуждалась в нескольких публикациях IEC 61499. В соответствии с [1, 12, 28] система управления IEC 61499 должна моделироваться с использованием подпрограмм на разных уровнях для формирования иерархической структуры. На самом верхнем уровне предлагается предоставить обзор различных компонентов, участвующих в приложении. Каждое вложенное приложение, представляющее конкретную функцию автоматизации, должно описывать свое соответствующее поведение.

В контексте SmartEST система управления делится на следующие части с использованием подпрограмм IEC 61499 в качестве основных элементов моделирования на верхнем и нижнем уровнях:

- взаимодействие SCADA (то есть SCADA_SUBAPP);
- функции управления, мониторинга и обработки данных (то есть CONTROL_SUBAPP);

- функции безопасности (то есть SAFETY_SUBAPP);
- интеграция I/O (то есть IO_SUBAPP);
- интеграция симуляции (то есть SIM_SUBAPP).

Взаимодействие подпрограмм реализуется с использованием интерфейсов адаптера и общих шаблонов интерфейсов связи (то есть СЕРВЕР–КЛИЕНТ), как предложено в [1, 12, 28]. Интерфейсы адаптера приведены на рисунке 21.4, где также представлен вид верхнего уровня приложения автоматизации с основными частями, перечисленными выше.

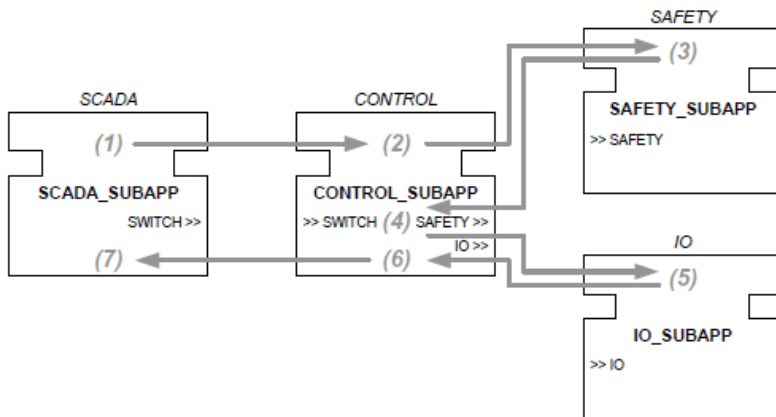


Рисунок 21.4. Схема приложения автоматизации и соответствующий поток сигналов

При отделении приложения автоматизации возможна независимая от оборудования реализация системы автоматизации SmartEST (кроме IO_SUBAPP и SIM_SUBAPP). Следующие разделы предоставляют более подробную информацию о реализации различных подприложений.

Взаимодействие SCADA

SCADA_SUBAPP обеспечивает интерфейс между уровнем управления и уровнем SCADA. Управляющие сигналы от системы SCADA передаются через это приложение для вызова функций управления и для подачи сигналов обратной связи (этапы (1) и (7) на рисунке 21.4). Кроме того, информация о состоянии от датчиков и измерительного оборудования также передается в систему SCADA, чтобы предоставить оператору обзор состояния лаборатории. Шаблоны универсального функционального блока интерфейса службы связи (SIFB) IEC 61499 КЛИЕНТ–СЕРВЕР являются основными частями этой подпрограммы. Управляющее устройство представляет функциональность СЕРВЕРА, тогда как система SCADA является соответствующим КЛИЕНТОМ.

Функции управления, мониторинг и обработка данных

Эта прикладная часть используется для настройки, контроля и мониторинга лабораторной конфигурации (симулятор переменного / постоянного тока, регулируемый трансформатор, система заземления, топология лабораторных решеток: радиальная, ячеистая, островная и т. д.). Она обеспечивает функции автоматизации для управления переключателями питания вручную через систему SCADA или через автоматическую последовательность переключения (например, корректировку эталонного значения для активной и/или реактивной мощности нагрузки RLC, последовательность тестирования для настройки лабораторных сетей).

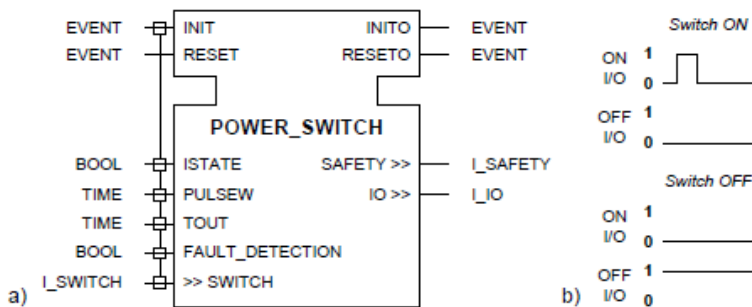


Рисунок 21.5. Выключатель питания FB: (a) интерфейс FB и (b) генерируемый сигнал переключения

На рисунке 21.5 показан соответствующий функциональный блок IEC 61499 (FB) для управления выключателем питания. Цифровые сигналы, передаваемые через децентрализованные I/O (см. интеграцию I/O ниже), необходимы для включения или выключения питания. Этот FB отвечает за проверку возможности выполнения командного переключателя и генерации соответствующих сигналов I/O (этапы (2) и (4) на рисунке 21.4).

Кроме того, мониторинг лабораторного состояния (например, положения переключателя, аварийных сигналов) также выполняется и передается в SCADA (этап (6) на рисунке 21.4). Для этой задачи используется универсальный коммуникационный КЛИЕНТ SIFB по стандарту IEC 61499 для взаимодействия с датчиками и измерительным оборудованием через технологический интерфейс IEC 61499. Modbus/TCP в основном используется в качестве протокола связи для взаимодействия с измерительными устройствами.

Функции безопасности

Лабораторные сетки SmartEST содержат несколько переключателей и компонентов, и многие из них имеют особые условия безопасности. Например, существует несколько случаев, когда два переключателя или более не могут быть включены одновременно. Некоторые из наиболее критических условий

безопасности покрываются с помощью аппаратных функций; все остальные условия должны быть включены в программное обеспечение.

Условия безопасности могут быть обеспечены контрольным уровнем или уровнем SCADA. Кроме того, если условия безопасности проверяются аппаратными средствами, необходимо надлежащее взаимодействие с пользователем и визуализация. Однако на практике система SCADA не используется для критических проверок безопасности – главным образом из-за низкого приоритета задачи программного обеспечения SCADA. Чтобы приложения безопасности были как можно более надежными, они должны быть реализованы на уровне управления с использованием элементов IEC 61499 (то есть FB).

Подпрограмма безопасности (SAFETY_SUBAPP) содержит функции, которые проверяют различные условия безопасности в лаборатории. Перед тем как управляющее действие (например, запрос на переключение, инициированное оператором через SCADA или через функции управления) может быть выполнено с использованием I/O, подключенных к действующему субъекту (например, выключатель питания), оно должно пройти безопасное приложение (шаг (3) на рисунке 21.4). Это предотвращает любые управляющие действия, направленные на то, чтобы обойти приложение безопасности и подключиться непосредственно к I/O. Укажем две основные функции безопасности, реализованные для лабораторной среды.

- Одна из общих функций безопасности – это так называемая функция блокировки. Она была реализована с использованием IEC 61499 FB, который показан на рисунке 21.6. Основная функциональность этого FB – предотвращение одновременного включения переключателей. FB получает от двух коммутаторов один сигнал запроса и один сигнал обратной связи с текущим состоянием коммутатора. Разрешенные состояния передаются на выходы, как также показано на этом рисунке.

- Более полной функцией безопасности является определение пускового тока. Из-за высокого пускового тока, который может возникнуть при включении напряжения, некоторые компоненты сети в нагрузке RLC могут быть повреждены (см. рисунок 21.1). Чтобы предотвратить это, используется определенный порядок, в котором разрешено включение переключателей, то есть переменное напряжение должно быть включено до нагрузки RLC.

Интеграция I/O

Чтобы позволить контрольным устройствам IEC 61499 взаимодействовать с лабораторным оборудованием, были использованы децентрализованные I/O (шаг (5) на рисунке 21.4). I/O были подключены через интерфейс Industrial Ethernet с использованием Ethernet POWERLINK (EPL), инкапсулированного в FB IEC 61499 [4]. Спецификация EPL определяет главные узлы (MN) и управляемые узлы (CN).

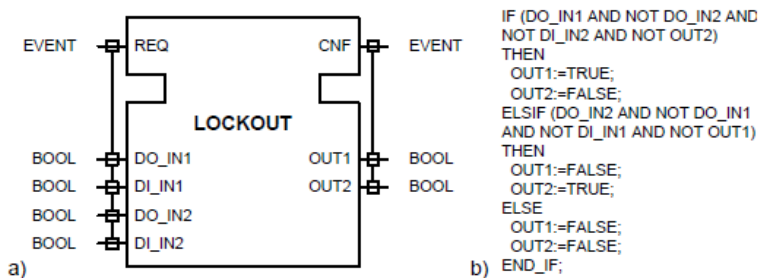


Рисунок 21.6. Реализация функции блокировки с использованием FB IEC 61499: (a) интерфейс FB и (b) инкапсулированный алгоритм

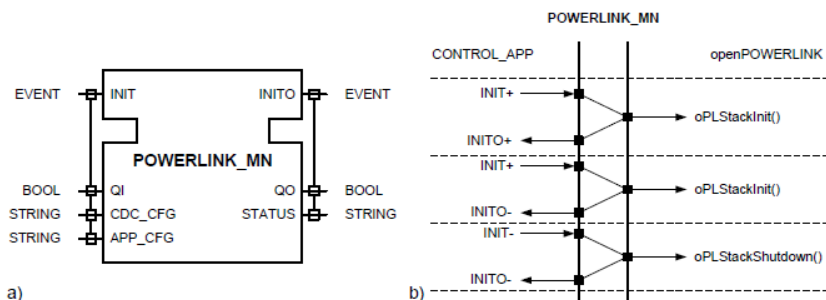


Рисунок 21.7. Главный узел EPL SIFB: (a) интерфейс FB и (b) последовательность обслуживания [4]

Обычно программируемые логические контроллеры (PLC) или промышленные ПК (IPC) действуют как MN, тогда как CN (то есть устройства I/O, приводы и т. д.) подключаются через Ethernet. Как показано на рисунке 21.3, устройство IEC 61499 представляет службы MN (то есть IO_SUBAPP). Чтобы инкапсулировать функции MN и CN, для определения IO_SUBAPP необходимы специальные FB IEC 61499 – блоки функций интерфейса службы (SIFB). В связи с этим были разработаны следующие два SIFB [4].

- **Главный узел EPL SIFB:** этот SIFB представляет MN. Он имеет два ввода данных: CDC_CFG содержит отображение I/O в образ процесса конфигурации EPL в виде двоичного файла, а APP_CFG представляет конфигурацию сети в виде файла XML. С помощью этого SIFB драйвер EPL можно запускать и останавливать. Во время фазы запуска MN разворачивает конфигурацию сети (то есть CDC_CFG) для CN. На рисунке 21.7 показан интерфейс и сервисные последовательности MN SIFB.

- **SIFB узла I/O клиента EPL:** доступ к устройствам I/O EPL инкапсулирован в этот SIFB. Он предоставляет услуги для чтения входных данных и записи выходных данных. Поскольку в сети EPL поддерживается несколько CN, идентификация конкретного CN должна быть предоставлена в этот SIFB IEC

61499 (то есть ввод данных CNID). Децентрализованные операции I/O обычно имеют несколько операций I/O. Поэтому необходимо указать правильный канал I/O. Это покрывается входом PARAM этого SIFB. OUTDATAm и INDATAm представляют собой соответствующие входы и выходы децентрализованных I/O, инкапсулированных в SIFB IEC 61499. На рисунке 21.8 показан интерфейс SIFB и сервисные последовательности CN, представляющие доступ к децентрализованным I/O через EPL.

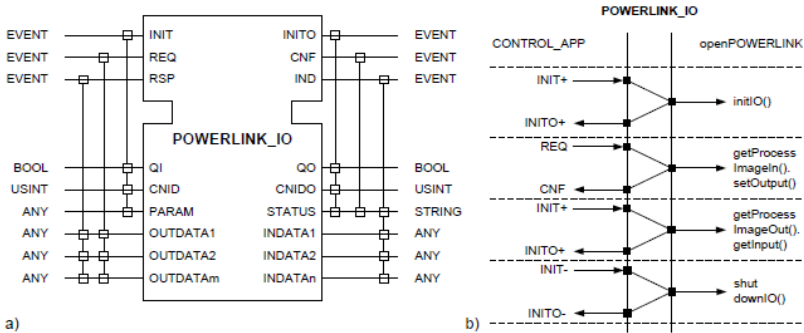


Рисунок 21.8. Управляемый EPL узел SIFB (то есть узел I/O): (а) интерфейс FB и (b) последовательность обслуживания [4]

Поскольку спецификация EPL использует поведение циклического выполнения и поведение выполнения, основанное на событиях IEC 61499, необходимо сопоставление между обеими сферами. Была выбрана следующая процедура: POWERLINK_IO SIFB уведомляет драйвер EPL о необходимости обновления выходных данных. В течение следующего цикла EPL соответствующие выходы обновляются. Входные данные обычно считываются при каждом цикле EPL, но событие вывода POWERLINK_IO SIFB (то есть IND) генерируется только в том случае, если этот SIFB обнаруживает изменения в значениях входов.

Интеграция симуляции

Как говорилось в разделе 21.2, для выполнения требований к интеллектуальным сеточным лабораториям также важное значение имеет интеграция моделируемых компонентов [24]. Эта проблема решается с помощью предлагаемой архитектуры и конфигурации системы, показанной на рисунке 21.3, путем взаимодействия с автономными имитаторами энергосистемы (MATLAB®/Simulink SimPowerSystems, PSAT, PowerFactory и т. д.) или цифровыми симуляторами реального времени (OPAL-RT, Typhoon HIL и т. д.). SIFB КЛИЕНТА–СЕРВЕРА, использующие TCP/IP, в основном применяются для подключения к автономным симуляторам, в то время как цифровые I/O, управляемые EPL, доступны для связи с симуляторами в

реальном времени. Учитывая эту возможность, моделируемые компоненты энергосистемы могут быть интегрированы в конструкцию всей лаборатории.

21.4.3. Среда автоматизации с открытым исходным кодом

Для реализации представленной выше концепции и соответствующей конфигурации системы IEC 61499 используются программное обеспечение и инструменты с открытым исходным кодом [17]. Для реализации функций управления используется открытый источник и распределенная управляющая среда 4DIAC, соответствующая IEC 61499 [27, 29]. Это обеспечивает инженерную среду для систем IEC 61499 и соответствующую среду исполнения для встроенных устройств. Данная среда может выполнять различные функции управления и обеспечения безопасности и взаимодействовать с уровнем SCADA и уровнем компонентов системы.

Для взаимодействия с лабораторным оборудованием установлено около 1300 цифровых I/O с использованием 16 распределенных узлов I/O, управляемых через EPL. Их основными задачами являются управление исполнительными механизмами (силовыми выключателями и реле) и предоставление информации о состоянии от датчиков и измерительных устройств.

Обмен данными выполняется на двух управляемых Linux IPC, а система распределенного I/O реализуется с использованием EPL-драйвера с открытым исходным кодом openPOWERLINK, инкапсулированного в SIFB (см. раздел 21.4.2.2) [4]. Наряду с системой распределенного цифрового I/O SmartEST наблюдает большое количество данных измерений (около 700 каналов, охватывающих напряжения, токи, активную и реактивную мощность, частоту и т. д.). Всего 22 устройства измерения мощности интегрированы и используются как для эксплуатации, так и для целей управления. Для передачи измеренных значений из лабораторного оборудования на уровни управления и SCADA используется Modbus/TCP. Подобно интеграции openPOWERLINK, подход с открытым исходным кодом был интегрирован в стандарт SIFBS связи стандарта IEC 61499 (то есть КЛИЕНТ–СЕРВЕР). Для этой цели используется библиотека libmodbus.

Уровень SCADA для диспетчерского контроля и мониторинга также очень важен для работы лаборатории SmartEST, так как он обеспечивает HMI для оператора. Поскольку 4DIAC охватывает функции управления без каких-либо возможностей визуализации, используется реализация с открытым исходным кодом ScadaBR [17]. Она визуализирует доступное лабораторное оборудование и предоставляет информацию о состоянии, как показано на рисунке 21.9. Также в ScadaBR выполняются функции контроля и управления высокого уровня. Обзор выбранной конфигурации лабораторных сетей предоставляется оператору. Связь и обмен данными с уровнем управления (то есть средой 4DIAC) осуществляется с использованием протокола на основе TCP, предложенного IEC 61499, который реализован в ScadaBR и 4DIAC.

21.5. Сводка и выводы

Для разработки и проверки интеллектуальных сетевых решений необходима гибкая и многофункциональная среда. Это должно облегчить тестирование аппаратных и программных компонентов и их интеграцию в энергосистему. Это может быть достигнуто с помощью лабораторного оборудования или надлежащих имитационных моделей или их комбинации. Лабораторная среда SmartEST обеспечивает такую настройку для системы питания и области умных сетей электроснабжения.

Для автоматизации и управления этой исследовательской инфраструктурой должны быть выполнены различные сложные требования к аппаратному и программному обеспечению. Для этого предлагается архитектура автоматизации и соответствующая концепция с использованием эталонной модели IEC 61499. Для ее реализации используются инструменты с открытым исходным кодом 4DIAC, ScadaBR, openPOWERLINK и libmodbus. Эта трудоемкая интеграция свободно доступного программного обеспечения была необходима для создания очень гибкой и легко конфигурируемой программной среды. Подход 4DIAC, соответствующий IEC 61499, служит интегрирующим промежуточным программным обеспечением между лабораторными компонентами и SCADA. Это обеспечивает ядро предлагаемой системы автоматизации.

Будущая работа направлена на дальнейшее развитие функций автоматизации, необходимых для конкретных методов тестирования. Другие коммуникационные подходы, такие как IEC 61850, OPC UA и протоколы интеллектуального учета, должны быть интегрированы в систему автоматизации SmartEST. Подход с открытым исходным кодом позволяет осуществлять необходимую интеграцию новых концепций, методов и протоколов вместо проприетарных и закрытых решений от коммерческих поставщиков автоматизации.

Библиография

- [1] F. Andrén, R. Bründlinger, and T. Strasser. IEC 61850/61499 control of distributed energy resources: concept, guidelines, and implementation. *IEEE Transactions on Energy Conversion*, Vol. 29, No. 4, 1008–1017, 2014.
- [2] F. Andrén, G. Lauss, R. Bründlinger, P. Svec, and T. Strasser. An open source-based and standard-compliant smart grid laboratory automation system: The AIT SmartEST approach. In V. Marík, A. Shirmann, D. Trentesaux, and P. Vrba, editors, *Industrial Applications of Holonic and Multi-Agent Systems*, Lecture Notes in Computer Science, pages 195–205. Springer, Heidelberg, 2015.
- [3] F. Andrén, F. Lehfuss, and T. Strasser. A development and validation environment for real-time controller-hardware-in-the-loop experiments in smart grids. *International Journal of Distributed Energy Resources and Smart Grids*, Vol. 9, No. 1, 27–50, 2013.

- [4] F. Andr n and T. Strasser. Distributed open source control with Industrial Ethernet I/O devices. In *Proceedings of 16th IEEE Conference on Emerging Technologies Factory Automation*, pages 1–4, 2011.
- [5] R. Br ndlinger, D. Craciun, W. Heckmann, V. Helmbrecht, B. Klebow, P. Kotsampopoulos, L. Martini, S. Numminen, I. Papaioannou, T. Strasser, M. Sosnina, and P. Vaessen. European Distributed Energy Resources Laboratories Activity Report 2012/2013, 2012.
- [6] R. Br ndlinger, T. Strasser, G. Lauss, A. Hoke, S. Chakraborty, G. Martin, B. Kroposki, J. Johnson, and E. de Jong. Lab tests: verifying that smart grid power converters are truly smart. *IEEE Power and Energy Magazine*, Vol. 13, No. 2, 30–42, 2015.
- [7] DKE. The German Standardisation Roadmap E-Energy/Smart Grid, German Commission for Electrical, Electronic and Information Technologies of DIN and VDE, 2010.
- [8] EC. European SmartGrids Technology Platform: Vision and Strategy for Europe’s Electricity Networks of the Future, European Commission, 2006.
- [9] EC. Smart Grids: From Innovation to Deployment, Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions, European Commission, 2011.
- [10] H. Farhangi. The path of the smart grid. *IEEE Power and Energy Magazine*, Vol. 8, No. 1, 18–28, 2010.
- [11] V. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. Hancke. Smart grid technologies: communication technologies and standards. *IEEE Transactions on Industrial Informatics*, Vol. 7, No. 4, 529–539, 2011.
- [12] I. Hegny, T. Strasser, M. Melik-Merkumians, M. Wenger, and A. Zoitl. Towards an increased reusability of distributed control applications modeled in IEC 61499. In *Proceedings of 17th IEEE Conference on Emerging Technologies Factory Automation*, pages 1–8, 2012.
- [13] K. Heussen and O. Gehrke. Lab Survey: State of the Art Smart Grid Laboratories, Department of Electrical Engineering, DTU, Kongens Lyngby, 2014.
- [14] K. Heussen, A. Thavlov, and A. Kosek. Use Cases for Laboratory Software Infrastructure: Outline of Smart Grid Lab Software Requirements, Department of Electrical Engineering, DTU, Kongens Lyngby, 2014.
- [15] IEA. Smart grid insights, International Energy Agency, 2011.
- [16] IEA. Smart Grid International Research Facility Network (SIRFN), International Energy Agency, 2015.
- [17] G. Lauss, F. Andr n, M. Stifter, R. Br ndlinger, T. Strasser, K. Kn bl, and H. Fechner. Smart grid research infrastructures in Austria: examples of available laboratories and their possibilities. In *Proceedings of 13th IEEE International Conference on Industrial Informatics*, pages 1539–1545, 2015.
- [18] M. Liserre, T. Sauter, and J. Hung. Future energy systems: integrating renewable energy sources into the smart power grid through industrial electronics. *IEEE Industrial Electronics Magazine*, Vol. 4, No. 1, 18–37, 2010.
- [19] SGA. Roadmap Smart Grids Austria: Pathway to the Future of Electrical Power Grids!, Smart Grids Austria, 2015.

- [20] SMB Smart Grid Strategic Group (SG3). IEC Smart Grid Standardization Roadmap, International Electrotechnical Commission, 2010.
- [21] M. Steurer, C. Edrington, M. Sloderbeck, W. Ren, and J. Langston. A megawatt-scale power hardware-in-the-loop simulation setup for motor drives. *IEEE Transactions on Industrial Electronics*, Vol. 57, No. 4, 1254–1260, 2010.
- [22] T. Strasser, F. Andren, J. Kathan, C. Cecati, C. Buccella, P. Siano, P. Leitao, G. Zhabelova, V. Vyatkin, P. Vrba, and V. Marik. A review of architectures and concepts for intelligence in future electric energy systems. *IEEE Transactions on Industrial Electronics*, Vol. 62, No. 4, 2424–2438, 2015.
- [23] T. Strasser, F. Andrén, M. Merdan, and A. Prostejovsky. Review of trends and challenges in smart grids: an automation point of view. In V. Marik, J. Lastra, and P. Skobelev, editors, *Industrial Applications of Holonic and Multi-Agent Systems*, volume 8062 of Lecture Notes in Computer Science, pages 1–12. Springer, Heidelberg, 2013.
- [24] T. Strasser, M. Stifter, F. Andrén, and P. Palensky. Co-simulation training platform for smart grids. *IEEE Transactions on Power Systems*, Vol. 29, No. 4, 1989–1997, 2014.
- [25] TC 65/SC 65B. IEC 61499: Function blocks. International Electrotechnical Commission, Geneva, 2nd ed., 2012.
- [26] P. Vrba, V. Marik, P. Siano, P. Leitao, G. Zhabelova, V. Vyatkin, and T. Strasser. A review of agent and service-oriented concepts applied to intelligent energy systems. *IEEE Transactions on Industrial Informatics*, Vol. 10, No. 3, 1890–1903, 2014.
- [27] V. Vyatkin. IEC 61499 as enabler of distributed and intelligent automation: state-of-the-art review. *IEEE Transactions on Industrial Informatics*, Vol. 7, No. 4, 768–781, 2011.
- [28] A. Zoitl and H. Prähofer. Guidelines and patterns for building hierarchical automation solutions in the IEC 61499 modeling language. *IEEE Transactions on Industrial Informatics*, Vol. 9, No. 4, 2387–2396, 2013.
- [29] A. Zoitl, T. Strasser, and A. Valentini. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study. In *Proceedings of 2010 IEEE International Symposium on Industrial Electronics*, pages 3817–3819, 2010.