

Министерство образования и науки Российской Федерации
Государственное учебно-научное учреждение
Факультет вычислительной математики и кибернетики
Московского государственного университета имени М.В. Ломоносова
(Факультет ВМК МГУ имени М.В. Ломоносова)

УДК № госрегистрации Инв. №	УТВЕРЖДАЮ декан академик РАН _____ Е.И. Моисеев «__» _____ 2011 г.
-----------------------------------	--

Государственный контракт от «20» сентября 2010 г. № 14.740.11.0399
Шифр заявки «2010-1.1-215-138-007»

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

в рамках федеральной целевой программы «Научные и научно-педагогические кадры
инновационной России» на 2009-2013 годы

по теме:

«СОЗДАНИЕ ПРОТОТИПА ИНТЕГРИРОВАННОЙ СРЕДЫ И МЕТОДОВ
КОМПЛЕКСНОГО АНАЛИЗА ФУНКЦИОНИРОВАНИЯ РАСПРЕДЕЛЁННЫХ
ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ (РВС РВ)»

(промежуточный, этап № 2)

Наименование этапа: «Разработка первой очереди методов и инструментальных средств»

Руководитель работ, д.ф.-м.н.
профессор

_____ Смелянский Р.Л.
подпись, дата

Москва 2011

Обозначения и сокращения

АРМ	Автоматизированное рабочее место
БИ	Бортовые интерфейсы
БПМ	Библиотека поддержки моделирования
БРЭО	Бортовое радиоэлектронное оборудование
КБО	Контур бортового оборудования
ЛВК	Лаборатория вычислительных комплексов
ЛЖ	Линия жизни
МКИО	Мультиплексный канал информационного обмена
НИР	Научно-исследовательская работа
ПНМ	Полунатурное моделирование
ПЧМ	Последовательная частная модель
РВС РВ	Распределённая вычислительная система реального времени
РИМ	Распределённое имитационное моделирование
РЧМ	Распределённая частная модель
СМ	Система моделирования
СММ	Система математического моделирования
ТЗ	Техническое задание
ЧМ	Частная модель
ЭО ТПК	Экспериментальный образец типового программного комплекса
ЯОМ	Язык описания моделей

ACID	Атомарность, Согласованность, Изоляция и Долговечность (Atomicity, Consistency, Isolation and Durability)
API	Интерфейс программирования приложений (Application Programming Interface)
DDS	Сервис распределения данных (Data Distribution Service)
DOM	Программный интерфейс для доступа к документа (Document Object Model)
EPL	Общественная лицензия Eclipse (Eclipse Public License)
FOM	Федеративная объектная модель (Federation Object Model)
HLA	Высокоуровневая архитектура (High Level Architecture)
JDK	Комплект разработчика приложений на языке Java (Java Development Kit)
RCTL	Темпоральная логика реального времени (Real-time Computation Tree Logic)
RTI	Среда имитационного моделирования
SCXML	Расширяемый язык разметки для диаграмм состояний (State Chart eXtensible Markup Language)
STL	Стандартная библиотека шаблонов (Standard Template Library)
TCP	Протокол управления передачей (Transmission Control Protocol)
UDP	Протокол пользовательских дейтаграмм (User Datagram Protocol)
UML	Универсальный язык разметки (Universal Markup Language)
XMI	Расширяемый язык разметки для обмена метаданными (eXtensible Markup Language for Metadata Interchange)
XML	Расширяемый язык разметки (eXtensible Markup Language)

Реферат

Основной целью данной НИР является разработка прототипа интегрированной программной среды с открытыми исходными кодами для поддержки разработки и интеграции РВС РВ через моделирование, а также методов количественного и качественного анализа функционирования РВС РВ. Выполнение НИР должно обеспечивать достижение научных результатов мирового уровня, подготовку и закрепление в сфере науки и образования научных и научно-педагогических кадров, формирование эффективных и жизнеспособных научных коллективов.

Основной целью второго этапа НИР была разработка первой очереди методов и инструментальных средств поддержки анализа и разработки РВС РВ через моделирование. Основное содержание работ по второму этапу следующее: разработка архитектуры среды выполнения моделей компонентов РВС РВ; разработка первой очереди методов и инструментальных средств поддержки анализа и разработки РВС РВ; экспериментальное исследование первой очереди методов и инструментальных средств поддержки анализа и разработки РВС РВ; разработка средств поддержки единого формата описания РВС РВ.

Результатом работы по второму этапу являются: промежуточный отчёт о НИР за второй этап. Промежуточный отчёт о НИР за второй этап включает в себя: описание архитектуры среды выполнения моделей компонентов РВС РВ; описание первой очереди методов и инструментальных средств поддержки анализа и разработки РВС РВ; описание экспериментального исследования первой очереди методов и инструментальных средств поддержки анализа и разработки РВС РВ; описание средств поддержки единого формата описания РВС РВ, в том числе средств поддержки языка описания моделей компонентов РВС РВ.

Таким образом, задачи, поставленные в рамках второго этапа НИР, выполнены.

Содержание

Введение	6
1 Разработка архитектуры среды выполнения моделей компонентов PBC PB	7
1.1 Общее описание архитектуры среды выполнения моделей	7
1.2 Требования к среде выполнения моделей компонентов PBC PB	8
1.3 Проектные решения по интеграции средств среды выполнения моделей	13
1.4 Интеграция и доработка CERTI RTI	18
1.5 Алгоритмы временной синхронизации.....	28
2 Разработка средств поддержки единого формата описания PBC PB, средств поддержки языка описания моделей PBC PB	37
2.1 Два уровня единого формата описания PBC PB	37
2.2 Специализированные языки моделирования.....	39
2.3 Применение диаграмм состояний UML для описания моделей PBC PB	41
2.4 Особенности создания модели на HLA	48
3 Разработка методов и инструментальных средств поддержки анализа и разработки PBC PB первой очереди	56
3.1 Средства работы с трассами.....	56
3.2 Средства трансляции UML во временные автоматы.....	69
3.3 Средства трансляции средства трансляции UML в исполняемые модели, совместимые со стандартом HLA.....	108
4 Экспериментальное исследование первой очереди инструментальных средств поддержки анализа и разработки PBC PB	118
4.1 Экспериментальное исследование форматов трасс.....	118
4.2 Экспериментальное исследование средств трансляции UML во временные автоматы	135
4.3 Применимость CERTI для моделирования.....	155
4.4 Экспериментальное исследование средства трансляции UML в исполняемые модели совместимые со стандартом HLA.....	171
Заключение	178
Список использованных источников	183

Введение

Настоящий документ представляет собой научно-технический отчет по второму этапу НИР «Создание прототипа интегрированной среды и методов комплексного анализа функционирования распределённых вычислительных систем реального времени (РВС РВ)». Документ содержит отчет по пунктам 2.1-2.4 календарного плана, в соответствии с техническим заданием (ТЗ) по государственному контракту № 14.740.11.0399 от 20 сентября 2010 г. между Государственным учебно-научным учреждением Факультет вычислительной математики и кибернетики Московского государственного университета имени М.В. Ломоносова и Министерством образования и науки Российской Федерации.

В первой главе в соответствии с пунктом 2.1 календарного плана ТЗ приводится описание архитектуры среды выполнения моделей компонентов РВС РВ.

Во второй главе в соответствии с пунктом 2.4 календарного плана ТЗ приводится описание средств поддержки единого формата описания РВС РВ, в том числе средств поддержки языка описания моделей компонентов РВС РВ

В третьей главе в соответствии с пунктом 2.2 календарного плана ТЗ описываются первой очереди методы и инструментальные средства поддержки анализа и разработки РВС РВ.

В четвёртой главе в соответствии с пунктом 2.3 календарного плана ТЗ приводятся результаты экспериментального исследования первой очереди методов и инструментальных средств поддержки анализа и разработки РВС РВ.

В заключении изложены основные результаты второго этапа НИР.

1 Разработка архитектуры среды выполнения моделей компонентов PBC PB

В данном разделе описывается архитектура среды выполнения моделей компонентов распределённых вычислительных систем реального времени (PBC PB). В разделе 1.1 приводится общее описание архитектуры среды выполнения. В разделе 1.2 описываются требования к среде выполнения моделей компонентов PBC PB. В разделе 1.3 рассматриваются проектные решения по интеграции средств среды выполнения моделей. Раздел 1.4 посвящён описанию интеграции и доработки средства CERTI RTI, выбранного в качестве основы для построения библиотеки поддержки моделирования. В разделе 1.5 описываются алгоритмы временной синхронизации, используемые при моделировании.

1.1 Общее описание архитектуры среды выполнения моделей

На предыдущем этапе данной работы [1] была предложена общая схема архитектуры среды моделирования. В соответствии с этой схемой ЭО ТПК разделяется на две подсистемы: среду выполнения частных моделей (ЧМ) и средства АРМ инженера-экспериментатора.

Среда выполнения ЧМ включает следующий набор средств (см. **Рисунок 1**):

- библиотека поддержки моделирования,
- средства регистрации и трассировки событий моделирования,
- средства обеспечения взаимодействия ЧМ с аппаратным обеспечением,
- средства оперативной (динамической) проверки выполнения спецификаций интерфейсов и поведения.

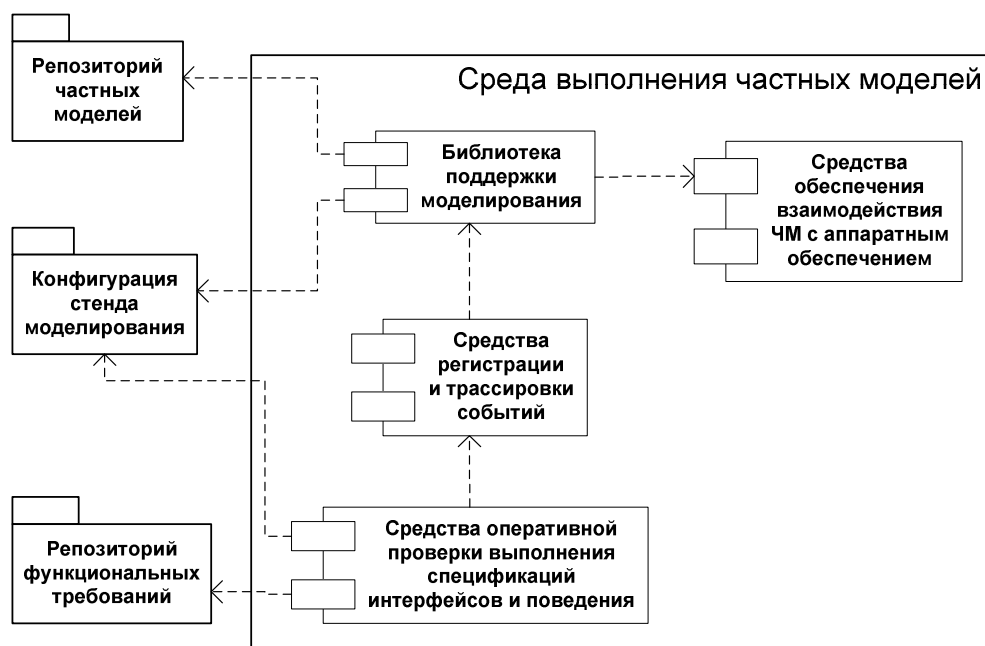


Рисунок 1. Состав средств среды выполнения ЧМ.

На втором этапе необходимо было детализировать данную архитектуру в соответствии с требованиями к среде выполнения моделей компонентов РВС РВ.

1.2 Требования к среде выполнения моделей компонентов РВС РВ

На первом этапе данной работы был выделен ряд требований к разрабатываемой системе моделирования (СМ). Рассмотрим эти требования подробно и детализируем их:

- **Представление СМ в виде чётко выделенных взаимодействующих компонентов.** Это требование вытекает из того, что нецелесообразно, исходя из возможных трудозатрат, создавать компоненты СМ “с нуля”. Поэтому необходимо максимально использовать, разработанные в открытых проектах программные компоненты. Для того чтобы это сделать СМ должно иметь чётко выделенную блочную структуру.
- **Организация выполнения набора моделей.** Разработанную модель необходимо запустить на выполнение. В имитационном моделировании выделяют распределённое имитационное моделирование (РИМ). РИМ имеет следующие достоинства [2]. Во-первых, это возможность использования вычислительных ресурсов нескольких процессоров (компьютеров) для

выполнения имитационного эксперимента. Компоненты имитационной модели распределяются по процессорам (компьютерам) и совместно участвуют в имитационном эксперименте с целью повышения производительности системы имитационного моделирования. Во-вторых, это возможность использования локальной памяти других процессоров (компьютеров). В-третьих, это возможность одновременного запуска нескольких репликаций параллельно на нескольких компьютерах, что позволяет снизить временные затраты на эксперимент. В-четвёртых, это возможность объединения уже готовых имитационных моделей и их участие в совместном имитационном эксперименте, что позволяет использовать один и тот же разработанный код в нескольких имитационных экспериментах. В-пятых, это возможность участия географически удаленных друг от друга пользователей в работе над одним имитационным проектом, что позволяет им одновременно разрабатывать модель, запускать имитационный эксперимент и одновременно наблюдать за выполнением разработанной модели. В-шестых, это возможность повышения надёжности при выполнении имитационного эксперимента, поскольку при выходе из строя процессора или компьютера, на котором выполняется один из компонентов имитационной модели, выполнение его может быть продолжено на другом процессоре (компьютере)

- ***Межмашинная синхронизация времени должна осуществляться в пределах 100 мкс.*** При проведении РИМ необходимо обеспечить корректный глобальный порядок событий. Поскольку при РИМ различные компоненты имитационной модели могут выполняться на разных процессорах (компьютерах), то между ними должна поддерживаться довольно точная синхронизация времени.
- ***Организация взаимодействия моделей.*** Современная РВС РВ представляет собой сложный программно-аппаратный комплекс, состоящий из большого количества взаимодействующих между собой устройств. Обычно каждый компонент РВС РВ описывается отдельной моделью или группой моделей, а значит актуальна задача организации взаимодействия между этими моделями.
- ***Сопряжение с аппаратурой в модельном и в реальном времени по натурным бортовым каналам.*** Одной из основных целей, поставленных перед разработчиками среды выполнения имитационных моделей в рамках

настоящей научно-исследовательской работы, является решение задачи полунатурного моделирования РВС РВ. Данный вид моделирования допускает использование имитационных моделей, часть компонентов которых представлена физическими устройствами, а часть – их программными моделями. При этом реальное оборудование, участвующее в моделировании, способно взаимодействовать с остальными компонентами исследуемой вычислительной системы лишь по predetermined набору физических каналов передачи данных и с помощью заданного ограниченного набора сетевых протоколов. Таким образом, разрабатываемая среда выполнения должна поддерживать возможность подключения устройств с помощью подходящих натуральных каналов и обеспечивать скорость выполнения программных моделей на уровне, достаточном для соблюдения спецификаций используемых протоколов передачи данных. Точность привязки модельного времени к физическому должна измеряться в десятках микросекунд. В рамках настоящего проекта в качестве участников моделирования могут выступать физические устройства, которые могут изменять своё состояние и отвечать на сообщения, полученные от программных моделей устройств за время, изменяемое в микро и даже наносекундах. Для возможности корректного построения имитационных моделей с такой точностью необходимо разработать среду выполнения с как можно меньшим временем отклика.

- **Возможность внесения отказов в бортовые каналы.** При создании бортовых вычислительных комплексов частым требованием является обеспечение заданного уровня устойчивости аппаратуры к отказам. Проверку данной характеристики распределённой системы можно частично осуществлять на этапе имитационного моделирования, задавая уровень случайных ошибок, которые возникают при передаче данных между отдельными компонентами комплекса.
- **Поддержка моделирования на различных уровнях детальности.** При детальном моделировании сложных РВС РВ зачастую не хватает вычислительных мощностей используемых для моделирования. На практике встречается такая ситуация, когда разработана очень детальная модель компонента РВС РВ, а свойства необходимые для проверки можно проверить на гораздо менее детальной модели. Например, при моделировании обработки

данных, выполняемых узлами РВС РВ, может быть вполне достаточным моделировать посылку случайных данных в нужные моменты времени. Одним из способов уменьшения сложности модели является автоматическое масштабирование детальной модели в менее детальную [3]. Поэтому при создании средства моделирования необходимо поддерживать моделирование на различных уровнях детальности.

- ***Возможность создания имитационных моделей приборов РВС РВ, а также вспомогательных моделей (например, модели внешней среды).*** Разработка вычислительного комплекса с использованием имитационного моделирования происходит поэтапно. На первой стадии каждый компонент комплекса представляется простейшей программной моделью. Постепенно модели усложняются, более точно отражая поведение реальных устройств. На более поздних этапах создания комплекса программные модели постепенно заменяются разработанными прототипами устройств, вплоть до полного исключения программных компонент. Таким образом, возможность построения моделей приборов в составе вычислительного комплекса является необходимым требованием, предъявляемым к разрабатываемой среде выполнения.
- ***Управление процессом моделирования в диалоговом режиме, либо выполнение автономного эксперимента без участия оператора.*** Как показывает практика решения задач моделирования, возможность вмешиваться в процесс проведения эксперимента является мощным и удобным средством построения, отладки и исследования свойств имитационной модели. Система моделирования должна предоставлять разработчику моделей средства для запуска, временной приостановки, возобновления и полной остановки проведения эксперимента. Так же необходимо обеспечить возможность изменения состояния модели «на лету», тем самым, давая разработчикам удобный способ интерактивной отладки имитационной модели. С другой стороны эксперименты часто состоят из набора рутинных проб с похожими или даже идентичными входными данными. Поэтому среда выполнения должна иметь встроенные средства для автономного запуска экспериментов без участия оператора.

- ***Оперативное отображение результатов моделирования в графическом и табличном виде.*** Представление результатов моделирования в графическом виде в процессе выполнения модели облегчает процессы разработки и отладки имитационной модели. Наглядное отображение процесса моделирования позволяет оценить его предварительные результаты без обработки полученных в результате моделирования трасс.
- ***Регистрация и обработка результатов моделирования, в том числе взаимодействие с аппаратными мониторами каналов передачи данных.*** Современные РВС РВ содержат сотни каналов передачи данных. Данные передаваемые через эти каналы должны не только передаваться корректно, но и вовремя [4]. Для того, чтобы проверить корректность этих данных необходимо регистрировать информацию обо всех событиях, происходящих при моделировании, но и сохранять её в форме удобной для последующей обработки. Например, графическое представление трассы проведённого эксперимента позволяет убедиться в корректности работы отдельного компонента вычислительного комплекса или же обнаружить ошибки в его поведении.
- ***Простота адаптации или автоматизация сторонних ЧМ к использованию совместно с библиотекой поддержки моделирования.*** Разработчики отдельных компонентов РВС РВ часто способны предоставить имитационные модели своего устройства, поддерживающие наиболее распространённые языки моделирования. Использование готовых имитационных моделей позволяет значительно снизить стоимость разработки нового вычислительного комплекса, поэтому среда выполнения должна поддерживать возможность подключения сторонних моделей, написанных с использованием как можно более широко диапазона языков моделирования.
- ***Интероперабельность стенда моделирования со сторонними стендами.*** Часто отдельные устройства в составе РВС РВ создаются конкурирующими или взаимно подозрительными организациями, которые отказываются предоставить разработчикам программную модель своего компонента из-за возможной утечки их технологий в процессе моделирования. Выходом из подобной ситуации может стать создание внутри организации-разработчика собственной системы с возможностью проведения совместного

моделирования. Таким образом, разрабатываемая в рамках настоящей научно-исследовательской работы система моделирования должна обладать возможностью интеграции с другими системами.

- **Стенд моделирования должен быть открытой системой.** Это требование напрямую вытекает из целей данной научно-исследовательской работы. Открытость исходных кодов комплекса позволит повысить прозрачность его функционирования, а также даст возможность применения этих исходных кодов в учебной и научно-исследовательской деятельности.

1.3 Проектные решения по интеграции средств среды выполнения моделей

В данном разделе приводятся решения об интеграции средств, обнаруженных в ходе патентного исследования, для реализации среды выполнения ЧМ в рамках ЭО ТПК.

1.3.1 Библиотека поддержки моделирования

Одним из необходимых компонентов среды выполнения частных моделей является библиотека поддержки моделирования (БПМ), выполняющая следующие функции:

- управление процессом моделирования;
- поддержка единого модельного времени;
- обеспечение ЧМ функциями работы с модельным временем и таймерами;
- поддержка взаимодействия ЧМ в процессе моделирования.

Среди возможных библиотек и сред поддержки распределённого моделирования, выявленных в ходе исследования на первом этапе, наиболее подходящими с точки зрения разработки прототипа ЭО ТПК являются следующие библиотеки [1]:

1. библиотека поддержки моделирования стенда полунатурного моделирования (Стенд ПНМ) [5];
2. библиотека OpenDDS поддержки среды выполнения распределённых систем реального времени [6];
3. библиотека CERTI RTI поддержки среды распределённого моделирования [7].

По результатам предварительного исследования в качестве основы БПМ была выбрана библиотека CERTI RTI [7]. Эта библиотека, основана на стандарте HLA [8].

Стандарт HLA утверждён организацией IEEE и применяется во многих средствах моделирования. По этой причине БПМ, разработанная в соответствии с HLA, должна прозрачно интегрироваться в одну из сред моделирования HLA RTI. Данный факт позволяет легко интегрировать ЧМ, разработанные при помощи средств, отличных от ЭО ТПК, и удовлетворяющие стандарту HLA. Возможность интеграции ЧМ в соответствии со стандартом HLA делает подход на основе этого стандарта довольно привлекательным. В тоже время в ходе анализа исследований в области средств моделирования на основе HLA не было обнаружено достоверных данных о точности моделирования на основе таких средств. Известно, что без дополнительной настройки средства RTI могут показывать неудовлетворительную производительность [9]. Тем не менее, для CERTI RTI известны рекомендации, позволяющие увеличить точность до 50 мкс [9]. Задача проверки точности средств, основанных на HLA, потребовала проведения экспериментального исследования, описанного в разделе 4.3. Данное исследование показало, что в существующем виде система моделирования CERTI может быть использовано лишь для моделирования довольно простых РВС РВ.

Одним из перспективных направлений исследований является проверка возможности интеграции БПМ, основанной на стандарте HLA, с более низким уровнем, основанным на DDS или уровнем БПМ Стенда ПНМ. Возможно, такой подход позволит совместить преимущества точного моделирования и использования распространённого интерфейса моделирования.

За основу БПМ предлагается взять библиотеку CERTI RTI, основанную на стандарте HLA. Такое решение также позволит упростить формат обмена между ЧМ, а также в перспективе позволит использовать средства анализа, основанные на HLA. Более того, если другие средства разрабатываемой ЭО ТПК будут использовать HLA, то их можно будет применять в других средах имитационного моделирования, совместимых с HLA. В данном разделе и далее описание средств ЭО ТПК приводится с учётом использования выбранного стандарта.

На **Рисунок 2** изображена структура стенда моделирования, основанного на среде моделирования HLA RTI.

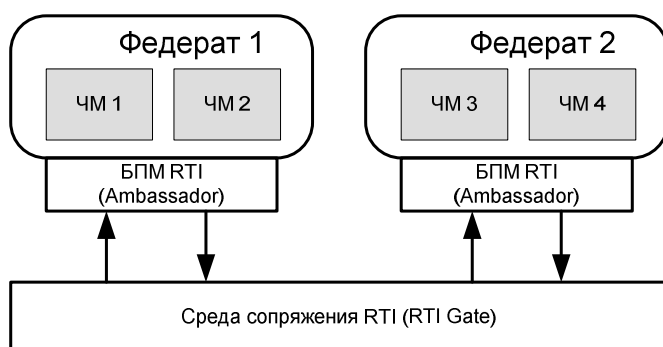


Рисунок 2. Структура стенда моделирования с использованием RTI.

Структура стенда моделирования, основанного на библиотеке CERTI RTI, изображена на рисунке 3. Одним из дополнительных преимуществ данной библиотеки является более эффективное взаимодействие между моделями посредством разделяемой памяти в том случае, если федерации, которым принадлежат модели, располагаются на одном хосте.

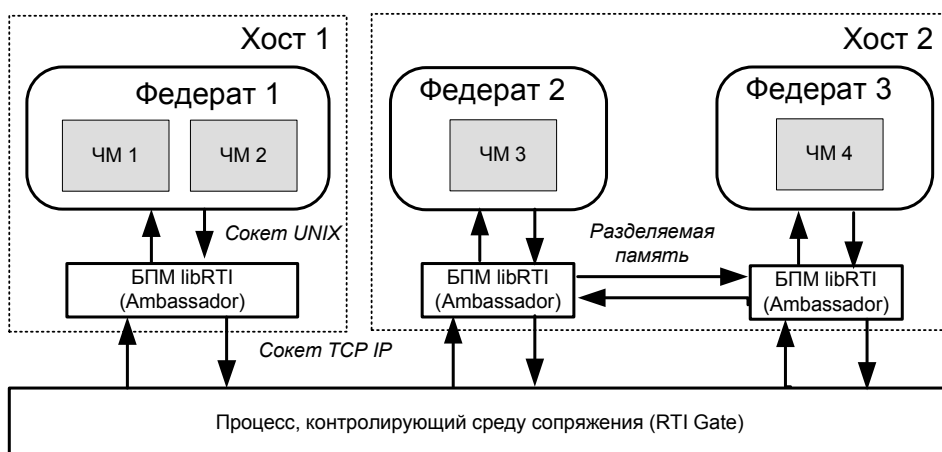


Рисунок 3. Архитектура стенда, использующего средство CERTI RTI.

1.3.2 Средства регистрации и трассировки событий моделирования

Средства регистрации и трассировки событий моделирования отслеживают изменения в атрибутах ЧМ (также называемых сигналами [5]) и обмен сообщениями. Каждый регистрируемый элемент называется событием, такой элемент обязательно сопровождается временной меткой. Регистрируемые события потребляются средствами оперативной визуализации, а также сохраняются в виде последовательности, называемой трассой. Сохранённые трассы потребляются средствами анализа поведения моделей РВС РВ, входящими в средства инженера-экспериментатора.

Известно несколько принципов регистрации событий и сбора трасс. В случае Стенда ПНМ [5] события заносятся основным процессом моделирования в область общей памяти. Вспомогательный процесс считывает из общей памяти события, передаёт средству оперативного управления и сохраняет периодически события в трассу. Схема работы изображена на рисунке 4.

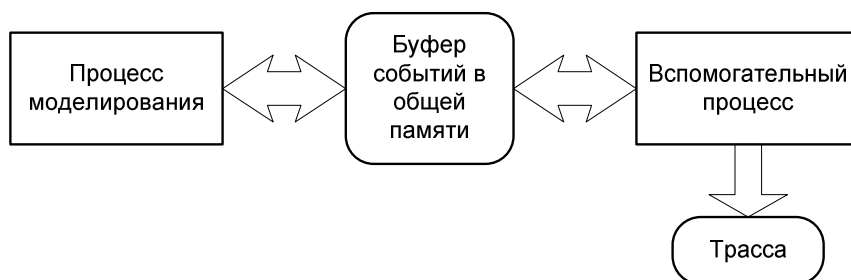


Рисунок 4. Схема сбора трасс в Стенде ПНМ.

Этот подход был адаптирован для использования с HLA RTI. Схема такой модификации изображена на рисунке 5.

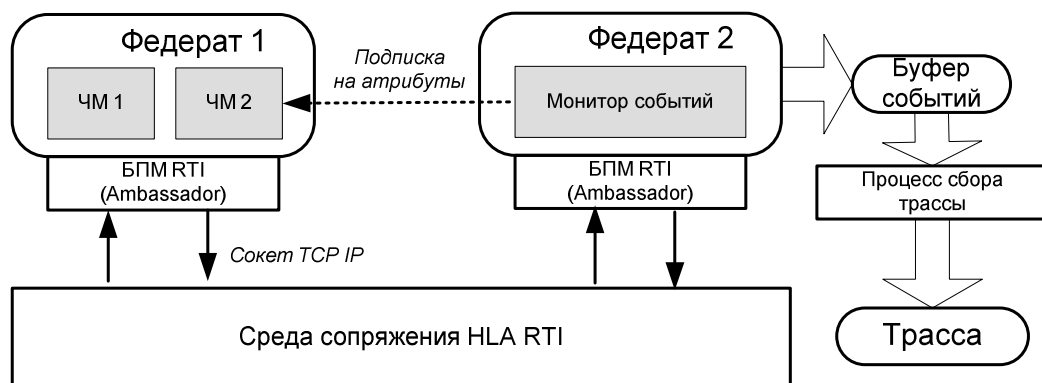


Рисунок 5. Схема сбора трассы с применением HLA RTI.

Следующий вопрос, который необходимо решить в средстве трассировки событий, это вопрос выбора формата, в который сохраняется трасса. С одной стороны, этот формат должен быть достаточно простым для того, чтобы процесс сбора трассы успевал сохранять трассу с учётом всех поступающих событий. С другой стороны, формат должен использовать компактные способы хранения трассы, так как в процессе моделирования объём накапливаемых данных может быть довольно большим, например, несколько терабайт.

В отчёте по первому этапу [1] данной работы рассматривались следующие форматы: TAU; группа ALOG, CLOG, SLOG2; EPILOG; STF; формат проекта V-Ray; TRC (формат трасс Стенд ПНМ); RaJe; OTF; CCG. Среди рассмотренных форматов были выделены

следующие форматы для дальнейшего практического исследования. Исследование, результаты которого описаны в разделах 3.1 и, показало, что наиболее перспективным является использование формата OTFz (OTF со сжатием).

1.3.3 Средства сопряжения с натурными интерфейсами

Данные средства должны обеспечивать взаимодействие библиотеки поддержки моделирования с натурными каналами, контролируемой аппаратурой. Для реализации этой цели необходимо реализовать драйвер соответствующего устройства. События, поступающие от аппаратуры, обрабатываются драйвером и транслируются изменение атрибутов ЧМ. Соответственно, изменения атрибутов транслируются обратно драйверу. При проведении экспериментального исследования, описанного в разделе 4.3, для сопряжения с натурным интерфейсом МКИО [10] использовался драйвер, разработанный в проекте стенд ПНМ.

При использовании библиотеки CERTI RTI возможен обмен такой ЧМ и процесса, взаимодействующего с драйвером, посредством общей памяти (рисунок 6).

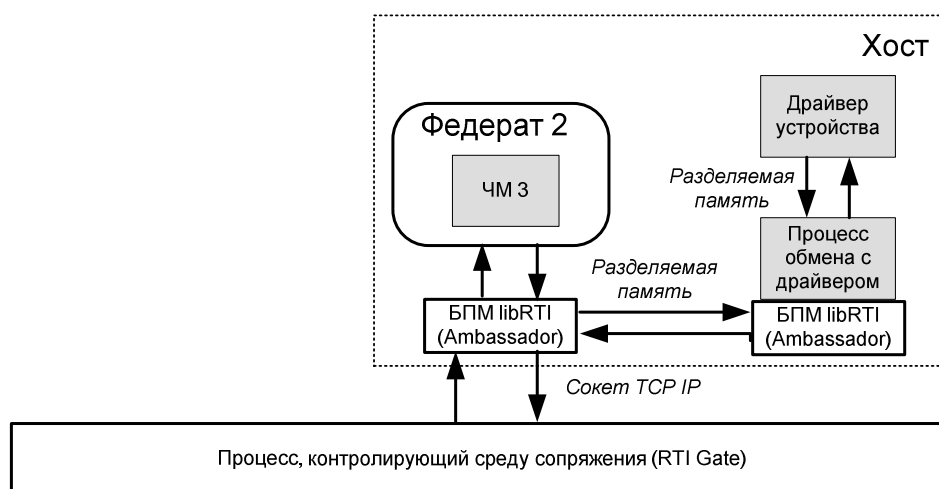


Рисунок 6. Схема сопряжения БПМ с устройством.

1.3.4 Средства оперативной (динамической) проверки выполнения спецификаций интерфейсов и поведения

Средства, которые предлагается разработать в данном разделе, основаны на принципе динамической верификации (runtime verification) формальных свойств. Обзор таких средств приводится в работе [11]. Основное преимущество таких средств заключается в проверке корректности определённых свойств на конечной трассе, сформированной в ходе процесса моделирования.

Основная идея принципа и его возможной реализации в предлагаемой архитектуре среды моделирования проиллюстрирована на **рисунке 7**. По спецификациям, заданным в виде логических формул, например, в виде формул логики линейного времени [12] формируется специальная ЧМ, называемая монитором. Основная цель этой модели – отслеживание изменений в контролируемой ЧМ. Монитор изменяет своё состояние в соответствии с изменением состояния контролируемой модели. В том случае, если монитором обнаруживается нарушение спецификации, формируется соответствующее событие о нарушении необходимого свойства.

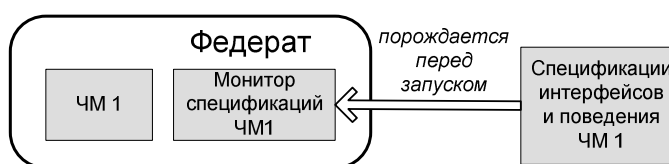


Рисунок 7. Схема применения методов динамической верификации.

Системы моделирования, рассмотренные на первом этапе данной работы [1], не используют методы динамической верификации. Поэтому актуальна задача создания такого средства. В разделе 3.2 приводится описание разработанной системы динамической верификации, а в разделе 4.2 результаты экспериментального исследования данного средства. Реализованный метод позволяет проверять соответствие интерфейса реализации ЧМ и её поведения, заданного в виде сообщений, посылаемых при помощи интерфейса, спецификации, заданной в виде логической формулы над сообщениями. В рамках такого средства предполагается проверять свойства лишь отдельных ЧМ, но не совокупности ЧМ. Такой подход должен обеспечить практическую применимость принципа динамической верификации.

1.4 Интеграция и доработка CERTI RTI

В данном разделе описывается архитектура среды выполнения, разрабатываемой на основе спецификаций стандарта распределённого моделирования HLA IEEE-1516 2000. Раздел 1.4.1 описывает архитектуру RTI в составе системы моделирования CERTI, которую было решено использовать в качестве базовой реализации RTI на первом этапе научно-исследовательской работы. В разделе 1.4.2 проводится критический анализ существующей архитектуры CERTI RTI и указываются её основные недостатки. Раздел 1.4.3 содержит набор возможных модификаций архитектуры CERTI RTI, способных повысить производительность

системы. При этом каждое предложение рассматривается не только с точки зрения потенциального увеличения производительности системы, но и оценивается с точки зрения трудоёмкости. Раздел 1.4.4 описывает ограничения, которые не позволяют использовать спецификации актуальной версии стандарта HLA, для решения задач полунатурного моделирования в реальном времени, и соответствующие доработки CERTI RTI. Раздел 1.4.5 содержит выводы по построению архитектуры среды выполнения на основе системы моделирования CERTI.

1.4.1 Архитектура CERTI RTI

Описание архитектуры

Стандарт HLA предусматривает возможность взаимодействия и синхронизации отдельных федератов имитационной модели с помощью сервисов и служб RTI. Так как федераты могут быть расположены на разных вычислительных узлах распределённой системы, то обмен данными между ними реализуется внутри RTI с помощью механизма передачи сетевых сообщений. Иногда для реализации части сервисов RTI достаточно использования информации, доступной внутри конкретного вычислительного узла. Например, логическое время федерата обычно хранится локально внутри части RTI, связанной с федератом непосредственно, и его запрос не требует дополнительного согласования с другими федератами. Использование остальных сервисов RTI приводит к отправке или получению, вообще говоря, нескольких сетевых сообщений между участниками моделирования. Стандарт HLA не предписывает разделение локальных и глобальных сервисов и оставляет данный аспект на усмотрение разработчиков его конкретных реализаций.

Архитектура CERTI RTI [13], используемой в рамках настоящего проекта, состоит из одного глобального процесса RTI Gate (RTIG), отвечающего за управление выполнением имитационной модели, нескольких локальных процессов RTI Ambassador (RTIA), и библиотеки libRTI, обеспечивающей взаимно-однозначную связь федерата с соответствующим ему процессом RTIA (см. [рисунок 8](#)). С одной стороны взаимодействие между глобальным процессом RTIG и локальными процессами RTIA происходит через сокеты сетевого протокола TCP/IP, с другой стороны RTIA обменивается сообщениями с процессами федератами через локальные каналы передачи данных (сокеты UNIX).

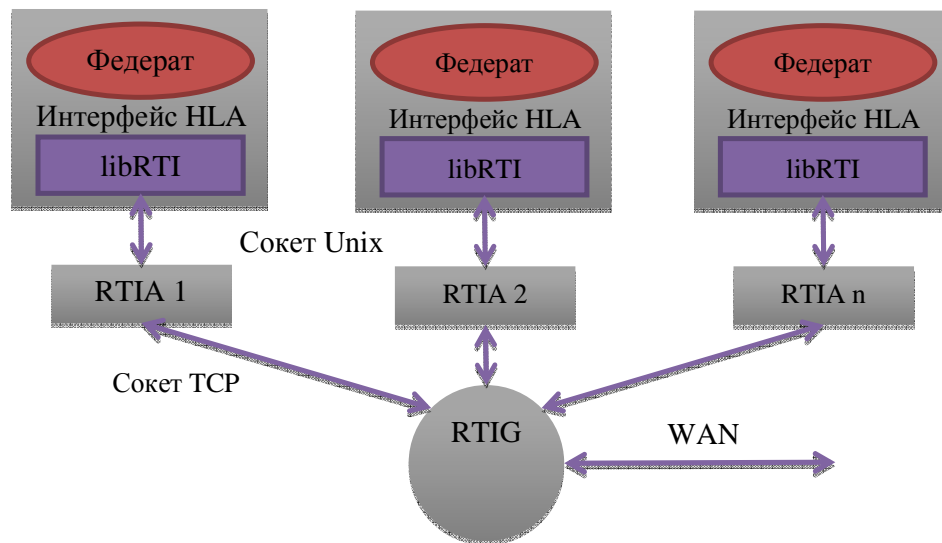


Рисунок 8. Архитектура RTI реализации CERTI.

Запуск RTI происходит в несколько этапов. Прежде всего, запускается глобальный процесс RTIG, который будет координировать работу подключённых компонентов RTI. В общем случае каждый участник моделирования может представлять собой независимую программу. При подключении нового участника к федерации, слинкованная с ним библиотека libRTI создаёт на инструментальной машине новый процесс RTIA, и связывается с ним через сокет UNIX. Далее созданный процесс RTIA устанавливает TCP соединение с глобальным процессом RTIG.

В CERTI основная часть сервисов RTI реализована внутри глобального процесса RTIG, хранящего всю информацию, необходимую для управления выполняемой федерацией. Остальные компоненты RTI служат для обмена информацией между процессом RTIG, и подключёнными федератами. Таким образом, в CERTI используется централизованная архитектура и часто синхронизация распределённой имитационной модели, сводится к последовательному выполнению запросов федератов на единственном вычислительном узле. Например, обычная схема выполнения запроса состоит из передачи сообщения от libRTI к связанному с федератом процессу RTIA, последующей передаче сообщения от RTIA к глобальному процессу RTIG, обработки запроса внутри процесса RTIG и отправки уведомления для федерата обратной дорогой.

Достоинства

Основное достоинство описанной архитектуры RTI – простота реализации глобальных алгоритмов синхронизации имитационной модели. Подобная архитектура автоматически приводит к выполнению так называемых принципов «ACID» [14]. ACID – это

акроним, описывающий требования к транзакционной системе, выполнение которых обеспечивает более надёжную и предсказуемую её работу. Он расшифровывается как Atomicity (Атомарность), Consistency (Согласованность), Isolation (Изоляция) и Durability (Долговечность). Эти принципы справедливы, например, для систем управления базами данных. Для имитационных моделей эти принципы заключаются в следующем:

1. Каждое изменение состояния имитационной модели *атомарно*, то есть происходят лишь в случае успешного выполнения запроса. Если поступивший запрос не может быть выполнен на момент его обработки, то он отклоняется. Подобная ситуация возможна, например, когда корректный на момент своего образования запрос не может быть выполнен из-за изменений в состоянии модели, вызванных обработкой других запросов;
2. Так как вся необходимая информация содержится внутри процесса RTIG, то разработчикам не нужно заботиться о *согласованности* состояния процессов модели – достаточно обеспечивать её для локальных данных;
3. Запросы, поступающие от федератов модели, обрабатываются процессом RTIG последовательно. Таким образом, выполнение поступивших запросов не может влиять друг на друга, что обеспечивает их изолированность;
4. Изменение состояния модели долговечно – ошибки выполнения имитационной модели не могут повлиять на результаты успешных запросов.

Простота архитектуры CERTI неоднократно позволяла данному проекту с открытым исходным кодом играть роль исследовательской базы для проверки новых веяний имитационного моделирования на практике. Например, в рамках проекта CERTI была разработана система обеспечения конфиденциальности данных подключаемого федерата, позволяющая предотвратить возможные утечки информации при моделировании, проводимом несколькими компаниями [15]. Так же на базе CERTI было разработано несколько эффективных алгоритмов временной синхронизации [16].

Таким образом, архитектура среды выполнения имитационных моделей, основанной на CERTI, позволяет относительно просто проводить эксперименты с алгоритмами, используемые в распределённых системах моделирования. Подобные наработки можно будет использовать в дальнейшем при реализации новой среды выполнения моделей.

1.4.2 Ограничения архитектуры CERTI RTI

Централизованная модель

Централизованная архитектура CERTI обладает и рядом недостатков, главным из которых является большая загруженность процесса RTIG. При активном обмене информацией между федератами глобальный процесс RTIG становится узким местом. Кроме того, централизация управления федерацией порождает необходимость пересылки большого числа сетевых сообщений, что пагубно сказывается на производительности системы в целом.

Пусть первый федерат пересылает данные другому федерату. При использовании ярко выраженной централизованной архитектуры пересылка будет происходить в два этапа: сначала первый федерат отправит данные центральному компоненту RTI, затем центральный компонент переправит их федерату-получателю. При использовании децентрализованной архитектуры часто можно обойтись без двойной пересылки данных, или отправлять данные одновременно федерату-получателю и центральному компоненту RTI. В последнем случае большая эффективность будет достигнута за счёт параллельной, а не последовательной отправки запросов.

Избыточное конвертирование параметров запроса

Рассмотрим процесс выполнения запроса федерата подробнее. При вызове федератом сервиса RTI, связанная с ним библиотека libRTI производит сериализацию параметров запроса и отправляет соответствующее сообщение локальному процессу RTIA. Процесс RTIA декодирует поступившее от федерата сообщение и производит его обработку. Ввиду высокой степени централизации архитектуры CERTI, в большинстве случаев это приводит к новой сериализации параметров запроса и дальнейшей отправке сообщения глобальному процессу RTIG. Там запрос федерата ожидает своей очереди на обработку. Как правило, обработка запроса приводит к отправке одного или нескольких сообщений заинтересованным процессам RTIA. Со своей стороны процесс RTIA декодирует поступившее сообщение и, возможно, посылает соответствующее сообщение обратно федерату. После этого федерат может считать запрос выполненным. Таким образом, практически каждый запрос федерата приводит к необходимости передачи двух локальных и двух сетевых сообщений, и, как следствие, сравнительно низкой производительности.

Поддержка полунатурного моделирования

Одним из обязательных требований к среде выполнения, разрабатываемой в рамках настоящего проекта, является возможность полунатурного моделирования реального времени. Таким образом, выполняемая имитационная модель должна обладать как можно

меньшим временем отклика, чтобы укладываться в директивные интервалы и успевать взаимодействовать с оборудованием, подключённым физически. Поэтому проблема эффективности работы среды выполнения становится особенно важной, а дистрибутив CERTI, вероятно, потребует дальнейшей доработки.

1.4.3 Предложения по развитию CERTI RTI

Перераспределение функциональности между компонентами RTI

Особенностью архитектуры CERTI является значительная централизация функциональности RTI внутри глобального процесса RTIG и сравнительно малая роль локальных процессов RTIA. Как показывают результаты тестирования нескольких коммерческих RTI [17], другое распределение обязанностей между глобальным и локальными компонентами этой программной прослойки позволяют достичь существенно большей производительности. Таким образом, в процессе усовершенствования и доработки CERTI может быть полезным произвести перераспределение ролей между отдельными компонентами этого программного продукта.

Упрощение взаимодействия между компонентами RTI

Проект CERTI ведётся Французской Аэрокосмической Лабораторией (ONERA) с 1996 года [18]. Сравнительно новая парадигма параллельного программирования – многонитевые программы – тогда только начинала внедряться на практике [19]. Видимо, это послужило причиной построения базовой архитектуры CERTI без использования процессов-нитей. До недавнего времени все процессоры были одноядерными, и использование только полновесных процессов не приводило к существенному падению производительности. Однако с появлением многоядерных архитектур, способных эффективно выполнять несколько нитей одновременно, потери стали более ощутимыми.

Механизм легковесных процессов выгодно использовать вместо связки RTIA & libRTI. Взаимодействие между отдельными процессами-нитеями может быть организовано эффективнее, чем взаимодействие полновесных процессов. В текущей архитектуре CERTI, федерат и соответствующий ему процесс RTIA обмениваются данными с помощью передачи сообщений, передающихся через сокет UNIX. При этом передаваемые параметры должны быть сериализованы перед передачей и декодированы после неё. Использование легковесных процессов позволяет корректно передавать данные с помощью использования механизма семафоров. При этом появляется возможность прямого обращения к параметрам

запроса и не возникает необходимости конвертирования их в специальный формат для передачи.

Использование процессов-нитей, очевидно, позволяет достичь большей производительности даже в рамках существующей архитектуры CERTI. Например, процесс RTIA часто вынужден обрабатывать запросы, поступающие со стороны RTIG и libRTI одновременно. В текущей реализации CERTI подобные запросы выполняются последовательно, однако, как правило, они не связаны между собой, и могут выполняться в разных нитях параллельно. Подобные рассуждения справедливы и для глобального процесса RTIG, где часто существует возможность параллельного выполнения независимых запросы федератов.

Децентрализация архитектуры RTI с использованием нитей

Централизованная архитектура CERTI особенно неэффективна в случае, если взаимодействующие федераты находятся на одной и той же инструментальной машине. В текущей реализации каждому федерату выполняемой имитационной модели однозначно соответствует локальный процесс RTIA. При этом процессы RTIA взаимодействуют между собой лишь через глобальный процесс RTIG. Таким образом, если два федерата обмениваются данными, то осуществляется их сетевая передача, даже в случае, если они находятся на одной и той же инструментальной машине.

Одним из решений данной проблемы может стать добавление дополнительного канала передачи данных между процессами RTIA. В качестве альтернативного решения можно использовать один процесс RTIA для каждой инструментальной машины, подключая к нему сразу несколько федератов. При таком подходе, вероятно, будет выгодным использовать внутри процесса RTIA несколько легковесных процессов. Стоит заметить, что оба предложенных решения ведут к децентрализации и усложнению архитектуры CERTI.

Объединение нескольких логических процессов внутри одного федерата

Существует ещё один путь решения проблемы – построение дополнительной функциональной прослойки поверх интерфейсов CERTI. Текущая версия CERTI предполагает, что существует взаимно однозначное соответствие между физическими процессами моделируемой системы и подключёнными к RTI федератами. Однако каждый федерат может объединять внутри себя несколько моделей таких процессов, называемых *логическими процессами* имитационной модели. Логические процессы могут работать параллельно друг с другом и представлять собой на аппаратном уровне, например, разные нити одного полновесного процесса.

На настоящий момент существует ряд проблем на пути данного решения. Стандарт HLA IEEE-1516, выпущенный в 2000 году, никак не рассматривал возможность использования нескольких нитей внутри федерата. Поэтому не удивительно, что реализация CERTI, которая стала поддерживать данный стандарт только в конце 2010 года, не позволяет использовать процессы нити. Стоит заметить, что более новый стандарт HLA IEEE-1516 версии 2010 уже содержит требования, необходимые для обеспечения работы мульти-нитевого режима и возможности вложенного вызова сервисов RTI, когда новые запросы к RTI производятся непосредственно при обработке поступивших от неё сообщений. К сожалению, на данный момент разработчики CERTI не ставят в ближайшие планы развития добавление подобных возможностей.

1.4.4 Ограничения текущей версии стандарта HLA для моделирования в реальном времени

Quality of Service (QoS)

Одним из сценариев использования разрабатываемой среды выполнения является полунатурное моделирование распределённых встроенных систем реального времени. При этом подключаемое физическое оборудование налагает свои требования к имитационной модели. В частности, поведение физического оборудования задаёт директивные интервалы, в течение которых выполняемая имитационная модель должна произвести определённое действие (например, отослать ответное сообщение). Для гарантированного обеспечения реакции системы в заданном доверительном интервале система реального времени должна обладать свойством *предсказуемости* – должна существовать возможность определения наихудшего времени выполнения задания и точного предсказания соответствующего ему времени [8].

Последняя доступная версия стандарта HLA IEEE-1516 не предусматривает своего использования для моделирования реального времени. Предположительно, такая возможность появится в спецификациях следующей версии стандарта HLA – HLA NG, разработка которого активно ведётся в настоящее время. Таким образом, на данный момент совместимая со стандартом HLA модель не вправе требовать какого-либо контроля качества от RTI.

В текущей версии стандарта HLA можно выделить следующие основные проблемы, препятствующие возможности проводить с его помощью моделирование систем реального

времени, где производительности и предсказуемость системы имеют первостепенное значение:

1. Стандарт HLA не предоставляет необходимых интерфейсов для задания требуемых параметров качества сервиса. Поэтому для проверки удовлетворения директивным интервалам требуется расширить существующий интерфейс или же разработать надстройку, предоставляющую возможности, необходимые для моделирования реального времени;
2. Программная прослойка RTI работает поверх существующей операционной системы и не может предоставить средств, необходимых для управления распределением ресурсов инструментальной машины. При таком положении, операционная система может, например, неожиданно откатить из оперативной памяти страницы процессов моделирования или дать другим существующим задачам больший приоритет в использовании процессорного времени. В то же время системы моделирования, не использующие стандарт HLA, позволяют интегрироваться с ОС [20];
3. В случае распределённого имитационного моделирования стандарт HLA поддерживает лишь два типа передачи данных: надёжный и ненадёжный (обычно выраженных в реализациях CERTI в виде использования протоколов транспортного уровня TCP и UDP соответственно). Однако таких каналов недостаточно для проведения моделирования реального времени.

Описанные ограничения справедливы и для реализации CERTI. Одним из возможных путей решения проблем может стать использование дополнительного программного слоя для передачи данных с поддержкой необходимых для моделирования реального времени политик QoS. Существует множество стандартов, обладающих необходимой функциональностью, и множество соответствующих им реализаций. Один из наиболее распространённых и прогрессивных стандартов в данной области – стандарт OMG Data Distributed Service (DDS) [21].

Спецификации OMG DDS определяют стандарт взаимодействия процессов, применимый к широкому классу распределённых систем реального времени и встроенных систем (DRE). В основе DDS лежит ориентированная на данные модель с архитектурой вида издатель-подписчик (DCPS). Процессы, использующие DDS, могут производить чтение и запись типизированных данных. При этом модель DCPS образует прослойку, которая позволяет читающим данные процессам получить доступ к самой последней их версии. В

рамках DCPS определяются глобальное пространство данных, которые можно читать и изменять, и глобальное пространство имён, позволяющее создавать новые и искать существующие разделяемые объекты. Стандарт DDS определяет большое количество политик QoS и способов обмена данными между взаимодействующими процессами.

Суммируя вышесказанное, разрабатываемая среда выполнения имитационных моделей в итоге должна быть сформирована вокруг концепций заложенных в стандарт распределённого моделирования HLA. Из-за предъявляемых спектром предполагаемых задач требований, таких как поддержка разнообразных QoS и не описанных спецификациями стандарта HLA, в новую среду выполнения необходимо добавить дополнительный уровень для передачи данных и расширить функциональность, предоставляемую сервисами RTI с помощью системы, основанной на стандарте DDS.

1.4.5 Выводы

Централизованная архитектура CERTI RTI обладает качествами, полезными для разработки новых алгоритмов работы распределённой среды выполнения имитационных моделей, их быстрой реализации и тестирования на практике. Однако простота существующей архитектуры не позволяет CERTI RTI достичь показателей производительности своих коммерческих конкурентов.

В данной главе был рассмотрен ряд возможных изменений архитектуры CERTI RTI, способных значительно увеличить её эффективность. Каждое из предложенных изменений обладает различным потенциалом и требует разного объёма усилий от разработчиков среды выполнения. Наиболее выгодной правкой представляется внедрение в прототип легковесных процессов-нитей, позволяющих упростить синхронизацию между компонентами RTI. Более сложным для разработчиков, но не менее перспективным является перераспределение функциональности между компонентами RTI.

Необходимость поддержки полунатурного моделирования в реальном времени приводит к возникновению более фундаментальных архитектурных проблем. Интерфейсы стандарта HLA версии IEEE-1516 2000 не рассчитаны на применение для решения подобных задач. Таким образом, от разработчиков среды выполнения требуется не только построить средство, эффективно реализующее требования существующего стандарта, но и расширить его спецификации. При этом возникает новое множество технических проблем. В частности, необходимо обеспечить возможность выбора контроля качества для каналов передачи данных. Для этого предполагается использовать стандарт OMG DDS.

1.5 Алгоритмы временной синхронизации

В данном разделе описываются алгоритмы временной синхронизации. Раздел 1.5.1 описывает методы продвижения времени, поддерживаемые в стандарте HLA. В разделе 1.5.2 описывается консервативная схема продвижения модельного времени. Раздел 1.5.3 содержит описание смешанной схемы продвижения модельного времени. Раздел 1.5.4 содержит выводы по внедрению смешанной схемы продвижения модельного времени в среду выполнения на основе системы моделирования CERTI

1.5.1 Методы продвижения времени стандарта HLA

Особенностью распределённых программ имитационного моделирования является необходимость согласования составляющих её процессов в едином модельном времени. Соответствующий механизм согласования процессов называется алгоритмом временной синхронизации имитационной модели. Существует множество различных алгоритмов синхронизации [22].

В рамках стандарта имитационного моделирования HLA IEEE-1516 2000 [23] предполагается, что состояние изучаемой системы изменяется мгновенно в дискретные моменты времени, причём каждому такому изменению соответствует событие – действие одного из участвующих в моделировании федератов. Каждый присоединённый к RTI федерат обладает своим *логическим временем* и способен генерировать собственный поток событий, независимо от других федератов. Основная задача группы сервисов продвижения времени HLA – упорядочивание разрозненных потоков событий, поступающих от отдельных федератов.

С точки зрения RTI состояние выполняемой имитационной модели может изменяться только вследствие вызова федератом одного из сервисов системы. Таким образом, вводится соответствие между событиями и действиями федератов. Для описания механизмов продвижения модельного времени стандарт HLA описывает каждое действие любого федерата в виде пересылки сообщения между этим федератом и RTI. Например, в терминах данной абстракции каждое изменение значения разделяемого объекта соответствует отправке сообщения от федерата к RTI, а уведомление федерата о произведённом изменении – сообщению в обратном направлении.

Любое сообщение, получаемое федератом, соответствует запросу со стороны RTI. Однако стандарт HLA допускает подобные запросы только в случае готовности федерата их обработать. Таким образом, сообщения, адресованные федератам, содержатся внутри RTI,

пока адресат не сможет их принять. Для хранения таких сообщений внутри RTI предусмотрены специальные упорядоченные контейнеры.

В зависимости от правил сортировки внутри контейнера выделяется два типа сообщений: упорядоченные по времени их получения инструментальной машиной (Receive Order, RO) и ранжируемые в соответствии с модельным временем (Time Stamp Order, TSO). Для сообщений типа RO внутри федерата создаётся очередь типа FIFO: новые сообщения помещаются в конец очереди, а обработка сообщений производится с её начала. Пусть два федерата пересылают сообщения типа RO третьему федерату, используя соответствующие сервисы RTI. Подобные сообщения передаются получателю в порядке их появления. Поэтому даже кратковременная ошибка канала передачи данных может изменить порядок получения событий и, как следствие, результаты эксперимента. Таким образом, порядок получения сообщений RO определён нестрогим и, вообще говоря, может меняться от эксперимента к эксперименту. В отличие от сообщений типа RO, сообщения TSO позволяют определить порядок доставки сообщений более точно. Для сообщений типа TSO внутри каждого федерата создаётся отдельная очередь, сообщения в которой ранжируются по возрастанию прикреплённых к ним временных меток. Метка сообщения не может быть меньше текущего значения логического времени федерата, то есть федерат не может отправлять сообщения «задним числом».

С точки зрения сервисов управления продвижением времени стандарта HLA существует два типа федератов – *управляющие* (Time Regulating) и *сдерживаемые* (Time Constrained). Один и тот же федерат может одновременно являться и управляющим и сдерживаемым или не принадлежать ни к одной из вышеперечисленных категорий (в данном случае обмен информацией с этим федератом возможен лишь через сообщения типа RO). Сообщения типа TSO могут отправляться лишь управляющими федератами и приниматься лишь сдерживаемыми федератами. Если федератом-отправителем было создано сообщение типа TSO, а федерат-получатель не может его принять, то сообщение конвертируется к формату RO, с которым способен работать любой федерат.

В процессе выполнения имитационной модели каждый регулирующий или сдерживаемый федерат может попытаться увеличить своё логическое время с помощью соответствующих сервисов RTI. Стандарт HLA описывает несколько способов продвижения времени:

1. Time Advance Request (TAR) – получить все адресованные ему RO-сообщения и TSO-сообщения с временной меткой меньше или равной заданному значению, а

затем продвинуть до него логическое время федерата. При выполнении данного запроса RTI не только доставляет федерату все подходящие сообщения, которыми располагает на данный момент, но и гарантирует, что временный метки всех TSO-сообщений, которые будут доставлены федерату в будущем, будут превосходить запрошенное логическое время;

2. Time Advance Request Available (TARA) – семантика данного запроса продвижения времени аналогична запросу вида TAR. Разница заключается в том, что RTI не даёт гарантии получения всех TSO-сообщений. Некоторые или даже все сообщения с временной меткой равной запрашиваемому значению могут не быть доставлены. Такие сообщения будут получены федератом лишь при следующей попытке продвижения своего логического времени;
3. Next Message Request (NMR) – ждать новых сообщений типа TSO с временной меткой меньше заданного значения. При появлении такого TSO-сообщения прервать ожидание федерата, получить все адресованные федерату сообщения типа RO и TSO-сообщения с временной меткой меньше или равной значению временной метки доступного TSO-сообщения, продвинуть до этого времени логическое время федерата. Таким образом, при выполнении запроса продвижения типа NMR логическое время федерата может быть продвинуто до величины отличной от значения, заданного в момент вызова запроса. Если же после выполнения федератом запроса NMR не может быть получено ни одного TSO-сообщения, удовлетворяющего поставленным условиям, то RTI будет действовать аналогично запросу TAR;
4. Next Message Request Available (NMRA) – метод продвижения времени обладающий чертами вышеописанных запросов TARA и NMR. Отличие от запроса NMR заключается в отсутствии гарантии получения всех TSO-сообщений, с временной меткой равной новому логическому времени федерата;
5. Flash Queue Request (FQR) – получить все хранящиеся в очередях сообщения обоих типов и продвинуть логическое время федерата до заданного значения. В отличие от предыдущих методов продвижения времени, данный запрос не даёт никаких гарантий. Федерату могут быть доставлены сообщения «из прошлого», то есть с временной меткой меньшей его текущего логического времени;

Представленные способы продвижения времени можно условно разделить на две группы. Первая группа запросов позволяет использовать так называемую *консервативную*

схему продвижения модельного времени, при которой RTI не позволяет выполнить событие до тех пор, пока в распределённой системе существуют факторы, способные на него повлиять. Пусть, например, все участники моделирования посылают сообщения о своём текущем задании через равные интервалы модельного времени. Вспомогательный федерат ведёт статистику о простаивающих федератах и регулярно сохраняет их число. В данном случае RTI не позволит вспомогательному федерату продвинуться до следующей контрольной точки, пока не сможет гарантировать, что все уведомления с подходящим модельным временем доставлены. К данной группе относятся все описанные запросы за исключением FQR, при выполнении которого RTI не даёт федерату никаких гарантий.

Запрос продвижения модельного времени FQR относится ко второй группе, позволяющей реализовать *оптимистическую* схему. Федераты, использующие подобную схему продвижения времени должны быть готовы к появлению устаревших сообщений, то есть сообщений, которые должны быть получены в прошлом. При получении подобных сообщений федератам часто приходится откатывать своё состояние назад и изменять свои действия. Поэтому федераты должны обладать возможностью сохранения своего состояния и отката до заданной контрольной точки.

Приоритетным направлением для настоящего проекта является полунатурное моделирование распределённых вычислительных систем реального времени. В данном виде моделировании в экспериментах допускается участие физических устройств, для корректной работы которых имитационная модель должна укладываться в директивные интервалы. Среда выполнения обязана успевать передать сообщения, адресованные оборудованию, в заданный временной интервал и гарантировать, что устройства не будут получать устаревшие сообщения. Таким образом, в рамках настоящего проекта большой интерес представляет консервативная схема продвижения времени и, соответствующие ей сервисы стандарта HLA: TAR, TARA, NMR и NMRA.

1.5.2 Консервативная схема продвижения модельного времени

Рассмотрим консервативную схему продвижения времени подробнее [24]. Пусть к каждому участнику моделирования прикрепляется по одной упорядоченной очереди TSO-сообщений для каждого из остальных участников. Тогда простейший алгоритм продвижения логического времени федерата можно представить в виде повторения следующих инструкций:

1. Подождать, пока в каждой очереди появится хотя бы одно сообщение;

2. Удалить из очереди сообщение с минимальной меткой среди всех сообщений;
3. Обработать удалённое сообщение и увеличить логическое время федерата до значения временной метки этого сообщения.

Однако если участники моделирования будут действовать по приведённому алгоритму, то возможна тупиковая ситуация. Например, на **рисунке 9** ни один федерат не может увеличить своё логическое время. Для исключения подобных ситуаций RTI нуждается в дополнительной информации об участниках моделирования. Поэтому в стандарт HLA вводятся понятия Lookahead – минимальная разница между текущим логическим временем федерата и временными метками отправляемых им сообщений и Greatest Available Logical Time (GALT) – наибольшее значение, до которого может быть безопасно увеличено логическое время федерата.

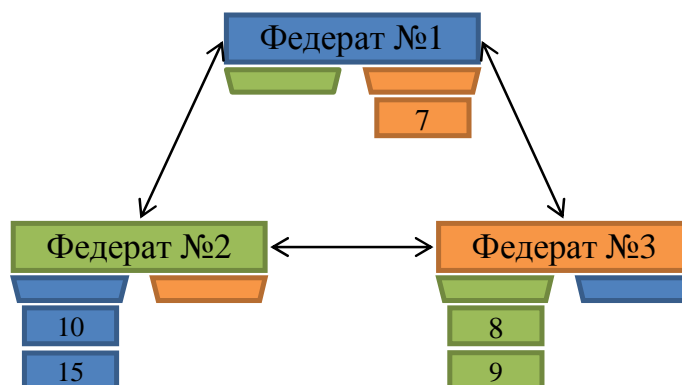


Рисунок 9. Модельное время федерации не может быть увеличено.

В соответствии с известным алгоритмом Chandy-Mistra-Briant [24] после завершения обработки сообщения федерат передаёт остальным участникам моделирования специальное NULL-сообщение с временной меткой равной значению его GALT. Сообщения данного вида обрабатываются как и любые другие, но не вызывают планирования федератом новых событий. Вернёмся к рассмотренному ранее примеру. Пусть параметр Lookahead равен 3 для всех федератов (см. **рисунок 10**). Федерат №3 только что закончил обработку события и его логическое время равно 5. Тогда данный федерат рассылает остальным участникам моделирования NULL-сообщение с временной меткой 8 (сумма его логического времени и значения параметра Lookahead). Федерат №2 в соответствии с рассмотренным алгоритмом продвигает своё логическое время до 8, и отправляет остальным участникам NULL-сообщение с меткой 11. Теперь федерат №1 может обработать хранящееся в его очереди обычное сообщение.

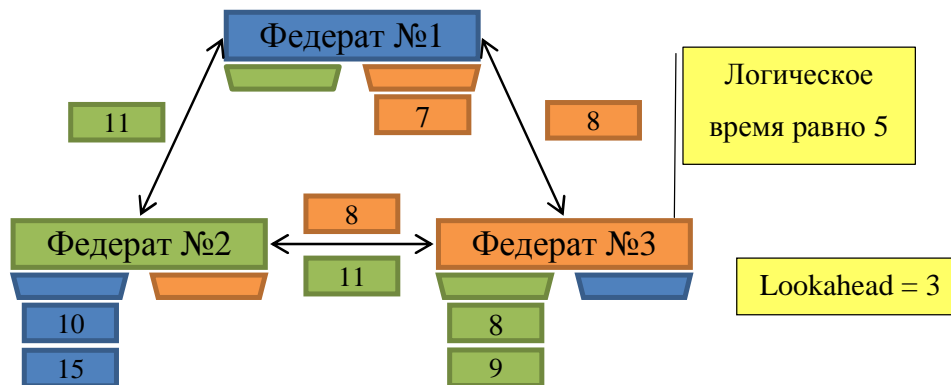


Рисунок 10. Пересылка NULL-сообщений.

Таким образом, для разрешения тупика потребовалось произвести две пересылки сообщений. В то же время, при небольших значениях параметра Lookahead возможно появление циклической пересылки сообщений, что приводит к резкому падению эффективности системы. Например, при значении параметра Lookahead, равного половине условной единицы, для разрешения тупика в рассмотренном примере потребуется уже 5 пересылок NULL-сообщений.

Существует множество модификаций алгоритма Chandy-Mistra-Briant [16,24,25], которые могут улучшить производительность системы в целом и уменьшить синхронизационные потери системы моделирования. К сожалению, для всех подобных алгоритмов справедливо общее правило – при уменьшении значения параметра Lookahead они становятся менее эффективными. Для таких имитационных моделей RTI не может гарантировать безопасности продвижения времени на сколь-нибудь значимый промежуток времени и федераты выполняются практически последовательно.

В то же время в рамках настоящего проекта в качестве участников моделирования могут выступать физические устройства, которые могут обладать значением предварительного просмотра Lookahead, изменяемым в микро- и даже наносекундах. Таким образом, разработка более эффективных и оптимизация существующих алгоритмов продвижения модельного времени представляет собой важнейшую задачу данного научно-исследовательского проекта.

1.5.3 Смешанная схема продвижения модельного времени

Консервативная и оптимистическая схема

В сравнении с рассмотренной консервативной схемой продвижения времени оптимистическая схема может быть более эффективной, особенно при небольших значениях предварительного просмотра Lookahead. Там где консервативная схема позволяет организовать выполнение федератов, близкое к последовательному, способность оптимистической схемы к продвижению времени без необходимости ожидания получения гарантии безопасности от RTI даёт возможность их параллельного выполнения.

Выигрыш в производительности, связанный с применением оптимистической схемы продвижения времени, существенно уменьшается из-за необходимости время от времени сохранять состояние федерата и восстанавливать его в случае получения им «просроченного» сообщения. На практике восстановленные федераты часто посылают новые сообщения «из прошлого», что приводит к необходимости откатывать федераты циклически и пагубно влияет на производительность системы. Существует множество вариантов оптимистического алгоритма продвижения времени, различающихся, например, числом необходимых контрольных точек. Однако, любой из них требует принципиальной возможности сохранения состояния федерата, что делает разработку среды выполнения и имитационной модели в целом более трудоёмкой. Кроме того, реализация оптимистической схемы продвижения времени требует существенно больше памяти, по сравнению с рассмотренной консервативной схемой.

Каждая из приведённых схем продвижения времени обладает своими достоинствами и недостатками. Выбор эффективного алгоритма временной синхронизации в значительной мере зависит от решаемой имитационной задачи. Обычно приводятся следующие алгоритмы в пользу использования той или иной схемы [24]:

1. Если имитационная задача позволяет использовать достаточно большое значение параметра предварительного просмотра Lookahead и его вычисление незначительно увеличивает накладные расходы на разработку модели, то рекомендуется использовать консервативный алгоритм.
2. В остальных случаях оптимистический алгоритм, вероятнее, будет демонстрировать лучшую производительность. Однако не стоит забывать о его

недостатках: реализация оптимистического алгоритма более сложная и требует большого количества памяти.

Смешанная схема

Стандарт HLA IEEE-1516 2000 позволяет использовать для разных федератов различные методы продвижения модельного времени. В рамках одной и той же имитационной модели можно совмещать консервативные и оптимистические схемы. Поэтому, несмотря на специфику задачи полунатурного моделирования встроенных систем, в разрабатываемой среде выполнения может использоваться и оптимистическая схема синхронизации. При этом некоторые участники моделирования, реализованные программно, могут не дожидаться получения всех TSO-сообщений от остальных участников.

Смешанная схема продвижения модельного времени объединяет элементы как консервативного, так и оптимистического подхода. Это делает среду выполнения более сложной и требует больших усилий разработчиков, как на этапе построения распределённой системы, так и при создании имитационных моделей. Необходимость давать гарантии при использовании консервативной схемы продвижения времени пагубно воздействует на возможность опережающего выполнения федератов, увеличивающих своё логическое время с помощью механизмов оптимистической схемы. Поэтому использование смешанной схемы продвижения времени эффективно только в случае, когда имитационная модель может быть представлена в виде набора кластеров логических процессов, связанных между собой слабыми связями. Таким образом, использование смешанной схемы требует разработки средств анализа компонентов имитационной модели, позволяющих выбрать более выгодные алгоритмы для отдельных федератов. Данные средства анализа могут включать как статический, так и динамический анализатор степени связности и интенсивности обменов между отдельными участниками моделирования.

Многоуровневая среда выполнения

Одним из возможных вариантов внедрения смешанной схемы продвижения времени может стать построение среды выполнения из нескольких уровней. Для групп логических процессов с одинаковым механизмом продвижения времени могут использоваться свои вспомогательные среды выполнения, играющие роль федерата в основной среде. Таким образом, можно получить иерархию из сред выполнения. Понятно, что каждый федерат или группа федератов может быть подключена на любом уровне такой организации. При этом в рамках каждой используемой среды выполнения может использоваться свой алгоритм синхронизации, оптимальный для группы подключённых к нему логических процессов.

Предложенный подход позволяет дополнительно провести индивидуальную настройку каждой используемой среды выполнения. Например, в среде выполнения может быть выключена дополнительная функциональность. Тем самым, будет достигнуто упрощение среды выполнения и, как следствие, увеличение производительности системы моделирования в целом.

Ручное построение имитационной модели, состоящей из нескольких уровней логических процессов, не представляется возможным. Поэтому для использования иерархической структуры среды выполнения требует разработки и реализации средств автоматизированной кластеризации логических процессов и определения оптимальных алгоритмов синхронизации для каждой группы. Подобное распределение логических процессов требует анализа имитационной модели до начала её выполнения.

1.5.4 Выводы

На данный момент реализация CERTI RTI поддерживает только консервативную схему продвижения модельного времени. Однако задача моделирования распределённых вычислительных систем в реальном времени может быть решена более эффективно с использованием смешанной схемы продвижения времени, объединяющей преимущества консервативной и оптимистической схем.

Внедрение смешанной схемы продвижения модельного времени потребует разработки дополнительных средств анализа имитационной модели. Перед данными средствами будет поставлена задача эффективного распределения участников моделирования по группам и определение для них оптимальных алгоритмов временной синхронизации.

Стандарт HLA IEEE-1516 2000 не является преградой для внедрения смешанной схемы. Таким образом, интеграция новых алгоритмов продвижения модельного времени в реализацию CERTI RTI не потребует его модификации.

2 Разработка средств поддержки единого формата описания PBC PB, средств поддержки языка описания моделей PBC PB

В данном разделе описываются средства поддержки единого формата описания PBC PB. В разделе 2.1 описаны два уровня представления единого описания PBC PB. приведены результаты экспериментов средств работы с трассами. Раздел 2.2 содержит результаты обзора специализированных языков моделирования. В разделе 2.3 описано применение диаграмм состояний UML для описания моделей PBC PB. В разделе 2.4 приводятся особенности создания модели на HLA.

2.1 Два уровня единого формата описания PBC PB

На первом этапе [1] данной работы в качестве единого формата описания PBC PB и их моделей предложено использовать программные компоненты, разработанные с применением стандарта HLA [8]. В ходе экспериментов на втором этапе НИР выяснилось, что интерфейс HLA представляет разработчику модели довольно низкоуровневый набор примитивов. Такой набор примитивов удобен для сопряжения имитационных моделей, предоставляемых различными разработчиками. Однако разработка новых имитационных моделей с применением лишь примитивов стандарта сложна и чревата ошибками.

Стандарт HLA IEEE-1516 2000 описывает в своих спецификациях набор сервисов среды выполнения RTI, необходимый для обеспечения синхронизации и взаимодействия отдельных участников моделирования между собой. Обращения федератов к сервисам RTI называется их прямым вызовом. Существуют так же и обратные вызовы, уведомляющие федератов об изменении глобального состояния имитационной модели. Обратные вызовы производятся со стороны среды выполнения RTI. На программном уровне реализация механизма обратных вызовов приводит к необходимости наследования предопределённого базового класса и переопределении его методов всеми участвующими в моделировании федератами. При этом интерфейсы стандарта HLA описывают процесс моделирования на высоком уровне абстракции, используя для этого, например, собственные и инородные для языка C++ типы данных. Таким образом, при создании каждого федерата разработчики вынуждены описывать большое количество громоздких и малопонятных языковых конструкций, связанных в частности с постоянным преобразованием формата данных.

Кроме того, существующий набор сервисов RTI зачастую предоставляет разработчикам слишком низкоуровневый набор интерфейсов, вынуждающий их строить собственные классы-обёртки для более удобного решения конкретной имитационной задачи. Например, стандарт HLA допускает возможность динамически изменяющегося состава участников моделирования, при котором новые федераты могут подключаться и прекращать работу непосредственно в процессе моделирования. В то же время широкий класс практических имитационных задач обладает статическим набором участников, известным до начала выполнения модели. Однако стандарт HLA не содержит встроенных средств начальной синхронизации всей федерации, необходимой при решении подобных задач. Поэтому разработчики модели вынуждены реализовывать собственные механизмы начальной синхронизации, используя сервисы RTI более низкого уровня. Таким образом, создание имитационной модели с использованием лишь набора сервисов стандарта HLA часто неудобно и может приводить к нерациональному расходу усилий разработчиков.

По этой причине необходим дополнительный способ описания имитационных моделей (рис. 11). В практике моделирования встречаются два распространённых подхода :

- описание моделей на одном из языков программирования общего назначения с использованием функций, предоставляемых дополнительной библиотекой [26];
- описание моделей при помощи специализированного языка моделирования [26].

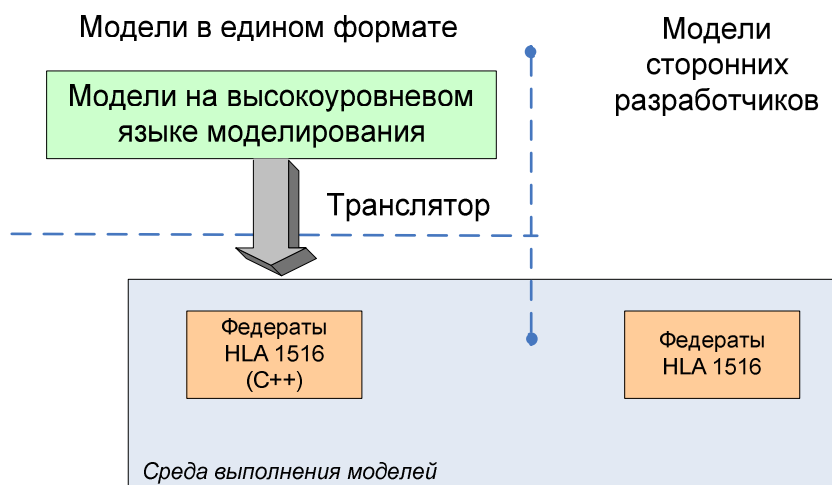


Рисунок 11. Высокоуровневое описание моделей и единый формат HLA.

При использовании первого подхода разработчику предоставляется базовый набор примитивов моделирования в виде структур данных и функций языка программирования общего назначения, на котором разрабатывается модель. В принципе, интерфейс HLA может

сам по себе выступать в качестве такой библиотеки. Однако, как было отмечено выше, этот интерфейс, сам по себе, неудобен для разработчика моделей.

Второй подход подразумевает использование специального языка моделирования [27], а иногда – расширения языка программирования общего назначения дополнительными конструкциями моделирования [28]. Специализированный язык моделирования может использоваться специалистами в предметной области, не имеющими значительного опыта в разработке собственно систем поддержки моделирования.

В рамках второго этапа было решено выбрать один из распространённых языков моделирования PBC PB в качестве высокоуровневого единого формата для описания моделей. Учитывая цели НИР такой язык должен удовлетворять следующим требованиям:

- Поддержка таких основных примитивов моделирования как взаимодействие моделей, явное или неявное продвижение модельного времени.
- Поддержка планирования эксперимента. Например, задание параметров модели и связей между моделями.
- Наличие средств интеграции с аппаратными устройствами с целью поддержки полунатурного моделирования.
- Наличие семантики языка моделирования, позволяющей применять формальные методы к описаниям моделей.

В следующем разделе приводится обзор специализированных языков описания моделей.

2.2 Специализированные языки моделирования

Как правило, при создании средств описания имитационных моделей разработчики используют один из следующих подходов [26]:

- Создание языка с функциями поддержки моделирования.
- Написание специализированных модулей (библиотек классов), решающих задачи поддержки моделирования, для языка общего назначения.
- Создание графического интерфейса для описания моделей (например, по средствам UML)

В первом случае создается (как правило, на основе синтаксиса существующего языка) новый язык, в котором разработчиками предусматриваются специальные конструкции, предназначенные для решения задач имитационного моделирования. Примерами таких

конструкций могут быть конструкции управления модельным временем, унифицированные механизмы взаимодействия компонентов модели, функции автоматической трассировки эксперимента и другие.

При использовании второго подхода описанные выше функции реализуются в виде модуля языка общего назначения, например библиотеки классов и шаблонов C++ или Java.

Применение уже существующего языка позволяет упростить разработку моделей за счет того, что не требуется изучение разработчиками нового языка и возможно использование существующих программных средств (таких как среда разработки, транслятор, отладчик) и существующих библиотек программных компонентов.

В то же время в языках общего назначения отсутствуют или ограничены возможности контроля за соблюдением разработчиком схемы моделирования, что усложняет возможности формального логического анализа. Написание таких анализаторов для языков общего назначения сопряжено с большими сложностями, учитывая богатые выразительные возможности таких языков. К тому же реализация средств описания моделей на языке общего назначения может быть не совсем удобной с точки зрения программирования или может оказаться непривычной для специалиста в предметной области, в интересах которой осуществляется моделирование.

В противоположность этому, специализированные языки моделирования позволяют жестко задать схему моделирования (механизмы управления модельным временем, способы взаимодействия компонентов и пр.), благодаря чему снимается вероятность как умышленного, так и неумышленного нарушения ее программистом, становится возможен формальный анализ модели в рамках заданной схемы моделирования и т.д. Специализированный язык моделирования может быть адаптирован для конкретной предметной области, в нем могут быть учтены практически любые пожелания специалистов, которым придется работать с этим языком.

Однако, анализ открытой литературы в данной области показывает, что текстовые специализированные языки описания моделей используются только для внутреннего представления. На практике же разработчики специализированных языков, предоставляют графический интерфейс описания моделей (например, в виде диаграмм классов).

В разделе 2.2.3 рассмотрены некоторые существующие средства имитационного моделирования.

2.3 Применение диаграмм состояний UML для описания моделей РВС РВ

В данном разделе приводится краткое описание универсального языка моделирования UML и особенностей диаграмм состояний и возможные направления их использования для описания поведения РВС РВ. Так как язык UML предназначен для описания структуры и поведения разнообразных систем, в том числе программных, применение языка для описания РВС РВ требует дополнительных договорённостей. В частности, требуется определить способ описания временных аспектов поведения таких систем.

2.3.1 Язык UML

Универсальный язык моделирования UML применяется для проектирования и моделирования различных аппаратных и программных систем, в том числе, и РВС РВ [29]. В [1, гл. 3] рассматривался вариант применения UML для описания поведения РВС РВ и имитационных моделей.

При помощи языка UML описываются различные аспекты систем. Описывается статическая структура системы, а также динамические аспекты их поведения. Среди основных диаграмм можно перечислить следующие: диаграммы прецедентов, диаграммы классов, диаграммы компонентов, диаграммы состояний, диаграмм последовательностей и деятельности, диаграммы развёртывания. Все перечисленные диаграммы могут использоваться для описания РВС РВ. В отличие от многих программных систем в системах реального времени на ранних этапах проектирования выделяются режимы работы компонентов системы, способы описания интерфейсов компонентов, а также ограничения на время пребывания в определённых режимах. По этой причине мы считаем, что диаграммы состояний являются одним из ключевых способов описания РВС РВ. Вывод об удобстве описания поведения таких систем, например, авиационных, был сделан десятки лет назад Д. Харелом [30]. Также похожий способ описания (с несколько отличной семантикой) можно указать в популярном коммерческом решении Simulink Stateflow [31].

Язык UML прост для понимания и использования. Несмотря на свои достоинства, нет чёткого стандарта языка, поэтому использование для описания моделей описательной мощности языка без ограничений не представляется возможным. Канонические диаграммы UML - это сложившаяся практика группирования сущностей и отношений.

Заметим, что помимо сущностей и отношений на диаграмме присутствуют другие элементы модели, которые можно назвать конструкциями языка. Это тексты, которые могут

быть написаны внутри фигур сущностей или рядом с линиями отношений, рамки диаграмм и их фрагментов, значки, присоединяемые к линиям или помещаемые внутрь фигур. Эти элементы не только помогают представить модель в более наглядной форме, но подчас несут значительную смысловую нагрузку.

Стандарт UML специфицирует большое количество диаграмм [32]:

- Диаграмма использования
- Диаграмма классов
- Диаграмма состояний
- Диаграмма деятельности
- Диаграмма последовательности
- Диаграмма коммуникации
- Диаграмма компонентов
- Диаграмма размещения
- Диаграмма объектов
- Диаграмма внутренней структуры
- Обзорная диаграмма взаимодействия
- Диаграмма синхронизации
- Диаграмма пакетов

Из всего многообразия способов представления системы в виде диаграмм UML в первую очередь интересны диаграммы, описывающие поведение системы. Для описания поведения системы используются диаграммы последовательностей и диаграммы состояний [33]. В РВС РВ понятие состояния вычислительного компонента зачастую вводится явно. В таком случае состояние соответствует активной задаче или наблюдаемому режиму аппаратного устройства. Поэтому для описания моделей будет достаточно использовать только диаграммы состояний (Statechart Diagram).

2.3.2 Основные примитивы диаграмм состояний UML

Диаграммы состояний [34, гл. 7] представляют множество *состояний* и *переходов* между ними (автомат). Состояния соответствуют пребыванию объекта (например, компонента РВС РВ) в определённом режиме. Из состояния может быть осуществлён переход, вызванный *событием* в системе. Примерами события служат приём сигнала, вызов функции или приём сообщения. Другим видом события, важным для описания РВС РВ, служит событие срабатывания таймера. Срабатывание перехода приводит к тому, что автомат переходит в новое состояние и, возможно, выполняет некоторое *действие*.

События. События подразделяются на четыре вида: событие вызова, событие изменения условия, событие приёма сигнала, временное событие. Событие вызова задаётся в виде $op(a:T)$, оно соответствует синхронному вызову метода op с параметрами a типов T . Событие изменения условия описывается выражением $when(exp)$ и срабатывает в том случае, когда выполняется логическое условие exp . Событие приёма сигнала $sname(a:T)$ соответствует приёму сообщения $sname$ с телом a типа T от другого объекта. Как правило, таким способом описывается приём асинхронного сообщения. Временное событие имеет вид $after(time)$ и срабатывает по истечении времени $time$. Стоит заметить, что в общем случае логические выражения и временные события могут описываться как формально (на языке программирования), так и неформально (на естественном языке).

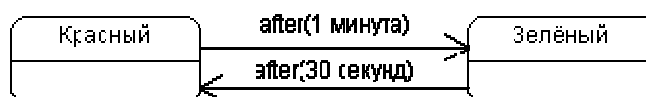


Рисунок 13. Пример состояний и переходов между ними.

Состояния. Состояния подразделяются на несколько видов. Основными строительными блоками диаграмм можно назвать *простые* состояния. На рисунке 12 приведены два таких состояния, с названиями «Красный» и «Зелёный». Такие состояния могут описывать, например, два режима работы светофора.

Большие системы могут содержать десятки и сотни состояний. Поэтому важным средством организации диаграмм состояния являются *композиционные* состояния. Композиционные состояния содержат другие состояния, в том числе, и композиционные. На рисунке 13 изображено композиционное состояние, соответствующее обнаружению неисправности.

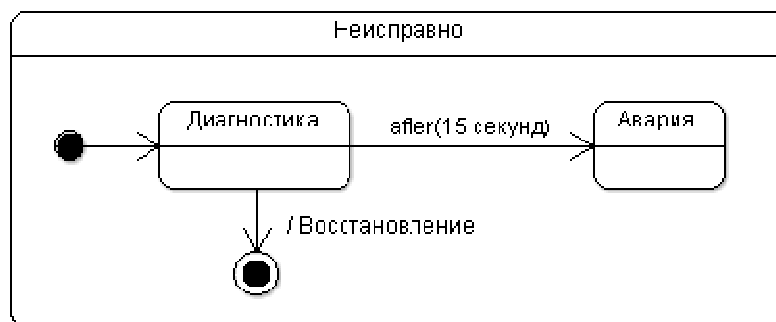


Рисунок 13. Композитное состояние, содержащее начальное, заключительное и два простых состояния.

В диаграммах так же выделяются служебные состояния, называемые *псевдосостояниями*. В таких состояниях система не может пребывать, они нужны лишь для объединения простых и композитных состояний. К псевдосостояниям относятся следующие:

- *Начальное* состояние – состояние i , с которого начинается обработка деятельности объекта, описываемой композитным состоянием, непосредственно содержащим i . На **рисунке 13** иллюстрируется начальное состояние, которое изображается в нотации UML при помощи круга, заполненного чёрным цветом.
- *Заключительное* состояние. Переход в такое состояние f соответствует окончанию деятельности, описываемой композитным состоянием, непосредственно содержащим f . На **рисунке 13** иллюстрируется заключительное состояние, которое изображается в нотации UML при помощи круга, заполненного чёрным цветом, внутри окружности.
- *Терминальное* состояние соответствует окончанию жизни объекта, поведение которого задаётся при помощи диаграммы.
- *Соединение* используется для объединения потока управления, заданного при помощи нескольких псевдосостояний. Поток, объединённый при помощи состояния вида «соединение», воспринимается как один переход. Состояние соединения изображено на **рисунке 14**.
- *Выбор* соответствует разветвлению потока управления в соответствии с условиями переходов, исходящих из псевдосостояния. Состояние выбора изображено на **рисунке 14**.

- *История* – специальное состояние, переход на которое приводит к переключению на такое состояние, из которого ранее был осуществлён выход из композитного состояния. Состояние истории изображено на **рисунке 15**. В зависимости от того, из какого состояния (*A* или *B*) был совершён выход из композитного состояния, при входе в это состояние активируется соответствующее состояние.
- *Поддиаграмма* – специальное состояние, ссылающееся на другую диаграмму. Вместо такого состояния непосредственно подставляется диаграмма, на которую состояние ссылается. На **рисунке 16** иллюстрируется такое состояние *B*.

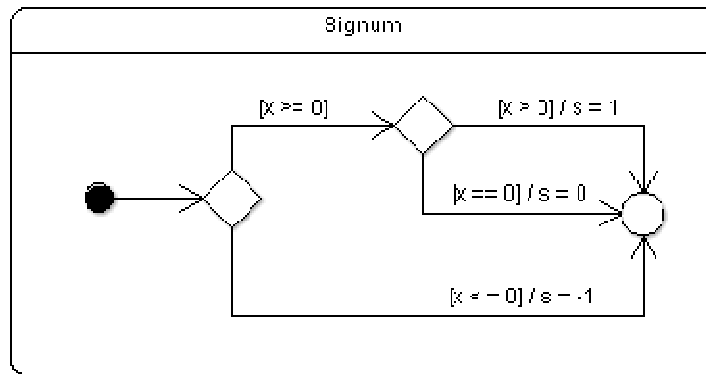


Рисунок 14. Иллюстрация двух состояний выбора и одного объединения.

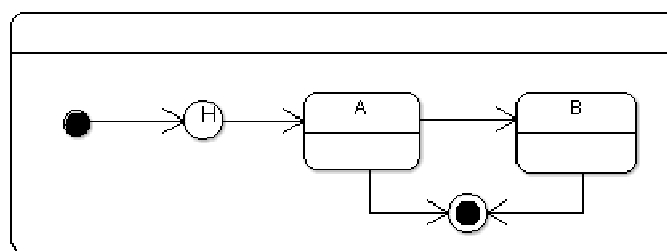


Рисунок 15. Иллюстрация состояния истории.

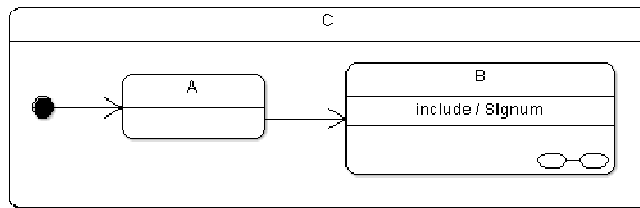


Рисунок 16. Иллюстрация состояния поддиаграммы.

Все виды состояний, приведённые ранее, описывают последовательный поток управления. Диаграммы состояний могут также описывать параллельные компоненты. Достигается это при помощи *параллельных регионов* композитного состояния. Объект пребывает одновременно в одном из состояний каждого параллельного региона. На **рисунке 17** изображены два параллельных региона. Вложенные композитные состояния так же могут содержать параллельные регионы.

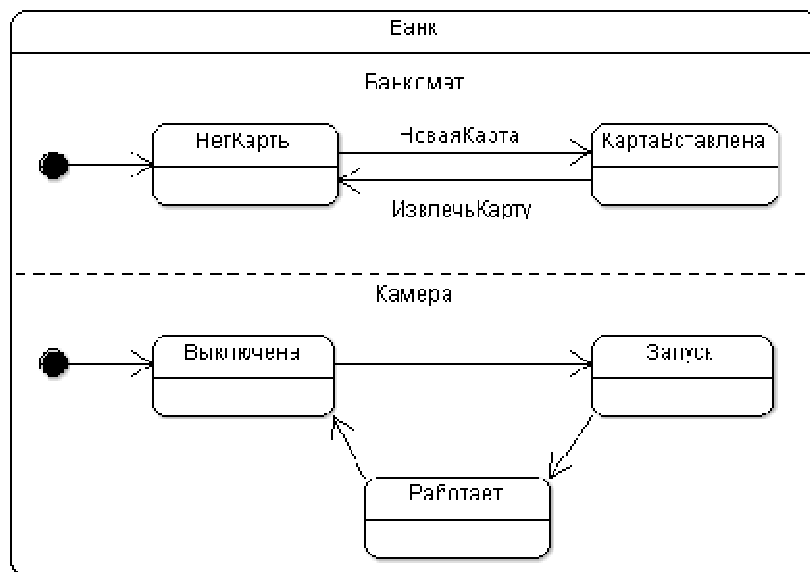


Рисунок 17. Пример параллельных регионов.

Переходы. Объект может перейти из состояния *S* в состояние *T* посредством перехода, соединяющего состояния *S* и *T*. Переход может быть размечен необязательными компонентами: триггером, предусловием и эффектом.

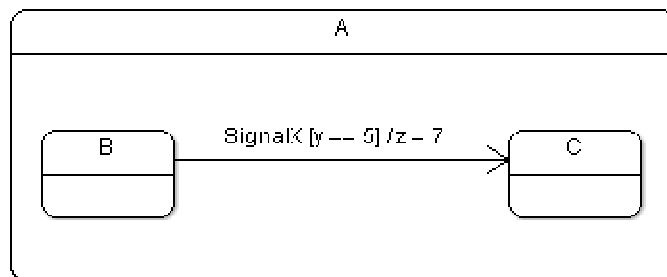


Рисунок 18. Иллюстрация триггера, предусловия и эффекта перехода.

Триггер задаёт событие, которое приводит к выполнению перехода. Считается, что к каждому состоянию прикреплена очередь событий. Все события обрабатываются по очереди. Одновременно произошедшие события так же обрабатываются последовательно, в неопределённом порядке. При обработке объектом события E в состоянии S *срабатывает* переход, помеченный триггером E . На **рисунке 18** изображён переход из состояния B в состояние C , активирующийся при появлении события – приёма сигнала $SignalX$.

Переход может сработать не во всех случаях. Ограничения на срабатывание перехода накладываются *предусловием* – булевой формулой над атрибутами объекта и параметрами события. Если обрабатывается событие E , а предусловие перехода с триггером E оценивается как ложь, то переход срабатывает. Если ни один из переходов в активном состоянии не срабатывает, событие E игнорируется. На **рисунке 18** изображён переход из состояния B в состояние C , срабатывающий при появлении события $SignalX$ и выполнении предусловия $y == 5$.

При срабатывании перехода применяется *эффект* перехода. Эффект может выражаться в виде *действия* или *деятельности*. Действие заключается в выполнении таких простых вычислений, как присваивание, выполнение арифметических операций, посылка сообщения, вызов функции, создание и уничтожение объекта. Более сложные вычисления описываются в виде деятельности – последовательности действий и/или деятельностей. На **рисунке 18** изображён переход из состояния B в состояние C , при выполнении которого выполняется действие $z = 7$.

Важным свойством эффекта является его выполнение от начала до конца, без влияния на действия или деятельность других объектов. Таким образом, эффект в UML похож на

транзакцию базы данных. Однако, эффект должен выполняться относительно быстро, условно, за период времени, близкий к нулю.

Переход необязательно помечается триггером. Такой переход срабатывает, как только прекращается вычисление, связанное с исходным состоянием перехода. Считается, что при завершении вычисления в исходном состоянии должен быть выполнен переход в новое состояние. Такие переходы обладают высоким приоритетом по отношению к переходам, размеченным триггерами.

Вычисление в состоянии. К каждому состоянию так же может быть прикреплено несколько видов деятельности: деятельность при входе, деятельность при выходе и внутренние переходы. При активации состояния S выполняется связанная с S деятельность при входе. При срабатывании перехода из состояния S выполняется связанная с S деятельность при выходе. С состоянием могут быть связаны внутренние переходы, которые срабатывают при условиях, аналогичных условиям обычных, внешних, переходов. Однако внутренние переходы не приводят к изменению состояния, поэтому не выполняется деятельность при входе и выходе.

2.4 Особенности создания модели на HLA

2.4.1 HLA – особенности интерфейса

Ключевыми понятиями HLA являются Federation и Runtime Infrastructure (RTI). Федерация – это объединение компонентов имитационного моделирования, называемых федератами.

Федерат может быть имитационной моделью, программой, драйвером для управления аппаратурой, интерфейсов с реальным объектом, например с летательным аппаратом. Взаимодействие федератов осуществляется посредством обмена данными. Обмен данными и исполнением федератов в едином модельном времени осуществляется с помощью программной оболочки (инфраструктуры) RTI. RTI – по сути, распределенная операционная система для федерации. Взаимодействие федератов с RTI происходит посредством вызова федератами сервисов.

Стандарт HLA не предъявляет требований к реализации федератов и RTI, а только определит правила оформления федератов и федерации и интерфейс между федератами и RTI. Стандарт состоит из трех частей:

- Правила HLA, определяющие базисные принципы.

- Шаблоны федераты и федерации, описывающие формат обмена данными между федератами.
- Спецификация интерфейса федератов с сервисами RTI.

Сервисы RTI делятся на следующие группы:

- Управление федерацией. Эти сервисы используются для создания и функционирования федерации в целом.
- Управление декларациями. Эти сервисы используются для объявления экспортируемых и импортируемых данных.
- Управление объектами. Эти сервисы используются для работы с объектами и их атрибутами.
- Управление правом доступа. Эти сервисы используются для передачи прав между федератами на модификацию значений атрибутов объектов.
- Управление распределением данных. Эти сервисы позволяют уменьшать объем данных, пересылаемых между федератами, за счет более эффективного распределения данных.
- Управление временем. Эти сервисы синхронизируют продвижение локального модельного времени федератов при выполнении федерации.

Инфраструктура RTI реализует передачу сообщения между федератами и синхронизирует из выполнение в едином модельном времени. Взаимодействие федерата с RTI осуществляется с помощью вызова интерфейсных функции (сервисов). Состав и синтаксис этих функции определены в [8].

2.4.2 Сложность написание HLA-совместимых моделей

Для построения HLA-совместимых моделей, требуется трудоёмкая работа по описанию всех интерфейсов для связи с RTI, что влечёт за собой большие временные издержки разработчика, и увеличивает вероятность ошибки. В связи с чем, актуальна задача использование готовой, или разработки собственной среды разработки HLA-совместимых моделей, где построение моделей осуществлялось при помощи текстового или графического языка описания моделей.

2.4.3 Языки описания моделей

В данном разделе рассмотрены некоторые существующие средства описания имитационных моделей.

2.4.3.1 Modelica

Modelica – свободно распространяемый объектно-ориентированный язык для моделирования сложных физических систем [27].

Язык имеет хорошую техническую поддержку разработчиков, для него существует большое количество библиотек компонентов, как уже существующих, так и новых. **Modelica** обеспечивает создание различных моделей: механических, электрических, гидравлических, химических, и др.

В основе языка **Modelica** лежит концепция соединяемых блоков. При соединении в соответствии с требуемой схемой автоматически генерируются соответствующие уравнения. Это делает язык простым для понимания и использования специалистами нематематического профиля. Пример графического описания модели на языке Modelica приведён на рисунке 19.

Modelica не ограничивает количество компонентов моделируемой системы базовыми компонентами, поставляемыми разработчиками. Пользователь может создавать свои собственные компоненты, используя при этом внутренний язык описания блоков. Язык **Modelica** поддерживает интеграцию с пакетами моделирования как MATLAB и SimuLink, обеспечивает поддержку стандартов ACSL, M-file, Simnon. Также поддерживается возможность использования функций и процедур написанных на языке C. Пример текстового описания модели на языке Modelica приведён на рисунке 20.

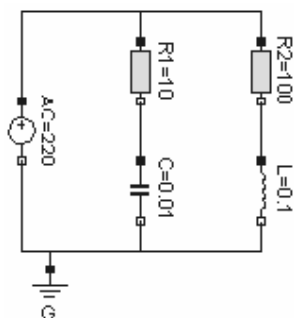


Рисунок 19. Пример графического описания модели на языке Modelica.

```

model circuit
Resistor R1(R=10);
Capacitor C(C=0.01);
Resistor R2(R=100);
Inductor L(L=0.1);
VsourceAC AC;
Ground G;
equation
connect (AC.p, R1.p); // Capacitor circuit
connect (R1.n, C.p);
connect (C.n, AC.n);
connect (R1.p, R2.p); // Inductor circuit
connect (R2.n, L.p);
connect (L.n, C.n);
connect (AC.n, G.p); // Ground
end circuit;

```

Рисунок 20. Пример текстового описания модели на языке Modelica.

Таким образом, Modelica – это язык динамического моделирования, предназначенный для моделирования физических моделей. Она включает в себя наборы библиотек для работы с математическими, электрическими и механическими системами, а также библиотеку для моделирования тепловых процессов. Modelica использует объектно-ориентированный подход программирования, что позволяет удобнее создавать модели и быстрее осуществлять их расчет. В Modelica поддерживается графическое отображение процессов, 3D анимация, симуляция в реальном времени, возможность использования моделей, созданных в Modelica, в других программах для моделирования, например, таких как MatLab. Пакет Dynamic Modeling Laboratory, поддерживающий язык моделирования Modelica, является комплексным инструментом для моделирования и исследования сложных систем в таких областях, как механика, автоматика, аэрокосмические исследования и др. Возможность объединения в одной модели компонентов различной физической природы позволяет строить модели сложных систем, более соответствующих реальности, и получать более точные результаты. Кроме собственного языка, Modelica поддерживает интеграцию с такими программными средами, как Fortran, C, Simulink, и некоторыми др. Возможность взаимодействия разработанных моделей с системой MATLAB/Simulink позволяет объединить сильные стороны структурного и физического моделирования. Modelica

представляет собой среду визуального моделирования, включающую универсальный объектно-ориентированный язык Modelica для моделирования сложных физических систем.

2.4.3.2 SLX

Язык SLX основывается на широких возможностях языка имитационного моделирования GPSS/H. Язык обеспечивает большие возможности для моделирования современных систем, описываемых языками, похожими на C. Язык представляет собой многоуровневую (послойную) структуру с ядром SLX в основании пирамиды, далее в середине — традиционный язык моделирования GPSS/H.

Основные особенности SLX обеспечивают возможности создания разнообразной логики, петель управления, расширения, диагностики. Большинство языков, использующих макросы, требуют наличия специальных подпрограмм или команд для их определения. Обычно эти подпрограммы отличаются от тех, которые используются в основной программе. Например, команды `if`, `else`, `endif` в языке C плохо используются для условного описания макросов, а их синтаксис отличается от принятого в языке C. В отличие от C, язык SLX не имеет специальных команд, используемых для определения утверждений, а сами новые команды `time delays`, `fork`, `wait until` позволяют решать многие задачи моделирования. Причем информация считывается из файла модели и запоминается в структурах данных, определяемых пользователем. В дополнение к новым директивам SLX поддерживает традиционные макросы и модули расширения, применимые как при компиляции, так и в процессе исполнения.

Интерфейс для SLX может быть использован для соединения моделей SLX с инфраструктурой моделирования (RTI), позволяющей осуществлять расширенное моделирование на уровнях HLA. Интеграция заключается в соединении функций C++, которые располагаются между SLX и RTI. На [рисунках 21 и 22](#) приведены примеры описания модели на языке SLX.

```

class car
{
    int          counter;

    actions
    {
        ++n_car;
        counter = n_car;
        print (counter,time) "Car ___.__ approaches crossroad      ___.__\n";
        advance 20;

        print (counter,time) "Car ___.__ has arrived at                ___.__\n";
        wait until (GhosedTrafficLight->CarLight && crossroad_clear);

        print (counter,time) "Car ___.__ passes crossroad at          ___.__\n";

        crossroad_clear = FALSE;
        advance 1.8;                // Car needs 1.8
        crossroad_clear = TRUE;

        advance 30;
        terminate;
        //actions
    }
};

```

Рисунок 21. Пример описания модели на языке SLX.

```

forever
{
    NextEventTime= next_imminent_time();           //determine next event time
    print ( NextEventTime ,time) "NextEventRequest:  ___.__| at time ___.__\n";
    grantTime = RTI_NextEventRequest( NextEventTime); //Request advancement
    advance grantTime-time;                        //Advance to grantTime
    RTI_ReflectControlVariableChanges();          //important for logical correctness: only AFTER
    // advancing to the grantTime may SLX be told to re-evaluate control variables

    if (SLX_StateObjectPtr->ObjectsDiscovered > 0)
    {
        print (SLX_StateObjectPtr->ObjectsDiscovered) "Count of Objects discovered is ___.__\n";
        print (SLX_StateObjectPtr->DiscoveredObjectClass) "Name is*****.\n";
        if (SLX_StateObjectPtr->DiscoveredObjectClass == "TrafficLight")
        {
            GhosedObjectID = RTI_RegisterGhosedObject("TrafficLight",GhosedTrafficLight);
            if (GhosedObjectID != -1)
            {
                SLX_StateObjectPtr->ObjectsDiscovered--;
                print(GhosedTrafficLight->CarLight) "Carlight after ghosting is ___.__\n";
            }
            else
                print "Error while ghosting the object.\n";
        }
        else
            print "Wrong class.\n";
    }
    if (SLX_StateObjectPtr->AttributeUpdatesReceived > 0 )
    {
        print(GhosedTrafficLight->CarLight) "Carlight after update is ___.__\n";
        SLX_StateObjectPtr->AttributeUpdatesReceived--;
    }
    yield;
}
}

```

Рисунок 22. Пример главного цикла модели на языке SLX.

2.4.3.3 Simulink

Simulink – это интерактивная система для анализа линейных и нелинейных динамических систем. Это графическая система позволяет вам моделировать систему простым перемещением блоков в рабочую область и последующей установкой их

параметров. Simulink может работать с линейными, нелинейными, непрерывными, дискретными, многомерными системами [35].

Наиболее интересным с точки зрения моделирования встроенных систем является пакет Stateflow Simulink [32]. **Stateflow** является интерактивным инструментом разработки в области моделирования сложных, управляемых событиями систем. Он основан на теории конечных автоматов. Stateflow предлагает решение для проектирования встроенных систем с контролирующей логикой.

Для описания логики модели Stateflow использует визуальный формализм - Statechart (диаграммы состояний и переходов). Основные неграфические компоненты таких диаграмм - это событие и действие, основные графические компоненты - состояние и переход.

Событие - нечто, происходящее вне рассматриваемой системы, возможно требуя некоторых ответных действий. События могут быть вызваны поступлением некоторых данных или некоторых задающих сигналов со стороны человека или некоторой другой части системы. События считаются мгновенными (для выбранного уровня абстрагирования).

Действия - это реакции моделируемой системы на события. Подобно событиям, действия принято считать мгновенными.

Состояние - условия, в которых моделируемая система пребывает некоторое время, в течение которого она ведет себя одинаковым образом. В диаграмме переходов состояния представлены прямоугольными полями со скруглёнными углами.

Переход - изменение состояния, обычно вызываемое некоторым значительным событием. Как правило, состояние соответствует промежутку времени между двумя такими событиями. Переходы показываются в диаграммах переходов линиями со стрелками, указывающими направление перехода.

Каждому переходу могут быть сопоставлены условия, при выполнении которых переход осуществляется. С каждым переходом и каждым состоянием могут быть соотнесены некоторые действия. Действия могут дополнительно обозначаться как действия, выполняемые однократно при входе в состояние; действия, выполняемые многократно внутри некоторого состояния; действия, выполняемые однократно при выходе из состояния.

Stateflow позволяет использовать диаграммы потоков (flow diagram) и диаграммы состояний и переходов (state transition) в одной диаграмме Stateflow. Система обозначений диаграммы потоков - логика, представленная без использования состояний. В некоторых случаях диаграммы потоков ближе логике системы, что позволяет избежать использования ненужных состояний. Система обозначений диаграммы потоков - эффективный способ

представить общую структуру программного кода как конструкцию в виде условных операторов и циклов.

Stateflow также обеспечивает ясное, краткое описание поведения комплексных систем, используя теорию конечных автоматов, диаграммы потоков и диаграммы переходов состояний. Stateflow делает описание системы (спецификацию) и проект ближе друг другу. Создавать проекты, рассматривая различные сценарии и выполняя итерации, намного проще, если при моделировании поведения системы используется Stateflow. Пример описания модели на языке StateFlow Simulink приведён на [рисунке 23](#).

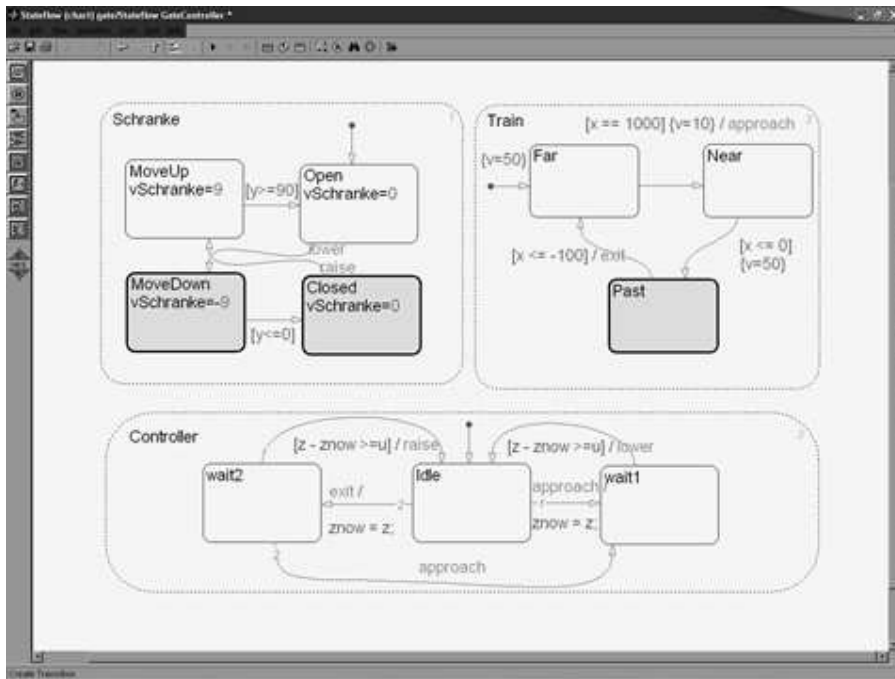


Рисунок 23. Пример описания модели на языке StateFlow Simulink.

2.4.4 Выводы

Обзор специализированных языков моделирования показал отсутствие необходимого языка для удобного описания HLA-совместимых моделей. Поэтому был сделан выбор в пользу создания собственного инструмента для описания моделей на HLA, при этом за основу берутся диаграммы состояний UML. Дополнительная логика, связанная с низкоуровневой реализацией компонентов или со специальными алгоритмами моделирования, может быть написана на языке C++. На основании такого графического описания, возможно, дополненного кодом на C++, порождается код HLA-совместимой модели на C++.

3 Разработка методов и инструментальных средств поддержки анализа и разработки РВС РВ первой очереди

В данном разделе описываются методы и инструментальные средства поддержки анализа и разработки РВС РВ первой очереди. В разделе 3.1 приводится описание средств работы с трассами. В разделе 3.2 описываются средства трансляции UML во временные автоматы. В разделе 3.3 средства трансляции UML в модель на HLA.

3.1 Средства работы с трассами

В ходе первого этапа НИР [1] были обозначены следующие форматы трасс: TAU; группа ALOG, CLOG, SLOG2; EPILOG; STF; формат проекта V-Ray; TRC (формат трасс Стенд ПНМ); Paje; OTF; CCG. Среди рассмотренных форматов были выделены следующие форматы для дальнейшего практического исследования: TRC (формат трасс проекта «Диана»), TAU, OTF и OTF с сжатием (OTFz). Соответствие данных форматов и средств их анализа приведено в таблице 1.

Таблица 1. Форматы трасс, выбранные на первом этапе НИР.

№	Формат	Средство анализа и визуализации	Кодировка
1	TRC	Vis	Бинарный
2	TAU	Vite, Jumpshot, Paje	Бинарный
3	OTF, OTFz	Vite, Vampir	Бинарный + спец. ACSII

Каждый формат характеризуется файловой структурой и структурой записей, которые допустимы в нём, и от их организации зависит гибкость, эффективность запросов к трассе, возможности работы с трассой. Далее будут описаны формат TRC среды моделирования Диана, формат TAU и OTF, разработанные для трассировки многопроцессорных систем.

В рамках экспериментального исследования был разработан ряд программных средств, которые описываются в данном разделе.

3.1.1 Формат TRC

3.1.1.1 Основные понятия среды моделирования «Диана»

В 1998-2000 годах в лаборатории вычислительных комплексов (ЛВК) была разработана среда моделирования и анализа ДИАНА [36]. Среда ДИАНА представляет собой

интегрированную среду для имитационного моделирования и анализа функционирования вычислительных систем, в том числе вычислительных систем реального времени. В основу среды ДИАНА положена математическая модель «модели с сообщениями», которая подробно описана в [37]. Операции приема и передачи сообщений являются примерами событий системы моделирования, которые влекут за собой изменение состояний инициировавших их компонентов модели.

На основе среды моделирования ДИАНА в лаборатории был разработан Стенд математического моделирования комплекса бортового оборудования (СММ КБО), предназначенный для разработки имитационных моделей элементов КБО с целью полунатурного моделирования КБО и для отработки комплексного взаимодействия устройств КБО по каналам бортовых интерфейсов (КБИ). Полунатурное моделирование – это способ моделирования, при котором часть моделей – имитационные, а часть – натурные образцы. СММ КБО позволяет осуществлять оценку технических решений, разработку предложений и проведение исследований в области структуры КБО и характеристик бортового радиоэлектронного оборудования (БРЭО). Конечной целью использования стенда является комплексная отработка КБО, включая проверку работоспособности натуральных образцов аппаратуры [38]. Программные средства стенда обеспечивают выполнение программных моделей, взаимодействие моделей между собой и с аппаратными компонентами, управление процессом моделирования, а также ведение истории (трасс) событий.

В рамках среды моделирования был разработан формат трасс TRC, средство визуализации и анализа Vis (визуализатор временной диаграммы) и ряд средств для работы с трассами, которые непосредственно использовались в СММ КБО. Для понимания формата TRC и содержимого трасс ниже приводятся основные понятия среды моделирования ДИАНА и СММ КБО.

Стенд представляет собой набор иерархически организованных *компонентов*. *Компонентами являются*

- последовательные частные модели (ПЧМ);
- распределённые частные модели (РЧМ);
- интерфейсы частных моделей;
- каналы бортовых интерфейсов (БИ);
- «искусственные» компоненты, введённые для повышения наглядности отображения.

С подробным описанием приведенных понятий можно ознакомиться в отчёте по первому этапу данной НИР [1] .

Функционирование каждого компонента характеризуется событиями и состояниями. *Событием* называется любое изменение, существенное с точки зрения логики функционирования моделируемой системы, которое может произойти во время работы стенда и которое отражается в трассе результатов эксперимента. Событие – это единица наблюдения, фиксации (средой выполнения моделей) и отображения (визуализатором Vis) хода работы стенда. Событие характеризуется:

- типом;
- временем возникновения;
- местом возникновения;
- набором дополнительных атрибутов, в зависимости от типа события.

Состояние компонента обозначает неизменный режим работы компонента в течение некоторого промежутка времени. Состояние характеризуется:

- типом;
- временем начала;
- продолжительностью.

Для всех типов интерфейсов определены состояния «Включён» и «Выключен». Для РЧМ и искусственных компонентов понятие состояния не определяется.

3.1.1.2 Основные события среды моделирования «Диана»

Для понимания формата TRC и информации, которую он позволяет хранить, необходимо описать события, которые возможны в среде моделирования ДИАНА и которые подвергаются трассировке.

В **таблице 2** приводятся основные типы событий, возникающих в ПЧМ, в интерфейсах различных типов, в моделях каналов БИ в стенде КБО.

Таблица 2. Типы событий.

№	Тип	Семантика события	Где возникает
1	Init	Начало имитационного эксперимента	Глобальное
2	Pause	Приостановка имитационного эксперимента	Глобальное
3	Continue	Продолжение приостановленного эксперимента	Глобальное
4	Flush	Запрос со стороны ЧМ некоторого действия над интерфейсом ЧМ	ЧМ
5	EndFlush	Завершение отработки интерфейсом запроса от частной модели	Интерфейс
6	Arrive	Приход сообщения на интерфейс	Интерфейс, модель КБИ
7	Update	Обновление значения выходного параметра	ЧМ
8	Event	«Пробуждение» ЧМ, ожидающей на операторе RTS_WAIT	ЧМ
9	Stop	Завершение имитационного эксперимента	Глобальное
10	Warning	Предупреждение, выдаваемое средой выполнения ЧМ	Глобальное, интерфейс, ЧМ
11	Error	Уведомление о фатальной ошибке, выдаваемое средой выполнения ЧМ	Глобальное, интерфейс, ЧМ
12	SetState	Установка режима работы ЧМ (режим понимается в смысле ЯОМ)	ЧМ
13	Finish	Завершение работы компонента стенда (например, выход из метода MainLoop ПЧМ).	ЧМ
14	Send	Отправка данных с интерфейса в канал или из канала на интерфейс по инициативе интерфейса или модели канала	Интерфейс, модель КБИ
15	Delivery	Доставка данных с интерфейса в ЧМ	ЧМ
16	Transfer	Отправка данных с интерфейса в модель по инициативе интерфейса	Интерфейс
17	Debug	Соответствуют выводу отладочного сообщения оператором RTS_DEBUG	Глобальное

События, отмеченные в таблице как «глобальные», относятся к функционированию стенда в целом и отображаются на всех линиях жизни (ЛЖ) компонентов.

Событие «Flush» возникает в момент *запроса* любой операции над интерфейсом (например, отправка сообщения, размещение данных в памяти адаптера БИ, отправка команды по КБИ, установка флагов адаптера БИ, и т.д.), требующей передачи данных в буфер интерфейса. Событие «EndFlush» (в интерфейсе) отмечает момент *окончания* загрузки данных в буфер интерфейса. Это событие является парным к событию «Flush» в ЧМ. Событие «Arrive» отмечает момент прихода сообщения на *входной* интерфейс. Событие «Send» отмечает момент отправки *выходным* интерфейсом сообщения в канал. Событие

«Delivery», происходящее в ЧМ, отмечает момент окончания доставки данных с интерфейса с каналом БИ в модель. Событие «Transfer», происходящее в интерфейсе, возникает в момент окончания передачи данных (по инициативе интерфейса) из буфера интерфейса в ЧМ.

Некоторые события могут быть *логически связаны* между собой. Именно, одно событие может быть *причиной* другого события, которое называется *следствием*. Событие-причина и событие-следствие могут происходить в разных компонентах. Возможные связи между событиями показывает **таблица 3**.

Таблица 3. Связи между событиями.

Событие-причина	Событие-следствие	Комментарий
<i>Flush</i>	<i>EndFlush</i>	ЧМ выполняет операцию над интерфейсом, требующую передачи данных в буфер интерфейса. Начало операции отмечается событием <i>Flush</i> в ЧМ, окончание — событием <i>EndFlush</i> в интерфейсе.
<i>EndFlush</i>	<i>Arrive</i>	Обновлён буфер интерфейса (событие <i>EndFlush</i>) и интерфейс начинает передачу данных в канал. Событие <i>Arrive</i> возникает в модели канала БИ, когда данные переданы.
<i>Send</i>	<i>Arrive</i>	Интерфейс начал передачу данных по собственной инициативе, без непосредственного воздействия со стороны ЧМ (событие <i>Send</i>); данные получены моделью канала (событие <i>Arrive</i>)
<i>Arrive</i>	<i>Delivery</i>	Данные приняты из канала интерфейсом во внутренний буфер (событие <i>Arrive</i>); завершена их передача в ЧМ (событие <i>Delivery</i>).
<i>Transfer</i>	<i>Delivery</i>	Интерфейс начал передачу данных из внутреннего буфера в ЧМ по собственной инициативе (событие <i>Transfer</i>); данные получены ЧМ (событие <i>Delivery</i>).

3.1.1.3 Описание формата TRC

Файловая структура формата TRC приведена на **рисунке 24**.

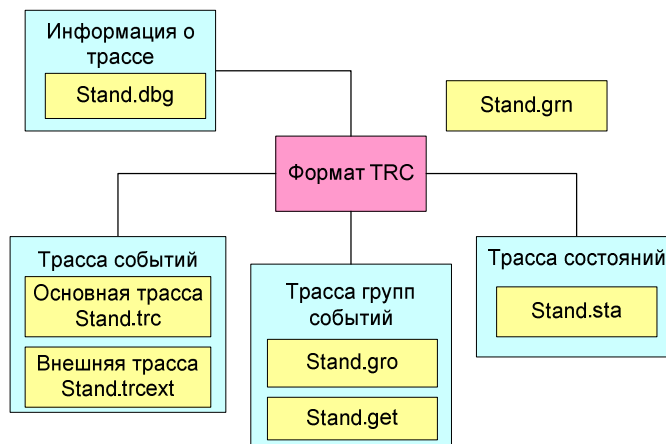


Рисунок 24. Файловая структура формата TRC.

Для записи событий, происходящих в моделируемой системе, в трассе используется файл основной трассы – файл с расширением trc и файл внешней трассы – файл с расширением trcext. Файл основной трассы имеет заголовок (4 байта) и набор последовательно записанных событий по 32 байта. Структура сообщений основной трассы приведена на **рисунке 25**.

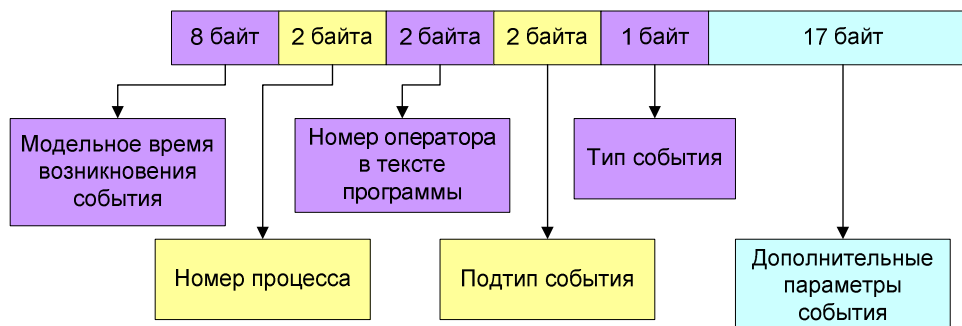


Рисунок 25. Структура событий в основной трассе.

Список дополнительных параметров событий приведён в **таблице 4**.

Файл внешней трассы представляет собой совокупность записей нефиксированного размера. В trcext-файл записывается дополнительная информация о событии, а для соответствующего события в основной трассе указывается ссылка (extLink) на запись внешней трассы. Например, для события типа Update во внешнюю трассу помещается значение изменяемого параметра. Таким образом, файл с расширением trcext был введен для расширения формата трассы TRC.

Таблица 4. Дополнительные параметры событий.

Событие	Параметр	Семантика
<i>Flush</i> <i>EndFlush</i> <i>Send</i> <i>Arrive</i> <i>Delivery</i> <i>Transfer</i> <i>Event</i>	link	Идентификатор группы, которой принадлежит событие. В событии типа Arrive таких параметров два – Link и Link2. В событии типа Event заполняется мусором и не используется.
<i>Flush</i> <i>Send</i> <i>Delivery</i> <i>Transfer</i> <i>Test</i>	param	Не используется во всех событиях, за исключением событий типа Test
<i>Update</i>	param	Номер обновляемого параметра частной модели в некоторой таблице
<i>SetState</i>	state	Тип состояния, в которое перешла частная модель
<i>Pause</i>	duration	Продолжительность паузы
<i>Debug</i>	size	Размер выводимого текстового сообщения
<i>Update</i> <i>Flush</i> <i>Arrive</i> <i>Event</i> <i>Debug</i> <i>Delivery</i> <i>Transfer</i>	extlink	Смещение в файле дополнительной трассы, где находится содержимое пакета, то есть дополнительной информации для события.

Файл формата dbg содержит общую информацию о трассе: количество компонентов, типов состояний и событий, количество параметров, названия событий, состояний и компонентов и т.п.

Состояния моделируемой системы последовательно записываются в sta-файл. Каждая запись имеет размер 24 байта. Файл не имеет заголовка. Структура записи о состоянии в sta-файле приведена на рисунке 26.

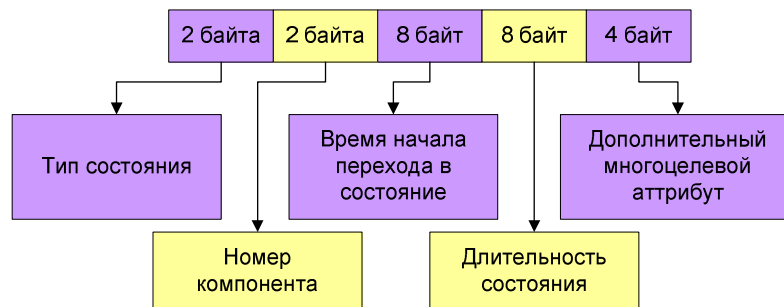


Рисунок 26. Структура состояния в sta-файле.

Таким образом, наиболее значимыми являются файлы `trc`, `trcext`, `sta` и `dbg`, остальные файлы трассы в исследовании рассматриваться не будут.

Нужно отметить, что в формате TRC поддерживается возможность слияния трасс, записанных разными компонентами моделируемой системы. При слиянии события основной трассы и состояния файла в формате `sta` упорядочиваются по значению временных меток, при этом для внешних трасс производится процедура конкатенации (склеивания, сцепления внешних файлов в один) с соответствующим преобразованием ссылок на внешнюю трассу.

Основными недостатками данного формата являются:

- Сложность структуры.
- Наличие внешней трассы, обращение к которой увеличивает скорость чтения и записи трассы.
- Жесткая структура события и сложность расширения этой структуры.

К достоинствам формата TRC можно отнести:

- Наличие достаточно хорошего средства визуализации и анализа трасс `Vis`.
- Поддержка возможности слияния трасс.

3.1.1.4 Описание средства трансляции трассы TRC в формат TXT

Для экспериментального исследования трасс формата TRC, который является бинарным форматом, было разработано средство `trc2txt`. Данное средство позволяет конвертировать трассу в формате TRC в читаемое представление в формате TXT. Схема работы средства представлена на [рисунке 27](#). Для чтения трасс средство использует библиотеки `tracexio` и `tracedb` среды моделирования ДИАНА. В качестве входных параметров средство получает трассу в TRC-формате (4 файла: `stand.trc`, `stand.trcext`, `stand.sta`, `stand.dbg`). В результате работы средства получаем два txt-файла: файл `events` содержит построчную запись событий трассы в читаемом виде, файл `states` содержит построчную запись состояний. Состояния и события упорядочены по временным меткам.

Для запуска средства необходимо выполнить строку

```
.\trc2txt stand.trc
```

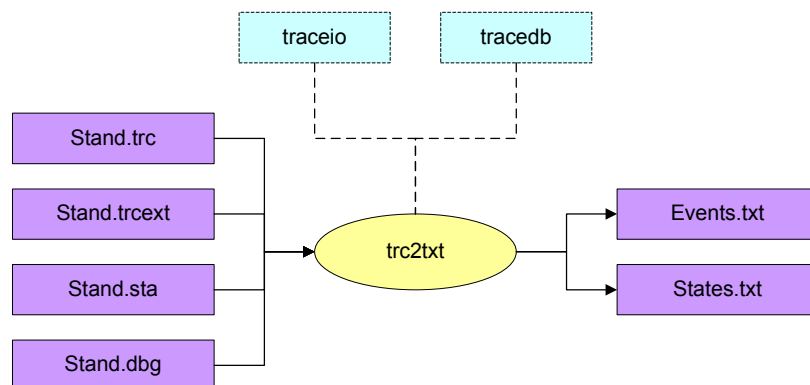


Рисунок 27.Схема работы средства trc2txt

3.1.1.5 Описание средств исследования трасс TRC - TraceAnalyzerTRC

Средство TraceAnalyzerTRC было разработано с целью автоматизации проведения экспериментов с трассами TRC-формата. Схема работы средства представлена на рисунке 28. В качестве входных параметров средство получает несколько трасс в TRC-формате и последовательно с каждой трассой проводит разработанные в методике эксперименты. Результаты каждого эксперимента в табличной форме фиксируются в файлах csv-формата, которые могут быть обработаны с помощью Microsoft Excel или других средств. В названии каждого csv-файла указывается номер эксперимента (ExpN), номер трассы (standM), количество запусков данного эксперимента над трассой. Для работы с трассами средство использует API, предоставляемый библиотеками traceio и tracedb. Методы библиотеки traceio используются для организации потоков и буферов чтения событий из трассы. Библиотека tracedb предоставляет непосредственно методы для работы с различными типами событий и состояний.

Для запуска средства необходимо выполнить строку

```
.\ main stand.trc
```

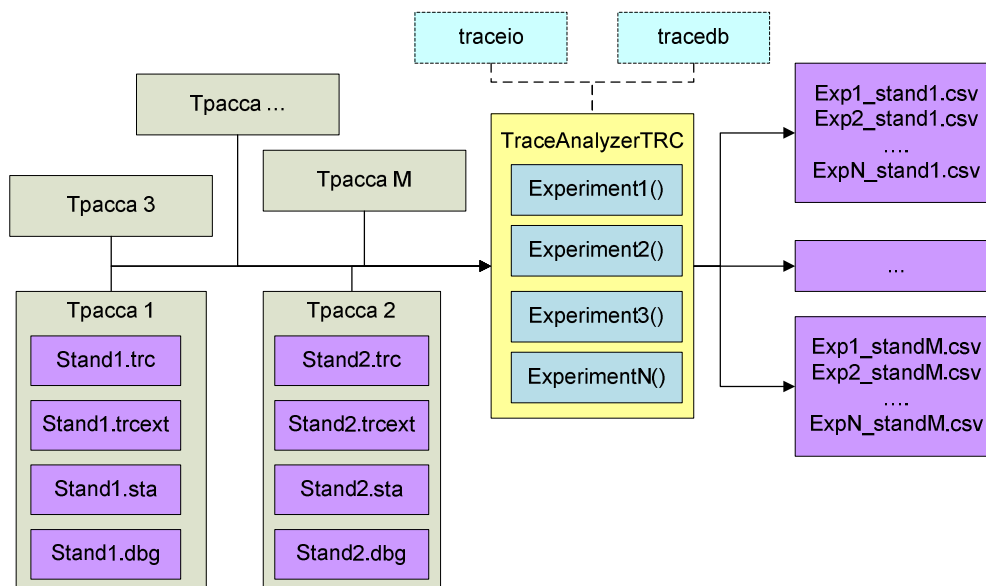



Рисунок 28.Схема работы средства TraceAnalyzerTRC.

3.1.2 Формат TAU

3.1.2.1 Описание формата TAU

TAU (Tuning and Analysis Utilities) – открытый формат трасс, нацеленный на анализ производительности параллельных программ, написанных на языках Fortran, C, C++, Java, Python. Формат был разработан Performance Research Lab (UOR) Орегонского университета США, of the Advanced Computing Laboratory (LAN) of the US-American Los Alamos National Laboratory (LANL) и центральным институтом прикладной математики (ZAM) в исследовательском центре Jülich/Германия.

Трасса в формате TAU состоит из двух файлов: небольшого файла определений и потенциально большого файла событий (см. рисунок 29). Записи кодируются в двоичном виде и, таким образом, формат явно зависит от платформы (порядок байт для платформы не корректируется). Все типы записей занимают фиксированное количество байт. Это упрощает внутреннее управление, но приводит к определенным накладным расходам хранения и к ограничениям расширяемости формата.

Формат TAU поддерживает возможность слияния трасс с помощью инструмента tau_merge.

Нужно отметить, TAU имеет целый ряд инструментов, позволяющих конвертировать трассы в формате TAU в большинство известных форматов: Paraver, SLOG-2, ALOG, EPILOG, OTF, VTF.

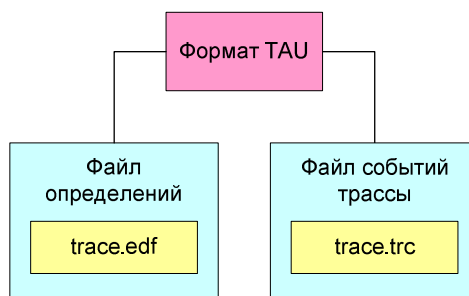


Рисунок 29. Файловая структура формата TAU.

В ходе практического исследования при попытке преобразования трасс, полученных при моделировании PBC PB была установлена недостаточная универсальность формата, поскольку он не позволяет хранить некоторые данные (вещественные значения, массивы char и т.п.), поэтому данный формат не может быть использован для трассировки PBC PB.

3.1.3 Формат OTF

3.1.3.1 Описание формата OTF

OTF (Open Trace Format) – открытый формат трасс, разработанный в Центре высокопроизводительных вычислений университета Дрездена для работы с large-scale параллельными платформами. Три основных цели, для достижения которых создавался формат – это открытость, гибкость и производительность. В силу определенной специфики PBC PB и высокопроизводительных параллельных платформ необходимо произвести сопоставление основных понятий и объектов PBC PB и понятий, используемых в формате OTF.

Формат OTF имеет файловую структуру, представленную на **рисунке 30**.

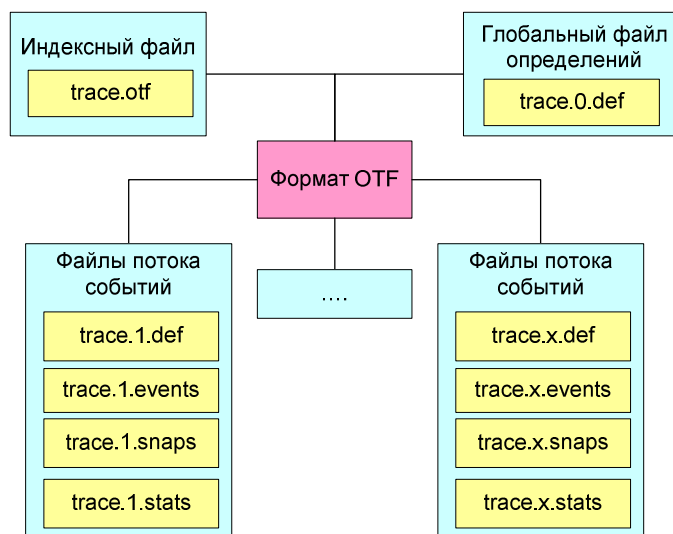


Рисунок 30. Файловая структура формата OTF.

Формат OTF имеет следующие особенности:

- поддерживает события: вход/выход из функции, отправка/прием сообщений, счетчики производительности аппаратного обеспечения;
- платформенная независимость;
- эффективное ACSII кодирование, Zlib сжатие;
- быстрый селективный доступ (для процессов и временных интервалов);
- гибкий контейнер для n процессов, использующий m потоков (файлов);
- API для чтения и записи на C/C++/Python;
- прямая и обратная совместимость.

3.1.3.2 Отображение элементов PBC PB в трассу формата OTF

В соответствии с форматом: процесс в OTF можно ассоциировать с моделью компонента (узлом) PBC PB, вход (выход) в некоторую функцию с входом (выходом) в некоторое состояние модели (узла) в PBC PB, метки о событиях, не связанных с обменами, в OTF с событиями в PBC PB.

3.1.3.3 Средство трансляции трасс TRC в трассу OTF

Схема работы средства представлена на [рисунке 31](#). Для запуска средства необходимо выполнить строку, предварительно положив в рабочую директорию трассу в формате TXT:

```
.\txt2otf
```

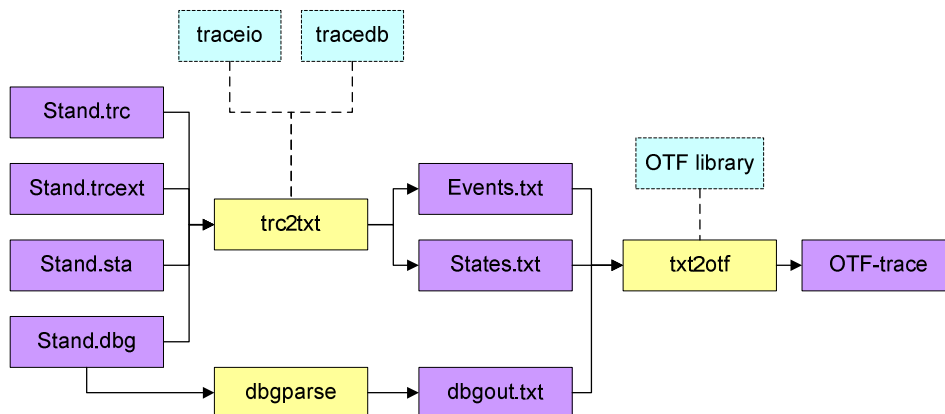


Рисунок 31. Схема работы средства trc2otf.

3.1.3.4 Описание средства исследования трасс OTF - TraceAnalyzerOTF

Для автоматизации процесса проведения экспериментов с трассами в формате OTF было разработано средство TraceAnalyzerOTF. Схема средства представлена на рисунке 32. Средство позволяет автоматически производить замеры времени последовательного линейного чтения трассы до некоторого события с учетом заданного числа итераций.

Для работы с трассами используется открытая библиотека OTF Library версии 1.9. Для запуска средства необходимо выполнить строку:

```
.\main stand1.otf stand2.otf ...
```

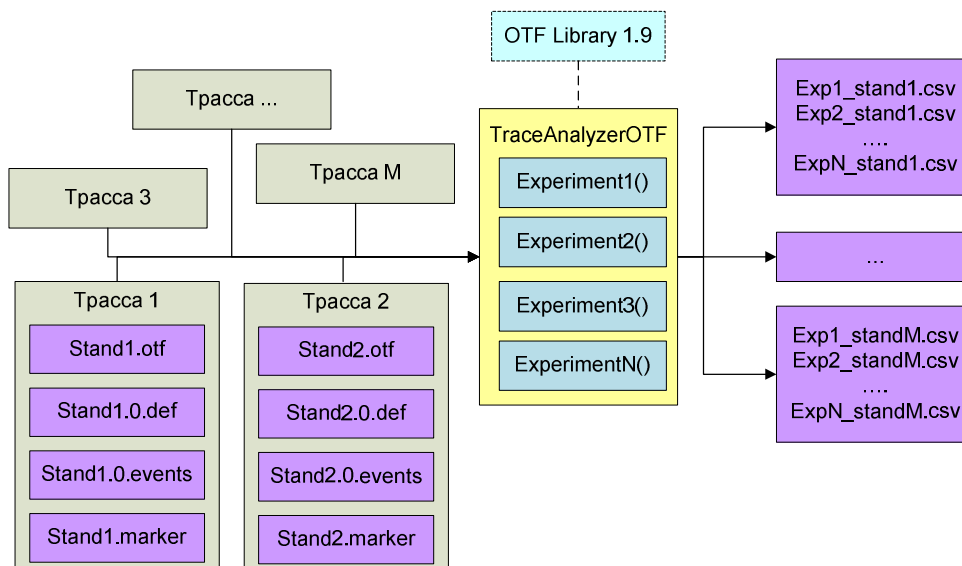


Рисунок 32. Схема работы средства TraceAnalyzerOTF.

3.1.4 Выводы

В данном разделе были подробно рассмотрены файловая структура и структура записей для форматов TRC, TAU и OTF, разработаны программные средства для проведения экспериментального исследования с трассами.

Таким образом, можно сделать следующие выводы:

- Формат TAU не может быть использован для трассировки поведения PBC PБ, поскольку не позволяет хранить данные вещественного типа и трудно расширяем для новых типов событий.
- Формат OTF расширяем и позволяет для событий использовать параметры различных типов.
- Поскольку экспериментальное исследование планировалось проводить на реальных трассах в формате TRC, были разработаны программные средства `trc2txt` и `txt2otf` для конвертирования трасс TRC в формат OTF.
- Для проведения экспериментального исследования и его автоматизации были разработаны средства `TraceAnalyzerTRC` и `TraceAnalyzerOTF`.

3.2 Средства трансляции UML во временные автоматы

Как правило, средства верификации работают с моделями, описанными на специализированных языках, предназначенных для удобной обработки алгоритмами верификации. В тоже время в качестве языка моделирования выбран универсальный язык моделирования UML, расширяемый описанием триггеров на языке C++. Поэтому для применения формальных методов необходима трансляция описания системы в специализированный язык верификатора.

Данный раздел построен следующим образом. Описание системы верификации PBC PБ UPPAAL приведено в разделе 3.2.1. В разделе 3.2.2 описаны ограничения, налагаемые на диаграммы UML общего вида, для того чтобы устранить неоднозначность интерпретации элементов этих диаграмм, но сохранить при этом все преимущества языка UML для описания поведения PBC PБ. Как известно, UML не имеет стандартной семантики, придающей однозначный смысл диаграммам этого языка и имеющей при этом строгое математическое описание, необходимое для проведения формальной верификации PBC PБ. Поэтому в разделах 3.2.3 и 3.2.4 мы описали математическую модель иерархических временных автоматов, которая используется для строгой интерпретации диаграмм

выбранного нами подмножества UML, и алгоритм преобразования диаграмм UML в систему иерархических взаимодействующих автоматов. Далее в разделе 3.2.5 описан алгоритм трансляции иерархических автоматов, используемых в качестве математических моделей диаграмм UML, в сети временных автоматов, используемых в качестве математической модели PBC PB в системе верификации UPPAAL. Общая структура созданного транслятора описана в разделе 3.2.6. Разработанный нами транслятор UML→UPPAAL позволяет подключить систему верификации UPPAAL для проверки правильности поведения PBC PB, описание которых представлено в виде диаграмм UML. Результаты экспериментального исследования сформированного таким образом программно-инструментального средства верификации PBC PB приведены в разделе 4.2.

3.2.1 Программно-инструментальное средство верификации конечных временных автоматов UPPAAL

В качестве средства верификации PBC PB было выбрано программно-инструментальное средство моделирования и верификации информационных систем реального времени UPPAAL. Система UPPAAL хорошо зарекомендовала себя во многих научно-исследовательских и промышленных проектах, связанных с анализом поведения PBC PB [39-46], она находится в открытом доступе, имеет удобный и развитый интерфейс. Система UPPAAL работает с математическими моделями PBC PC, представленными в виде сетей конечных временных автоматов, и спецификациями их поведения, выраженными формулами темпоральной логики RCTL.

Программно-инструментальное средство UPPAAL предназначено для проверки правильности поведения систем взаимодействующих процессов (распределенных информационных систем), работающих в реальном времени. Система верификации UPPAAL была разработана совместными усилиями исследователей университета Упсалы (Швеция) и университета Ольборга (Дания) в период с 1995 по 2004 гг. [47, 48]. Модернизация и расширение возможностей этой системы продолжается и в настоящее время. Математическую основу системы UPPAAL составляет модель конечных автоматов реального времени, предложенная Р. Алуром и Д. Диллом в 1990 г. [49]. Одна из особенностей этой модели состоит в том, что конечные автоматы реального времени, имея лишь конечное число состояний управления, допускают, тем не менее бесконечно много различных состояний вычисления за счет того, что значениями часов (таймеров), которыми снабжены управляющие состояния автоматов, могут быть любые неотрицательные

вещественные числа. Эффективный математический метод решения задач анализа поведения конечных автоматов реального времени был разработан Т. Хензингером в 1994 [50]. С тех пор эта модель вычислений нашла широкое применение при решении задач верификации информационных систем, в которых длительность и сроки выполнения имеют ключевое значение. Свойства поведения автоматов реального времени, нуждающиеся в проверке, описываются формулами темпоральной логики RCTL (Real-time Computation Tree Logic).

UPPAAL – это один из самых известных и успешных проектов создания программно-инструментального средства верификации информационных систем реального времени. Ядро системы составляют алгоритмы преобразования, моделирования поведения и формальной верификации конечных автоматов реального времени, а также процедуры трансляции, преобразующие описания (моделей) информационных систем реального времени в сеть конечных временных автоматов. Ядро системы UPPAAL выполнено на языке программирования C++, а пользовательский интерфейс – на языке программирования Java. Немаловажным достоинством системы верификации UPPAAL является ее общедоступность по адресу <http://www.UPPAAL.com/>.

Чтобы воспользоваться программно-инструментальным средством UPPAAL для верификации PBC PB, описанных в виде диаграмм UML, нами были

1. Разработано строгое описание синтаксиса и семантики того подмножества UML, которое наилучшим образом совместимо с математической моделью временных автоматов, используемой в системе верификации UPPAAL.
2. Создан транслятор, корректно преобразующий диаграммы выбранного подмножества UML в сети временных автоматов системы верификации UPPAAL.

3.2.1.1 Математическая модель конечных автоматов реального времени

Конечный автомат реального времени (далее просто автомат) – это абстрактная управляющая система (машина) с конечным множеством состояний управления, снабженная часовыми механизмами (таймерами). Значениями (показаниями) таймеров являются неотрицательные вещественные числа. Все таймеры изменяют свои значения (продвигают время) синхронно в одном и том же темпе. Показания таймеров можно аннулировать («сбрасывать время»). Можно также сравнивать показания одних таймеров с показаниями других таймеров и с натуральными числами. В зависимости от результатов сравнения показаний таймеров автоматы способны осуществлять переходы из одних состояний

управления в другие. При совершении каждого перехода автомат выполняет определенные действия. Разнообразие таких действия потенциально ничем не ограничено, но в системе UPPAAL разрешается использовать только действия следующих видов: отправление сообщения, прием сообщения, изменение значения переменной (внутренние действия). Действия отправления и приема сообщений применяются для организации взаимодействия (синхронизации) отдельных автоматов, работающих в составе сети конечных автоматов. В особую группу выделены действия сброса времени у выделенных таймеров.

Более формально конечные автоматы описываются так. Обозначим символом C множество таймеров. Элементарным сравнением будем называть всякий предикат вида $x \diamond n$ или $x - y \diamond n$, где x, y - таймеры, n - натуральное число, \diamond - одно из арифметических отношений $\{<, \leq, =, \geq, >\}$. Предохранителем называется всякая элементарная конъюнкция элементарных сравнений и их отрицаний. Множество всевозможных предохранителей над множеством таймеров C обозначим записью $B(C)$.

Конечный временной автомат – это система $U = \langle L, l_0, C, A, E, I \rangle$, состоящая из:

- конечного множества состояний управления L ,
- начального состояния управления l_0 ,
- множества таймеров C ,
- множества действий A , включающее действия отправления и приема сообщений, а также внутренние действия,
- множество переходов $E \subseteq L \times A \times B(C) \times 2^C \times L$,
- назначение инвариантов $I : L \rightarrow B(C)$.

Каждый переход (l, a, g, C', l') из множества E (далее такой переход будем обозначать записью $l \xrightarrow{a, g, C'} l'$) означает, что в том случае, если показания всех таймеров таковы, что предохранитель g принимает логическое значение true, то автомат переходит из состояния управления l в состояние управления l' , и при этом показания всех таймеров из множества C' аннулируются, и выполняется действие a . Инвариант $I(l)$, приписанный состоянию управления l , - это необходимое условие, которое должно быть соблюдено, для того чтобы автомат имел возможность пребывать в этом состоянии управления.

Для того, чтобы определить вычисления автомата $U = \langle L, l_0, C, A, E, I \rangle$, введем следующие понятия и обозначения. Оценкой таймеров будем называть всякое отображение $v: C \rightarrow R_{\geq 0}$, приписывающее каждому таймеру неотрицательное вещественное число в качестве значения. Оценку v_0 , приписывающую каждому таймеру число 0, назовем инициальной оценкой. Для каждой оценки таймеров v , неотрицательного вещественного числа d и множества таймеров C' будем использовать запись $v+d$ для обозначения оценки, приписывающей каждому таймеру x значение $v(x)+d$, а запись $v[C']$ для обозначения оценки, приписывающей каждому таймеру из множества C' значение 0 (сброс времени), а каждому таймеру, не принадлежащему множеству C' , значение $v(x)$. Также будем использовать запись $v \in g$ для обозначения того, что для оценки таймеров v предохранитель g принимает логическое значение true.

Множество всевозможных вычислений автомата $U = \langle L, l_0, C, A, E, I \rangle$ - это множество всевозможных начальных трасс в системе переходов $\langle S, s_0, \rightarrow \rangle$, где $S \subseteq L \times R_{\geq 0}^{|C|}$ - множество состояний вычислений, $s_0 = (l_0, u_0) \in S$ - начальное состояние всех вычислений, а $\rightarrow \subseteq S \times (R_{\geq 0} \cup A) \times S$ - отношение, описывающее один шаг каждого вычисления и удовлетворяющее следующим двум требованиям:

- $(l, v) \xrightarrow{d} (l, v+d)$, если для любого вещественного d' , удовлетворяющего неравенствам $0 \leq d' \leq d$ выполняется соотношение $u+d' \in I(l)$,
- $(l, v) \xrightarrow{a} (l', v')$, если существует такой переход $l \xrightarrow{a, g, C'} l' \in E$, для которого выполняются условия $v \in g$, $v[C'] \in I(l')$; при этом $v' = v[C']$.

На каждом шаге вычисления автомата:

- либо происходит продвижение времени, и автомат остается в прежнем состоянии управления, при условии, что это продвижение времени сохраняет истинность инварианта состояния,
- либо без продвижения, но со сбросом времени некоторых таймеров осуществляется некоторый переход из того состояния управления, в котором пребывает автомат, в новое состояние управления, при условии, что новые показания таймеров удовлетворяют инварианту нового состояния управления.

Несколько конечных автоматов могут вступать во взаимодействие друг с другом посредством действий синхронизации, образуя, таким образом, параллельную композицию (или, иначе, сеть) конечных автоматов реального времени. Вычисления сети конечных автоматов отличаются от вычислений отдельного автомата тем, что сочетаемые друг с другом действия синхронизации выполняются в двух автоматах сети одновременно (синхронно). Действия синхронизации подразделяются на активные действия (обозначаются записью $!a$) и парные им пассивные действия (обозначаются записью $?a$). Активные действия соответствуют отправлению синхронизирующего сигнала по каналу a , а пассивные действия – приему синхронизирующего сигнала по каналу a .

Предположим, что задано конечное множество (сеть) $N = \{U_1, U_2, \dots, U_n\}$, состоящее из автоматов $U_i = \langle L_i, l_{0i}, C, A, E_i, I_i \rangle$, $1 \leq i \leq n$. Тогда всевозможные вычисления сети автоматов N задаются системой переходов $\langle \bar{S}, \bar{s}_0, \Rightarrow \rangle$, в которой $\bar{S} = (L_1 \times L_2 \times \dots \times L_n) \times R_{\geq 0}^C$ – множество вектор-состояний вычислений, $\bar{s}_0 = (l_{01}, l_{02}, \dots, l_{0n}, v_0)$ – начальное вектор-состояние всех вычислений, $\Rightarrow \subseteq S \times S$ – отношение, описывающее один шаг каждого вычисления сети и удовлетворяющее следующим трем условиям:

- $(\bar{l}, v) \Rightarrow_N (\bar{l}, v + d)$, если для любого $i, 1 \leq i \leq n$, справедливо отношение $(l[i], v) \xrightarrow{d} (l[i], v + d)$,
- $(\bar{l}, v) \Rightarrow_N (\bar{l}[l'[i]/l[i]], v')$, если для некоторого $i, 1 \leq i \leq n$, справедливо отношение $(l[i], v) \xrightarrow{a} (l'[i], v')$, где a – это внутреннее действие, и для любого $j, 1 \leq j \leq n, j \neq i$, верно соотношение $v' \in I_j(\bar{l}[j])$,
- $(\bar{l}, v) \Rightarrow_N (\bar{l}[l'[i]/l[i], l'[j]/l[j]], v')$, если для некоторой пары $i, j, 1 \leq i, j \leq n$, справедливы соотношения $l[i] \xrightarrow{!a, g', C'} l'[i]$, $l[j] \xrightarrow{!a, g', C''} l'[j]$, $v' = v[C' \cup C'']$, $v' \in I_i(\bar{l}'[i])$, $v' \in I_j(\bar{l}'[j])$ и для любого $k, 1 \leq k \leq n, k \neq i, j$, справедливо соотношение $v' \in I_k(\bar{l}[k])$.

Вычислением сети автоматов $N = \{U_1, U_2, \dots, U_n\}$ является всякая максимальная последовательность вектор-состояний вычисления вида

$$\vec{s}_0 \Rightarrow_N \vec{s}_1 \Rightarrow_N \vec{s}_2 \Rightarrow_N \dots \Rightarrow_N \vec{s}_k \Rightarrow_N \dots$$

Анализ поведения сети автоматов заключается в доказательстве того, что все вычисления этой сети удовлетворяют некоторому свойству (спецификации), которое задается формулой темпоральной логики RCTL.

3.2.1.2 Автоматы в системе UPPAAL

Язык описания конечных временных автоматов в системе UPPAAL разрешает использовать некоторые дополнительные средства, расширяющие тот минимальный набор действий, которые присутствуют в стандартной математической модели автомата. Основные отличительные особенности автоматов UPPAAL таковы.

1. При описании автоматов разрешается использовать
 - целочисленные константы,
 - целочисленные переменные фиксированного диапазона значений,
 - целочисленные массивы фиксированной длины,
 - массивы таймеров фиксированной длины,
 - массивы каналов фиксированной длины.

В целочисленном типе данных определены булевы и арифметические операции и отношения. Разрешается также использовать сложные булевы и арифметические выражения, построенные из переменных, констант и операций так, как это общепринято в императивных языках программирования.

2. В качестве элементарных сравнений разрешается использовать также булевы выражения, содержащие целочисленные константы и переменные, таймеры, целочисленные массивы, массивы таймеров; при этом, однако, таймеры разрешается сравнивать только с константами и с другими таймерами.
3. В качестве каналов синхронизации разрешается использовать выражения целочисленного типа
4. В качестве внутренних действий разрешается использовать операторы присваивания вида $x:=E$, где x – переменная целочисленного типа. Эффект этого действия такой же, как и эффект оператора присваивания в императивных языках программирования.
5. В качестве предохранителей, фигурирующих в инвариантах состояний управления, разрешается использовать только элементарные отношения вида $t < E$ или $t \leq E$, где t – таймер, а E – выражение целочисленного типа.

6. Разрешается использование широковещательных каналов связи и широковещательного обмена сообщениями по таким каналам. Действие широковещательного отправления сообщения выполняется всегда, независимо от того, может ли быть выполнено хотя бы одно парное действие широковещательного приема сообщения. При каждом выполнении перехода, содержащего действие широковещательного отправления сообщения, *все дееспособные* переходы, содержащие парные действия широковещательного приема сообщения *обязаны* выполняться синхронно. Во избежание неоднозначности все переходы, содержащие действия широковещательного приема сообщений не имеют права сбрасывать время каких-либо таймеров.
7. Разрешается использовать так называемые *срочные каналы связи*. Особенность этого типа каналов связи состоит в том, что в любом состоянии, из которого исходит дееспособный переход, содержащий действие отправления сообщения по срочному каналу связи, *обязано* выполнить этот переход. Во избежание неоднозначности все переходы, содержащие такие действия, не имеют права использовать таймеры в своих предохранителях.
8. Некоторые состояния управления особо выделены как *срочные состояния*. В срочных состояниях управления запрещается осуществлять шаг вычисления, связанный с продвижением времени. Это равносильно тому, как если бы с каждым таким состоянием был ассоциирован специальный уникальный таймер x , который бы сбрасывал время на каждом переходе, ведущем в это состояние, а само это состояние содержало бы в своем предохранителе-инварианте элементарное отношение $x \leq 0$.
9. *Сверхсрочные состояния управления* налагают еще более строгие ограничения на порядок выполнения переходов в сетях автоматов. Если в состоянии вычисления \vec{s} сети автоматов хотя бы один автомат пребывает в сверхсрочном состоянии управления, то на следующем шаге вычисления обязан сработать хотя бы один переход, исходящий из этого состояния управления.

3.2.1.3 Темпоральная логика RCTL

Самая главная процедура, которую способна выполнять система верификации UPPAAL (и ради которой была создана вся система) – это процедура проверки выполнимости формул темпоральной логики RCTL на бесконечных моделях, представляющих собой множество всевозможных вычислений конечных временных

автоматов (сетей автоматов). Формулы логики RCTL используются в качестве спецификаций свойств поведения автоматов. Процедура верификации системы UPPAAL для заданного автомата (сети автоматов) и заданной темпоральной формулы проверяет, выполняется ли эта формула на модели, образованной множеством всевозможных вычислений этого автомата (сети автоматов). Для того, чтобы увеличить гарантии успешного завершения процедуры верификации, в UPPAAL разрешается использовать лишь темпоральные формулы, имеющие очень простое устройство. В качестве атомарных формул разрешается использовать любое элементарное отношение, а также формулы вида $in\ l$, где l - некоторое состояние управления автомата. Из атомарных формул при помощи обычных булевых связей конъюнкции, дизъюнкции и пр. конструируются простые формулы состояний. И наконец, темпоральные формулы состояний образуются за счет применения к простым формулам состояния одного из следующих пяти темпоральных операторов: $A[], A\langle\rangle, E[], E\langle\rangle, \mapsto$. Выполнимость формул каждого из перечисленных видов определяется следующим образом. Для каждого состояния вычисления $s = (l, v)$ заданного конечного автомата $U = \langle L, l_0, C, A, E, I \rangle$ и для каждого элементарного отношения g это отношение выполняется в состоянии s (для обозначения этого факта используется обозначение $s \models g$), если $v \in g$. Атомарная формула $in\ l'$ выполняется в состоянии вычисления $s = (l, v)$ тогда и только тогда, когда $l = l'$. Выполнимость простой формулы состояния φ определяется по законам булевой алгебры на основании выполнимости входящих в ее состав атомарных формул. И, наконец, выполнимость темпоральных формул состояния определяется по следующим правилам:

- $s \models A[]\varphi$ тогда и только тогда, когда в каждом состоянии всякого вычисления автомата U , которое начинается в состоянии s , выполняется формула φ ;
- $s \models E[]\varphi$ тогда и только тогда, когда в каждом состоянии хотя бы одного вычисления автомата U , которое начинается в состоянии s , выполняется формула φ ;
- $s \models A\langle\rangle\varphi$ тогда и только тогда, когда хотя бы в одном состоянии всякого вычисления автомата U , которое начинается в состоянии s , выполняется формула φ ;
- $s \models E\langle\rangle\varphi$ тогда и только тогда, когда хотя бы в одном состоянии хотя бы одного вычисления автомата U , которое начинается в состоянии s , выполняется формула φ ;

- $s \models \varphi \mapsto \psi$ тогда и только тогда, когда в каждом вычислении автомата U , которое начинается в состоянии s , после достижения такого состояния вычисления s' , в котором выполняется формула φ , обязательно будет достигнуто такое состояние вычисления s'' , в котором выполняется формула ψ .

Помимо перечисленных формул в системе UPPAAL разрешается использовать специальные формулы, которые пригодны для проверки свойств любого автомата. К числу таких формул относится выражение *deadlock*, которое выполняется во всех тех состояниях вычисления, из которых нельзя совершить ни одного шага вычисления, как за счет выполнения какого-либо перехода, так и за счет продвижения времени.

Все проверяемые в системе UPPAAL темпоральные свойства подразделяются на три класса: свойства достижимости, свойства безопасности и свойства живости. Свойства достижимости формулируются так: существует ли хотя бы одно вычисление автомата, по ходу которого достигается состояние, удовлетворяющее некоторому условию φ . Эти свойства выражаются формулами вида $E \langle \varphi \rangle$. Свойства безопасности – это утверждения, которые имеют формулировку: «каждое состояние некоторого (или всех) вычислений автомата удовлетворяет некоторому условию безопасности φ ». Свойства безопасности выражаются темпоральными формулами вида $A[\varphi]$ и $E[\varphi]$. Свойства живости – это утверждения, которые имеют формулировку: «когда-нибудь по ходу любого вычисления будет достигнуто состояние, удовлетворяющее некоторому условию φ ». Свойства живости выражаются формулами вида $A \langle \varphi \rangle$ и $\varphi \mapsto \psi$.

3.2.1.4 Возможности системы верификации UPPAAL

Программно-инструментальное средство верификации информационных систем реального времени UPPAAL состоит из трех основных компонентов: графический редактор, модуль симуляции (моделирования поведения), модуль верификации.

Графический редактор

Для того, чтобы проанализировать поведение информационной системы реального времени, ее необходимо представить в виде параллельной композиции (сети) конечных временных автоматов. Описание такой сети состоит из нескольких частей, каждая из которых помещается в отдельный файл.

1. **Описание глобальных элементов сети (Global declaration)** содержит объявления всех тех констант, переменных, таймеров, каналов связи, которые являются общими для всех автоматов сети.
2. **Шаблоны (Templates)** – набор файлов, каждый из которых содержит параметризованное описание некоторого автомата. Графический редактор позволяет пользователю работать с описанием каждого автомата как с размеченным ориентированным графом. Вершины графа соответствуют состояниям управления автомата. Каждое состояние управления имеет имя и две пометки, которые указывают на тип состояния (простое, срочное или глобально срочное) и инвариант состояния. Дуги графа соответствуют переходам в автомате. Каждая дуга графа помечена предохранителем и списком действий, включающим действия синхронизации и присваивания. Сброс времени у таймеров указывается явно посредством операторов присваивания. Наряду с глобальными элементами в описании автомата можно использовать локальные параметры. Редактор позволяет вводить новые состояния управления (вершины графа) и переходы (дуги графа), удалять имеющиеся элементы, изменять описания вершин и дуг графа.
3. **Назначения процессов (Process assignments)** – файл, содержащий конкретные описания автоматов. Каждое конкретное описание указывает шаблон, на основе которого определяется автомат, а также инициализацию всех параметров этого шаблона.
4. **Описание системы (System definition)** – файл, содержащий список всех процессов (автоматов) анализируемой системы.

Симулятор (модуль моделирования)

Назначение симулятора – построение и визуализация отдельных вычислений анализируемой сети процессов (автоматов) реального времени. Существуют три режима работы симулятора: режим случайного выбора, режим ручного управления и режим просмотра импортированного вычисления. В режиме случайного выбора симулятор строит трассу одного из возможных вычислений анализируемой сети, выбирая на каждом шаге случайным образом один из дееспособных переходов. В режиме ручного управления пользователь конструирует трассу вычисления сети автоматов самостоятельно, указывая на каждом шаге очередной переход, который должен быть выполнен. В режиме просмотра импортированного вычисления симулятор конструирует трассу некоторого вычисления,

заданного априори (как правило, того вычисления системы, которое строится верификатором в качестве контрпримера в том случае, когда проверяемое свойство безопасности не выполняется). Трасса вычисления сети автоматов представляет собой набор параллельных цепочек шагов каждого автомата, входящего в состав сети.

Верификатор (модуль проверки правильности)

Верификатор – это главный и наиболее сложный модуль системы UPPAAL. В строках меню верификатора пользователь должен указать имя файла, в котором содержится описание проверяемой поведения сети автоматов, а также формулу RCTL, которая формально описывает проверяемое свойство сети (в строке «Комментарии» можно указать формулировку этого же свойства на естественном языке). При запуске модуля верификации алгоритм верификации моделей применяется к описанной сети автоматов и заданной формуле RCTL. В том случае, если алгоритм обнаруживает, что невыполнима формула безопасности или выполнима формула достижимости, то наряду с констатацией этого факта также конструируется трасса вычисления, подтверждающая этот факт. Эта трасса может быть в дальнейшем использована (импортирована) симулятором для последующего визуального анализа ее устройства

3.2.2 Ограничения на диаграммы состояний UML

Язык UML предназначен для моделирования широкого класса систем, разрабатываемых на различных языках программирования под разные платформы. По этой причине авторы стандарта UML намеренно не фиксируют синтаксические ограничения и не определяют до конца семантику модели [34, гл. 13]. В языке определяется лишь *метамодель*, то есть ограничения на любую модель, описанную в нотации UML. Таким образом, в общем случае известно лишь, является ли модель *правильно-построенной*. При этом даже правильно-построенная модель может быть не до конца определена, а такие элементы как предусловия, действия и триггеры могут быть написаны на естественном языке, что ведёт к неоднозначной интерпретации.

Авторы языка предлагают для конкретного класса систем задавать свой *профиль* модели, не меняя при этом общую нотацию. Профиль модели определяется при помощи дополнительных стереотипов, ограничений и помеченных значений. Тем не менее, в случае диаграмм состояний создание профиля не решает проблем с интерпретацией триггеров, действий и предусловий. Так как конечной целью является формальная верификация

описания системы, то необходимо строго задать синтаксис и семантику всех используемых элементов диаграмм состояний. Поэтому в данной работе мы добавляем дополнительные ограничения на структуру диаграмм состояний и синтаксис выражений, устраняя таким образом неоднозначность интерпретации элементов диаграмм.

На **рисунках 33, 34, 35** приведены диаграммы, иллюстрирующие все допустимые элементы диаграмм состояний UML и используемую нотацию. Стоит заметить, что на структуру диаграмм не накладывается существенных ограничений, а те виды псевдосостояний, которые отсутствуют на рисунках, могут быть добавлены в алгоритм и средство трансляции без существенных изменений. На **рисунке 33** показана основная диаграмма, на **рисунках 34, 35** – диаграммы, вложенные в нее.

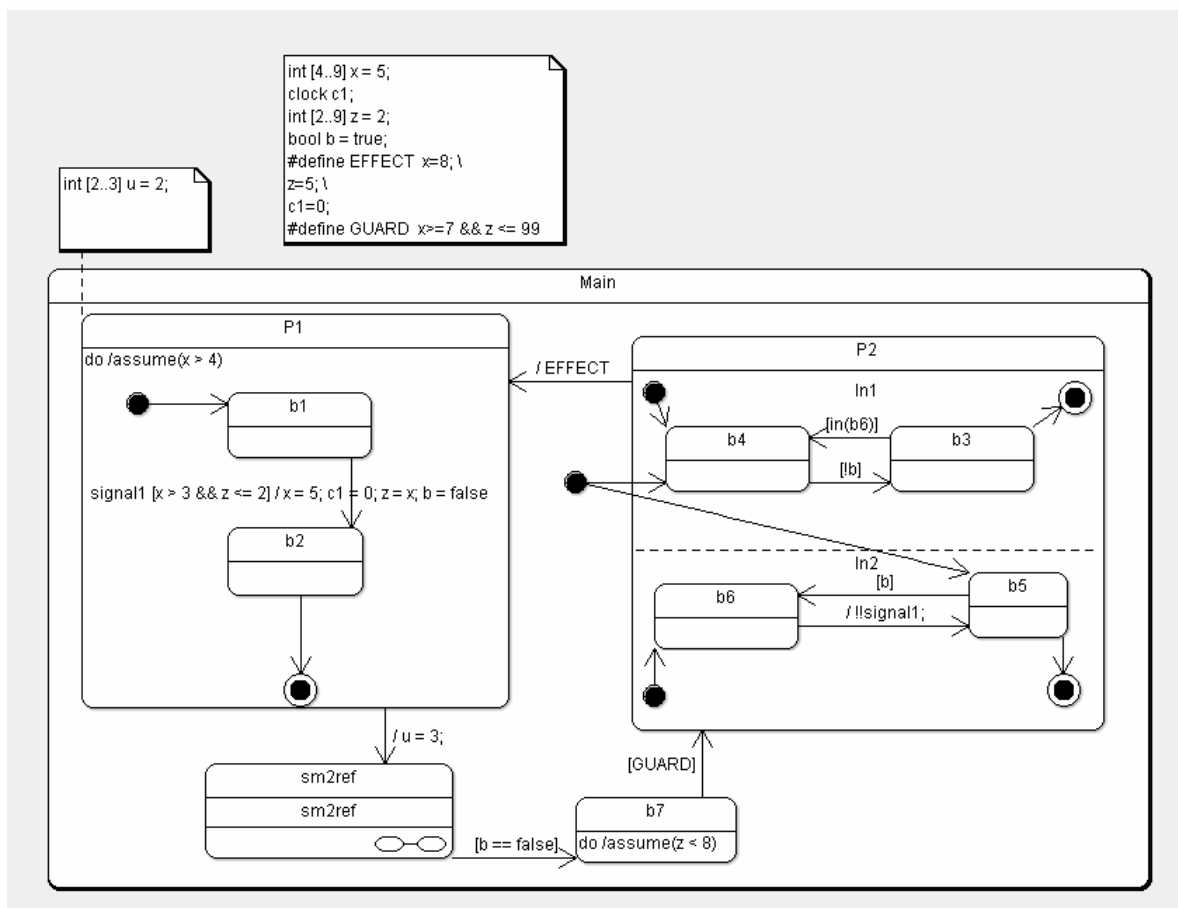


Рисунок 33. Пример диаграммы состояний UML.

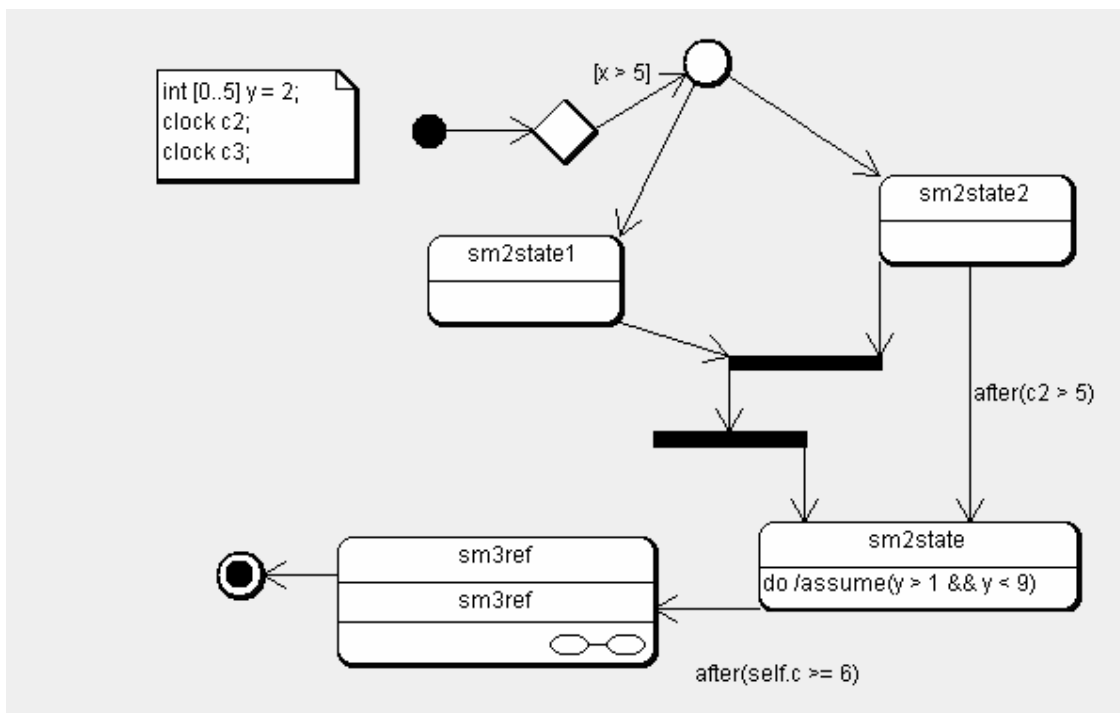


Рисунок 34. Пример диаграммы состояний UML(2).

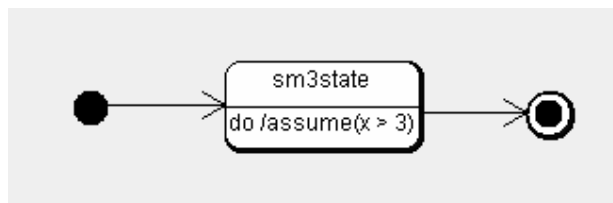


Рисунок 35: Пример диаграммы состояний UML(3).

Диаграмма на [рисунке 33](#) содержит четыре состояния, объединенных в цикл. Состояние b7 – простое, P2 и P1 – композитные. Состояние P1 – композитное состояние, и в нем последовательно осуществляется переход из простого состояния b1 в простое состояние b2. Состояние P2 содержит параллельный регион, поэтому в нем параллельно выполняется деятельность, описанная как регионом In1, так и регионом In2. Состояние sm2ref – ссылка на автомат, изображенный на [рисунке 34](#). В sm2ref также есть ссылка на автомат sm3ref.

Далее приводятся принятые нами ограничения и приводятся пояснения.

Структурные ограничения на диаграммы.

- Простые состояния удовлетворяют стандартной метамодели UML, их входы и выходы сохраняют стандартную интерпретацию.
- Допускаются переходы как в начальное состояние композитного состояния, так и непосредственно в композитное состояние. В последнем случае переход осуществляется в каждое начальное состояние композитного.

Ограничения на предусловия переходов. Предусловие задаётся как конъюнкция элементарных условий, то есть: $Atom1 \ \&\& \ \dots \ \&\& \ AtomN$. Элементарные условия $AtomI$ – это линейные неравенства над переменными, таймерами и константами, то есть выражения вида:

```

Atom ::= Expr < Expr | Expr <= Expr | Expr > Expr | Expr >= Expr
      | Expr == Expr | Expr != Expr
      | in(S)
      | BoolExpr == BoolExpr | BoolExpr != BoolExpr
Expr  ::= ClockVar | ClockVar - ClockVar
      | IntExpr | IntExpr + IntExpr | IntExpr - IntExpr | IntConst | (Expr)
ClockVar ::= <идентификатор>
IntExpr  ::= IntVar | IntConst
IntVar   ::= <идентификатор>
IntConst ::= <целочисленная константа>
BoolExpr ::= BoolVar | false | true
BoolVar  ::= <идентификатор>

```

При выборе синтаксиса мы в первую очередь руководствовались стандартным синтаксисом языков, подобных языку Си. Набор ограничений был выбран, исходя из возможностей современных средств верификации, в частности, средства UPPAAL. Семантика выражений полностью совпадает с семантикой операторов в языке Си. Выражение $in(S)$, заимствованное из языка STATEMATE [51], означает, что в автомате активно состояние S .

В графической нотации UML, как обычно, на диаграмме предусловия записываются в квадратных скобках.

Ограничения на действия переходов. В качестве действий допускаются присваивания языка Си над целочисленными переменными, константами и таймерами, а также операторы отправки сообщения и недетерминированного выбора значения. Такие операторы могут объединяться в последовательность действий, разделённых точкой с запятой.

```

Action ::= Assign | SendSignal | Choose | Sequence
Assign ::= IntVar = IntExpr | BoolVar = BoolExpr
Choose ::= IntVar = random() | BoolVar = random()
SendSignal ::= !!<идентификатор>
Sequence ::= Action; Sequence

```

Операция отправки сигнала соответствует широковещательной отсылке [51], часто используемой в РВС РВ. Автомат выставляет указанный сигнал X. На следующем шаге системы все автоматы, способные совершить переход с триггером по приёму сигнала X, должны совершить такой переход. Возможна ситуация, при которой ни один автомат не может совершить переход. В этом случае посланный сигнал считается потерянным.

Оператор **Choose** позволяет присвоить переменной одно из значений типа этой переменной недетерминированным образом. Несмотря на название функции `random()`, выбор осуществляется недетерминировано, без задания какого-либо распределения случайной величины. При моделировании системы для выбора значения может использоваться датчик случайных чисел.

В графической нотации UML действия указываются после косой черты, то есть '/

Ограничения на триггеры переходов. В качестве условий срабатывания переходов (триггеров) допускаются приём сигнала и срабатывание таймера:

```
Trigger ::= ReceiveEvent | TimerEvent
ReceiveEvent ::= <идентификатор сигнала>
TimerEvent ::= after(Expr)
```

В качестве выражений в `TimerEvent` допускаются лишь выражения над таймерами. Событие по приёму сигнала приходит после того, как один из автоматов выставил сигнал.

Дополнительно в событиях срабатывания таймера можно использовать выражение `self.c`, означающее специальный таймер, уникальный для каждого состояния.

Инварианты состояний. К состоянию может быть добавлен инвариант, задающий временное ограничение на пребывании в состоянии. Так как в диаграммах состояний нет явной поддержки таких выражений, инварианты задаются в качестве внутренних переходов состояния. Синтаксис инвариантов следующий: `assume(Expr)`, где `Expr` – выражение над таймерами.

Объявления переменных. В диаграммах UML можно добавлять комментарии. Комментарии используются для объявления переменных, используемых в предусловиях, инвариантах и действиях. Каждое объявление начинается с новой строки и заканчивается точкой с запятой. Имя переменной начинается с буквы и содержит буквы, цифры, точки и знаки подчеркивания. Для целых переменных обязательно указывать начальное значение и диапазон. Например: `int [4..7] x = 5; clock c; bool b = false;` Переменные видны в пределах диаграммы, где они объявлены и во всех вложенных диаграммах.

Семантически объявление таких переменных соответствует объявлению атрибута соответствующего класса. Конструкция объявления переменных добавлена ради удобства, так как часто для компонент РВС РВ требуется описание поведения, а не статической структуры соответствующих классов.

Макросы. В комментариях помимо переменных можно задавать макросы в стиле языка Си. Синтаксис макросов следующий:

```
#define X <пробел или табуляция> Y <перевод строки>
```

При этом X должно быть допустимым именем переменной. Y - произвольный текст, может включать в себя пробелы. Можно делать многострочные макросы, если ставить в конце каждой строки \. В отличие от языка C, нельзя объявить макрос, имеющий то же имя, что и некоторая переменная.

3.2.3 Модель иерархических временных автоматов

Модель иерархических временных автоматов может быть строго описана в виде системы общего вида и наложенных на эту систему ограничений.

Иерархический временной автомат — это система

$$(S, S_0, \eta, \text{Type}, \text{Var}, \text{Clocks}, \text{Chan}, \text{Inv}, T, \text{Chantype}),$$

где

1. S — множество состояний,
2. $S_0 \subseteq S$ — множество инициальных состояний,
3. $\eta : S \rightarrow 2^S$ — функция вложенности состояний,
4. $\text{Type} : S \rightarrow \{\text{AND}, \text{XOR}, \text{BASIC}, \text{ENTRY}, \text{EXIT}\}$ — функция типов состояний,
5. Var — множество переменных,
6. Clocks — множество таймеров,
7. Chan — множество каналов,
8. $\text{Inv} : S \rightarrow \text{Invariant}$ — функция инвариантов состояний,
9. $T \subseteq S \times (\text{Guard} \times \text{Sync} \times \text{Reset} \times \{\text{true}, \text{false}\}) \times S$ — множество переходов,
10. $\text{Chantype} : \text{Chan} \rightarrow \{\text{h}, \text{b}\}$ — функция типов каналов.

Поясним введенное понятие иерархического временного автомата.

Каждый элемент множества Invariant — либо константа true или false, либо множество выражений вида «x op n» или «(x-y) op n», где x, y — таймеры из множества Clocks , n — целочисленная константа, op — один из знаков <, ≤, =.

Каждый элемент множества *Guard* — либо константа *true* или *false*, либо множество предикатов. При этом каждый предикат имеет вид либо «*E'* оп *E''*», где *E'*, *E''* — арифметические выражения над переменными из множества *Var*, оп — один из знаков сравнения (<, =, >, ≤, ≥), либо «*x* оп *n*», где *x* — таймер из множества *Clocks*, *n* — целочисленная константа, оп — также один из знаков сравнения.

Каждый элемент множества синхронизаций *Sync* имеет вид либо «*c!*» (посылка сообщения по каналу *c*), либо «*c?*» (прием сообщения по каналу *c*), либо «*none*» (отсутствие посылки/приема сообщения). При этом функция *Chantype* для каждого канала определяет, является ли этот канал широковещательным (*broadcast*) или «точка-точка» (*handshake*).

Каждый элемент множества *Reset* есть множество присваиваний вида «*x* ← 0», где *x* — таймер из множества *Clocks*, и «*v* ← *E*», где *v* — переменная из множества *Var*, а *E* — арифметическое выражение над переменными из множества *Var*.

Состояния типов *AND* и *XOR* будем называть метасостояниями, состояния типа *ENTRY* — входами, состояния типа *EXIT* — выходами, состояния типа *BASIC* — простыми состояниями.

Если не учитывать функции вложенности состояний, иерархический автомат может рассматриваться как помеченный ориентированный мультиграф. В связи с этим к автомату будет также применяться терминология, используемая в теории графов. В частности, переходы при таком рассмотрении оказываются дугами мультиграфа, несущими четыре пометки. По порядку, обозначенному при определении модели, будем называть эти пометки, соответственно, предусловием, синхронизацией, присваиванием и флагом срочности. Если флаг срочности имеет значение *true*, соответствующий переход будем называть срочным, иначе — несрочным. Кроме того, каждая вершина рассматриваемого мультиграфа помечена ровно одним из пяти типов, определяемых функцией *Type*, инвариантом, определяемым функцией *Inv*, и может быть помечена как инициальная.

Опишем ограничения, при выполнении которых введенная система определяет корректный иерархический временной автомат. Ограничения системы могут быть разбиты на несколько классов:

- ограничения на структуру состояний:
 - функция η задает ориентированное корневое дерево с множеством *S* в качестве вершин и дугами, идущими от корня к листьям дерева (в дальнейшем будем называть его деревом состояний);
 - все метасостояния и только они имеют потомков в дереве состояний;

- потомками метасостояния типа AND в дереве состояний могут быть только входы, выходы и метасостояния;
- ограничения на множество инициальных состояний:
 - корень дерева состояний является инициальным состоянием;
 - инициальными могут быть только простые состояния и метасостояния;
 - если некоторое состояние является инициальным, то и его предок в дереве состояний является инициальным состоянием;
 - среди потомков состояния типа XOR в дереве состояний ровно один является инициальным;
 - все потомки состояния типа AND в дереве состояний являются инициальными;
- ограничения на встречающиеся выражения:
 - любые две дуги, исходящие из входа, вложенного в состояние типа AND, несут присваивания, использующие непересекающиеся множества переменных;
 - инварианты входов и выходов суть константы true;
- ограничения на переходы:
 - из входа, вложенного в метасостояние типа XOR, исходит ровно одна дуга;
 - из входа, вложенного в метасостояние типа AND, исходит ровно по одной дуге, ведущей в каждого потомка, вложенного в данное метасостояние и также являющегося метасостоянием, и больше не исходит никаких дуг;
 - все ограничения на соотношение инцидентных переходов состояний приведены на рисунке 36; на этом рисунке кругами обозначены простые состояния, тонкими линиями — входы, толстыми линиями — выходы, квадратами — метасостояния, стрелками — переходы, вложенность состояний соответствует геометрической вложенности на рисунке;
 - все псевдопереходы (дуги, исходящие из входа или ведущие в выход) являются несрочными и не несут синхронизации (т.е. несут синхронизацию «none»);
 - если дуга, являющаяся псевдопереходом, исходит из входа, то она не несет предусловия (т.е. несет предусловие true);
 - если дуга, являющаяся псевдопереходом, ведет в выход, то она не несет

присваиваний (т.е. несет присваивание, являющееся пустым множеством);

- если дуга, являющаяся псевдопереходом, исходит из выхода и ведет в выход, то она не несет ни предусловия, ни присваиваний.

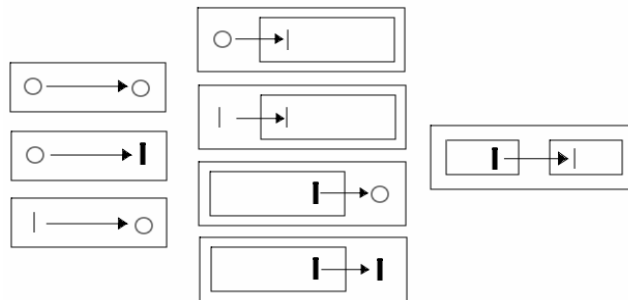


Рисунок 36. Ограничения на соотношение инцидентных переходов состояний.

3.2.4 Преобразование диаграммы состояний во временной автомат

Так как структура диаграмм состояний отличается от структуры иерархических временных автоматов, необходим промежуточный этап преобразования диаграмм UML перед непосредственной их трансляцией в UPPAAL.

В первую очередь, ещё на этапе синтаксического разбора диаграммы UML, проводится преобразование выражений, отсутствующих в синтаксисе UPPAAL. Вначале, до разбора предусловий и присваиваний, проводится подстановка всех макросов. Затем заменяется конструкция $\text{in}(S)$ в предусловиях: вместо нее заводится булевская переменная s_var , соответствующая состоянию S , на все дуги, входящие в S , добавляется присваивание $s_var = \text{true}$, а на все выходящие – $s_var = \text{false}$.

Затем все ссылки на вложенные автоматы заменяются самими этими автоматами до тех пор, пока ссылок не останется; если во вложенном автомате есть ссылка на еще один вложенный, он также будет вставлен в исходный автомат. На этом этапе предотвращается коллизия имен: если во вложенном автомате есть состояние с таким же именем, как и какое-либо состояние главного автомата, состояние переименовывается. Если в двух разных областях видимости (например, в двух автоматах, вложенных в один и тот же) определены переменные с одинаковыми именами, то в одной из областей видимости переменная переименовывается. В результате формируется описание единственного иерархического автомата в виде диаграммы UML, который далее будет транслироваться в систему временных автоматов UPPAAL.

Следующий этап трансляции – корректировка композитных состояний. Во временном автомате допускаются только переходы между простыми состояниями, входами и выходами,

поэтому необходимо так преобразовать переходы, инцидентные композитным состояниям диаграммы UML, чтобы они стали инцидентны нужным входами и выходам.

В первую очередь, явно добавляются входы и выходы в And-состояния. В редакторе их добавить нельзя, поэтому никаких дополнительных проверок не делается. Затем каждое Хог-состояние преобразуется по следующим правилам. Пусть S – состояние типа Хог.

- Если S содержит только композитные состояния, то вход и выход добавляются явно.
- Если S содержит простые состояния, и есть хотя бы один переход в S , то в диаграмме UML обязательно должен быть вход, причем единственный. Если есть хотя бы один переход из S , то в диаграмме UML должен быть выход, причем единственный. Иначе выдается ошибка: не определено однозначно, где вход/выход из сложного состояния.
- Если S – параллельный регион (оно вложено в And-состояние), то ищутся добавленные на предыдущем шаге вход и выход в And-состояние, добавляется переход из входа в And-состояние во вход в S и переход из выхода из S в выход And-состояния.
- Если S вложено в другое Хог-состояние, то каждая входящая в S дуга перенаправляется во вход в S , а источником каждой исходящей из S дуги становится выход из S .

Следующий этап преобразования – добавление недостающих входов и выходов. В иерархическом временном автомате возможен переход во вложенное состояние, если он заканчивается во входе этого вложенного состояния, и переход из вложенного состояния в одно из состояний объемлющего автомата, если этот переход начинается в выходе. Все остальные переходы допускаются только в пределах одного композитного состояния. На UML-диаграмме же есть возможность добавить переходы «вглубь», в состояние не являющееся вложенным в текущее. Поэтому перед трансляцией в UPPAAL необходимо добавить все промежуточные входы и выходы. Это делается по следующему алгоритму:

1. Найти в автомате все переходы (A, B) , удовлетворяющие одному из следующих четырех условий:

- a. $(\text{Basic}(B) \vee \text{Exit}(B)) \& \text{not Exit}(A) \& \eta^{-1}(A) \neq \eta^{-1}(B)$
- b. $(\text{Basic}(B) \vee \text{Exit}(B)) \& \text{Exit}(A) \& \eta^{-1}(\eta^{-1}(A)) \neq \eta^{-1}(B)$
- c. $\text{Entry}(B) \& \text{not Exit}(A) \& \eta^{-1}(A) \neq \eta^{-1}(\eta^{-1}(B))$
- d. $\text{Entry}(B) \& \text{Exit}(A) \& \eta^{-1}(\eta^{-1}(A)) \neq \eta^{-1}(\eta^{-1}(B))$

2. Если не найдено ни одного перехода, завершить работу

3. Если $B \in \eta^*(\eta^{-1}(A))$, то добавить новый вход E в $\eta^{-1}(B)$, если $\text{not Entry}(B)$, и в $\eta^{-1}(\eta^{-1}(B))$ – в противном случае. Заменить переход (A, B) на (A, E) и (E, B) , оставив предусловия и присваивания на переходе (A, E) .
4. Иначе, добавить новый выход F в $\eta^{-1}(A)$, если $\text{not Exit}(A)$, и в $\eta^{-1}(\eta^{-1}(A))$ в противном случае. Заменить переход (A, B) на (A, F) и (F, B) , оставив предусловия и присваивания на переходе (A, F) .
5. Вернуться к пункту 1.

На последнем этапе нужно убрать предусловия, присваивания и синхронизации с выходных переходов. Если переход (A, B) заканчивается в выходном состоянии и имеет непустое предусловие, действие или синхронизацию, то в композитное состояние, содержащее A и B , добавляется новое простое состояние E , а вместо перехода (A, B) добавляются переходы (A, E) и (E, B) , причем предусловие, действие и синхронизация перехода (A, B) переносятся на переход (A, E) .

3.2.5 Алгоритм трансляции иерархических временных автоматов в плоские временные автоматы.

В данном разделе приводится алгоритм, не зависящий от средств написания UML-диаграмм и переводящий математическую модель иерархических временных автоматов в модель плоских временных автоматов. Модель плоских временных автоматов при этом максимально приближена к ее реализации в средстве верификации UPPAAL.

За основу алгоритма трансляции взят алгоритм, приведенный в статье [52]. Отличие от алгоритма, приведенного в этой статье, состоит в наличии широкоэмиттерных каналов, отсутствии истории, строгом описании алгоритма в виде псевдокода и ряде технических особенностей трансляции.

В следующих подразделах приводится описание моделей иерархических временных автоматов, после чего в виде псевдокода и пояснений к нему описывается алгоритм трансляции.

3.2.5.1 Модель системы плоских временных автоматов

Модель системы плоских временных автоматов может быть строго описана следующим образом.

Сеть плоских временных автоматов — это система

$$(A, \text{Vars}, \text{Clocks}, \text{Chan}, \text{Chantype}, \text{Chanurg}),$$

где

- $A = (A_1, A_2, \dots, A_n)$ – вектор процессов (объектов, схожих с плоскими временными автоматами, определенными в разделе 2.2.1.1; определение процесса приведено далее),
- Vars – множество переменных, каждая из которых принимает значения из заранее ограниченного с обеих сторон целочисленного отрезка,
- Clocks – множество таймеров,
- Chan – множество каналов синхронизации,
- $\text{Chantype} : \text{Chan} \rightarrow \{h, b\}$ – функция типов каналов,
- $\text{Chanurg} : \text{Chan} \rightarrow \{c, u\}$ – функция срочности каналов.

Каналы, как и в модели иерархических временных автоматов, могут быть широкопередаточными и «точка-точка». Кроме того, каждый канал может быть помечен функцией Chanurg как срочный (*urgent*).

Каждый процесс A_i в векторе процессов является системой

$$(L_i, T_i, \text{Type}_i, \text{Inv}_i, l_i^0),$$

где

- L_i — множество состояний,
- $T_i \subseteq L_i \times (\text{Guard} \times \text{Sync} \times \text{Reset}) \times L_i$ — множество переходов,
- $\text{Type}_i : L_i \rightarrow \{o, u, c\}$ — функция типов состояний,
- $\text{Inv}_i : L_i \rightarrow \text{Invariant}$ — функция инвариантов,
- l_i^0 — начальное состояние.

Множества *Guard*, *Sync*, *Reset*, *Invariant* полностью совпадают с одноименными множествами в модели иерархических временных автоматов.

Посредством функции Type_i каждое состояние процесса может быть помечено как обычное, срочное и сверхсрочное.

3.2.5.2 Описание алгоритма трансляции

Алгоритм принимает на вход корректный иерархический временной автомат

$$\text{HTA} = (S, S_0, \eta, \text{Type}_{\text{hta}}, \text{Var}_{\text{hta}}, \text{clocks}_{\text{hta}}, \text{chan}_{\text{hta}}, \text{Inv}, T, \text{Chantype}_{\text{hta}}).$$

В процессе работы алгоритмом формируется система плоских временных автоматов

$$M = (A, \text{Vars}, \text{Clocks}, \text{Chan}, \text{Chantype}, \text{Chanurg}).$$

По окончании работы алгоритма сформированная система выдаётся в качестве результата.

В псевдокоде, описывающем алгоритм, объекты

$HTA, S, S_0, \eta, Type_{hta}, Var_{hta}, clocks_{hta}, chan_{hta}, Inv, T, Chantype_{hta}$

используются как входные данные, а объекты

$M, A, Vars, Clocks, Chan, Chantype, Chanurg$ —

как глобальные переменные.

Также в упомянутом псевдокоде используется запись $[X]$, где X – произвольный объект (например, состояние иерархического временного автомата или процесса, переменная, таймер, канал, дерево и т.п.). Такая запись подразумевает, что вместо соответствующего объекта подставляется уникальная строка (т.е. строка, отличная от строк, подставляемых вместо всех других объектов). Для удобства записи алгоритма подставляемая вместо объекта строка отождествляется с самим объектом.

Опишем работу алгоритма.

В начале работы алгоритма создается система плоских временных автоматов, не содержащая ни одного процесса и содержащая все каналы и таймеры, соответствующие каналам и таймерам HTA.

В процессе работы алгоритм последовательно обрабатывает все метасостояния HTA от корня к листьям, транслируя каждое метасостояние в отдельный процесс в системе M . Каждый получаемый при трансляции метасостояния процесс содержит состояние «idle», соответствующее неактивности исходного метасостояния в HTA. После добавления состояния «idle» метасостояние обрабатывается в зависимости от его типа — AND или XOR.

Обработка метасостояния типа AND.

При трансляции состояния типа AND в получаемый процесс добавляется состояние «active», соответствующее его активности в HTA. Каждый вход, вложенный в обрабатываемое метасостояние, порождает дополнительные сверхсрочные вершины, число которых равно числу вложенных в метасостояние компонент, также являющихся метасостояниями. Дополнительные вершины упорядочиваются, после чего добавляется дуга из состояния «idle» в первую вершину и дуга из последней вершины в состояние «active». Кроме того, каждая дополнительная соединяется со следующей по порядку.

Добавленные дуги несут синхронизацию, соответствующую активации исходного метасостояния и всех его вложенных компонент, также являющихся метасостояниями.

Для обеспечения корректной деактивации метасостояния в получаемый процесс добавляется дуга, несущая особую синхронизацию.

Обработка метасостояния типа XOR.

Для каждого метасостояния и каждого простого состояния в получаемом процессе заводится состояние, соответствующее активности этого вложенного состояния. Кроме того, для каждого входа, вложенного в каждую вложенную компоненту, в процессе заводится отдельное сверхсрочное метасостояние. Из каждого состояния процесса, соответствующего входу, вложенному во вложенную компоненту, ведется дуга в соответствующую вложенную компоненту, отвечающая активации данной компоненты с задействованием данного входа.

Переходы в НТА, соединяющие два простых состояния, простое состояние со входом, вход с простым состоянием, два входа, простое состояние с выходом или два выхода, при трансляции порождают одну дугу с соответствующими метками. Остальные переходы при трансляции порождают дополнительные сверхсрочные вершины, последовательно соединенные дугами и обеспечивающие деактивацию не только вложенной компоненты, но и всех ее потомков в дереве состояний вплоть до листьев. Более подробно это рассказано в подпараграфе «Обеспечение деактивации метасостояний».

Обеспечение корректного начала работы.

Предполагается, что в начале работы каждый получаемый процесс находится в состоянии «idle». Для перехода системы плоских временных автоматов в состояние, соответствующее начальному состоянию НТА, делается следующее.

До начала основной обработки в систему М добавляется процесс `Global_kickoff`, представляющий собой вершины, последовательно соединенные дугами, несущими особую синхронизацию, обеспечивающую корректное начало работы системы. Все вершины, кроме последней, являются сверхсрочными, что гарантирует немедленное выполнение всех переходов процесса, т.е. немедленное приведение системы в нужное состояние.

В каждом обрабатываемом инициальном метасостоянии, помимо прочего, добавляется дуга, корректно активизирующая данное метасостояние и несущая синхронизацию, парную к одной из синхронизаций дуг процесса `Global_kickoff`.

Трансляция срочных переходов НТА.

Из-за того, что в модели иерархических временных автоматов могут быть срочные переходы, тогда как в модели системы плоских временных автоматов их нет, приходится моделировать срочные переходы средствами результирующей модели.

Для обработки срочных переходов, не несущих синхронизации, в систему М перед началом основной обработки добавляется процесс, состоящий из одного состояния и одной дуги, несущей синхронизацию «Hurry?» по срочному каналу типа «точка-точка». При обработке срочного перехода, не несущего синхронизации, соответствующей дуге процесса приписывается синхронизация «Hurry!».

Каждый канал «с» в множестве каналов НТА порождает в результирующей системе четыре канала для моделирования взаимодействия двух несрочных, несрочного со срочным, срочного с несрочным и двух срочных переходов с синхронизацией по каналу «с» в НТА. При обработке перехода, несущего синхронизацию, вместо соответствующей дуги процесса может порождаться несколько дуг с синхронизацией по различным каналам.

Обеспечение деактивации метасостояний.

При обработке дуги, ведущей из выхода вложенной в метасостояние типа XOR компоненты во вход другой вложенной компоненты или во вложенное простое состояние, необходимо добавить в получаемый процесс дополнительные сверхсрочные вершины и соединить дугами состояния, соответствующие активности вложенных компонент, через эти вершины.

Добавление вершин связано в том, что помимо деактивации вложенной компоненты провести также деактивацию всех ее потомков вплоть до листьев, если они были активны. Для обеспечения такой деактивации делается следующее.

Для каждого выхода вложенной в метасостояние типа XOR компоненты вычисляются все возможные наборы простых состояний ее потомков в дереве состояний вплоть до листьев, из которых потенциально возможен (хотя может и не быть осуществлен, например, из-за несовместности наборов предусловий) одновременный переход с деактивацией всех активных потомков. Для этого строится ориентированное дерево, дуги которого направлены к корню — рассматриваемому выходу, листья которого являются простыми состояниями и остальные вершины которого образуют все выходы, из которых переходом по дугам через другие выходы достигим рассматриваемый.

После построения дерева достаточно перебрать все его поддеревья, удовлетворяющие следующим условиям: если предок выхода, являющегося вершиной дерева, имеет тип XOR, то у выхода ровно один потомок, если же предок имеет тип AND, то у выхода столько потомков, сколько в исходном дереве.

После получения всех поддеревьев у них отбрасываются листья (простые вершины). При этом, возможно, некоторые деревья станут одинаковыми — тогда можно отбросить все

повторения. В получаемый процесс добавляются сверхсрочные вершины, соответствующие выходам в поддереве, и они последовательно соединяются дугами, обеспечивающими корректную деинициализацию. Каждое дерево порождает свою последовательность последовательно соединенных вершин.

Кроме того, в процессе трансляции заводятся переменные, по значению которых можно утверждать, может или не может сработать каждая последовательность деинициализаций метасостояний, и предусловия, соответствующие возможности деинициализации, добавляются дуге, ведущей в первую вершину добавленной для деинициализации последовательности.

3.2.5.3 Формальное описание алгоритма

В данном разделе приведено формальное описание алгоритма трансляции иерархических временных автоматов в автоматы URPAAL, записанное на псевдокоде.

```
// Имя обычного канала
function nonurg_sync(syn) : Sync
begin
  if syn = c! then
    result := '[c]_send!'
  fi
  if syn = c? then
    result := '[c]_rcv'?
  fi
  return result;
end;

// Имя срочного канала
function urg_sync(syn) : Sync
begin
  if syn = c! then
    result := '[c]_rcv!'
  fi
  if syn = c? then
    result := '[c]_send'?
  fi
  return result;
end;

//Добавление канала
procedure add_chan(var M, name, t, urg)
begin
  Chan := Chan U { name };
  Chantype := Chantype U { (name, t) };
  Chanurg := Chanurg U { (name, urg) };
end;

// Добавление состояния
procedure add_location(var P, name, t, inv)
begin
  P.L := P.L U { name };
  P.Type := P.Type U { (name, t) };
  P.Inv := P.Inv U { (name, inv) };
end;
```

```

end;

// Добавление перехода
procedure add_transition(var P, source, target, g, syn, r)
begin
  P.T := P.T U { (source, (g, syn, r), target) };
end;

// Указание начального состояния
procedure set_init(var P, name)
begin
  P.l0 := name;
end;

// Добавление перехода со всеми требуемыми синхронизациями
procedure
  add_sync_transition(var P, source, target, g, syn, r, urg, t)
begin
  switch (urg, syn, t)
  of
    (false, none, ⊥),
      add_transition(P, source, target, g, none, r);
    (false, ch!, h):
      add_transition(P, source, target, g, ch!, r);
      add_transition(P, source, target, g, nonurg_sync(ch!), r);
    (false, ch!, b):
      add_transition(P, source, target, g, ch!, r);
    (false, ch?, h):
      add_transition(P, source, target, g, ch?, r);
      add_transition(P, source, target, g, nonurg_sync(ch?), r);
    (false, ch?, b):
      add_transition(P, source, target, g, ch?, r);
      add_transition(P, source, target, g, '[ch]_urg?', r);
    (true, none, ⊥),
      add_transition(P, source, target, g, 'Hurry!', r);
    (true, ch!, h):
      add_transition(P, source, target, g, '[ch]_urg!', r);
      add_transition(P, source, target, g, urg_sync(ch!), r);
    (true, ch!, b):
      add_transition(P, source, target, g, '[ch]_urg!', r);
    (true, ch?, h):
      add_transition(P, source, target, g, '[ch]_urg?', r);
      add_transition(P, source, target, g, urg_sync(ch?), r);
    (true, ch?, b):
      add_transition(P, source, target, g, ch?, r);
      add_transition(P, source, target, g, '[ch]_urg?', r);
  fo
end;

// Добавление процесса Hurry
procedure add_hurry(var M)
begin
  add_chan(M, 'Hurry', h, u);
  P := empty_process;
  add_location(P, loc);
  set_init(P, loc);
  add_transition(P, loc, loc, true, 'Hurry!?', {});
end;

// Создание инициализирующего процесса
procedure add_global_kickoff(HTA, var M, root)
begin
  S0 := S0 ∩ { B in S | XOR(B) or AND(B) };
  let S0 = { Bi | i = 1..n };

  P := empty_process;

  for i := 1..n

```



```

do
  add_location(P, L_i, c, true);
od

add_location(P, L_(n+1), o, Inv(root));
set_init(P, L_1);

for i := 1..n
do
  add_chan(M, 'init_[B_i]', h, c);
  add_transition(P, L_i, L_(i+1), true, 'init_[B_i]!', {});
od
end;

// Построение множества деревьев выходов
function make_tree_set(HTA, B) : set of state-sets,
  i.e. { {s_(i,1), s_(i,2), ..., s_(i,n_i)} | i = 1..k }
begin
  result := {};
  if EXIT(B) then
    In_set := { v | (v, (g, syn, r, urg), B) in T };
    let In_set = { v_i | i = 1..m };

    temp_set_i := make_tree_set(HTA, v_i) | i = 1..m;

    if AND( $\eta^{-1}$ (B)) then
      forall (s_1, s_2, ..., s_m) such that
        s_i in temp_set_i, i = 1..m
      do
        result := result U { {B} U  $\cup_{i=1..m} s_i$  };
      od
    fi
    if XOR( $\eta^{-1}$ (B)) then
      forall s such that
        exists i : i = 1..m and s in temp_set_i
      do
        result := result U { {B} U s };
      od
    fi
  fi
  return result;
end;

// Добавление синхронизаций для корректного выхода из вложенных метасостояний
procedure add_exit_cascades(HTA, s, var P)
begin
  forall EX in  $\eta$ ( $\eta$ (s)) such that
    exists (EX, (g, syn, r, urg), B) in T : BASIC(B) or ENTRY(B)
  do
    Tree_set := make_tree_set(HTA, EX);
    forall t in Tree_set
    do
      let t = { v_i | i = 1..m };
      let t_leaves = { v | v in t and
        not( exists v_1 in t such that
          (v_1, (g, syn, r), v) in T) };

      Vars := Vars U { 'exit_[E]_ready' | E in t_leaves };

      k := 0;
      forall (EX, (g, syn, r, urg), B) in T such that
        BASIC(B) or ENTRY(B)
      do
        k := k + 1;

        for i := 1..m
        do
          add_location(P, 'exit_cascade_[t]_[k]_[i]', c, true);
        od
      od
    od
  od
end;

```

```

od

add_sync_transition(P,
    '[ $\eta^{-1}$ (EX)]_active_in_[s]',
    'exit_cascade_[t]_[k]_[1]',
    g and  $\forall_{EX \text{ in } t\_leaves}$  'exit_[EX]_ready = 1',
    syn,
    {}),
    urg,
    Chantypehta(syn));

for i := 1..m-1
do
    add_transition(P,
        'exit_cascade_[t]_[k]_[i]',
        'exit_cascade_[t]_[k]_[i+1]',
        true,
        'exit_[ $\eta^{-1}$ (vi)]!',
        {});
od

if BASIC(B) then
    add_transition(P,
        'exit_cascade_[t]_[k]_[p]',
        '[B]_active_in_[s]',
        true,
        'exit_[ $\eta^{-1}$ (vm)]!',
        r);
fi

if ENTRY(B) then
    add_transition(P,
        'exit_cascade_[t]_[k]_[p]',
        '[s]_aux_[ $\eta^{-1}$ (B)]_[B]',
        true,
        'exit_[ $\eta^{-1}$ (vm)]!',
        r);
fi
od
od
od
end;

// Добавление предусловий для корректного выхода из метасостояний
procedure add_exit_assignments(HTA, s, var P)
begin
    forall EX in  $\eta$ (s) such that
        EXIT(EX)
    do
        let In_set = { B |
            (B, (g, syn, r, urg), EX) in T and
            BASIC(B) };

        forall B in In_set
        do
            forall ('[B]_active_in_[s]', (g, syn, r), v) in P.T such that
                not ( exists u : v = '[u]_active_in_[s]' and
                    BASIC(u) and u in In_set )
            do
                Vars := Vars U { 'exit_[EX]_ready' };
                r := r U 'exit_[EX]_ready := 0';
            od

            forall (v, (g, syn, r), '[B]_active_in_[s]') in P.T such that
                not ( exists u : v = '[u]_active_in_[s]' and
                    BASIC(u) and u in In_set )
            do
                Vars := Vars U { 'exit_[EX]_ready' };
                r := r U 'exit_[EX]_ready := 1';
            od
        od
    od
end;

```

```

        od
    od
od
end;

// Обработка состояния типа Хор
procedure process_xor(HTA, s, var P)
begin
    forall B in  $\eta(s)$  such that
        XOR(B) or AND(B) or BASIC(B)
    do
        add_location(P, '[B]_active_in_[s]', o, Inv(B));

        if B in  $S_0$  then
            add_transition('[s]_idle',
                '[B]_active_in_[s]',
                true,
                'init_[s]?',
                {});
        fi

        if XOR(B) or AND(B) then
            forall E in  $\eta(B)$  such that
                ENTRY(E)
            do
                add_chan(M, 'enter_[B]_via_[E]_in_[S]', h, c);
                add_location(P, '[s]_aux_[B]_[E]', c, true);
                add_transition(P,
                    '[s]_aux_[B]_[E]',
                    '[B]_active_in_[s]',
                    true,
                    'enter_[B]_via_[E]_in_[s]!',
                    {});
            od
        fi
    od

    forall (B_1, (g, syn, r, urg), B_2) in T such that
        B_1, B_2 in  $\eta(s)$  and BASIC(B_1) and BASIC(B_2)
    do
        add_sync_transition(P,
            '[B_1]_active_in_[s]',
            '[B_2]_active_in_[s]',
            g,
            syn,
            r,
            urg,
            Chantypehta(syn));
    od

    forall (B, (g, syn, r, urg), E) in T such that
        B,  $\eta^{-1}(E)$  in  $\eta(s)$  and BASIC(B) and ENTRY(E)
    do
        add_sync_transition(P,
            '[B]_active_in_[s]',
            '[s]_aux_[ $\eta^{-1}(E)$ ]_[E]',
            g,
            syn,
            r,
            urg,
            Chantypehta(syn));
    od

    forall (E, (true, none, r, false), B) in T such that
        E, B in  $\eta(s)$  and ENTRY(E) and BASIC(B)
    do
        add_chan(M, 'enter_[s]_via_[E]_in_[ $\eta^{-1}(s)$ ]', h, c);
        add_transition(P,

```

```

        '[s]_idle',
        '[B]_active_in_[s]',
        true,
        'enter_[s]_via_[E]_in_[ $\eta^{-1}(s)$ ]'? ,
        r);
od

forall (E_1, (true, none, r, false), E_2) in T such that
    E_1,  $\eta^{-1}(E_2)$  in  $\eta(s)$  and ENTRY(E_1) and ENTRY(E_2)
do
    add_chan(M, 'enter_[s]_via_[E_1]_in_[ $\eta^{-1}(s)$ ]?', h, c);
    add_transition(P,
        '[s]_idle',
        '[s]_aux_[ $\eta^{-1}(E_2)$ ]_[E_2]',
        true,
        'enter_[s]_via_[E_1]_in_[ $\eta^{-1}(s)$ ]'? ,
        r);
od

forall (B, (g, none, {}, false), EX) in T such that
    B, EX in  $\eta(s)$  and BASIC(B) and EXIT(EX)
do
    add_chan(M, 'exit_[s]', h, c);
    add_transition(P,
        '[B]_active_in_[s]',
        '[s]_idle',
        g,
        'exit_[s]'?,
        {});
od

forall (EX_1, (true, none, {}, false), EX_2) in T such that
    EXIT(EX_1) and EXIT(EX_2) and EX_1 in  $\eta(\eta(s))$  and EX_2 in  $\eta(s)$ 
do
    add_chan(M, 'exit_[s]', h, c);
    add_transition(P,
        '[ $\eta^{-1}(EX_1)$ ]_active_in_[s]',
        '[s]_idle',
        true,
        'exit_[s]'?,
        {});
od

add_exit_cascades(HTA, s, P);
add_exit_assignments(HTA, s, P);
end;

// Обработка состояния типа And
procedure process_and(HTA, s, var P)
begin
    let  $\eta(s) \cap \{B \text{ in } S \mid \text{BASIC}(B) \text{ or } \text{AND}(B) \text{ or } \text{XOR}(B)\} = \{B_i \mid i = 1..n\}$ ;

    add_location(P, '[s]_active', o, Inv(s));

    if s in  $S_0$  then
        add_transition('[s]_idle', '[s]_active', true, 'init_[s]'?, {});
    fi

    forall E in  $\eta(s)$  such that
        ENTRY(E)
    do
        let (E, (g, s, r, u), E_i) in T
            such that E_i in  $\eta(B_i)$  and
                i = 1..n;

        for i := 1..n
            do

```

```

    add_location(P, 'enter_[E]_loc_[B_i]', c, true);
  od

  add_chan(M, 'enter_[s]_via_[E]_in_[ $\eta^{-1}(s)$ ]', h, c);
  add_transition(P,
    '[s]_idle',
    'enter_[E]_loc_[B_1]',
    true,
    'enter_[s]_via_[E]_in_[ $\eta^{-1}(s)$ ]?',
    {});

  for i := 1..n-1
  do
    add_transition(P,
      'enter_[E]_loc_[B_i]',
      'enter_[E]_loc_[B_(i+1)]',
      true,
      'enter_[B_i]_via_[E_i]_in_[s]!',
      {});
  od

  add_transition(P,
    'enter_[E]_loc_[B_n]',
    '[s]_active',
    true,
    'enter_[B_n]_via_[E_n]_in_[s]!',
    U(E, (g, syn, r), B) \in T(X));
  od

  add_chan(M, 'exit_[s]', h, c);
  add_transition(P, '[s]_active', '[s]_idle', true, 'exit_[s]?', {});
end;

// Входная точка программы
begin
  let root = root_s such that root_s in S and  $\eta^{-1}(\text{root}_s) = \{\}$ ;

  A := {};
  Vars := VarHTA;
  Clocks := ClocksHTA;

  Chan := {};

  forall ch in ChanHTA such that
    ChantypeHTA(ch) = h
  do
    add_chan(M, ch, h, c);
    add_chan(M, '[ch]_urg', h, u);
    add_chan(M, '[ch]_recv', h, u);
    add_chan(M, '[ch]_send', h, u);
  od

  forall ch in ChanHTA such that
    ChantypeHTA(ch) = b
  do
    add_chan(M, ch, b, c);
    add_chan(M, '[ch]_urg', b, u);
  od

  add_global_kickoff(HTA, M, root);
  add_hurry(M);

  To_process := {};
  To_process.push_back(root);

  while not_empty(To_process)
  do
    s := pop_front(To_process);

```

```

forall B such that
  B in  $\eta(s)$  and
  (XOR(B) or AND(B))
do
  push(To_process, B);
od

P := empty_process;
add_location(P, '[s]_idle', o, true);
set_init(P, '[s]_idle');

if XOR(s) then process_xor(HTA, s, P);
if AND(s) then process_and(HTA, s, P);

A.push_back(P);
od
end.

```

3.2.6 Средство трансляции диаграмм состояний UML в автоматы URPAAL

3.2.6.1 Проверка синтаксических ограничений

Транслятор позволяет обнаруживать следующие ошибки в диаграммах и выдает предупреждения:

- Начальное значение целочисленной переменной находится вне допустимого диапазона
- Нет названия у состояния или диаграммы. В этом случае в качестве названия состояния или диаграммы вводится уникальное служебное имя UNNAMED_STATE_i, где i – натуральное число.
- Состояние-ссылка на вложенный автомат содержит некорректную или пустую ссылку
- Некорректный синтаксис комментария, предусловия или инварианта. Эти сущности игнорируются. Некорректный синтаксис комментария приводит к тому, что переменные, которые там должны быть описаны, считаются необъявленными, и все предусловия и присваивания с ними также игнорируются.
- Объявлена переменная с тем же именем, что и одна из уже существующих переменных. Повторное описание игнорируется.
- Сигнал отправляется, но нигде не принимается.
- Сигнал принимается, но нигде не отправляется.
- Ошибка в типах (выражение, в разных частях которого разные типы).

3.2.6.2 Структура транслятора

Транслятор реализован в виде набора библиотек на языке Python 2.7. Помимо стандартной библиотеки Python для работы транслятора требуются следующие модули:

1. Antlr3 – библиотека для автоматического создания анализаторов по формальным грамматикам и ее runtime для языка Python. Используется для разбора выражений предусловий и присваиваний.
2. PyUPPAAL – библиотека, позволяющая преобразовывать диаграммы UPPAAL в удобочитаемый вид (без пересечений ребер и перекрывающихся надписей).
3. Xmlparser – библиотека для разбора xml-файлов с диаграммами UML. Из нее удалено все, что не относится к диаграммам состояний, а в оставшуюся часть внесены изменения для поддержки требуемого синтаксиса.

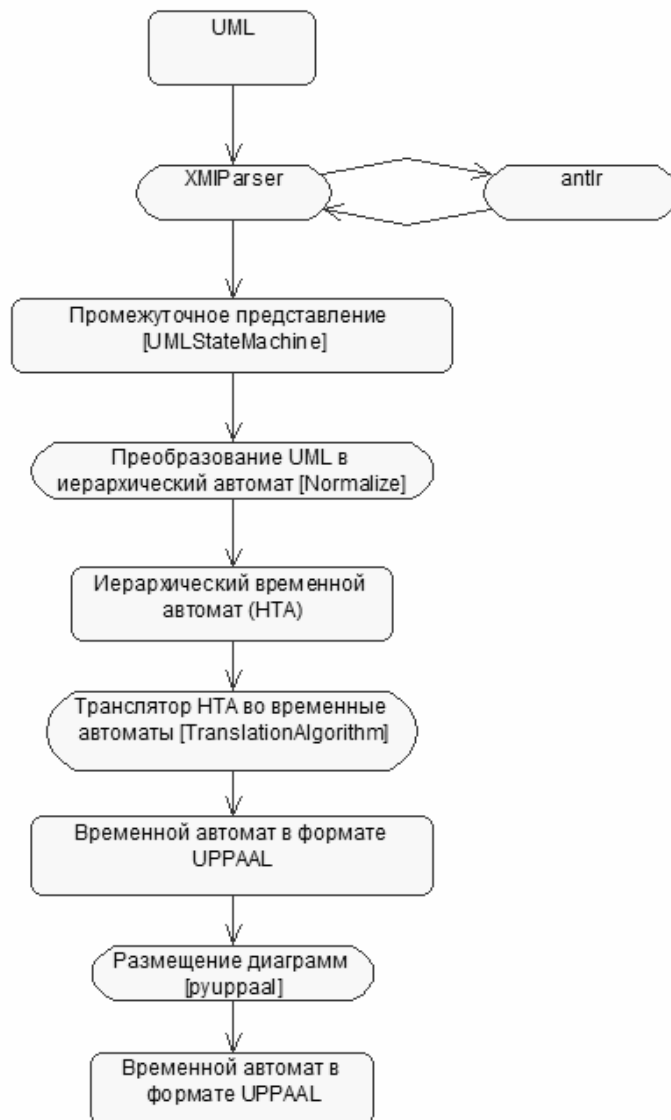


Рисунок 37. Схема работы транслятора.

Общая схема работы транслятора приведена на **рисунке 37**. Программа выполняет следующие действия:

1. Библиотека `xmparser` осуществляет синтаксический разбор поданного на вход `xmi`-файла, преобразуя его в структуры из пакета `UMLStateMachine`. В процессе используются анализаторы, генерируемые `antlr`.
2. Диаграмма, заданная пользователем, преобразуется во временной автомат по алгоритму, описанному в предыдущем разделе. Это действие выполняет функция `Normalize` из одноименного модуля.

3. Преобразованная диаграмма подается на вход функции `main_block` из модуля `TranslationAlgorithm`. Выполняется преобразование по алгоритму, описанному в разделе 3.2.7.3.
4. Полученный в результате предыдущего пункта объект типа `TimedAutomaton` из пакета `UPPAAL` экспортируется в строку в формате XML.
5. Полученная строка в формате XML подается на вход библиотеке `pyUPPAAL`, которая приводит диаграмму в удобочитаемый вид и сохраняет в файл.

Главный исполняемый модуль программы называется `uml2ta.py` и запускается командной строкой вида:

```
python uml2ta.py <xml-файл> <имя главной диаграммы>
```

В той же директории находятся модули `Normalize.py` и `TranslationAlgorithm.py`, реализующие 2 и 3 пункт описанной выше схемы.

Структуры данных автоматов UML и UPPAAL реализованы с помощью, соответственно, пакетов `UMLStateMachine` и `UPPAAL`.

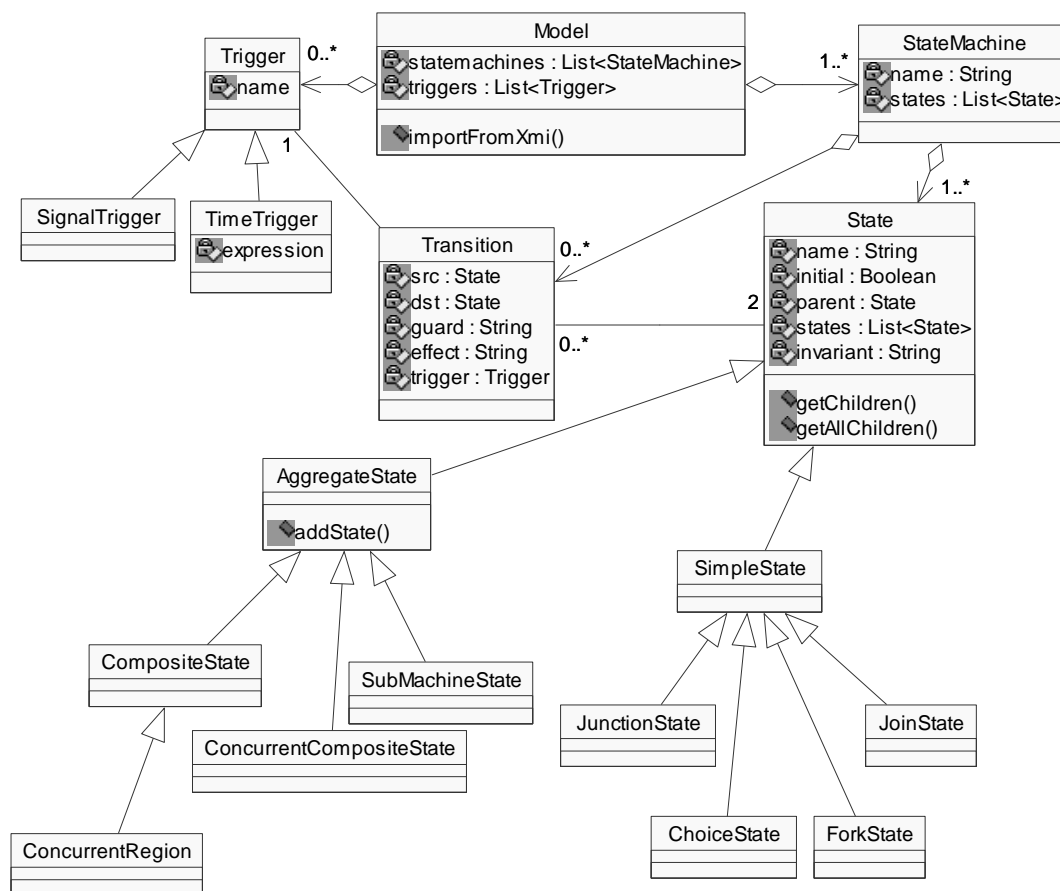


Рисунок 38. Диаграмма классов пакета UMLStateMachine.

На [рисунке 38](#) приведена диаграмма классов пакета UMLStateMachine.

Модель – это набор диаграмм (StateMachine) и триггеров (Trigger), которые могут присутствовать на разных диаграммах. Триггеры используются как для синхронизации (SignalTrigger), так и для таймаутов (TimeTrigger), в последнем случае в триггере присутствует выражение-условие (expression). Диаграмма состоит из состояний (State) и переходов (Transition). Каждый переход содержит два состояния, которое он связывает, предусловие, присваивание и триггер, возможно, пустые. Каждое состояние имеет имя, инвариант, дочерние состояния, ссылку на родительское состояние (или на диаграмму, если состояние самого верхнего уровня), и может быть помечено как начальное. Для состояний введена иерархия классов. Все состояния, наследуемые от AggregateState, могут иметь дочерние состояния, а все состояния, наследуемые от SimpleState – не могут (поле states содержит пустой список). Классы CompositeState и ConcurrentRegion реализуют Хор-

состояния, класс ConcurrentCompositeState – And-состояния, класс SubMachineState – ссылки на вложенные автоматы. Метод `getChildren()` возвращает прямых потомков данного состояния, `getAllChildren()` – все состояния, рекурсивно вложенные в данное.

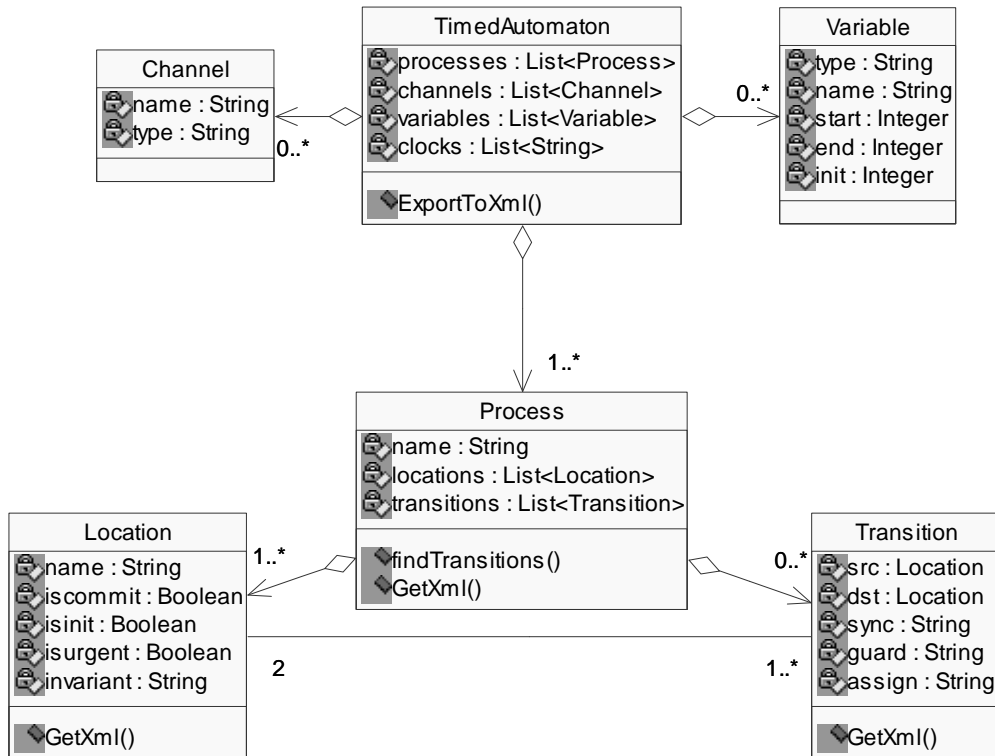


Рисунок 39. Диаграмма классов пакета UPPAAL.

На [рисунке 39](#) приведена диаграмма классов пакета UPPAAL.

Автомат UPPAAL (`TimedAutomaton`) – это список каналов (`Channel`), список переменных (`Variable`), список таймеров (для их хранения достаточно только имен) и список процессов (`Process`). У каналов есть имя и тип (`broadcast/handshake`). Переменные имеют тип (целый/логический), имя, начальное значение и диапазон (для логических он всегда от 0 до 1). Процесс – это список состояний (`Location`) и переходов (`Transition`). Каждое состояние имеет имя и инвариант (по умолчанию пустой) и может быть начальным, срочным или сверхсрочным. У каждого перехода есть начало и конец, а также предусловие, присваивание и синхронизация (по умолчанию пустые). Метод `ExportToXml()` возвращает строку в формате XML, описывающую весь автомат. Методы `GetXml()` в различных классах строят объекты из стандартного модуля `xml.dom.minidom`, которые затем компонуются в один документ XML.

3.2.7 Выводы

В результате проделанной работы удалось создать новую систему моделирования и верификации РВС РВ, описанных в виде диаграмм состояний UML. Главное достоинство разработанной системы состоит в том, что с ее помощью удалось объединить два известных, общедоступных и хорошо зарекомендовавших себя на практике программно-инструментальных средства проектирования и анализа сложных информационных систем: ArgoUML – графическое средство разработки и моделирования диаграмм состояния UML, и UPPAAL – программно-инструментальное средство моделирования и верификации сетей временных автоматов. Благодаря этому, появилась возможность не только использовать диаграммы состояний UML для описания устройства и поведения РВС РВ, но и применять математическую модель временных автоматов и язык темпоральной логики для проверки и строгого доказательства свойств корректности и безопасности поведения проектируемых РВС РВ, описанных при помощи диаграмм состояний UML.

3.3 Средства трансляции средства трансляции UML в исполняемые модели, совместимые со стандартом HLA

В данном разделе описываются средства трансляции диаграммы состояний языка UML в исполняемые модели, совместимые со стандартом HLA. В разделе 3.3.1 приводится общее описание схемы работы средства трансляции. Раздел 3.3.2 содержит описание текстового представления диаграммы состояний языка UML. В разделе 3.3.3 приводится описание шаблонов Cheetah, используемых в средстве. Раздел 3.3.4 содержит описание отображения примитивов языка UML на шаблон исходного кода HLA-совместимой модели. В разделе 3.3.5 описан генератор исходных файлов из SCXML. Вывод об описываемом средстве формируется в разделе 3.3.6.

3.3.1 Общая схема работы средства трансляции

Как было показано в разделе 2 в качестве средства описания моделей выбраны диаграммы состояний UML, расширенные описанием триггеров на языке C++. Поэтому необходимо было разработать средства для генерации исходных кодов модели, описанной в качестве диаграмм состояний. Основным требованием к генерируемой модели является совместимость с HLA.

Для создания и редактирования диаграмм состояний UML использовалось средство ArgoUML [53]. ArgoUML является открытым программным обеспечением и распространяется под лицензией EPL (Eclipse Public License). ArgoUML полностью написан на Java и для работы ему подходит любая операционная система с установленной Java 2 JRE или JDK версии 1.4 или выше.

Однако, в ходе проведения экспериментов было выявлено ряд неудобств при работе с данным средством, в частности отсутствие функций копирования и вставки (copy/paste function) для диаграмм состояний. Также в средстве отсутствует возможность сохранения в формат state chart XML. Следовательно, в дальнейшем необходимо дополнительное исследование средств создания и редактирования диаграмм состояний UML.

На рисунке 40 представлена схема работы генератора кода моделей совместимых со стандартом HLA. Ниже приводится описание основных компонентов средства трансляции.

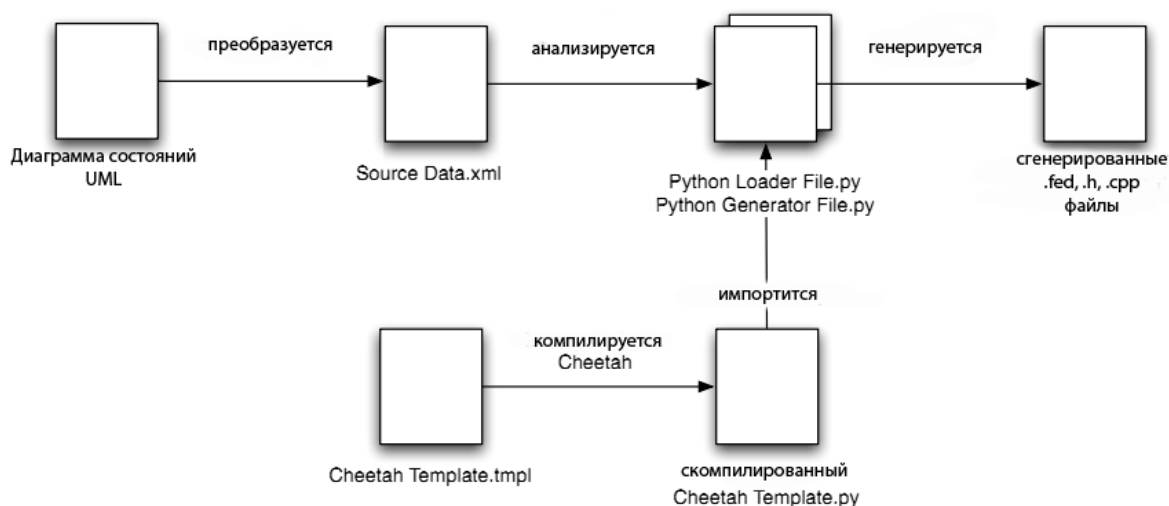


Рисунок 40. Схема работы генератора кода.

3.3.2 Текстовое представление диаграммы состояний UML

В качестве текстового представления диаграмм состояний UML был выбран XML формат – SCXML. Более подробное описание данного формата представлено в следующем подразделе.

3.3.2.1 SCXML

SCXML расшифровывается как State Chart XML [54]. Эта нотация позволяет описывать конечные автоматы в общем виде на основе диаграмм состояний Харела (диаграммы состояний UML). Она основана на XML.

Используя SCXML можно описать различные типы структур конечных автоматов. В качестве примера можно привести такие случаи, как вложенность, параллельность, синхронизация или параллельность подавтоматов.

В соответствии с W3C State Chart XML (SCXML): State Machine Notation for Control Abstraction specification [54], SCXML является событийно - ориентированным языком описания конечных автоматов общего вида, который может быть использован множеством способов включая следующие:

- Как высокоуровневый диалоговый язык управления
- Как метаязык голосовых приложений, где также управлять доступом к базам данных и модулям, реализующим бизнес – логику.
- Как фреймворк конечных автоматов для будущих версий SCXML

На рисунке 41 приведён пример представление диаграммы состояний UML.

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" initial="ready">
  <state id="ready">
    <transition event="watch.start" target="running"/>
  </state>
  <state id="running">
    <transition event="watch.split" target="paused"/>
    <transition event="watch.stop" target="stopped"/>
  </state>
  <state id="paused">
    <transition event="watch.unsplit" target="running"/>
    <transition event="watch.stop" target="stopped"/>
  </state>
  <state id="stopped">
    <transition event="watch.reset" target="ready"/>
  </state>
</scxml>
```

Рисунок 41. Окно редактирования SCXML GUI.

3.3.2.2 XMI to SCXML

Так как ArgoUML использует для описания диаграмм состояний XMI формат [55]. Как было отмечено ранее в ArgoUML отсутствует конвертация в формат SCXML, следовательно, необходимо создание скрипта для перевода XMI представление в SCXML, используемого генератором кода модели. Создание такого скрипта планируется на следующем этапе проекта. На этом этапе данное преобразование проводилось вручную.

3.3.2.3 SCXML GUI

Это графическое приложение для редактирования диаграмм состояний в виде SCXML. Оно написано на основе JGraphX [56] библиотеки и представляет из себя графический редактор диаграмм состояний.

Пример окна работы SCXML GUI показан на [рисунке 42](#).

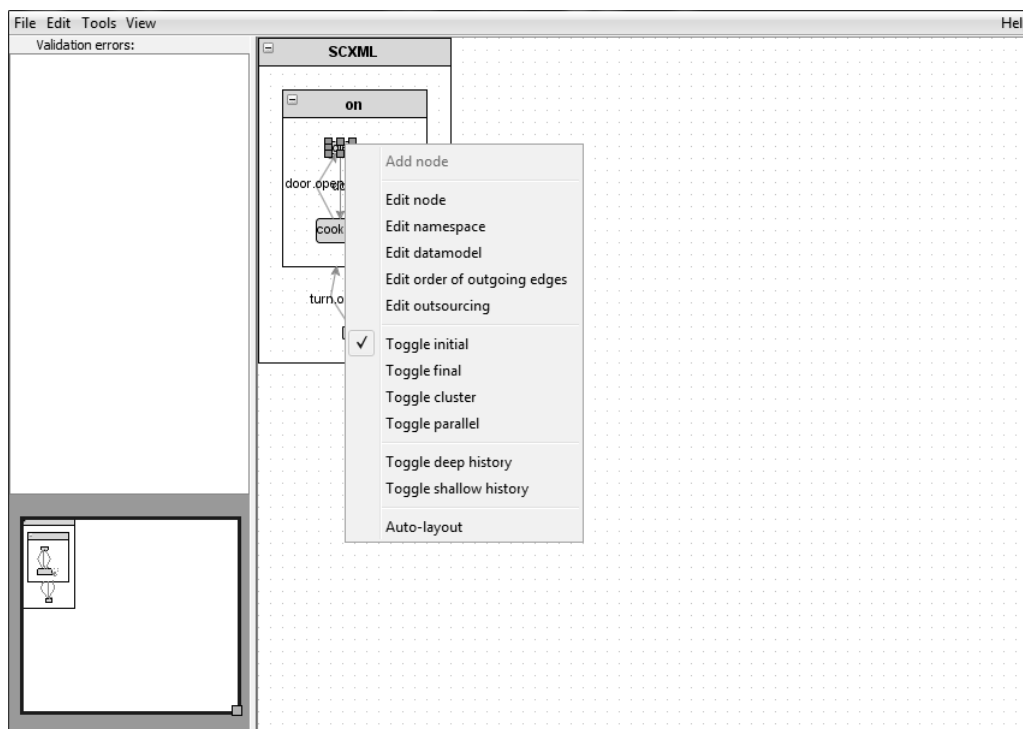


Рисунок 42. Окно редактирования SCXML GUI.

После получения XML файла, нужно сконфигурировать синтаксический анализатор для работы с данным типом файлов. В качестве базового средства использовался синтаксический анализатор проекта gnosis [57], так как данный проект:

- распространяется по свободной лицензии;
- имеет открытый исходный код;
- обладает достаточной функциональностью для перевода XML представление в представление на языке Питон.

Существует несколько подходов для разбора XML файла с помощью языка Питон. Для простоты будет использоваться высокоуровневая (opensource) библиотека работы с XML файлами. Существует несколько открытых библиотек, в основном они различаются в скорости поиска в XML файле. Но так как обычно SCXML файлы в виду своей простоты

небольшого размера для нашего проекта это не принципиально. Следовательно в генераторе будет использоваться DOM синтаксический анализатор XML файлов [58].

Основная задача синтаксического анализатора - это перевод XML файлов в объекты на языке Питон. Далее с помощью библиотеки шаблонов Cheetah генерируются исходные файлы модели.

3.3.3 Шаблоны Cheetah

Cheetah [59] – это библиотека поддержки шаблонов для питона. С помощью данной библиотеки однажды созданный шаблон можно использовать несколько раз с разными файлами. Шаблон компилируется с помощью cheetah компилятора в представление шаблонов на языке Питон. Данный файл используется загрузчиком питона для генерации выходного файла.

Шаблон Cheetah состоит из комбинации выходного текста и кода, похожего на питон. Cheetah это преобработчик на языке Питон предназначенный для получения выходных текстов. Поэтому вместо того чтобы текст отмечать кавычками, как это сделано в питоне и других языках программирования, при использовании cheetah команды на языке Питон выделяются ограничителями.

Далее на [рисунке 43](#) приведен пример шаблона написанного на cheetah.

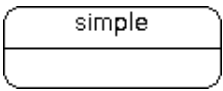
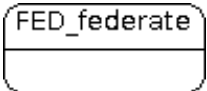


```
## statechart_debugout.tpl - Cheetah template for states and transitions
#for $state in $stateChartXML.state:
$state.id state
  #if $state.__dict__.has_key('transition'):
    #for transition in state.transition:
      on $transition.event event
      transition to $transition.target.next state
    #end for
  #end if
#end for
```

Рисунок 43. Пример шаблона для библиотеки cheetah.

3.3.4 Отображение примитивов языка UML на шаблон исходного кода HLA-совместимой модели

В данном разделе в [таблице 5](#) приведено отображение UML примитивов на XML объекты и шаблоны генерации исходных кодов модели.

Таблица 5. Отображение UML примитивов на XML объекты и шаблоны кода.

Название	Примитив языка UML	XML объект	Шаблон исходного кода HLA-совместимой модели
<i>Обычное состояние модели</i>		<code><state id="simple"></state></code>	Простой шаблон для генерации кода h и cpp файла
<i>Состояние обозначающее федерат</i>		<code><state id="FED_federate"></state></code>	Шаблон федерата для генерации кода h и cpp файла
<i>Обычное событие</i>		<code><transition event="event"></transition></code>	Простое добавление методов в h и cpp файл
<i>Событие, обозначающее взаимодействие федератов</i>		<code><transition event="IRT_interaction"></transition></code>	Добавление взаимодействия федератов в h и cpp файл

3.3.5 Генератор исходных файлов из SCXML

После того как создан SCXML файл, описывающий модель в виде диаграммы состояний, он подаётся на вход генератору исходных кодов, который на основании шаблонов генерирует .h .cpp .fed файлы. Пример шаблона представлен на [рисунке 44](#).

Для генерации кода федератов (с правильными интерфейсами для подключения к RTI) были созданы особые шаблоны: отдельно для .h, .cpp и .fed файлов. Так же потребовалась модификация генератора для создания файла федерации (.fed), так как базовая функциональность не позволяла передать синтаксическому анализатору шаблонов cheetah всю необходимую информацию.

Так же стоит отметить, что некоторые состояния и переходы в диаграмме именовются по особому: состояние обозначающее федерат – именовются начиная с префикса “FED_”, а взаимодействие федератов - “IRT_”.

```

1 ;; ${fedName}
2
3 (Fed
4   (Federation ${fedName})
5   (FedVersion v1516)
6 #for $state in $stateChartXML.state:
7   #if ($state.id.find('FED_') == 0):
8     (Federate "${state.id}", "Public")
9   #end if
10 #end for
11 (Spaces
12 )
13 (Objects
14 )
15
16 (Interactions
17 #for $state in $stateChartXML.state:
18   #if $state.__dict__.has_key('onentry'):
19     #for onentry in $state.onentry:
20       #for step in onentry.function:
21         #if ($step.name.find('IRT_') == 0):
22           (Class InteractionRoot BEST_EFFORT RECEIVE
23             (Class ${step.name}Message RELIABLE TIMESTAMP
24               (Sec_Level "Public")
25               (Parameter ${step.name}X)
26               (Parameter ${step.name}Id)
27             )
28           )
29         #end if
30       #end for
31     #end for
32   #end if
33 #end for
34 )
35 )
36 █

```

Рисунок 44. Пример шаблона для генерации исходных кодов модели.

Главная функция генератора выглядит как показано на [рисунке 45](#):

```

from sm_state_generic_cpp import sm_state_generic_cpp

from loader import Loader
from generator import Generator

if __name__ == '__main__':

    loader = Loader()
    loader.load('../yourfile.xml')
    Перевод XML файла в
    питоновские объекты

    generator = Generator()

    entireChartFiles = {'../statemachine/sm_events.h':
                        sm_events_h(),
                        '../statemachine/sm_stateChart.cpp':
                        sm_stateChart_cpp() }

    for fileName in entireChartFiles:
        print 'Generating:', fileName
        generator.generateChart(loader.stateChartXML,
                               fileName,
                               entireChartFiles[fileName])
    Загрузка шаблонов в
    генератор исходных
    файлов

    stateFiles = {'.h': sm_state_generic_h(),
                 '.cpp': sm_state_generic_cpp() }
    Обозначение - какие типы
    файлов по каким шаблонам
    создавать

    for extension in stateFiles:
        for state in loader.stateChartXML.state:
            fileName = '../statemachine/sm_state_' + state.id + extension
            print 'Generating:', fileName
            generator.generateState(loader.stateChartXML,
                                   state,
                                   fileName,
                                   stateFiles[extension])

```

Рисунок 45. Главная функция генератора.

В результате получаются исходные файлы моделей с интерфейсами для подключения к RTI. Пример представлен на [рисунках 46 и 47](#).

```

2 #ifndef SM_STATE_FED_ARMED_H
3 #define SM_STATE_FED_ARMED_H
4
5 #include "sm_state.h"
6 #include "federate.h"
7
8 class SM_State_FED_Armed : public SM_State, Federate
9 {
10 public:
11
12 //-----
13 SM_State_FED_Armed (std::wstring_federationName, std::wstring_fddName);
14
15 //-----
16 ~SM_State_FED_Armed ();
17
18 //-----
19
20
21 //-----
22
23 void onentry ();
24
25 protected:
26
27 private:
28 //-----
29 void LockDoors();
30
31 //-----
32 // copy constructor not implemented
33 SM_State_FED_Armed( const SM_State_FED_Armed& );
34
35 // assignment operator not implemented
36 SM_State_FED_Armed& operator=( const SM_State_FED_Armed& );
37
38 //-----
39
40
41 void getHandles();
42 void customInit();
43 void loop();
44
45 //-----
46
47 }; // SM_State_FED_Armed
48

```

Рисунок 46. Пример сгенерированного header файла.

```

12
13 //-----
14
15 #include "sm_state_FED_Armed.h" // class SM_State_FED_Armed
16
17
18
19 #include <iostream>
20
21
22
23 //-----
24 //
25 // Class Name : SM_State_FED_Armed
26 //
27 //-----
28
29 SM_State_FED_Armed::SM_State_FED_Armed ()
30 {}
31
32 //-----
33 SM_State_FED_Armed::~SM_State_FED_Armed ()
34 {}
35
36 //-----
37 void SM_State_FED_Armed::onentry ()
38 {
39 LockDoors();
40 }
41
42 //-----
43 void SM_State_FED_Armed::LockDoors ()
44 {
45
46 std::cout << "SM_State_FED_Armed::LockDoors\n";
47
48
49 }
50
51 //-----
52

```

Рисунок 47. Пример сгенерированного src файла.

3.3.6 Вывод

Создание описанного в данном разделе средства трансляции позволяет существенно сократить время разработки моделей совместимых со стандартом HLA, а также не обращать пользователю внимание на создание нужных HLA интерфейсов для подключения к RTI. Также к достоинствам данного генератора можно отнести, что пользователь описывает модели с помощью диаграмм Харела (диаграмм состояний UML).

4 Экспериментальное исследование первой очереди инструментальных средств поддержки анализа и разработки PBC PB

В данном разделе описываются результаты экспериментального исследования методов и инструментальных средств поддержки анализа и разработки PBC PB первой очереди. В разделе 4.1 приведены результаты экспериментов средств работы с трассами. Раздел 4.2 содержит результаты экспериментов со средствами трансляции UML во временные автоматы. В разделе 4.3 приводятся результаты анализа применимости CERTI для моделирования PBC PB. Раздел 4.4 содержит результаты функционального тестирования средства трансляции UML в исполняемые модели, совместимые со стандартом HLA.

4.1 Экспериментальное исследование форматов трасс

В данном разделе приводится методика и результаты экспериментального исследования работы с трассами.

4.1.1 Цели и методика экспериментального исследования

Для проведения экспериментального исследования форматов TRC, OTF и OTF с сжатием (OTFz) были выбраны 5 трасс различных размеров полученных в результате моделирования PBC PB морского назначения. Эти трассы обозначаются следующими именами: Test_2010_06_01, Test_2010_10_28, Pohod88, Kingstown, VeryBigTrace.

Целью экспериментального исследования является сравнение форматов TRC и OTF. Ключевыми количественными параметрами при работе с трассами являются размер трассы и скорость обработки запросов к трассе, по которым и будет производиться сравнение. Для проведения экспериментального исследования необходимы трассы в формате TRC и OTF с одинаковой исходной информацией

Поскольку исследование проводится на основе информации, содержащейся в трассах TRC, в разделе 4.1.2 проводится исследование и сравнение их содержимого, а именно:

- по общим параметрам моделирования (количество состояний, событий, компонентов и так далее),
- по размерам файлов формата TRC и по распределению информации между ними,
- по распределению событий в трассах.

Для сравнения форматов TRC и OTF по размеру необходимо, чтобы идентичная информация была записана в этих форматах, а затем произвести сравнение размеров полученных трасс. Используя средства `trc2txt` и `txt2otf`, описанные в разделе 3.1, можем из трассы TRC получить трассы TXT и OTF с одинаковой содержащейся информацией. Используя средство `txt2otf` с флагом сжатия, установленным в 1, можем получить сжатую трассу в формате OTF (OTFz). Сравнение форматов по размеру получаемых трасс проводится в разделе 4.1.3.

Для удобства анализа трасс обычно используют средства визуализации трасс. Поэтому для сравнения по скорости обработки запросов к трассе сформулируем перечень возможных запросов к трассе со стороны визуализаторов и ситуаций, когда они возникают:

- 1. Запрос «Полное чтение трассы».** Промежуток времени устанавливается в весь временной отрезок, охватываемый трассой. Запрашиваются все события, группы и состояния из этого промежутка. Этот запрос моделирует ситуацию, когда пользователь хочет увидеть диаграмму трассы полностью в одном окне в визуализаторе трасс. Таким образом, данная ситуация сводится к последовательному линейному чтению трассы с начала и до конца.
- 2. Запрос «Последовательный просмотр трассы».** Устанавливается временной промежуток в четверть времени эксперимента. Из этого промежутка запрашиваются все события и состояния. Затем все события, состояния и группы запрашиваются из следующей четверти трассы, затем из третьей и четвертой. Эти запросы моделируют последовательный просмотр трассы от начала до конца. Таким образом, данная ситуация сводится к линейному последовательному чтению фрагментов трассы.
- 3. Запрос «Масштабирование трассы».** Устанавливается промежуток времени в половину времени эксперимента, из него запрашиваются все события и состояния. Затем берется промежуток меньшего размера, полностью лежащий в предыдущем, и все события и состояния берутся из него. Затем промежуток уменьшается еще раз и запрос повторяется. Этот запрос моделирует масштабирование трассы в визуализаторе. Пользователь сначала задает какой-то промежуток времени, затем, увидев временную диаграмму, пытается увидеть участок трассы более подробно. Данный запрос также сводится к линейному поиску по трассе первого фрагмента, а затем к линейному поиску меньшего фрагмента в найденном.
- 4. Запрос «Выборка событий в некотором временном интервале»** заключается в выборке событий определенного типа для некоторого промежутка времени из разных

концов трассы. Данный запрос сводится к последовательному линейному поиску фрагмента трассы, соответствующего временному интервалу, и последовательному просмотру данного фрагмента с целью обнаружения событий.

Таким образом, перечислив основные возможные запросы к трассе, можно сделать вывод, что необходимо измерить и сравнить скорость линейного последовательного чтения трасс в форматах TRC, OTF и OTFz. Данное экспериментальное исследование проводится в разделе 4.1.4. Общие выводы по экспериментальному исследованию трасс приведены в разделе 4.1.5.

4.1.2 Исследование содержимого трасс TRC

В таблице 6 представлены общие параметры трасс, выбранных для экспериментального исследования: количество компонентов в моделируемой системе РВ, время моделирования функционирования системы, количество состояний и событий, произошедших в системе. В таблице 7 представлены размеры файлов трасс в формате TRC и общий размер трасс.

Таблица 6. Общие параметры трасс в формате TRC.

Характеристика	Test_2010_06_01	Test_2010_10_28	Pohod88	Kingstown	VeryBig Trace
<i>Количество событий</i>	54 003	283 712	351 701	2 984 870	33 876 442
<i>Количество изменений состояний</i>	4	42 839	45 317	703 764	9 894 104
<i>Количество компонентов</i>	16	16	96	96	96
<i>Время моделирования (мкс.)</i>	60 000 000	60 000 000	10 000 000	100 000 000	600 000 000

Таблица 7. Размеры трасс в формате TRC.

	Test_2010_06_01	Test_2010_10_28	Pohod88	Kingstown	VeryBigTrace
<i>stand.sta</i>	96 б	1 004 Кб (1.028.136 б)	1,0 Мб (1.087.608 б)	16,1 Мб (16.890.336 б)	226,5 Мб (237.458.496 б)
<i>stand.trc</i>	1,6 Мб (1.728.100 б)	8,7 Мб (9.078.788 б)	10,7 Мб (11.254.436 б)	91,1 Мб (95.515.844 б)	1,0 Гб (1.084.046.148 б)
<i>stand.trcext</i>	269,5 Кб (275.962 б)	772,1 Кб (790.674 б)	2,2 Мб (2.320.198 б)	111,2 Мб (15.272.872 б)	149,7 Мб (156.934.626 б)
<i>stand.dbg</i>	19,5 Кб (19.920 б)	33,5 Кб (34.335 б)	150,9 Кб (154.543 б)	151,0 Кб (154.656 б)	151,0 Кб (154.656 б)
<i>Общий размер трассы</i>	1,88 Мб	10,47 Мб	14,05 Мб	218,55 Мб	1,37 Гб

В **таблицах 8 и 9** приведено распределение информации в процентах и байтах между файлами основной трассы, внешней трассы и трассы состояний в формате TRC.

Таблица 8. Распределение информации между файлами в трассах формата TRC.

Трасса	Test_2010_06_01		Test_2010_10_28		Pohod88	
	байты	%	байты	%	байты	%
<i>stand.trc</i>	1728100	86,23	9078788	83,31	11254436	76,76
<i>stand.trcext</i>	275962	13,77	790674	7,26	2320198	15,82
<i>stand.sta</i>	96	0,00	1028136	9,43	1087608	7,42
<i>stand.trc + stand.trcext + stand.sta</i>	2004158	100,0	10897598	100,0	14662242	100,0

Таблица 9. Распределение информации между файлами в трассах формата TRC.

Трасса	Kingstown		VeryBigTrace	
	байты	%	байты	%
<i>stand.trc</i>	95515844	74,81	1084046148	73,32
<i>stand.trcext</i>	15272872	11,96	156934626	10,61
<i>stand.sta</i>	16890336	13,23	237458496	16,06
<i>stand.trc + stand.trcext + stand.sta</i>	127679052	100,0	1478439270	100,0

Из **таблиц 8 и 9** видно, что:

- Файл основной трассы (*stand.trc*) составляет от 73% до 86% от размера трассы.
- В файле внешней трассы (*stand.trcext*) хранится в среднем не более 15% информации от размера трассы.

- Файл трассы состояний (stand.sta) составляет до 16% от размера трассы.

В **таблице 10** приведено общее количество событий в рассматриваемых трассах формата TRC и количество событий в трассе каждого типа.

Таблица 10. Статистика событий в трассах TRC

	Test_2010_ 06_01	Test_2010_ 10_28	Pohod88	King- stown	VeryBig- Trace
<i>Init</i>	0	0	0	0	0
<i>Pause</i>	0	0	0	0	0
<i>Continue</i>	0	0	0	0	0
<i>Wait</i>	0	0	0	0	0
<i>Flush</i>	6001	21921	27710	170446	1210211
<i>EndFlush</i>	6001	21921	27710	170404	1210211
<i>Arrive</i>	5999	30025	17928	264088	3296219
<i>Update</i>	11999	108757	199889	1241845	12973675
<i>Event</i>	11998	17405	13781	127751	1376127
<i>Stop</i>	1	1	1	1	1
<i>Debug</i>	5999	1	0	0	0
<i>SetState</i>	6	42851	45816	712017	9969215
<i>Interval</i>	0	0	0	0	0
<i>Send</i>	0	16219	10146	179238	2574882
<i>Delivery</i>	5999	22361	8720	119080	1265901
<i>Transfer</i>	0	2250	0	0	0
<i>Finish</i>	0	0	0	0	0
<i>Test</i>	0	0	0	0	0
Всего	54003	283712	351701	2984870	33876442

На приведенных ниже графиках изображенных на **рисунках 48, 49, 50, 51, 52** приведено распределение в процентах по типам событий для каждой из рассматриваемых трасс. События типов *Init*, *Pause*, *Continue*, *Interval*, *Finish*, *Test* не отображаются, поскольку они отсутствуют во всех рассматриваемых трассах.

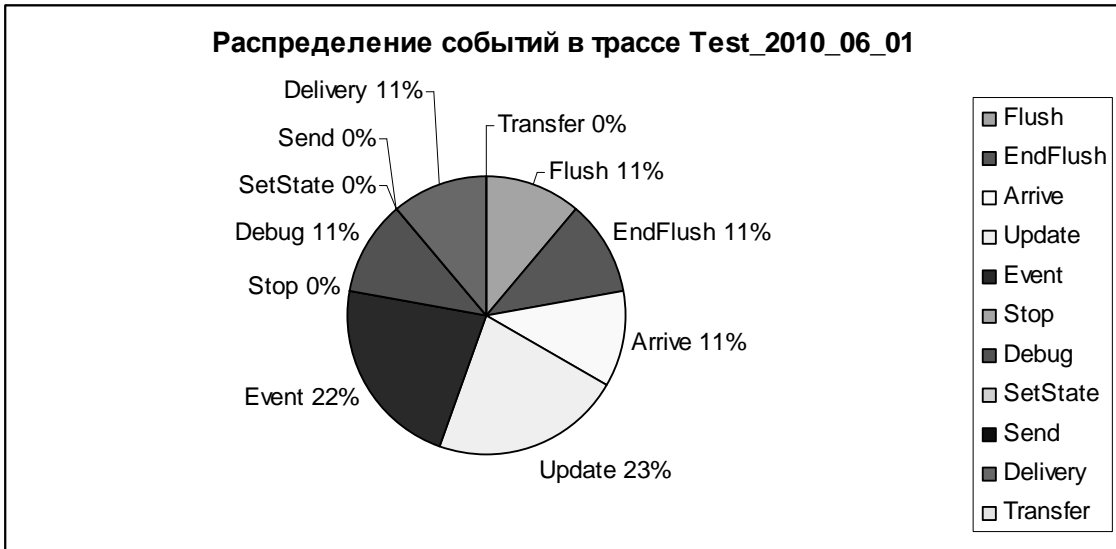


Рисунок 48. Распределение событий в трассе Test_2010_06_01

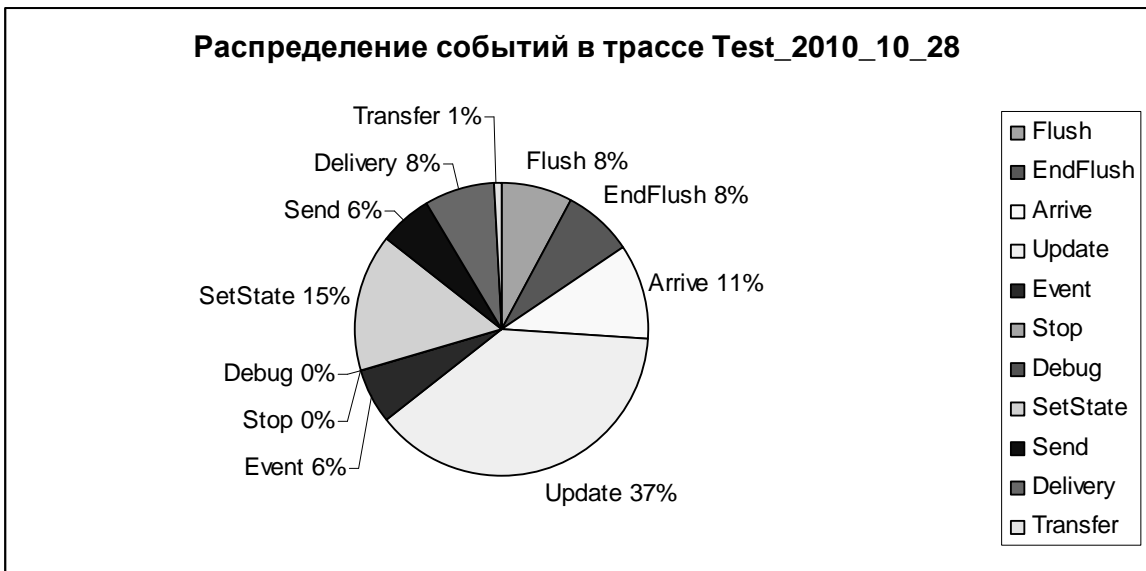


Рисунок 49. Распределение событий в трассе Test_2010_10_28.

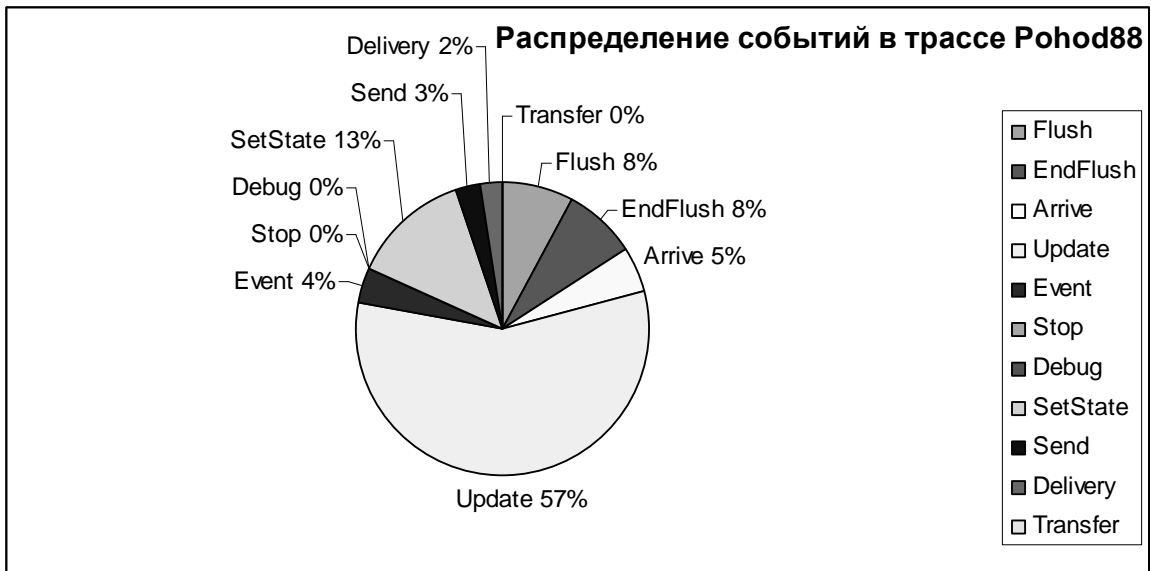


Рисунок 50. Распределение событий в трассе Pohod88.

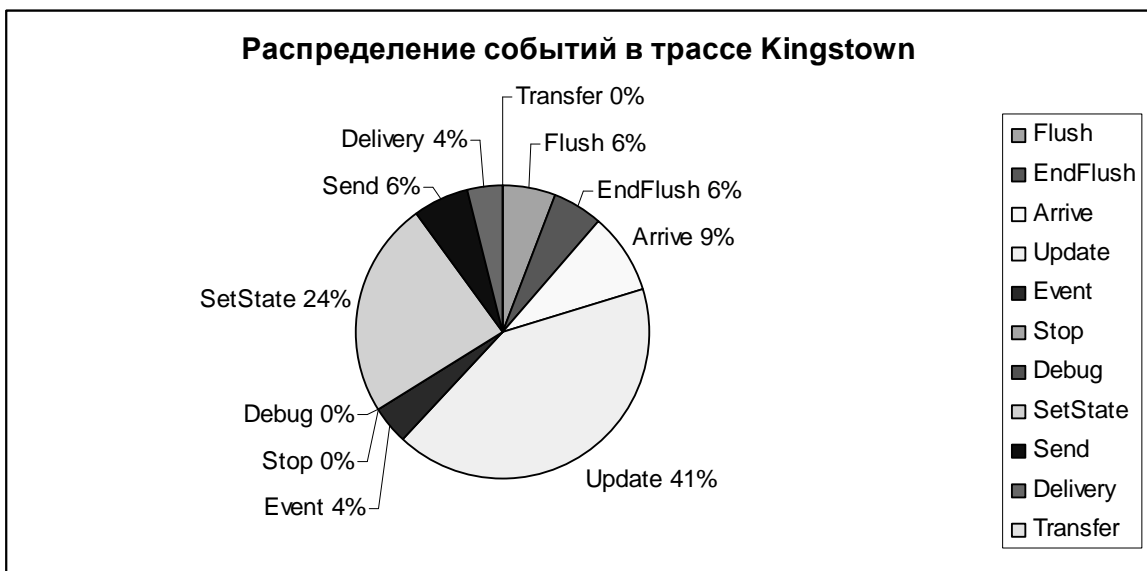


Рисунок 51. Распределение событий в трассе Kingstown.

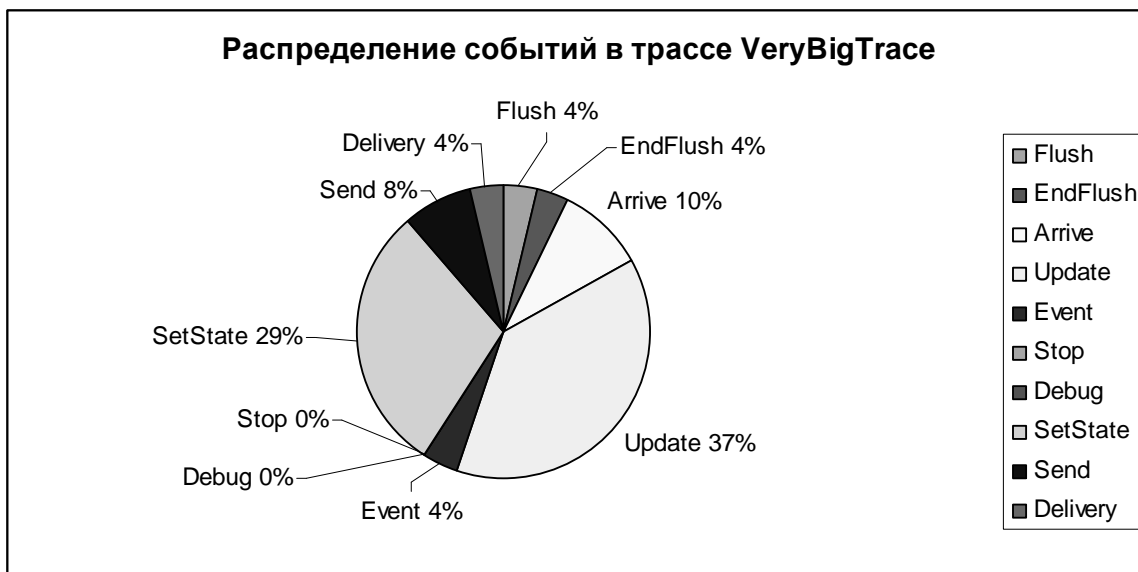


Рисунок 52. Распределение событий в трассе VeryBigTrace.

Приведённые диаграммы показывают, что большинство событий в трассе – это события типа Update (от 23% до 57%). На втором месте - события типа SetState (от 13% до 24%). На третьем месте – события типа Arrive и парные события типа Flush-EndFlush (до 11%). События обмена – не превышают 6-8% от общего количества событий. Это означает, что во-первых, более половины происходящих в РВС РВ относится к изменению значений параметров внутри моделей компонентов РВС РВ, а во-вторых, что изменения режимов функционирования РВС РВ внутри компонентов РВС РВ происходит реже, чем обмены сообщениями между компонентами моделируемой РВС РВ.

4.1.3 Исследование размера трасс различных форматов

С помощью средств trc2txt и txt2otf было произведено конвертирование рассматриваемых трасс в формате TRC в форматы TXT, OTF и OTF с сжатием (OTFz). Результаты приведены в **таблицах 11 и 12.**

Таблица 11. Сравнение размеров трасс в форматах TRC, TXT, OTF и OTFz.

Трасса	Test_2010_06_01	Test_2010_10_28	Pohod88
Трасса в формате TRC			
<i>stand.sta</i>	96 б	1 004 Кб	1,0 Мб
<i>stand.trc</i>	1,6 Мб	8,7 Мб	10,7 Мб
<i>stand.trcext</i>	269,5 Кб	772,1 Кб	2,2 Мб
<i>stand.dbg</i>	19,5 Кб	33,5 Кб	150,9 Кб
<i>всего:</i>	1,88 Мб	10,47 Мб	14,05 Мб
Трасса в формате TXT			
<i>states.txt</i>	80 б	859 Кб	861 Кб
<i>events.txt</i>	2,06 Мб	10,44 Мб	14,0 Мб
<i>dbgout.txt</i>	31 Кб	51 Кб	345 Кб
<i>всего:</i>	2,09 Мб	11,25 Мб	15,2 Мб
Трасса в формате OTF-acsii			
<i>otftrace.otf</i>	1 Кб	1 Кб	1 Кб
<i>otftrace.0.def</i>	2 Кб	3 Кб	7 Кб
<i>otftrace.1.events</i>	331 Кб	772 Кб	774 Кб
<i>otftrace.0.marker</i>	2,4 Мб	12,1 Мб	15,7 Мб
<i>всего:</i>	2,7 Мб	12,9 Мб	16,5 Мб
Трасса в формате OTF-z			
<i>otftrace.otf</i>	1 Кб	1 Кб	1 Кб
<i>otftrace.0.def.z</i>	1 Кб	2 Кб	3 Кб
<i>otftrace.1.events.z</i>	82,8 Кб	145 Кб	200 Кб
<i>otftrace.0.marker.z</i>	0,6 Мб	1,86 Мб	3,38 Мб
<i>всего:</i>	0,68 Мб	2,0 Мб	3,58 Мб

Таблица 12. Сравнение размеров трасс в форматах TRC, TXT, OTF и OTFz.

	Kingstown	VeryBigTrace
Трасса в формате TRC		
<i>stand.sta</i>	16,1 Мб	226,5 Мб
<i>stand.trc</i>	91,1 Мб	1,0 Гб
<i>stand.trcext</i>	111,2 Мб	149,7 Мб
<i>stand.dbg</i>	151,0 Кб	151,0 Кб
<i>всего:</i>	218,55 Мб	1,37 Гб
Трасса в формате TXT		
<i>states.txt</i>	13,5 Мб	198,0 Мб
<i>events.txt</i>	217,2 Мб	1,38 Гб
<i>dbgout.txt</i>	345 Кб	345 Кб
<i>всего:</i>	231,2 Мб	1,58 Гб
Трасса в формате OTF-acsii		
<i>otftrace.otf</i>	1 Кб	1 Кб
<i>otftrace.0.def</i>	7 Кб	7 Кб
<i>otftrace.1.events</i>	12,4 Мб	181,0 Мб
<i>otftrace.0.marker</i>	135,0 Мб	1,57 Гб
<i>всего:</i>	147,0 Мб	1,74 Гб
Трасса в формате OTF-z		
<i>otftrace.otf</i>	1 Кб	1 Кб
<i>otftrace.0.def.z</i>	3 Кб	3 Кб
<i>otftrace.1.events.z</i>	2,82 Мб	39,7 Мб
<i>otftrace.0.marker.z</i>	38,02 Мб	189,1 Мб
<i>всего:</i>	40,84 Мб	228,8 Мб

Соотношение размеров экспериментальных трасс в форматах TRC, TXT, OTF и OTFz приведено на графиках на рисунках 53 и 54.

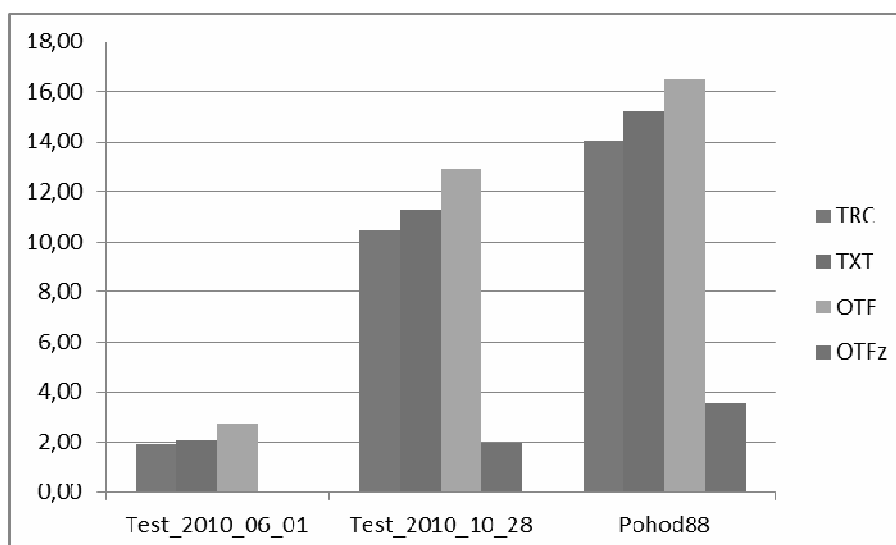


Рисунок 53. Соотношение экспериментальных трасс в разных форматах (в Мб.).

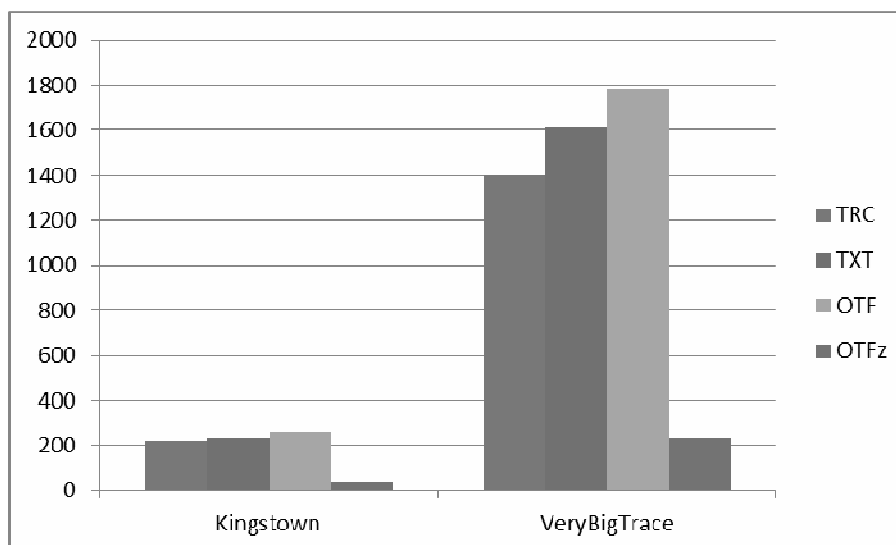


Рисунок 54. Соотношение экспериментальных трасс в разных форматах (в Мб.).

Из графиков 53 и 54 видно, что трасса в формате OTF практически всегда превосходит по размеру трассы в форматах TRC и TXT, однако в трассы в формате OTFz сжатием имеют наименьший размер и также могут читаться визуализаторами (Vite, Vampir) без дополнительных действий по их распаковке. В таблицах 13 и 14 приведены значения коэффициента сжатия трасс в форматах TXT, OTF и OTFz по сравнению с форматом TRC и видно, что:

- Коэффициент сжатия для трасс в формате TXT составляет от 0,87 до 0,95.

- Коэффициент сжатия для трасс в формате OTF составляет от 0,70 до 0,85.
- Формат OTFz позволяет сжимать трассу в формате TRC примерно в 3-6 раз.

Таблица 13. Коэффициент сжатия трасс в форматах TRC, TXT, OTF и OTFz.

	Test_2010_06_01		Test_2010_10_28		Pohod88	
	Мб	Коэффициент сжатия	Мб	Коэффициент сжатия	Мб	Коэффициент сжатия
<i>TRC</i>	1,88	1,00	10,47	1,00	14,05	1,00
<i>TXT</i>	2,09	0,90	11,25	0,93	15,2	0,92
<i>OTF</i>	2,70	0,70	12,9	0,81	16,5	0,85
<i>OTFz</i>	0,68	2,76	2	5,24	3,58	3,92

Таблица 14. Коэффициент сжатия трасс в форматах TRC, TXT, OTF и OTFz.

	Kingstown		VeryBigTrace	
	Мб	Коэффициент сжатия	Мб	Коэффициент сжатия
<i>TRC</i>	218,55	1,00	1402,88	1,00
<i>TXT</i>	231,2	0,95	1617,92	0,87
<i>OTF</i>	258,6	0,85	1781,76	0,79
<i>OTFz</i>	40,84	5,35	228,8	6,13

Таким образом, из проведённого исследования форматов TRC, OTF и OTFz по размеру получаемой трассы наиболее предпочтительным является формат OTFz с сжатием.

4.1.4 Последовательный поиск событий в трассе

Наиболее частый и простейший запрос к трассе заключается в поиске некоторого события, обладающего определенным свойством, например, поиск события с определенной временной меткой. На базе таких запросов строятся более сложные, например, найти группу событий определенного типа, найти фрагмент трассы в определенном временном диапазоне и так далее. Для поддержки таких запросов к трассе используется алгоритм линейного, последовательного поиска. Он имеет невысокую скорость работы, но достаточно прост. Принцип работы заключается в том, что каждый элемент (событие) трассы сравнивается с ключом поиска (эталонным событием) на случай совпадения. В случае нахождения ключа поиска в трассе, программный модуль, реализующий данный алгоритм, выводит соответствующее сообщение и сообщает индекс события в трассе.

Для исследования скорости (времени) последовательного линейного чтения трасс были разработаны средства ResearchModul, TraceAnalyzerTRC и TraceAnalyzerOTF для

форматов TRC и OTF соответственно. Средство ResearchModul по соответствующим трассам в формате TXT формирует перечень запросов событий к трассам в форматах TRC и OTF, время поиска которых измеряется с помощью средств TraceAnalyzerTRC и TraceAnalyzerOTF.

Ниже приведены результаты исследования линейного поиска в трассах в формате TRC и OTF(и OTFz). Для каждой трассы с помощью модуля ResearchModul были определены 4 события, расположенные на отметках 25%, 50%, 75%, 100% от длины трассы. Далее был произведен замер времени линейного поиска этих событий и 100 итерациям в автоматическом режиме с помощью модуля TraceAnalyzerTRC и TraceAnalyzerOTF.

В **таблицах 15, 18, 21, 24, 27** приведена информация о событиях, поиск которых осуществляется в трассах Test_2010_06_01, Test_2010_10_28, Pohod88, Kingstown, VeryBigTrace соответственно. В **таблицах 16, 19, 22, 25, 28** приведены результаты замеров времени поиска событий 1-4 по 100 итерациям – минимальное (min), среднее(av) и максимальное (max) время поиска событий в трассах в формате TRC, OTF и OTFz. Поиск событий осуществляется соответственно по временной метке (time), типу события (type) и номеру компонента (proc). Время чтения трасс в формате OTF и OTFz в процентах относительно времени чтения трассы в формате TRC приведено для всех трасс в **таблицах 17, 20, 23, 26, 29**.

Последовательный поиск в трассе Test_2010_06_01

Таблица 15. События для поиска в трассе Test_2010_06_01.

Test_2010_06_01					
		1	2	3	4
Событие	<i>Event №</i>	13500	27001	40502	54003
	<i>type</i>	DELIVERY	EVENT	DEBUG	STOP
	<i>time</i>	14990184	29990179	44990176	60000000
	<i>proc</i>	3	3	3	0

Таблица 16. Время прохождения фрагментов трассы.

Test_2010_06_01 (100 итераций)					
Формат	Показатель	1	2	3	4
TRC	<i>min time</i>	0,002724	0,005556	0,008721	0,011720
	<i>av time</i>	0,003803	0,007046	0,010928	0,013766
	<i>max time</i>	0,006706	0,011797	0,023998	0,018736
OTF	<i>min time</i>	0,002611	0,006161	0,00883	0,011355
	<i>av time</i>	0,003320	0,006412	0,009311	0,012266
	<i>max time</i>	0,004689	0,009401	0,012962	0,017017
OTFz	<i>min time</i>	0,003012	0,005668	0,008456	0,010904
	<i>av time</i>	0,003404	0,006553	0,009343	0,012321
	<i>max time</i>	0,003897	0,007346	0,010569	0,014107

Таблица 17. Относительное среднее время прохождения трассы (в %).

Test_2010_06_01 (av time в %)				
Формат	1	2	3	4
TRC	100,0	100,0	100,0	100,0
OTF	87,3	91,0	85,2	88,1
OTFz	89,5	93,3	85,5	89,5

Последовательный поиск в трассе Test_2010_10_28

Таблица 18. События для поиска в трассе Test_2010_10_28.

Test_2010_10_28					
Событие		1	2	3	4
	<i>Event №</i>	70928	141856	212784	283712
	<i>type</i>	UPDATE	FLUSH	UPDATE	STOP
	<i>time</i>	21720000	34480000	47240000	60000000
	<i>proc</i>	9	1	1	0

Таблица 19. Время прохождения фрагментов трассы.

Test_2010_10_28 (100 итераций)					
Формат	Показатель	1	2	3	4
TRC	<i>min time</i>	0,014872	0,029888	0,048001	0,06199
	<i>av time</i>	0,018382	0,035997	0,054295	0,071692
	<i>max time</i>	0,024609	0,053032	0,066666	0,105337
OTF	<i>min time</i>	0,014746	0,02772	0,041982	0,058743
	<i>av time</i>	0,015680	0,031317	0,047074	0,063806
	<i>max time</i>	0,019574	0,039113	0,055442	0,074296
OTFz	<i>min time</i>	0,014235	0,027432	0,044813	0,056849
	<i>av time</i>	0,016084	0,031713	0,049517	0,064236
	<i>max time</i>	0,018416	0,035551	0,056014	0,073550

Таблица 20. Относительное среднее время прохождения трассы (в %).

Test_2010_10_28 (av time в %)				
Формат	1	2	3	4
TRC	100,0	100,0	100,0	100,0
OTF	85,3	87,0	86,7	89,0
OTFz	87,5	88,1	91,2	89,6

Последовательный поиск в трассе Pohod88

Таблица 21. События для поиска в трассе Pohod88.

Pohod88					
Событие		1	2	3	4
	<i>Event №</i>	87925	175850	263775	351701
	<i>type</i>	SETSTATE	SETSTATE	SETSTATE	STOP
	<i>time</i>	2440010	4989878	7479637	10000000
	<i>proc</i>	24	24	9	0

Таблица 22. Время прохождения фрагментов трассы.

Pohod88 (100 итераций)					
Формат	Показатель	1	2	3	4
TRC	<i>min time</i>	0,017489	0,034666	0,051655	0,069239
	<i>av time</i>	0,019749	0,039437	0,059419	0,077069
	<i>max time</i>	0,025542	0,047428	0,135977	0,095499
OTF	<i>min time</i>	0,015059	0,032968	0,048951	0,066074
	<i>av time</i>	0,017438	0,034902	0,055497	0,070133
	<i>max time</i>	0,023918	0,042304	0,066322	0,083297
OTFz	<i>min time</i>	0,015293	0,031111	0,048881	0,061113
	<i>av time</i>	0,017280	0,035967	0,054012	0,069054
	<i>max time</i>	0,019786	0,040318	0,061098	0,079067

Таблица 23. Относительное среднее время прохождения трассы (в %).

Pohod88 (av time в %)				
Формат	1	2	3	4
TRC	100,0	100,0	100,0	100,0
OTF	88,3	88,5	93,4	91,0
OTFz	87,5	91,2	90,9	89,6

Последовательный поиск в трассе Kingstown

Таблица 24. События для поиска в трассе Kingstown.

Kingstown					
Событие		1	2	3	4
	<i>Event №</i>	746217	1492435	2238652	2984870
	<i>Type</i>	SETSTATE	SETSTATE	SETSTATE	STOP
	<i>Time</i>	35932374	57297259	78630815	100000000
	<i>Proc</i>	11	34	11	0

Таблица 25. Время прохождения фрагментов трассы.

Kingstown (100 итераций)					
Формат	Показатель	1	2	3	4
TRC	<i>min time</i>	0,167065	0,334217	0,529633	0,690258
	<i>av time</i>	0,185442	0,374599	0,563527	0,741026
	<i>max time</i>	0,212038	0,414445	0,606784	0,879618
OTF	<i>min time</i>	0,184289	0,33686	0,528686	0,708447
	<i>av time</i>	0,165229	0,333019	0,527461	0,664700
	<i>max time</i>	0,201274	0,414839	0,632046	0,815367
OTFz	<i>min time</i>	0,145243	0,284497	0,462563	0,584981
	<i>av time</i>	0,164116	0,328898	0,511119	0,660995
	<i>max time</i>	0,187913	0,368695	0,578178	0,756839

Таблица 26. Относительное среднее время прохождения трассы (в %).

Kingstown (av time в %)				
Формат	1	2	3	4
TRC	100,0	100,0	100,0	100,0
OTF	89,1	88,9	93,6	89,7
OTFz	88,5	87,8	90,7	89,2

Последовательный поиск в трассе VeryBigTrace

Таблица 27. События для поиска в трассе VeryBigTrace.

VeryBigTrace					
Событие		1	2	3	4
	<i>Event №</i>	8469110	16938221	25407331	33876442
	<i>Type</i>	EVENT	UPDATE	SETSTATE	STOP
	<i>Time</i>	160105901	306711566	453335165	600000000
	<i>Proc</i>	5	20	79	0

Таблица 28. Время прохождения фрагментов трассы.

Test_2010_06_01 (100 итераций)					
Формат	Показатель	1	2	3	4
<i>TRC</i>	<i>min time</i>	1,007288	2,055112	3,226732	4,044420
	<i>av time</i>	1,140289	2,303422	3,465147	4,556594
	<i>max time</i>	1,303829	2,548437	3,731136	5,408801
<i>OTF</i>	<i>min time</i>	1,00012	2,00527	3,207587	4,020641
	<i>av time</i>	1,039944	2,015494	3,146353	4,023473
	<i>max time</i>	1,402163	2,723701	3,743056	4,815367
<i>OTFz</i>	<i>min time</i>	1,021120	2,002040	3,092587	4,020641
	<i>av time</i>	1,050206	2,089204	3,160214	4,078152
	<i>max time</i>	1,279163	2,723701	3,629656	4,701367

Таблица 29. Относительное среднее время прохождения трассы (в %).

Test_2010_06_01 (av time в %)				
Формат	1	2	3	4
<i>TRC</i>	100,0	100,0	100,0	100,0
<i>OTF</i>	91,2	87,5	90,8	88,3
<i>OTFz</i>	92,1	90,7	91,2	89,5

Из таблиц 18, 21, 23, 26, 29 видно, что по скорости чтения трасс в форматах *OTF* и *OTFz* примерно одинаковы, из *OTF* чтение осуществляется несколько быстрее. Скорость чтения из трасс в формате *OTF* на 9-12% выше скорости чтения из трасс в формате *TRC*.

4.1.5 Выводы

Таким образом, в результате экспериментального исследования было проведено сравнение трасс в форматах *TRC*, *OTF* и *OTFz* и получены следующие результаты:

- Скорость чтения из трасс в формате *OTF*, *OTFz* оказалась на 9-12% выше скорости чтения из трасс в формате *TRC*.
- Коэффициент сжатия для трасс в формате *OTF*, использующий кодировку *ASCII* составляет от 0,70 до 0,85.
- Формат *OTFz* позволяет сжимать трассу в формате *TRC* примерно в 3-6 раз.

Для дальнейшего использования в рамках данного НИР предпочтительным является формат *OTF* с сжатием, то есть *OTFz*.

4.2 Экспериментальное исследование средств трансляции UML во временные автоматы

4.2.1 Функциональное тестирование алгоритма

Для проверки правильности работы алгоритма трансляции диаграмм состояний UML в автоматы UPPAAL был создан набор функциональных тестов, задействующих различные этапы алгоритма. Далее приведены исходные диаграммы состояний UML, эквивалентные им системы автоматов UPPAAL, созданные вручную, и диаграммы UPPAAL, сгенерированные транслятором.

В первом тесте (см. **рисунок 55**) присутствуют базовые элементы диаграмм состояния UML: простые и композитные состояния, переходы между ними.

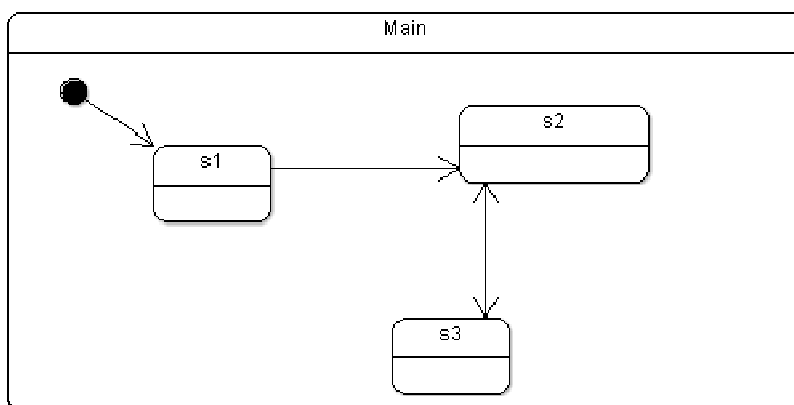


Рисунок 55. Первый тест

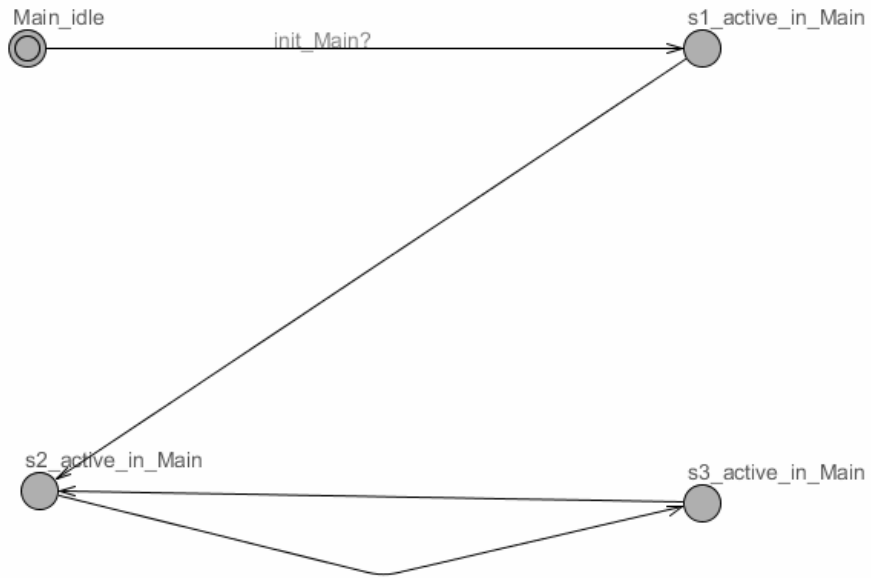


Рисунок 56. Ожидаемая диаграмма UPPAAL для первого теста.

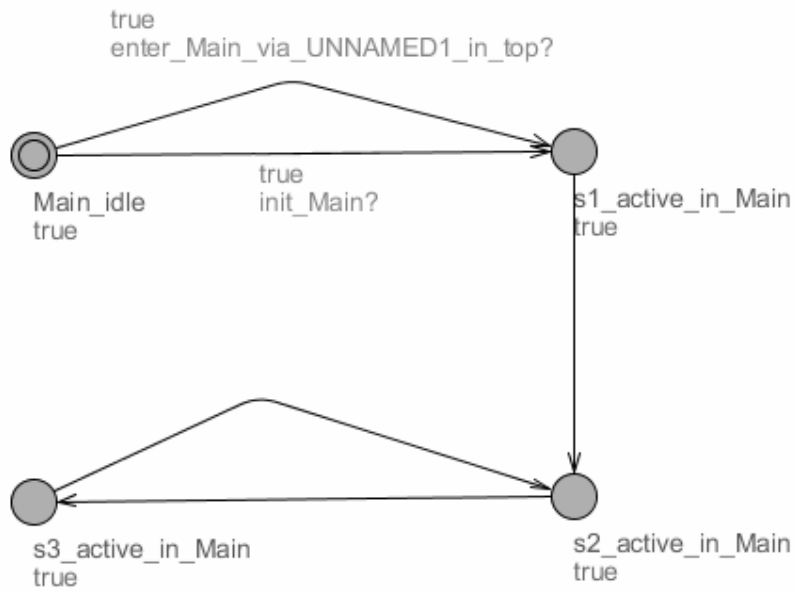


Рисунок 57. Полученная диаграмма UPPAAL для первого теста.

Во втором тесте (см. **рисунок 58**) присутствует композитное состояние типа And

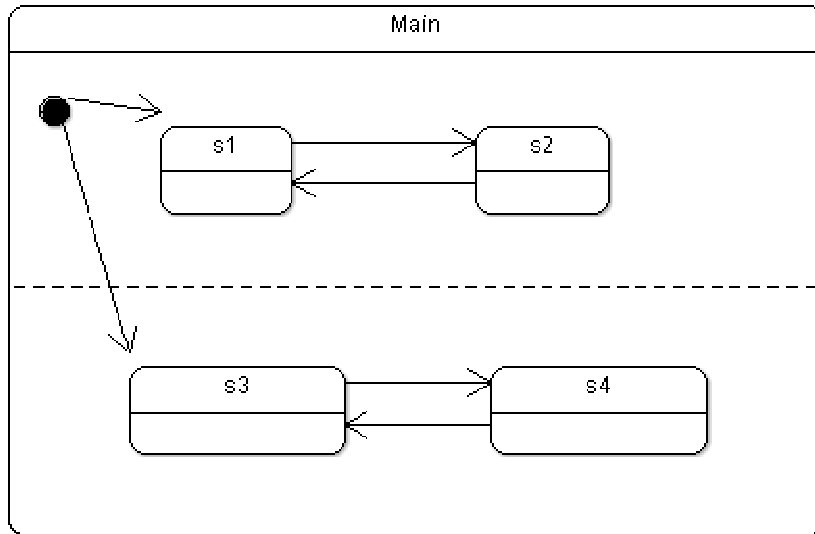


Рисунок 58. Второй тест.

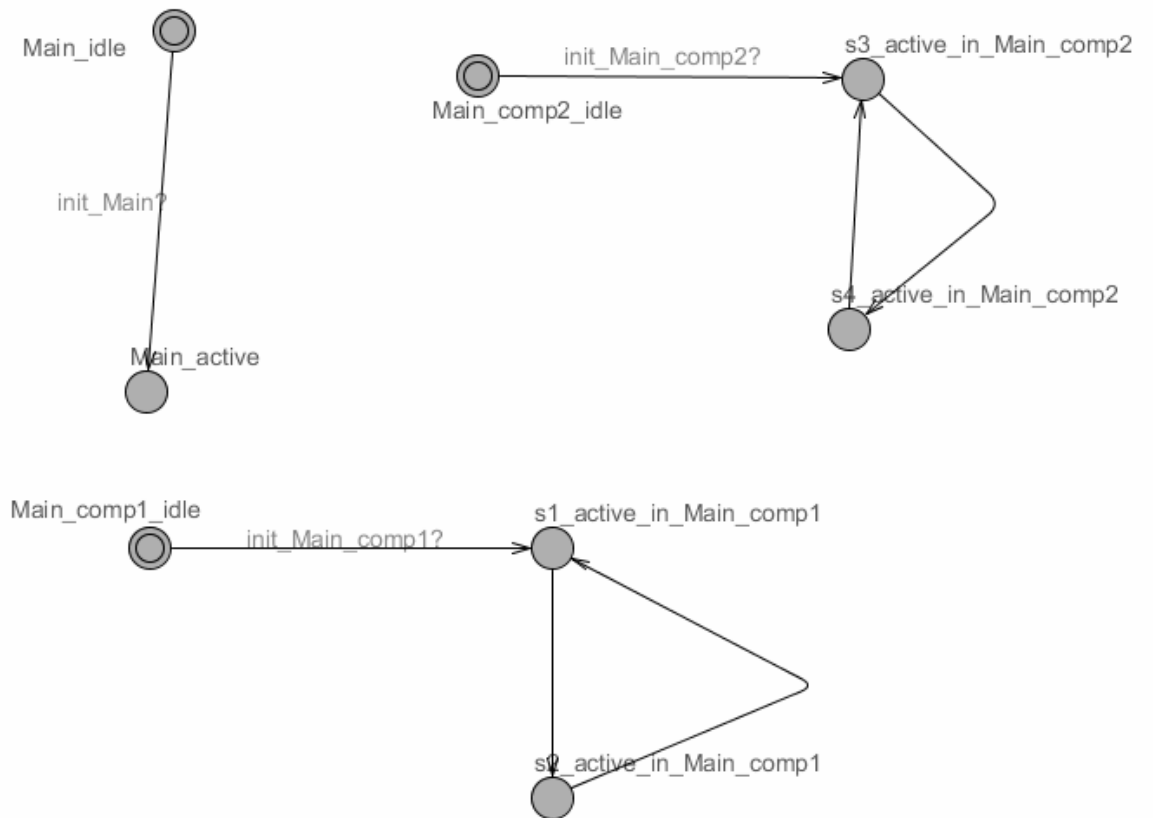


Рисунок 59. Ожидаемая диаграмма UPPAAL для второго теста.

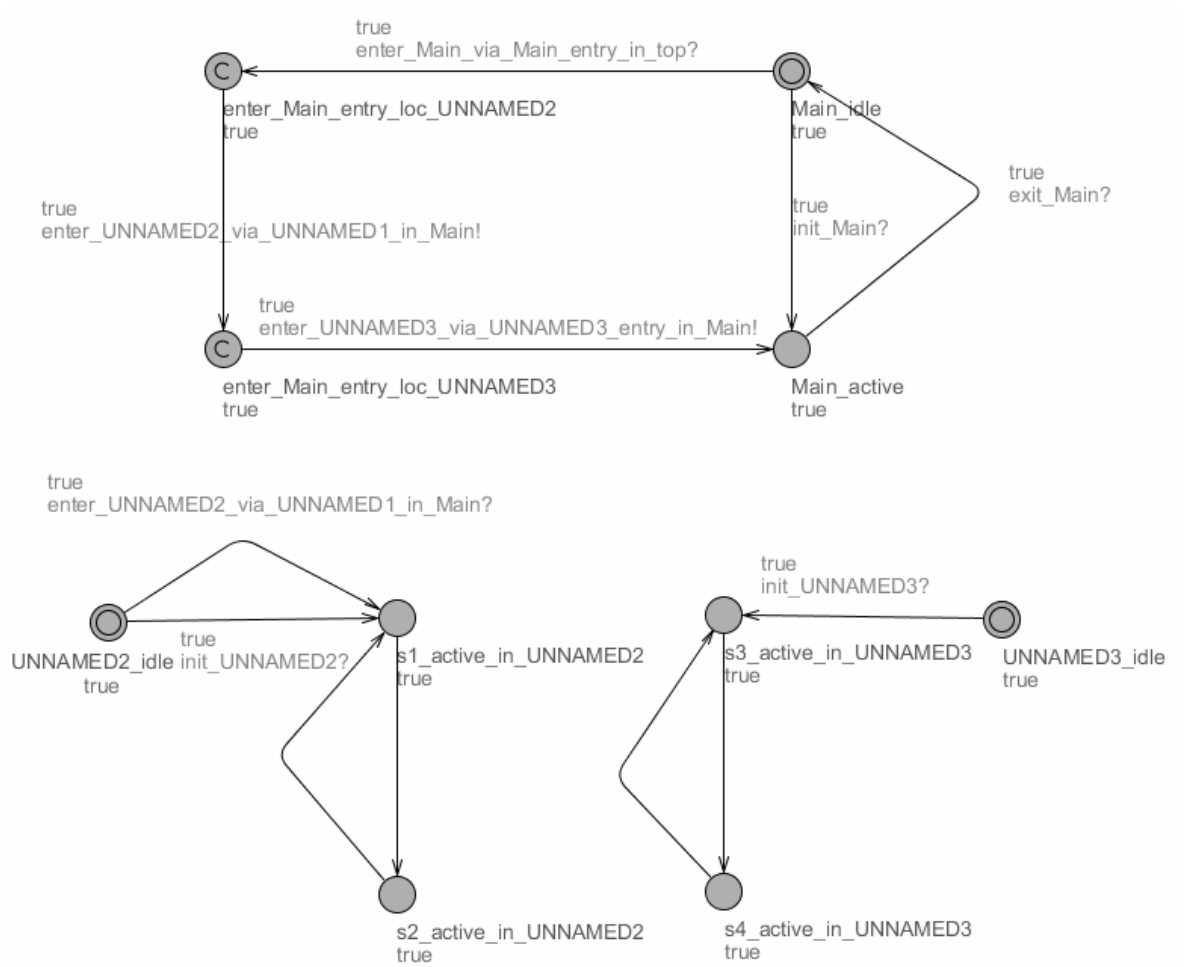


Рисунок 60. Полученная диаграмма UPPAAL для второго теста.

В третьем тесте (см. рисунок 61) присутствует нетривиальное множество выходных деревьев (функция `make_tree_set` в алгоритме из раздела 3.2.1)

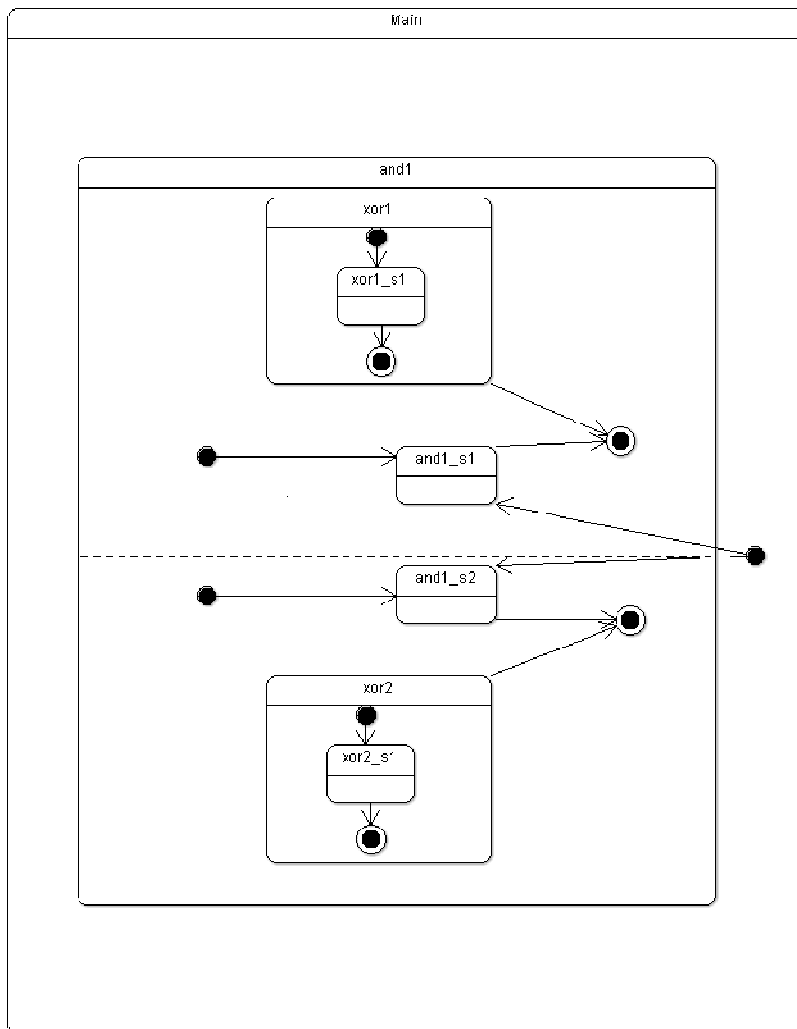


Рисунок 61. Третий тест.

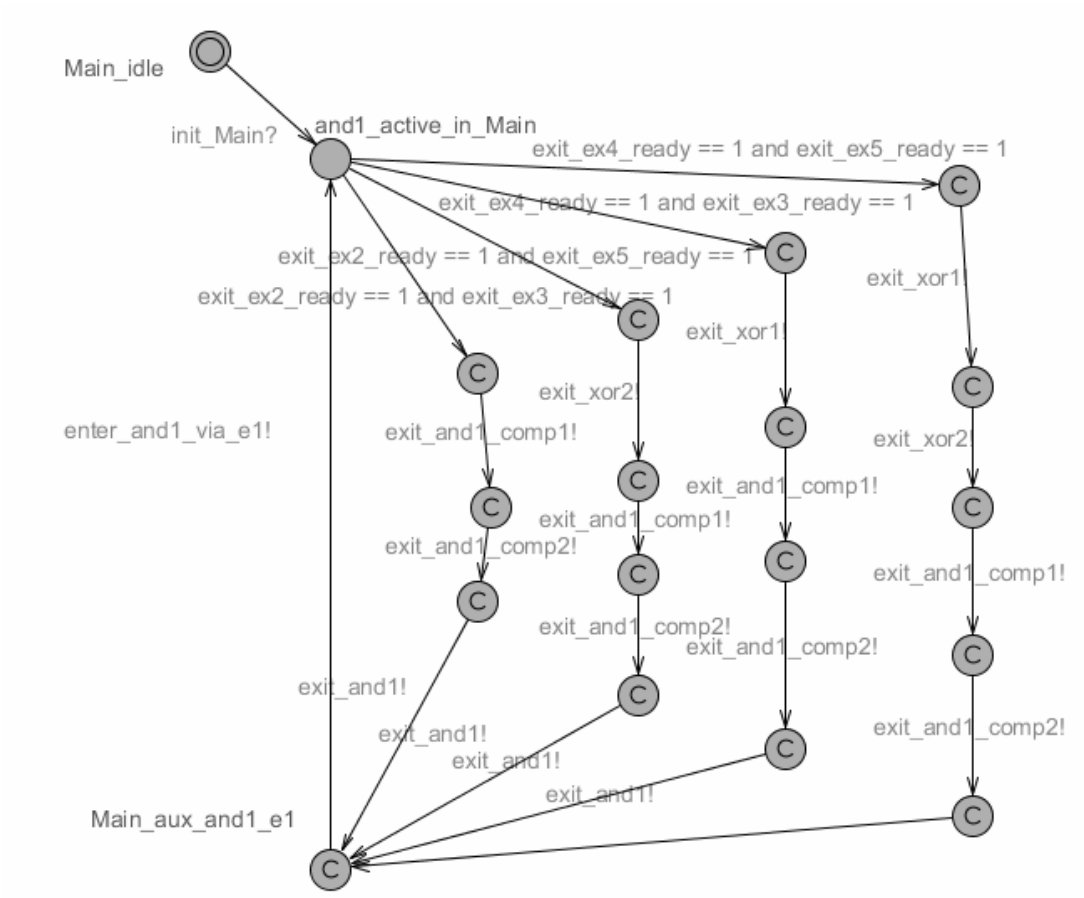


Рисунок 62. Ожидаемая диаграмма UPPAAL для третьего теста (фрагмент с деревом выходов).

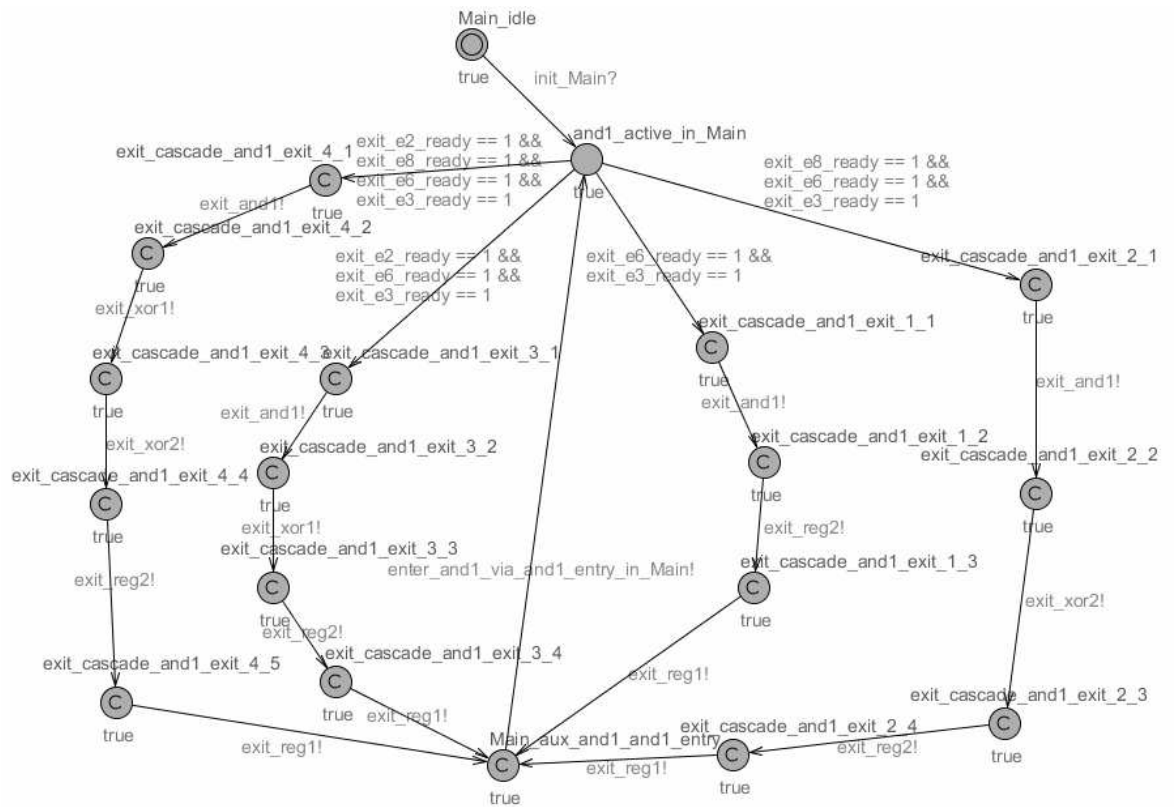


Рисунок 63. полученная диаграмма UPPAAL для третьего теста (фрагмент с деревом выходов).

В четвертом тесте (см. рисунок 64) требуется нетривиальное назначение предусловий (функция `add_exit_guard`s в алгоритме из раздела 3.2.1)

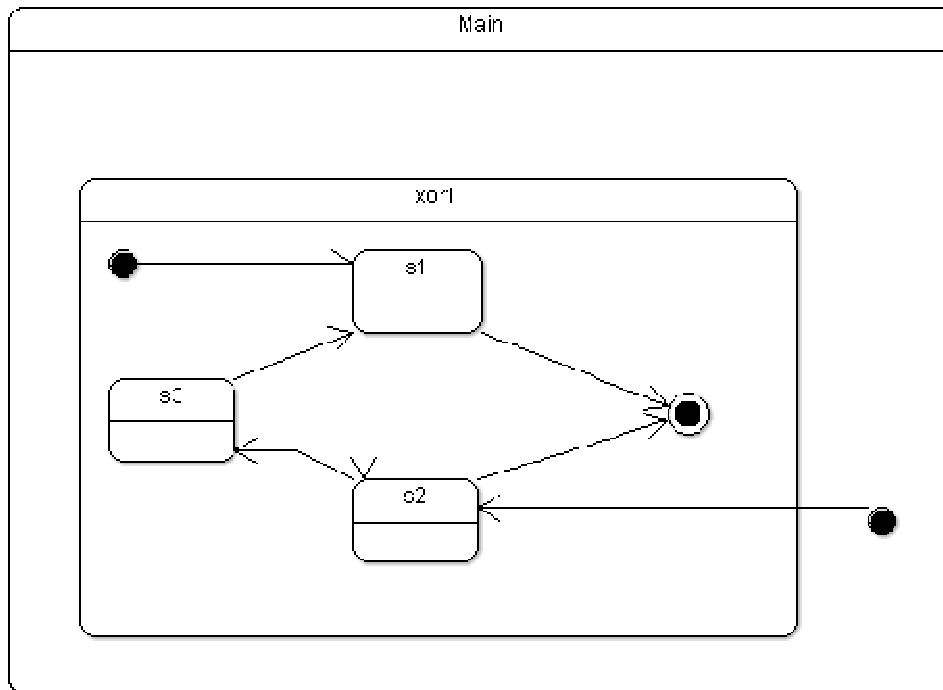


Рисунок 64. Четвертый тест.

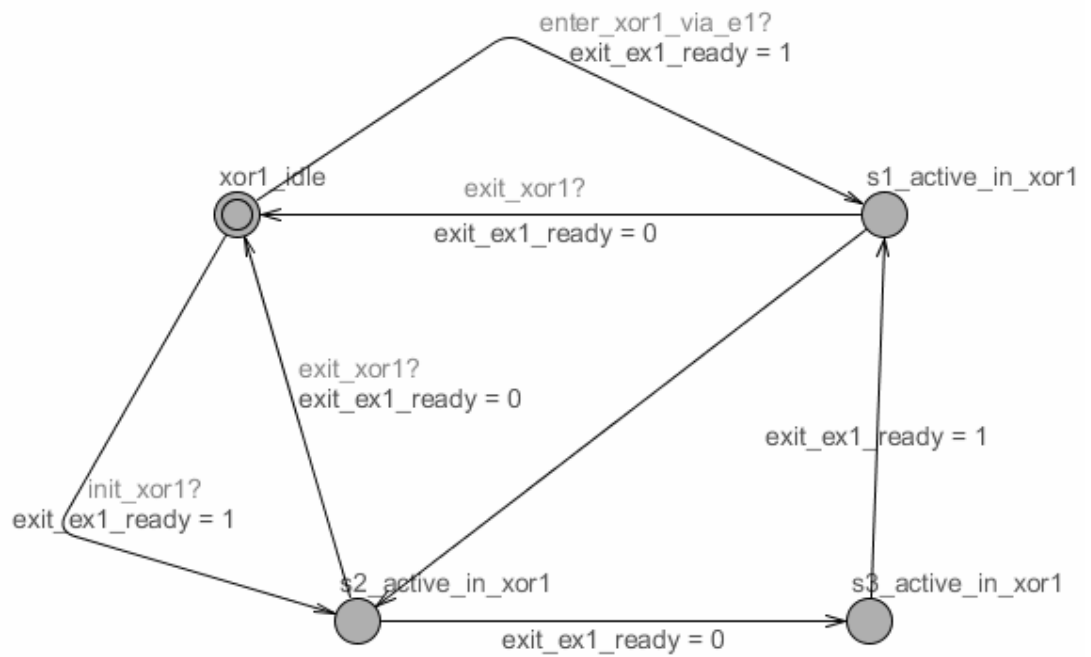


Рисунок 65. Ожидаемая диаграмма UPPAAL для четвертого теста.

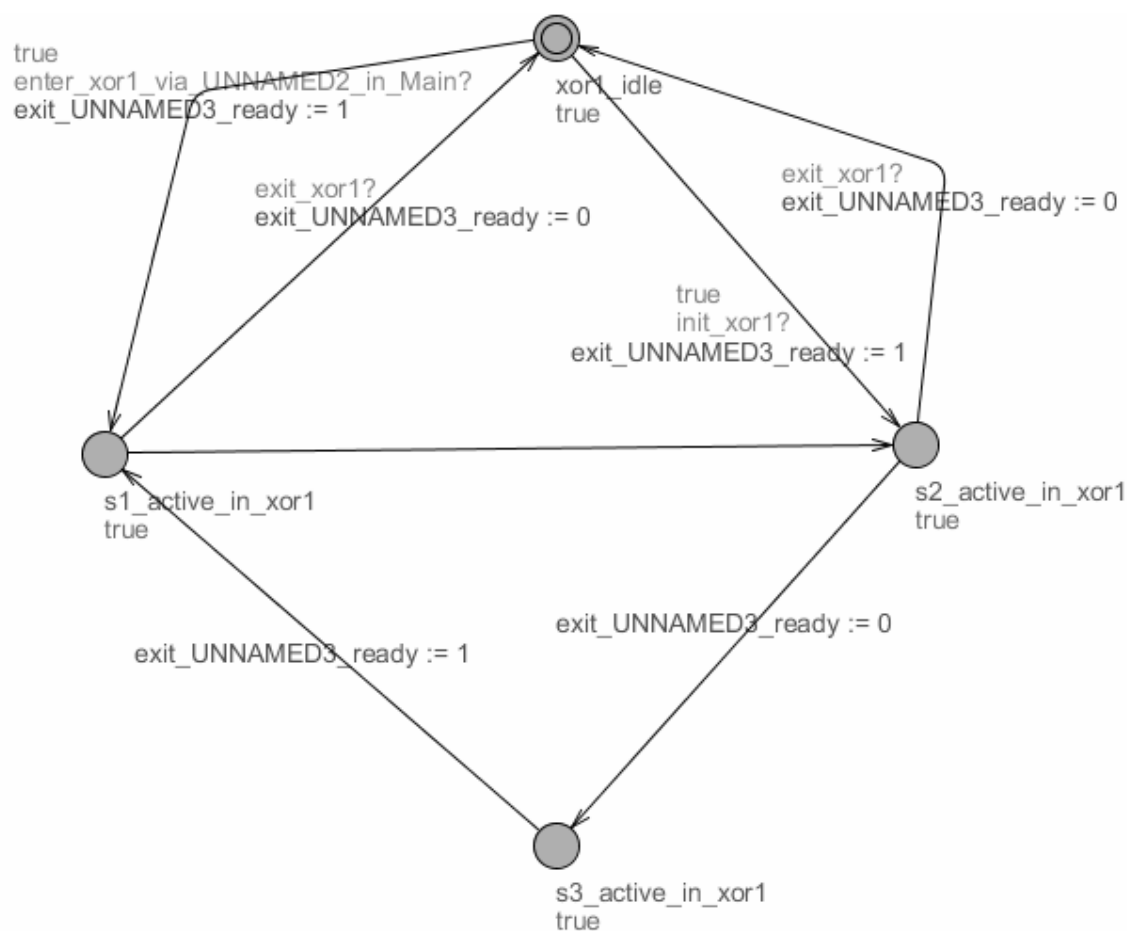


Рисунок 66. Полученная диаграмма UPPAAL для четвертого теста.

Из приведенных примеров видно, что алгоритм автоматически строит системы автоматов, эквивалентные ожидаемым с точностью до переименования состояний и переменных и применения эквивалентных логических преобразований. Также в автоматически построенных автоматах UPPAAL могут присутствовать дополнительные переходы, связанные с автоматическим добавлением входных состояний в соответствии с преобразованиями, описанными в разделе 3.2.3.

4.2.2 Моделирование системы управления уличным движением

В качестве одного из простых, но очень наглядных примеров РВС РВ можно рассмотреть систему управления уличным движением на перекрестке. Эта система призвана осуществлять правильное управление сигналами двух светофоров на перекрестке проспекта и улицы. Светофоры работают под управлением контроллера. В обычных условиях светофоры согласованно изменяют цвет сигнала так, чтобы периодически открывать движение по обеим магистралям. Но, помимо этого, контроллер наделен способностью

обнаруживать приближение к перекрестку автомобиля скорой помощи. В таком случае контроллер переходит в особый режим работы, чтобы обеспечить как можно более быстрое, но вместе с тем безопасное пересечение перекрестка автомобилем скорой помощи. Для простоты предполагается, что каждый раз не более одного автомобиля скорой помощи может приближаться к перекрестку. В обычном режиме работы сигналы светофора чередуются строго периодически, поочередно открывая движение по обеим магистралям: зеленый свет горит на протяжении 45 условных единиц (у.е.) времени, желтый – на протяжении 5 у.е., и красный – на протяжении 1 у.е. На перекрестке установлен сенсор, который позволяет обнаруживать приближение автомобиля скорой помощи к перекрестку. Сенсор калиброван так, чтобы различать три разновидности расположения автомобиля скорой помощи относительно перекрестка. Когда автомобиль скорой помощи появляется на одной из магистралей, ведущих к перекрестку, сенсор отправляет контроллеру сигнал «объект приближается к перекрестку» (arrg); когда автомобиль скорой помощи оказывается в непосредственной близости от перекрестка, контроллер получает предупреждение «объект на перекрестке» (before); и, наконец, когда автомобиль скорой помощи минует перекресток, сенсор отправляет контроллеру сигнал «объект миновал перекресток» (After). Контроллер, получив от сенсора сигнал «объект приближается к перекрестку», обеспечивает безопасный режим проезда автомобиля скорой помощи; для этого он включает красный свет у обоих светофоров. Как только автомобиль скорой помощи оказывается в непосредственной близости к перекрестку, и контроллер получает сигнал «объект на перекрестке», он включает зеленый свет на той магистрали, по которой движется автомобиль скорой помощи. Когда автомобиль скорой помощи минует перекресток, контроллер сменяет зеленый свет на красный на той магистрали, по которой проехала «скорая помощь», и переходит в обычный режим работы.

Модель управляющей информационной системы реального времени состоит из четырех компонентов – двух светофоров (светофор проспекта и светофор улицы), машины скорой помощи и контроллера, - взаимодействующих друг с другом. Фактически, поведение машины скорой помощи моделируется сенсором, который определяет ее местоположение. Общая схема взаимодействия этих компонентов изображена ниже на [рисунке 67](#).

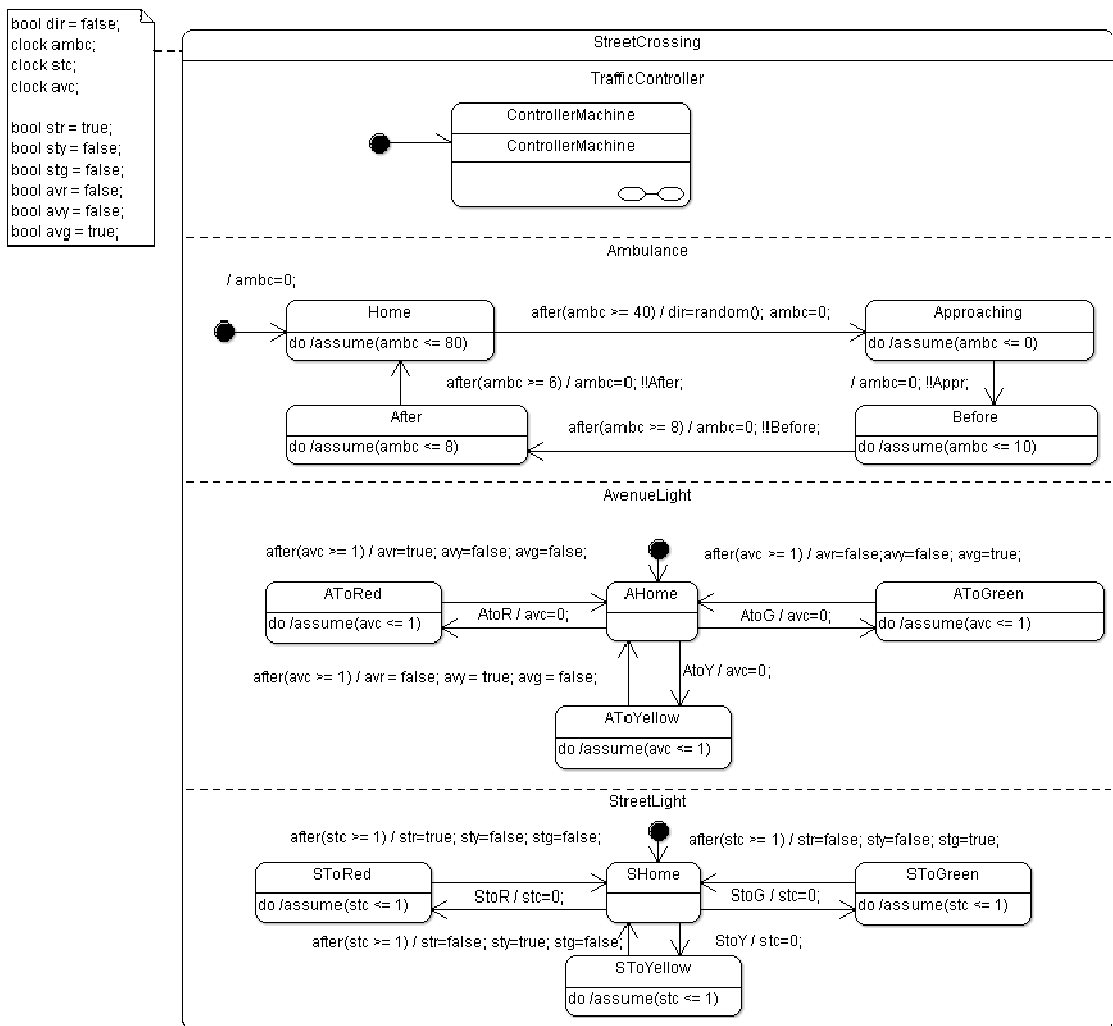


Рисунок 67. Главная диаграмма системы управления уличным движением

Контроллер получает сигналы от сенсора, следящего за подъездом машины скорой помощи и, в случае необходимости, переводит работу светофоров из регулярного режима в экстренный. Сигналы, которыми обмениваются компоненты рассматриваемой системы таковы.

Сенсор – контроллеру:

<Appr> - «скорая помощь» приближается к перекрестку,

<Before> - «скорая помощь» стоит на перекрестке,

<After> - «скорая помощь» миновала перекресток,

Контроллер – светофорам:

<StoR> - светофору улицы включить красный свет,

- <StoY> - светофору улицы включить желтый свет,
- <StoG> - светофору улицы включить зеленый свет,
- <AtoR> - светофору проспекта включить красный свет,
- <AtoY> - светофору проспекта включить желтый свет,
- <AtoG> - светофору проспекта включить зеленый свет.

Диаграммы, описывающие поведение каждого компонента, приведены на рисунках ниже. Оба светофора имеют одинаковое устройство (см. рисунки 68, 69), они являются двумя разными вариантами одного и того же прототипа (шаблона).

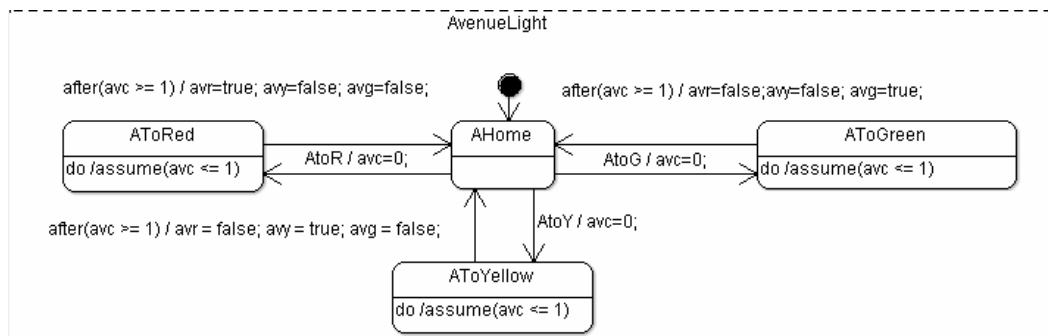


Рисунок 68. Диаграмма светофора на проспекте.

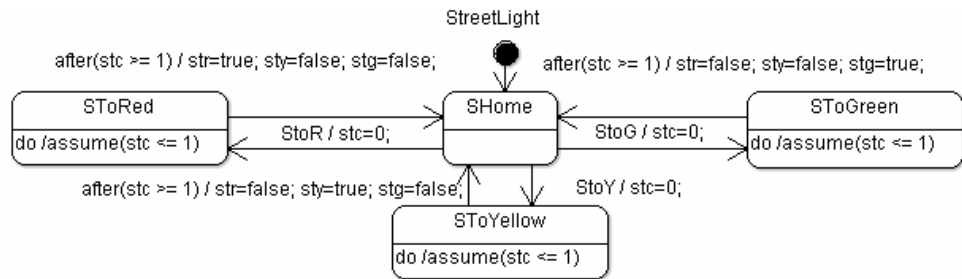


Рисунок 69. Диаграмма светофора на улице.

Наряду с состояниями управления в диаграмме контроллера присутствуют две переменные: *dir* и *amb*. Первая из них нужна контроллеру для того, чтобы помнить ту магистраль, по которой движется автомобиль скорой помощи, а вторая – для того, чтобы помнить о том, в каком режиме, регулярном или экстренном, работает вся система. Каждый светофор имеет три булевы переменные *r*, *y*, *g*, которые играют роль индикаторов сигналов светофора (*r*=1 – у светофора включен красный свет, *r*=0 – у светофора не включен красный свет).

Рассмотрим коротко устройство каждого компонента системы.

Обычное состояние управления светофора – это *Home*. В этом состоянии он может пребывать неограниченно долго. Однако поступление сигналов *<X,to_red>*, *<X,to_yellow>*,

$\langle X, to_green \rangle$ вынуждает светофор совершить переход в одно из состояний управления ToRed, ToYellow, ToGreen соответственно. В каждом из этих состояний светофор пребывает ровно 1 у.е. времени. Так моделируется выполнение задания перемены сигнала светофора. По истечении этого срока светофор возвращается в управляющее состояние Home, совершив на этом переходе присваивание переменным r, y, g соответствующих значений.

Модель машины скорой помощи осуществляет переходы по циклу. В состоянии управления Home машина скорой помощи может провести любой отрезок времени, длительность t которого располагается в пределах $40 \leq t \leq 80$. После этого процесс переходит в состояние APPR, выбрав произвольным образом направление движения к перекрестку. Не задерживаясь в этом состоянии, модель совершает переход в состояние управления BEFORE, отправив контроллеру сообщение о той магистрали, по которому движется машина скорой помощи. В состоянии BEFORE процесс проводит некоторое время t , длительность которого находится в пределах $8 \leq t \leq 10$. Это величина соответствует оценке времени, необходимого для того, чтобы машина скорой помощи могла приблизиться к перекрестку. Затем процесс совершает переход в состояние AFTER, отправив контроллеру сообщение о том, что машина скорой помощи находится на перекрестке. В состоянии AFTER процесс проводит некоторое время t , длительность которого находится в пределах $6 \leq t \leq 8$ (это соответствует оценке времени, необходимого машине скорой помощи для того, чтобы миновать перекресток). Далее процесс переходит в состояние управления Home, отправив при этом контроллеру сообщение о том, что машина скорой помощи миновала перекресток и скрылась из зоны видимости.

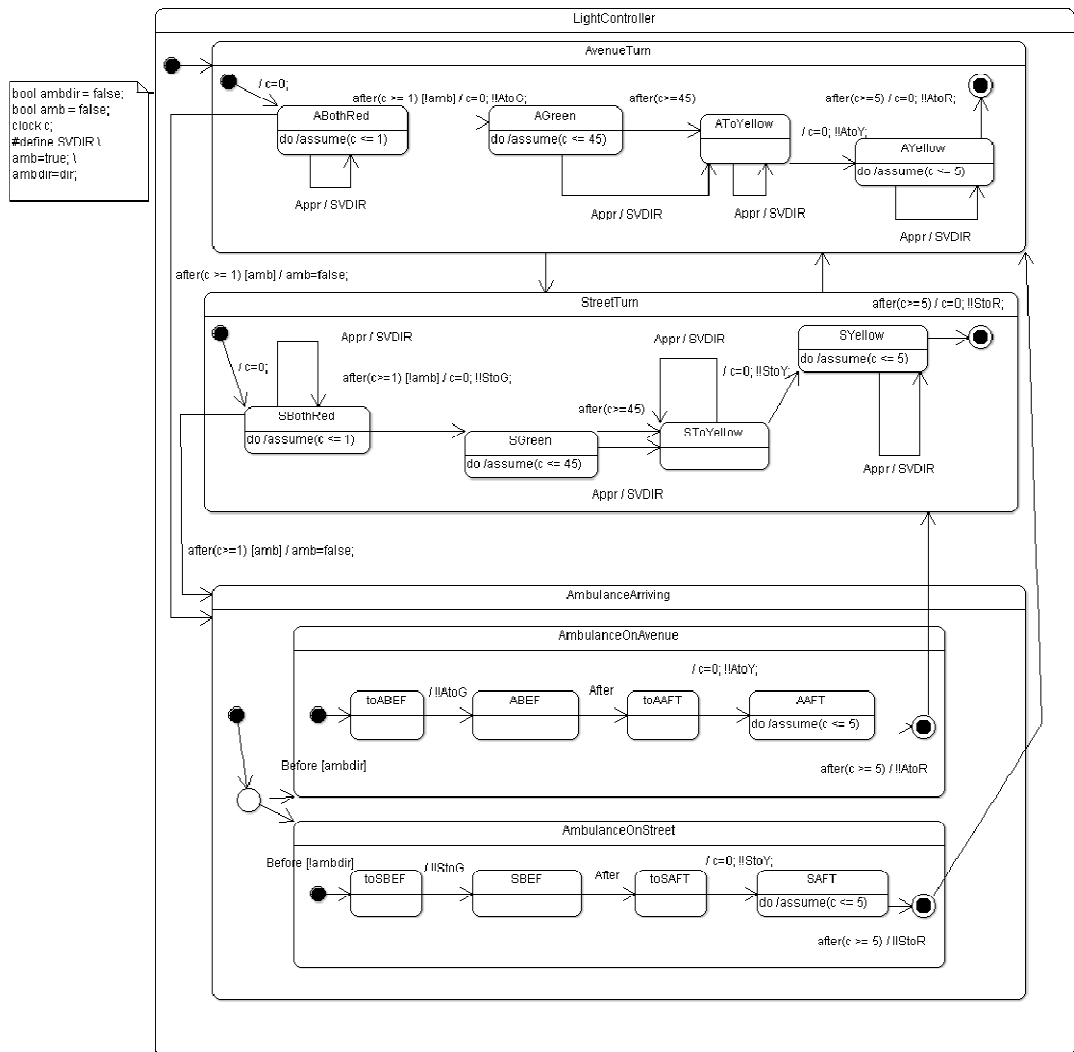


Рисунок 70. Диаграмма контроллера.

Поведение контроллера имеет две фазы – регулярную и экстренную. Начальным состоянием управления считается состояние ABothRed (both lights red, оба светофора горят красным, преимущество дано проспекту). В регулярной фазе контроллер работает по следующему сценарию:

1. из состояния ABothRed (у обоих светофоров включен красный свет, преимуществом пользуется проспект), в котором контроллер пребывает ровно 1 у.е. времени, он переходит в состояние управления AGreen (у светофора на проспекте включен зеленый свет, а у светофора на улице – красный свет) с отправлением команды <AtoG> светофору проспекта и пребывает в состоянии AGreen в течение 45 у.е. времени;

2. из состояния AGreen переходит в состояние управления AYellow (у светофора на проспекте включен желтый свет, а у светофора на улице – красный свет) с отправлением команды <AtoY> светофору проспекта и пребывает в состоянии AYellow в течение 5 у.е. времени;
3. из состояния AYellow переходит в состояние управления SBothRed (у обоих светофоров включен красный свет, преимуществом пользуется улица) с отправлением команды <AtoR> светофору проспекта и пребывает в состоянии BR2 в течение 1 у.е. времени;
4. далее по той же схеме в регулярной фазе контроллер проходит последовательно через состояния управления SGreen (у светофора на улице включен зеленый свет, а у светофора на проспекте – красный свет), SYellow (у светофора на улице включен желтый свет, а у светофора на проспекте – красный свет), ABothRed.

В каждом из перечисленных состояний контроллер имеет возможность принять сообщения от процесса- сенсора о приближении к перекрестку автомобиля скорой помощи и отметить факт получения этого сообщения и направление движения автомобиля изменением значения переменных-индикаторов amb и dir. В состояниях управления ABothRed и SBothRed контроллер проверяет значение переменной amb (индикатор приближения автомобиля скорой помощи) и в зависимости от значения индикатора либо продолжает выполнять переходы регулярной фазы работы, либо совершает переход в состояние управления AmbulanceArriving (у обоих светофоров включен красный свет, приближается автомобиль скорой помощи), начиная тем самым выполнять действия экстренной фазы работы.

В экстренной фазе работы контроллер в состоянии управления AmbulanceArriving ожидает поступления сообщения Before от сенсора, свидетельствующего о том, что автомобиль скорой помощи подъезжает к перекрестку, проверяет значение индикатора dir и в зависимости от того, по какой магистрали движется автомобиль скорой помощи переходит в одно из двух состояний управления ABEF (автомобиль скорой помощи приближается к перекрестку по проспекту) или SBEF (автомобиль скорой помощи приближается к перекрестку по улице), отправляя сообщение соответствующему светофору о необходимости включения зеленого света. В состоянии управления ABEF (или SBEF) контроллер пребывает, до тех пор пока не получит от сенсора сообщение after о том, что автомобиль скорой помощи миновал перекресток, и затем переходит в состояние управления AAFT

(автомобиль скорой помощи миновал перекресток по проспекту) или, соответственно, SAFT (автомобиль скорой помощи миновал перекресток по улице), отправив сообщение соответствующему светофору о необходимости включения желтого света. В этих состояниях контроллер пребывает 5 у.е. времени И, наконец из состояния управления AAFT (или SAFT) контроллер переходит в состояние управления ABothRed (или SBothRed соответственно), отправляя при этом приказ светофору переключить свет с желтого на красный с тем, чтобы впоследствии открыть движение по другой магистрали.

Простота устройства компонентов системы управления дорожным движением не должна вводить в заблуждение: главная особенность ее состоит в том, чтобы обеспечить корректную и безопасную синхронизацию взаимодействия всех ее составных частей в реальном времени. Поэтому для проверки свойств корректности и безопасности поведения этой системы реального времени нужно убедиться в том, что она удовлетворяет ряду требований.

1. Одновременное движение по обеим магистралям невозможно. Для проверки этого требования безопасности нужно убедиться в том, что система никогда не достигнет такого состояния вычисления, при котором у обоих светофоров включен зеленый свет.
2. В то время когда автомобиль скорой помощи приближается к перекрестку, у обоих светофоров всегда включен красный свет.
3. Во время пересечения перекрестка машиной скорой помощи на светофоре соответствующей магистрали всегда включен зеленый свет.
4. На каждой магистрали всегда соблюдается правильный порядок смены цвета сигнала соответствующего светофора.
5. Автомобиль скорой помощи всегда находится в движении.

4.2.3 Результаты

После трансляции модели светофора в UPPAAL получилась система из 12 автоматов:

- Global_kickoff
- StreetCrossing
- TrafficController
- Ambulance

- AvenueLight
- StreetLight
- LightController
- AvenueTurn
- StreetTurn
- AmbulanceArriving
- AmbulanceOnAvenue
- AmbulanceOnStreet

На рисунках 71, 72, 73 приведены некоторые из автоматов.

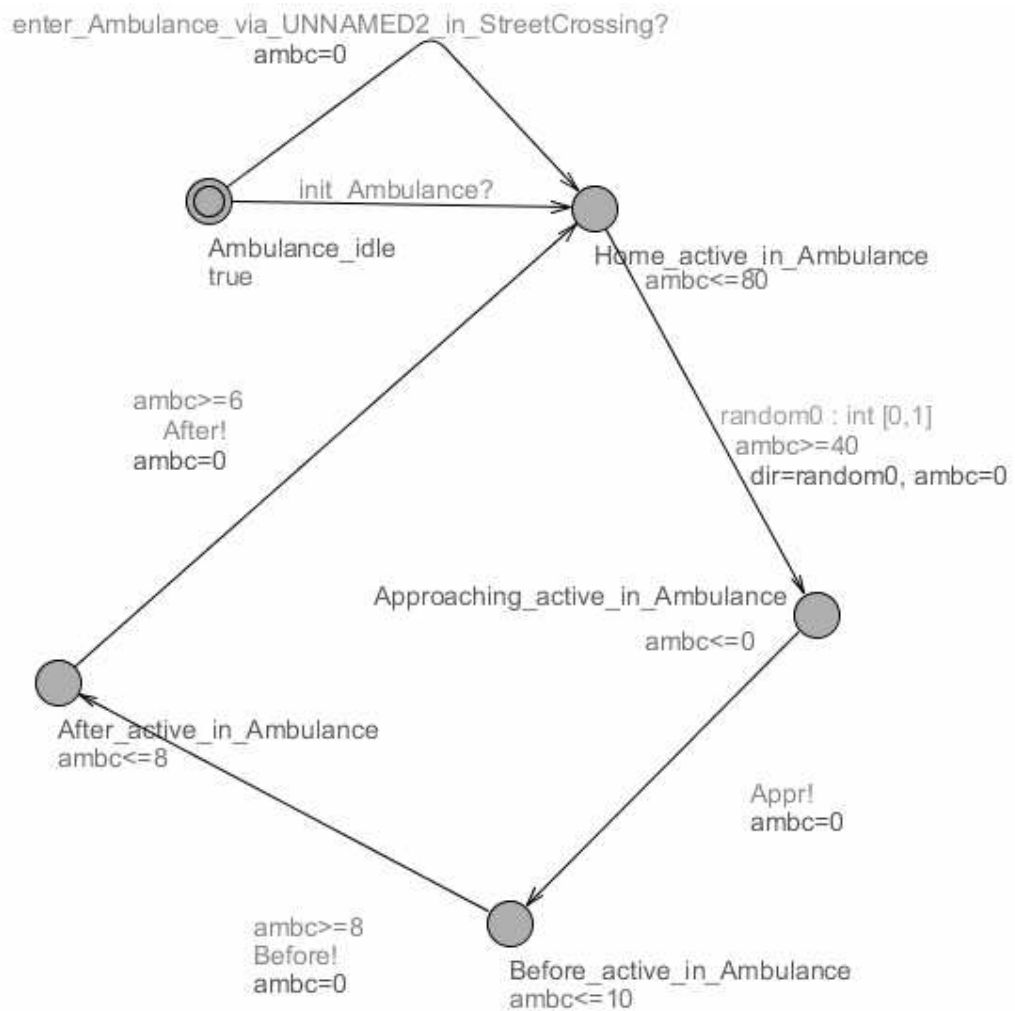


Рисунок 71. Автомат Ambulance.

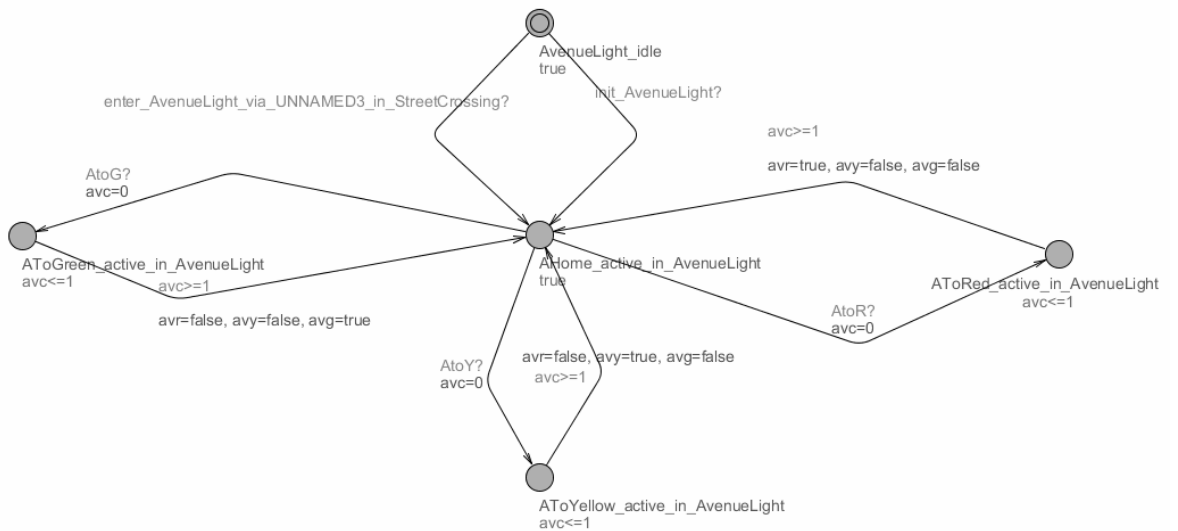


Рисунок 72. Автомат AvenueLight.

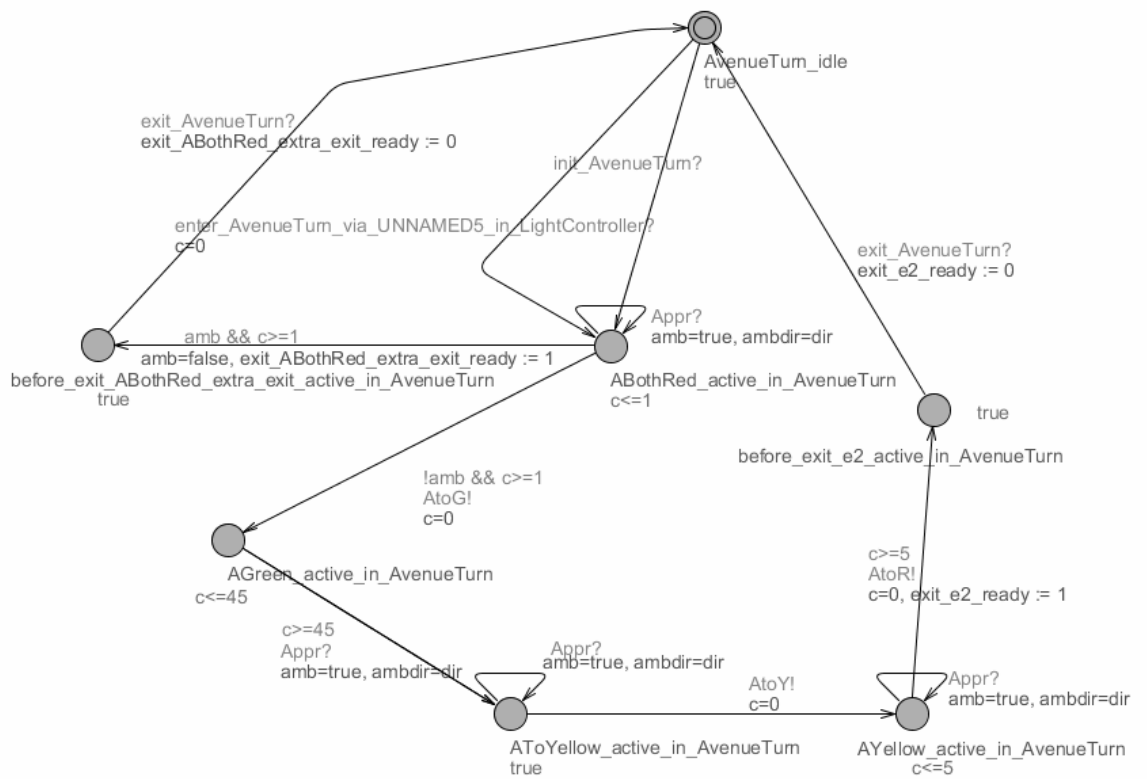


Рисунок 73. Автомат AvenueTurn.

Видно, что хотя полученная система и эквивалентна исходной диаграмме состояний UML, но она плохо читается из-за того, что разбита на 12 частей, и из-за этого добавлено множество служебных состояний и переходов. Потенциальной темой для дальнейшего

исследования является поиск способа улучшить алгоритм, добавив автоматическое уменьшение числа процессов и состояний.

На этой системе были проверены следующие свойства:

$A[]! \text{ deadlock}$

Свойство, гарантирующее отсутствие потенциальных тупиков в работе системы.

Свойство выполняется.

$A[]! (stg==1 \parallel sty==1) \text{ imply } avr==1$

$A[]! (avg==1 \parallel avy==1) \text{ imply } str==1$

Свойства, гарантирующие корректное переключение светофоров. Если на первом светофоре горит зеленый или желтый свет, то на втором горит обязательно красный, и наоборот. Свойство выполняется.

$E<> stg==1 \ \&\& \ avg==1$

Свойство, утверждающее, что существует трасса, на которой оба светофора горят зеленым. Это свойство не выполняется.

$A[] (stg==1 \parallel avg==1)$

Свойство, утверждающее, что всегда один из светофоров горит зеленым светом. Это свойство не выполняется, так как может быть ситуация, когда один светофор горит красным, а другой – желтым. На рисунке 74 приведен фрагмент трассы, обнаруженной верификатором, на которой данное свойство не выполняется.

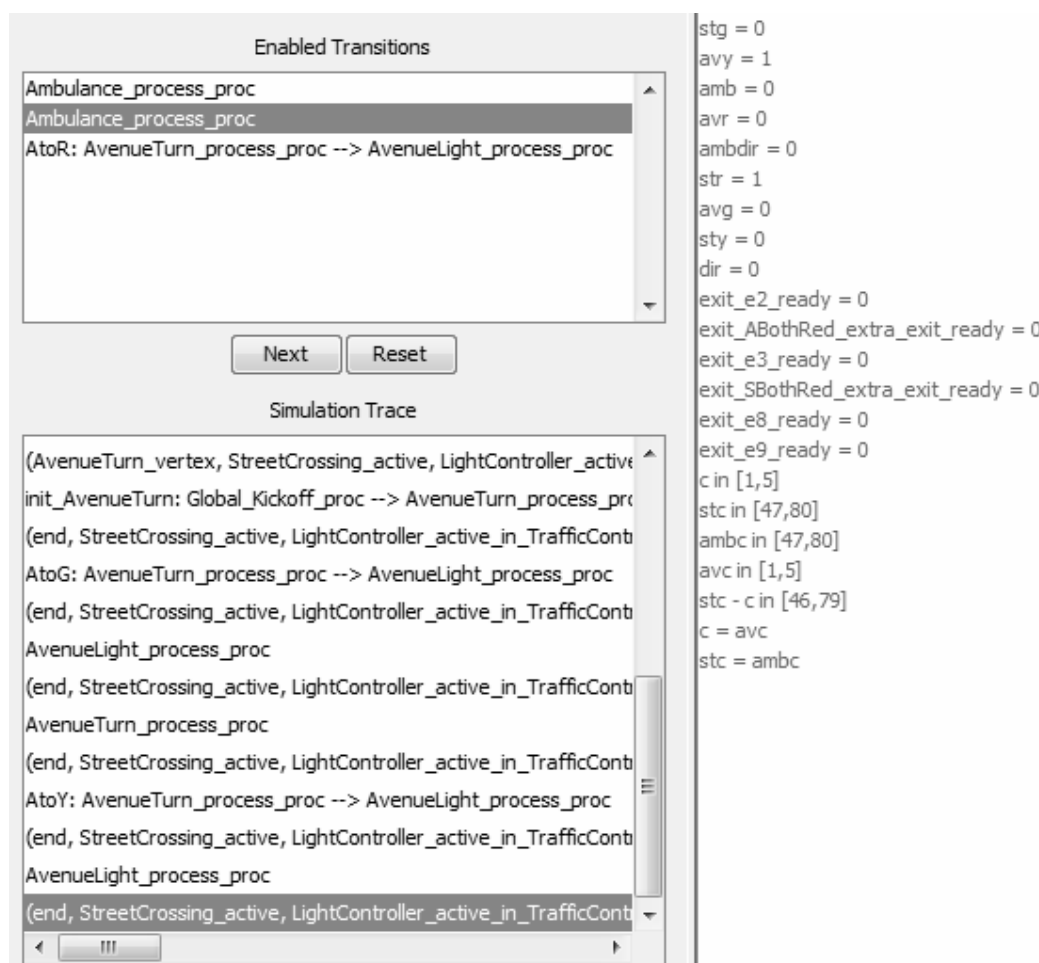


Рисунок 74. Пример трассы UPPAAL.

Ambulance_process_proc.Approaching_active_in_Ambulance -->
Ambulance_process_proc.Home_active_in_Ambulance

Свойство, гарантирующее достижимость состояния Home из состояния Approaching. Содержательно это означает, что если скорая помощь появилась, что через некоторое время она проедет через перекресток.

4.2.4 Выводы

Исследование алгоритма показало, что автоматическая трансляция диаграмм состояний UML в системы автоматов UPPAAL работает корректно. Результаты, полученные в статье [52] ручной работой, были полностью воспроизведены при автоматической трансляции. Проведенные эксперименты показали, что созданная система эффективно применима для проверки правильности поведения сравнительно простых распределенных систем, состоящих из нескольких взаимодействующих процессов.

Вместе с тем, результаты экспериментальных исследований показали, что для эффективного применения к сложным распределенным системам реального времени наша система нуждается в дальнейшем совершенствовании. Основным недостатком построенного двухступенчатого транслятора UML→UPPAAL состоит в том, что количество временных автоматов в сети, которая формируется на выходе транслятора, значительно превосходит число взаимодействующих процессов в исходном UML описании проектируемой РВС РВ. Это объясняется тем, что многие временные автоматы в полученной сети играют лишь вспомогательную роль, обеспечивая корректность перехода от иерархического, многоуровневого описания РВС РВ посредством диаграмм UML к одноуровневому, «плоскому» описанию той распределенной системы, но уже посредством сети временных автоматов, взаимодействующих друг с другом. Поскольку быстродействие UPPAAL имеет тенденцию стремительно понижаться с ростом числа временных автоматов в сети, можно предполагать что вычислительные возможности процедур верификации, заложенных в UPPAAL, окажутся недостаточными для проверки правильности поведения описаний РВС РВ, состоящих из большого числа диаграмм UML.

Для преодоления этой трудности на следующих этапах проекта планируется разработать и реализовать алгоритм оптимизации сети временных автоматов, конструируемой на выходе транслятора UML→UPPAAL. Предполагается, что этот алгоритм сможет выделять группы временных автоматов, совместное функционирование которых осуществляется по последовательным схемам передачи управления, и объединять автоматы одной группы в один последовательно работающий временной автомат. Есть основания полагать, что этот прием позволит сократить размер сети автоматов, получаемых на выходе построенного нами транслятора UML→UPPAAL, и сделает нашу систему пригодной для анализа поведения РВС РВ, состоящих из большого числа процессов.

4.3 Применимость CERTI для моделирования

4.3.1 Описание методики проведения экспериментов

Задачи экспериментального исследования

Первостепенной задачей на текущем этапе настоящей научно-исследовательской работы является оценка применимости системы CERTI к задачам по моделированию РВС РВ. Таким образом, необходимо провести экспериментальное исследование и оценить производительность данной системы моделирования.

Базовая версия CERTI RTI может и не продемонстрировать качества, достаточные для решения задач полунатурного моделирования. Такой результат будет означать, что использование CERTI RTI потребует внесения правок, улучшающих необходимые для данного класса задач характеристики системы. Отсюда вытекает следующий набор подзадач – оценить фронт предстоящих работ и выделить в системе моделирования «узкие места», снижающие показатели системы моделирования в целом.

На текущем этапе настоящей научно-исследовательской работы уже было сделано несколько предложений по улучшению архитектуры CERTI RTI и адаптации данного продукта к задаче имитационного моделирования распределённых встроенных систем реального времени. Результаты экспериментов с участием CERTI должны послужить отправной точкой для оценки важности предложенных изменений, расстановке приоритетов между ними и соответствующим распределением усилий разработчиков между найденными проблемами.

Подход к решению задачи

Возможность использования конкретной системы имитационного моделирования для выполнения заданной модели в реальном времени определяется двумя основными численными характеристиками: средним и максимально возможным временем обработки события. Первый из перечисленных параметров даёт общую оценку эффективности применения среды выполнения к заданной имитационной задаче. Чем меньше среднее время обработки события, тем большим потенциалом обладает система моделирования, и тем меньшие требования она предъявляет к вычислительным ресурсам. Таким образом, данный параметр важен для любой системы моделирования. Кроме того, если рассматривать имитационную задачу без ограничений на скорость выполнения, то меньшее время обработки события позволяет получать результат быстрее.

Существование такой характеристики, как наибольшее время обработки события позволяет гарантировать выполнение событий внутри заданного директивного интервала и обеспечивает систему свойством предсказуемости, необходимым для моделирования в реальном времени. Чем меньше максимальное время обработки события, тем большей гибкостью будет обладать система моделирования в реальном времени. Система с лучшими показателями позволяет проводить моделирование с использованием меньших директивных интервалов, и, как следствие, большей точностью. Кроме того, такая система способна обрабатывать более сложные имитационные модели и позволяет предоставить дополнительную свободу действий их разработчикам.

Если среднее время обработки события не будет удовлетворять требованиям, предъявляемым существующими имитационными задачами, то в среду выполнения неминуемо потребуются вносить правки. Поэтому точный расчёт наибольшего времени обработки события будет неоправданным и преждевременным. Таким образом, в рамках данного экспериментального исследования наиболее приоритетной задачей является измерение среднего времени обработки события.

Проведение моделирования без временных ограничений позволяет получить оценку для среднего времени обработки события в виде отношения времени выполнения модели и числа обработанных при этом событий. Все проведённые эксперименты были основаны на данном подходе.

Непосредственные оценки применимости системы моделирования CERTI для решения имитационных задач в реальном времени могут быть получены сравнением её численных результатов с аналогичными показателями системы моделирования «Стенд ПНМ», предназначенной для решения подобных задач. Данный подход потребует значительно меньших усилий разработчиков при использовании в качестве тестов для среды выполнения CERTI RTI имитационных задач, уже решённых с помощью системы «Стенд ПНМ» [20]. В результате были выбраны простейшие задачи, в частности, применяемые для проверки корректности работы системы «Стенд ПНМ».

4.3.2 Построение имитационных моделей совместимых со стандартом HLA

Разделяемые объекты и взаимодействия

Стандарт распределённого имитационного моделирования HLA IEEE-1516 2000 предоставляет два основных механизма для обмена информацией между участвующими в моделировании федератами: разделяемые объекты и взаимодействия. Основное отличие между данными механизмами взаимодействия – время жизни данных. Объекты похожи на разделяемую память, используемую для организации взаимодействия между процессами. Они предназначены для долгосрочного хранения данных и предполагают возможность многократного к ним доступа. Объекты хорошо подходят для представления, например, боевых единиц на поле боя или количества товара на складе. Значения таких параметров могут часто изменяться или запрашиваться несколькими участниками моделирования.

Взаимодействия представляют собой аналог механизма обмена сообщениями. В отличие от разделяемых объектов, взаимодействия не хранятся длительное время, а

разрушаются автоматически сразу после того, как были доставлены получателю. Данный механизм удобно использовать для моделирования однократных или кратковременных событий и явлений, таких как полёт снаряда после выстрела из орудия.

Описанные механизмы взаимодействия федератов взаимозаменяемы. Например, реализовав хранение копии объекта внутри каждого федерата, можно получить функциональность разделяемых объектов, используя только передачу сообщений. Если не использовать способность объектов к модификации и своевременно производить их удаление, то можно получить механизм, аналогичный передаче взаимодействий.

Стандарт HLA предоставляет возможность использования сразу двух механизмов взаимодействия, потому что каждый из них может быть более эффективен при решении разных классов имитационных задач. Обычно взаимодействия используются в случае, когда основные данные хранятся внутри федерата-получателя локально, а передаваемая информация должна привести к их изменению. Разделяемые объекты, наоборот, удобнее использовать, когда основной интерес представляет передаваемая информация.

Управление разделяемыми объектами и взаимодействиями

Обмен данными с помощью разделяемых объектов и взаимодействий стандарта HLA производится через соответствующие сервисы RTI. Существует несколько категорий таких сервисов: управления декларациями, управления объектами и управления правом собственности.

Сервисы управления декларациями описывают взаимодействие федератов в терминах модели вида издатель-подписчик, дающей возможность анонимной пересылки данных между отправителем и получателем. Пусть изначально есть несколько федератов-издателей, создающих новые разделяемые объекты или модифицирующие атрибуты уже существующих объектов, и несколько федератов-подписчиков, желающих получать от RTI уведомления об этих изменениях. Все издатели и подписчики уведомляют RTI о своих намерениях с помощью сервисов управления декларациями. Тогда при изменении атрибута объекта RTI предоставит всем подписанным на этот атрибут федератам соответствующее уведомление. При этом фильтрация поступающих сообщений остаётся на усмотрение федерата-получателя. Средства для автоматизированной фильтрации уведомлений от RTI стандарт HLA предоставляет другую категорию сервисов – сервисы управления распределением данных.

Разделяемые объекты и взаимодействия стандарта HLA похожи на классы декларативных объектно-ориентированных языков программирования. Для использования

объектов и взаимодействий нужно сначала описать их тип, так же называемый классом. Классы могут быть составными и включать в свой состав несколько атрибутов, с заданными правилами доступа. Из классов можно выстраивать иерархии наследования. Однако существует и множество отличий, которые необходимо учитывать при использовании разделяемых объектов и взаимодействий. В частности, атрибуты разделяемых объектов могут использоваться несколькими федератами одновременно, что усложняет правила доступа и приводит к необходимости введения сервисов управления правом собственности.

На протяжении всего времени существования объекта, с момента его создания и до момента разрушения, состояние объекта может контролироваться с помощью сервисов управления объектами. Данная категория сервисов позволяет создавать новые экземпляры заданного класса, инициализировать атрибуты объектов и модифицировать их значения, находить и удалять существующие экземпляры.

Выбор механизма взаимодействия для написания моделей

Модели BasicTest и BCVMTest, выбранные для оценки эффективности применения системы моделирования CERTI к решению имитационных задач в реальном времени, воспроизводят поведение нескольких физических устройств, пересылающих сообщения друг другу. Подобный обмен данными может быть с переменным успехом представлен как разделяемыми объектами, так и взаимодействиями стандарта HLA. Однако механизм взаимодействий ближе к передаче сообщений, используемой компонентами моделируемого вычислительного комплекса.

Выбор взаимодействий стандарта HLA для реализации обмена данными между федератами в данном случае не был случайным. Одна из целей, поставленных перед разрабатываемой системой моделирования – решение задачи полунатурного моделирования вычислительных комплексов в реальном времени. Условие подобных имитационных задач часто предполагает, что часть компонентов вычислительного комплекса представлена физическими устройствами, а часть – их программными моделями. При этом подключённое устройство зачастую представляет собой «чёрный ящик», отправляющий или принимающий сообщения по физическим каналам передачи.

Обмен данными между разнородными компонентами имитационной модели должен производиться в формате, понятном для оборудования. Так как взаимодействия стандарта HLA больше, чем разделяемые объекты, походят на сообщения, то их использование, вероятно, позволит построить подключать физические устройства более эффективно. Таким

образом, взаимодействия представляются более удобным механизмом для реализации имитационных моделей данного класса.

Моделирование передаваемых сообщений

Модель BasicTest состоит из одного устройства-получателя и нескольких устройств-отправителей, с каждым из которых связан числовой параметр. Каждый отправитель итеративно пересылает получателю значение своего параметра и уменьшает его на единицу, пока это значение остаётся положительным. При этом единственная задача получателя заключается в распечатывании значений присылаемых параметров.

С помощью механизма взаимодействий стандарта HLA описанная имитационная модель была реализована следующим образом:

1. В рамках федерации определяется пользовательский класс взаимодействий, атрибутом которого является единственный числовой параметр;
2. Каждый федерат-отправитель заявляет через сервисы RTI о своём намерении опубликовать данные описанного класса;
3. Федерат-получатель подписывается на получение уведомлений об изменениях, связанных с атрибутами описанного типа.

Таким образом, каждое создание федератом-отправителем нового взаимодействия заданного типа приведёт к пересылке соответствующего уведомления получателю.

В модели BCVMTest бортовая цифровая вычислительная машина (БЦВМ) обменивается сообщениями с несколькими подключёнными к ней терминалами. С точки зрения федератов-терминалов алгоритм работы имитационной модели может быть кратко сформулирован следующим образом:

1. Терминал отсылает бортовому вычислителю сообщение, содержащее значение локального числового параметра;
2. БЦВМ получает сообщение от терминала, выводит информацию об отправителе и полученном значении, и пересылает его обратно отправителю без изменений;
3. Терминал получает сообщение от БЦВМ и уменьшает значение параметра на единицу. Если полученное значение остаётся положительным, то производится повторение цикла, в противном случае – завершение работы терминала.

В рамках данной имитационной модели основной и единственной задачей БЦВМ является ответ на поступающие от терминалов запросы. Модель БЦВМ работает на всём протяжении выполнения имитационной модели и завершается лишь после завершения всех подключённых к моделированию терминалов.

Модель BCVMTest сложнее модели BasicTest и требует пересылки сообщений между участниками моделирования в две стороны. Однако при её построении были использованы те же самые принципы. Для реализации BCVMTest было определено два пользовательских типа взаимодействия, каждый из которых включает в свой состав числовой параметр и идентификатор терминала. При этом каждый федерат-терминал публикует взаимодействия первого типа и подписан на получение взаимодействий второго типа, в то время как бортовой вычислитель, наоборот, подписан на взаимодействия первого типа и публикует взаимодействия типа два.

Передача идентификатора терминала при каждом обмене данными позволяет центральному вычислителю определить отправителя сообщения и направить ответ именно данному терминалу. Так как все федераты-терминалы подписаны на получение взаимодействий от центрального вычислителя, то после получения взаимодействия они должны сверять полученный идентификатор сообщения с собственным идентификатором. Если идентификаторы не совпадают, то полученное от центрального вычислителя сообщение предназначается другому терминалу, и его можно сбросить.

Продвижение модельного времени

Внутри федератов продвижение модельного времени было реализовано с помощью сервисов Time Advance Request (TAR) и Next Message Request (NMR), позволяющих реализовать консервативную схему. При этом запрос типа TAR использовался в случае, если увеличения логического времени федерата должно произойти вне зависимости от влияния внешних факторов. В частности, с помощью запроса типа TAR было реализовано продвижение времени, имитирующее обработку сообщения физическим устройством, во время которой на поведение устройство не могут повлиять приходящие сообщения.

Если для федерата важна способность своевременно реагировать на произошедшее в системе событие, то для реализации продвижения времени использовался сервис типа NMR. Например, центральный вычислитель должен как можно быстрее отвечать на запросы, поступающие от подключённых к комплексу терминалов.

Немаловажным фактором, влияющим на производительность среды выполнения, является значение параметра предварительного просмотра Lookahead. Федерат не может планировать событие на время меньше суммы его текущего логического времени и установленного значения Lookahead. Таким образом, параметр Lookahead ограничивает снизу модельную скорость реакции федерата на произошедшее изменение состояния. В частности, центральная вычислительная машина не может ответить на полученное от

терминала сообщение до того, как её модельное время продвинется на текущую величину предварительного просмотра.

Консервативные алгоритмы продвижения модельного времени неэффективны при маленьких значения параметра Lookahead. Однако, часто моделируемые физические устройства в составе вычислительного комплекса имеют значения параметра Lookahead, измеряемые в микро- и даже наносекундах. Таким образом, выбор значения параметра предварительного просмотра требует компромисса между точностью моделирования и временем выполнения полученной имитационной модели.

Начальная синхронизация имитационной модели

Задачи BasicTest и BCVMTest предполагают одновременный старт всех участников моделирования. Отсюда возникает необходимость начальной синхронизации федерации. Для подобных целей стандарт HLA предлагает использовать механизм барьеров.

Сервисы RTI позволяют установить барьер для группы подключённых федератов. При достижении установленного барьера каждым федератом группы RTI оповещает их об успешной синхронизации. Стандарт HLA описывает особый способ использования барьера – полную синхронизацию федерации. Во время полной синхронизации федераты, подключаемые в период между установкой барьера и его достижением всеми подсоединёнными ранее федератами, включаются в группу участников синхронизации и оповещаются об установленном барьере.

Описанный случай применения барьеров позволяет использовать его для начальной синхронизации федератов модели. Допустим, что первый подключённый федерат сразу устанавливает барьер для полной синхронизации модели, но достигает его лишь через достаточно большой промежуток времени. Таким образом, все подключённые в этот период федераты будут участвовать в начальной синхронизации. Использование временной задержки для начальной синхронизации федератов позволяет подключать произвольное их количество.

4.3.3 Трудности при создании моделей

Неудобный интерфейс HLA

Спецификации стандарта моделирования HLA IEEE-1516 2000 определяют интерфейсы RTI и участвующих в моделировании федератов на достаточно высоком уровне абстракции. В частности, стандарт определяет собственные типы, такие как, последовательность байт произвольной длины, множество дескрипторов для обращения к

атрибутам разделяемых объектов и параметрам взаимодействий или собственные типы для хранения логического времени и временного интервала. Кроме того, семантика использования каждого метода формализуется с помощью всех доступных языковых средств. Например, каждый аргумент функции, модификация которого внутри неё не предполагается, определяемых стандартом сервиса с помощью константного модификатора. Отдельного внимания заслуживает описание исключительных ситуаций, которые могут произойти во время выполнения каждого метода.

К сожалению, высокий уровень абстракции и точности спецификаций порождает громоздкие и неудобные интерфейсы. В то же время стандарт требует переопределения некоторых методов со стороны федерата. На практике это приводит к необходимости описывать сложные и малопонятные методы в пользовательском коде.

Кроме того, методы, определяемые стандартом HLA, часто обеспечивают слишком низкий уровень сервиса и вынуждают разработчиков механически реализовывать необходимую им функциональность при построении каждой новой имитационной задачи. Примером такой функциональности может быть начальная синхронизация имитационной модели, используемая в каждой из решённых экспериментальных задач.

В результате во время построения экспериментальных имитационных моделей вокруг интерфейсов, описываемых стандартом HLA, был разработан ряд классов-обёрток, предоставляющих функциональность, более удобную при решении поставленных задач. Кроме того, был создан набор классов облегчающих процесс конвертирования между произвольными пользовательскими типами данных и набором типов, определяемым спецификациями стандарта.

Ошибки дистрибутива CERTI

В ходе работы с дистрибутивом CERTI был выявлен ряд программных ошибок, потребовавших исправления. Основная часть ошибок была связана с недавним началом поддержки стандарта HLA IEEE-1516 версии 2000 (до этого CERTI поддерживал лишь стандарт HLA DMSO 1.3, выпущенный в 1998 году). Несмотря на тот факт, что дистрибутив CERTI содержит набор функциональных тестов, для проверки своей корректности, на момент выхода последней версии CERTI покрытие кода тестами было неполным. В частности, разработчики не располагали тестами для проверки интерфейсов стандарта HLA IEEE-1516, в котором было введено множество доработок и обобщений по сравнению с предыдущей версией.

Стандарт IEEE-1516 2000 предусматривает возможность передачи федератами данных лишь в нескольких абстрактных форматах. Предыдущая версия CERTI содержала специальный набор функций для конвертирования к необходимым форматам, которые, однако, несовместимы с последней версией стандарта HLA. Таким образом, необходимость приведения произвольных типов данных к заданному набору возлагается на разработчиков имитационной модели. При этом возникает новая проблема, связанная с порядком байтов в представлении числовых типов используемом архитектуре инструментальной машины.

Разные архитектуры процессоров могут использовать представления чисел, отличающиеся друг от друга порядком байт. Поэтому передача чисел требует более тонкого подхода, чем простое копирование содержащих его ячеек памяти. Традиционно существует несколько возможных вариантов представления [60]:

1. Порядок от младшего байта к старшему (англ. little-endian). В таком представлении шестнадцатеричное число DEADBEEF в ячейках памяти будет иметь вид EFBEADDE;

2. Порядок от старшего байта к младшему (англ. big-endian). В данной архитектуре шестнадцатеричное число DEADBEEF будет иметь первоначальный вид;

3. Смешанный порядок байтов (англ. middle-endian) может использоваться при работе с числами, превышающими размер машинного слова. При этом способе записи число представляется последовательностью машинных слов в естественной для данной архитектуры записи, но сами слова идут в обратном порядке.

Таким образом, если в моделировании участвуют инструментальные машины с разной архитектурой, то числа могут передаваться некорректно не только из-за некорректной работы RTI, но и по вине разработчика имитационной модели.

Существуют и другие ошибки, связанные с приведение пользовательских данных к абстрактным типам стандарта HLA. Например, одна из найденных в дистрибутиве CERTI ошибок была связана именно с конвертированием данных, производимых внутри реализации для использования функциональности RTI предыдущей версии. В RTI совместимой с HLA DMSO 1.3 любые параметры передавались в виде структуры, состоящей из последовательного участка памяти и его размера. При конвертировании размер описанной структуры рассчитывался с помощью стандартной функции, считающей концом строки первый нулевой байт. Однако при передаче произвольных данных нулевой байт может встречаться не только в конце содержащей их строки.

Другая категория ошибок была вызвана добавленной в рамках последнего стандарта функциональностью. Например, в рамках стандарта HLA IEEE-1516 2000 появилась возможность резервирования имени разделяемого объекта до его создания. При реализации нескольких новых методов были допущены ошибки. При разборе запросов, поступающих от федерата, процесс RTIA некорректно преобразовывал сообщения описанного типа, что приводило к критической ошибке и аварийному завершению моделирования.

Ещё одна категория ошибок была связана с редким использованием некоторых сервисов RTI. Например, при попытке использования сервисов управления правом собственности федератов над разделяемыми ресурсами имитационной модели иногда возникала критическая ошибка в центральном компоненте распределённой среды выполнения – процессе RTIG. В ходе исправления этой ошибки удалось выяснить, что её причиной послужил неудачный рефакторинг исходного кода, произведенный во время внедрения в существовавшую среду выполнения контейнеров из стандартной библиотеки STL. В одном из возможных вариантов работы программы производилась попытка записи элемента в контейнер без предварительного выделения под него памяти. На данный момент сервис RTI, приводивший к обнаруженной ошибке, практически не используется разработчиками, поэтому данная ошибка не была обнаружена такое долгое время.

Отдельной ошибкой, не входящей в уже описанные категории является некорректное поведение основного встроенного в дистрибутив CERTI алгоритма временной синхронизации. На одной из созданных в рамках настоящего исследования модели алгоритм работает неоправданно долго, в сравнении с теоретической оценкой его вычислительной сложности. Косвенным показателем программной ошибки может являться количество сообщений, передаваемых между процессами RTIG & RTIA, значительно превышающее предусмотренное алгоритмом значение. Причина данной ошибки так и остаётся неясной. К счастью, в дистрибутив CERTI встроен альтернативный алгоритм синхронизации, работающей на построенном примере корректно [16].

В ходе работы с RTI проекта CERTI в данном дистрибутиве было выявлено десять критических ошибок. Найденные ошибки были исправлены, а предложенные изменения исходного кода согласованы с разработчиками данного продукта. Кроме того, одна из разработанных имитационных моделей была включена в состав набора для функционального тестирования дистрибутива CERTI. Таким образом, после возможных будущих модификаций дистрибутива уже найденные в нём ошибки не будут сделаны снова.

4.3.4 Численные результаты исследования

Экспериментальный стенд

Основная часть экспериментального исследования прототипа среды выполнения производилось на машине с процессором Intel Core-i7 и 4-мя Гб оперативной памяти, находящейся под управлением 64-х битной версии операционной системы Ubuntu 11.04 Natty. Замеры производительности системы «Стенд ПНМ» производились на собственной инструментальной машине с процессором Core 2 Duo и частотой 2.60 ГГц, 2 Гб оперативной памяти и операционной системой Debian 5.0.8 Lenny. Поэтому сравнение показателей системы «Стенд ПНМ» с результатами CERTI потребовало запуска созданного прототипа на той же машине.

Для экспериментов была использована последняя доступная из репозитория разработчиков версия CERTI RTI. Данная версия CERTI основана на стабильной версии CERTI 3.4.0, но также содержит исправления ошибок, найденных с момента её выхода. Для сборки проекта использовался стандартный для данной операционной системы компилятор GCC версии 4.3.

Запуск моделирования

Для запуска и тестирования сред выполнения была использована система DTest, созданная разработчиками CERTI. Основная цель проекта DTest – проверка соответствия поведения распределённой системы предоставленному сценарию. Система реализована в виде подключаемой библиотеки языка Python.

Изначально целевая распределённая система представляется в виде набора специальных процессов-тестеров. При создании тестера с ним ассоциируется сессия по протоколу ssh, позволяющая управлять одним из вычислительных узлов системы. Поведение каждого тестера контролируется с помощью разбора поступающей от него информации.

Сценарий поведения процесса-тестера описывается в виде последовательности шагов. К каждому тестеру могут быть привязаны файлы, соответствующих потокам данных сессии. Запись потока данных в ассоциированный с ним файл производится автоматически, что позволяет облегчить получение и анализ экспериментальных данных.

Подсчёт результатов

В рамках настоящего экспериментального исследования время выполнения имитационной модели отождествлялось с интервалом между моментом завершения

начальной синхронизации и завершением всех подключённых федератов. Таким образом, результат эксперимента не учитывает затраты на инициализацию имитационной модели и системы моделирования, используемой для её выполнения.

Так как каждый из подключённых к RTI федератов представляет собой отдельную и независимую программу, которая, вообще говоря, может выполняться на собственной инструментальной машине, то в рамках федерации отсутствуют общие часы, и измерение общего времени выполнения имитационной модели представляется затруднительным. Поэтому в ходе проведённого экспериментального исследования замерялось выполнение каждого участника моделирования, а общее время выполнения модели считалось как среднее арифметическое от полученных значений.

Результаты исследования и их анализ

В рамках проведённой практической работы было создано несколько различных реализаций BasicTest и BCVMTest. Для каждой из перечисленных имитационных моделей была разработана версия, использующая механизмы продвижения модельного времени стандарта HLA, и версия, в которой федераты-участники обмениваются данными без учёта модельного времени. Кроме того, для модели BCVMTest была разработана версия, использующая вместо средств обмена данными, предоставляемых реализацией CERTI RTI, физический канал передачи данных MIL-STD-1553 [61].

В ходе первого эксперимента проводилось исследование зависимости между числом переданных сообщений и временем выполнения имитационной модели. При этом среднее время обработки одного события может быть получено как отношение этих величин (таблица 30).

При выключенной временной синхронизации увеличение числа передаваемых сообщений порождает дополнительную нагрузку на используемую сеть передачи данных. Таким образом, возможна ситуация, когда сеть станет узким местом передачи данных и приведёт к значительному падению производительности.

Таблица 30. Зависимость времени выполнения имитационных моделей от числа передаваемых сообщений.

Число сообщений	Время выполнения модели BasicTest, мкс		Время выполнения модели BCVMTest, мкс	
	Без учёта времени	С учётом времени	Без учёта времени	С учётом времени
10	4	7	9	14
100	27	45	51	124
1000	186	459	615	783
10000	1675	5499	3938	5994
100000	17411	37194	59171	69878

Кроме того, была произведено сравнение производительности среды выполнения при использовании оригинального механизма передачи данных и канала MIL-STD-1553 (таблица 31). Каналы данного типа имеют пропускную способность 1 Мбит/с, что существенно меньше в сравнении каналом Ethernet, используемым системой CERTI.

Таблица 31. Временя выполнения модели BCVMTest при использовании канала передачи данных MIL-STD-1553 и без него.

Число сообщений	Время выполнения модели BCVMTest, мкс	
	Без использования MIL-STD-1553	С использованием MIL-STD-1553
10	14	23
100	124	212
1000	783	2144
10000	5994	21849
100000	69878	216132

Второй эксперимент отражает зависимость между средним временем обработки события и числом участвующих в моделировании федератов. При увеличении числа компонентов модели время работы и сложность алгоритмов, требующих согласования действий нескольких федератов, существенно увеличиваются.

В системе моделирования CERTI большая часть глобальных алгоритмов для федерации реализована внутри глобального процесса RTIG, поэтому в большинстве случаев число передаваемых по сети сообщений увеличивается с ростом числа федератов не значительно. Однако данное утверждение неверно для используемых при построении моделей BasicTest и BCVMTest консервативных алгоритмов временной синхронизации.

Кроме того, рассмотренные модели являются существенно централизованными. В частности, вместе с ростом числа подключённых федератов-терминалов значительно увеличивается нагрузка на модель БЦВМ, что приводит к замедлению выполнения всей имитационной модели. Результаты исследования зависимости между временем выполнения имитационной модели BCVMTest и числом подключённых к центральному вычислителю терминалов приведены в [таблице 32](#).

Таблица 32. Зависимость времени выполнения модели BCVMTest от числа подключённых к центральному вычислителю терминалов.

Число сообщений	1 терминал	2 терминала	3 терминала	4 терминала
10	14	13	15	20
100	51	93	146	179
1000	615	871	1240	1771
10000	3938	9343	12776	15801
100000	59171	84864	118197	157660

На практике рассмотренные модели BasicTest и BCVMTest используются для проверки работоспособности среды выполнения системы моделирования «Стенд ПНМ». Поэтому время работы имитационных моделей, которое достигается с помощью системы «Стенд ПНМ», известно. Сравнение результатов, полученных с использованием CERTI, и результатов системы «Стенд ПНМ» приведены в [таблицах 33 и 34](#).

Результаты системы «Стенд ПНМ» подсчитывались как совокупность всего времени работы имитационной программы, включая инициализацию различных компонентов системы. Поэтому показатели системы при небольшом количестве переданных сообщений проигрывают аналогичным характеристикам системы CERTI. Однако вместе с увеличением числа передаваемых сообщений «Стенд ПНМ» начинает значительно превосходить своего конкурента.

Таблица 33. Сравнение времени выполнения модели BasicTest с использованием средств CERTI и системы «Стенд ПНМ»

Число сообщений	Время выполнения модели системой CERTI, мкс	Время выполнения модели системой Стенд ПНМ, мкс
10	7	638
100	43	733
1000	449	849
10000	5562	898
100000	39027	935

Таблица 34. Сравнение времени выполнения модели BCVMTest с использованием средств CERTI и системы «Стенд ПНМ»

Число сообщений	Время выполнения модели системой CERTI, мкс	Время выполнения модели системой Стенд ПНМ, мкс
10	14	802
100	128	1037
1000	795	3744
10000	5874	4598
100000	70128	5607

Выводы

Результаты, показанные базовой реализацией системы моделирования CERTI, позволяют использовать её в данном виде лишь для полунатурного моделирования небольших РВС РВ. Однако дальнейший анализ данного продукта показал, что его производительность может быть существенно увеличена, например, с помощью изменения существующей структуры процессов распределённой системы, использования более эффективных способов передачи данных и перераспределения функциональности между глобальными и локальными компонентами RTI.

Распределённая система моделирования CERTI предназначена для решения самых разных имитационных задач, включая разработку и построение эффективных систем массового обслуживания, дистанционное обучение персонала в интерактивном режиме и оказание помощи в принятии решения. В рамках настоящего проекта систему CERTI предполагается использовать для полунатурного моделирования РВС РВ, и её производительность системы может быть существенно повышена после проведения адаптации к данному классу задач. В частности, подобные задачи могут решаться более эффективно с помощью использования смешанных алгоритмов синхронизации.

Система моделирования «Стенд ПНМ» была изначально ориентирована на исследование свойств РВС РВ. Кроме того, система «Стенд ПНМ» является параллельной системой моделирования, исключая удалённое расположение участников. Поэтому неудивительно, что данная система выигрывает у своего более универсального аналога. В то же время отставание текущей версии RTI CERTI не является критическим и может быть преодолено.

4.4 Экспериментальное исследование средства трансляции UML в исполняемые модели совместимые со стандартом HLA

4.4.1 Описание экспериментального исследования

Основной задачей экспериментального исследования средств трансляции UML в исполняемые модели совместимые со стандартом HLA являлось функциональное тестирование разработанных средств. Для проверки правильности работы средств трансляции были созданы диаграммы состояний моделей BasicTest (см. рисунок 75) и BCVMTest (см. рисунок 76), описанных в разделе 4.3.2 и предварительно написанных вручную. Далее приведены диаграммы состояний этих моделей:



Рисунок 75. Диаграмма состояний BasicTest.

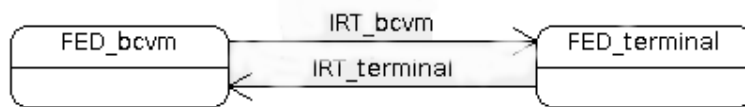


Рисунок 76. Диаграмма состояний BCVMTest.

Затем по диаграммам состояний были сгенерированы XML файлы (в формате scxml).
 Подробное описание данного формата приведено в разделе 3.3. Содержимое файлов
 приложено на **рисунках 77 и 78**.

```

<scxml id="SCXML" initial="FED_sender" xmlns="http://www.w3.org/2005/07/scxml"><!-- node-size-and-position x=0.0 y=0.0 w=654.0 h=595.0 -->
<state id="FED_sender"><!-- node-size-and-position x=70.0 y=160.0 w=100.0 h=100.0 -->
  <transition event="IRT_sender" target="FED_receiver"></transition>
</state>
<state id="FED_receiver"><!-- node-size-and-position x=380.0 y=230.0 w=100.0 h=100.0 --></state>
</scxml>
  
```

Рисунок 77. XML представление BasicTest.

```

<scxml id="SCXML" initial="FED_bcvm" xmlns="http://www.w3.org/2005/07/scxml"><!-- node-size-and-position x=0.0 y=0.0 w=654.0 h=595.0 -->
<state id="FED_bcvm"><!-- node-size-and-position x=70.0 y=160.0 w=100.0 h=100.0 -->
  <transition event="IRT_bcvm" target="FED_terminal"></transition>
</state>
<state id="FED_terminal"><!-- node-size-and-position x=390.0 y=160.0 w=100.0 h=100.0 -->
  <transition event="IRT_terminal" target="FED_bcvm"></transition>
</state>
</scxml>
  
```

Рисунок 78. XML представление BCVMTest.

По XML представлению генерируются исходные коды модели (.h, .cpp, .fed файлы).
 Далее представлено сравнение сгенерированных исходных кодов модели (**рисунки 80 и 82**) и
 исходных кодов написанных вручную (**рисунки 79 и 81**).

```

;; BasicTest

(Fed
  (Federation BasicTest)
  (FedVersion v1516)
  (Federate "receiver" "Public")
  (Federate "sender" "Public")
  (Spaces
  )
  (Objects
  )
  (Interactions
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class senderMessage RELIABLE TIMESTAMP
        (Sec_Level "Public")
        (Parameter senderX)
        (Parameter senderId)
      )
    )
  )
)

```

Рисунок 79. Код basicstest.fed из BasicTest, написанный вручную.

```

;; basicstest

(Fed
  (Federation basicstest)
  (FedVersion v1516)
  (Federate "FED_receiver" "Public")
  (Federate "FED_sender" "Public")
  (Spaces
  )
  (Objects
  )
  (Interactions
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class IRT_senderMessage RELIABLE TIMESTAMP
        (Sec_Level "Public")
        (Parameter IRT_senderX)
        (Parameter IRT_senderId)
      )
    )
  )
)

```

Рисунок 80. Сгенерированный код basicstest.fed из BasicTest.

```

;; BCVMTest

(Fed
  (Federation BCVMTest)
  (FedVersion v1516)
  (Federate "bcvm" "Public")
  (Federate "terminal" "Public")
  (Spaces
  )
  (Objects
  )
  (Interactions
    (Class InteractionRoot RELIABLE RECEIVE
      (Class bcvmMessage RELIABLE TIMESTAMP
        (Sec_Level "Public")
        (Parameter bcvmX)
        (Parameter bcvmId)
      )
      (Class terminalMessage RELIABLE TIMESTAMP
        (Sec_Level "Public")
        (Parameter terminalX)
        (Parameter terminalId)
      )
    )
  )
)

```

Рисунок 81. Код bcvmtest.fed из BCVMTest, написанный вручную.

```

;; bcvmtest

(Fed
  (Federation bcvmtest)
  (FedVersion v1516)
  (Federate "FED_bcvm", "Public")
  (Federate "FED_terminal", "Public")
  (Spaces
  )
  (Objects
  )
  (Interactions
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class IRT_bcvmMessage RELIABLE TIMESTAMP
        (Sec_Level "Public")
        (Parameter IRT_bcvmX)
        (Parameter IRT_bcvmId)
      )
    )
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class IRT_terminalMessage RELIABLE TIMESTAMP
        (Sec_Level "Public")
        (Parameter IRT_terminalX)
        (Parameter IRT_terminalId)
      )
    )
  )
)

```

Рисунок 82. Сгенерированный код bcvmtest.fed из BCVMTest.

В качестве примера генерации исходного кода модели приведем код файла sender.h из модели BasicTest (соответственно вручную написанный и сгенерированный вариант) на рисунках 83 и 84.

```
#ifndef SENDER_H
#define SENDER_H

#include "federate.h"

class Sender : public Federate {
private:
    rti1516::ParameterHandleValueMap senderMessageMap;
    rti1516::InteractionClassHandle senderMessageH;
    rti1516::ParameterHandle senderXH;
    rti1516::ParameterHandle senderIdH;

    int senderX;

    void getHandles();
    void customInit();
    void loop();

public:
    Sender( std::wstring _federationName, std::wstring _fddName );

    //=====
    // FEDERATE AMBASSADOR CALLBACKS
    //=====
};

#endif // SENDER_H
```

Рисунок 83. Вручную написанный код sender.h из BasicTest.

```

#ifndef SM_STATE_FED_SENDER_H
#define SM_STATE_FED_SENDER_H

#include "sm_state.h"
#include "federate.h"

class SM_State_FED_sender : public SM_State, Federate
{
public:

    //-----
SM_State_FED_sender (std::wstring _federationName, std::wstring _fddName);

    //-----
~SM_State_FED_sender ();

    //-----

    //-----

// void onentry ();

protected:

private:

    //-----

    //-----
// copy constructor not implemented
SM_State_FED_sender( const SM_State_FED_sender& );

```



```

// assignment operator not implemented
SM_State_FED_sender& operator=( const SM_State_FED_sender& );

//-----
rti1516::ParameterHandleValueMap IRT_senderMessageMap;
    rti1516::InteractionClassHandle IRT_senderMessageH;
    rti1516::ParameterHandle IRT_senderXH;
    rti1516::ParameterHandle IRT_senderIdH;

    int IRT_senderX;
void getHandles();
void customInit();
void loop();

//-----
}; // SM_State_FED_sender

//-----
#endif

```

Рисунок 84. Сгенерированный код FED_sender.h из BasicTest.

4.4.2 Вывод

Приведённые выше результаты функционального тестирования показали, что средства трансляции UML в исходные коды модели совместимых со стандартом HLA работают корректно. При трансляции создаются файлы в форматах .h, .cpp, .fed эквивалентные написанным вручную с точностью до переименования федератов и именования взаимодействия федератов.

Заключение

В результате выполнения работ по второму этапу НИР были получены следующие результаты:

1. Детализованы требования к среде выполнения моделей компонентов PBC PB (см. раздел 1.2).
2. Разработана архитектура среды выполнения моделей компонентов PBC PB (см. разделы 1.1, 1.3 и 1.4).
3. Проведён обзор алгоритмов временной синхронизации и выбран наиболее перспективный для реализации смешанный консервативно-оптимистический алгоритм (см. раздел 1.5).
4. Проведён обзор специализированных языков моделирования. За основу языка описания PBC PB взят универсальный язык моделирования UML, расширенный возможностью описания триггеров на языке C++ (см. раздел 2).
5. Реализованы средства преобразования трасс из формата стенда ПНМ в формат OTF, проведено экспериментальное исследование трасс PBC PB для форматов TRC и OTF (см. разделы 3.1 и 4.1).
6. Реализовано и проведено экспериментальное исследование средства трансляции UML во временные автоматы UPPAAL (см. разделы 3.2 и 4.2).
7. Проведена экспериментальная оценка применимости системы CERTI для моделирования PBC PB (см. раздел 4.3).
8. Реализованы средства трансляции UML в исполняемые модели, совместимые со стандартом HLA, проведено экспериментальное исследование применимости средств (см. разделы 3.3 и 4.4).

Ниже детализированы полученные результаты.

В рамках данной работы выбор требований к среде выполнения моделей компонентов PBC PB обусловлен необходимостью сопряжения с аппаратурой в реальном времени по каналам бортовых интерфейсов. Также среда выполнения моделей должна быть рассчитана на интеграцию с моделями, разработанными сторонними организациями.

Исходя из выработанных требований, разрабатываемая среда выполнения имитационных моделей должна быть сформирована вокруг концепций, заложенных в стандарт распределённого моделирования HLA. Из имеющихся открытых реализаций для построения новой среды выполнения моделей за основу взята среда CERTI RTI. Из-за того,

что не все предъявляемые требования описаны спецификациями стандарта HLA, в новую среду выполнения необходимо добавить дополнительный уровень для передачи данных и расширить функциональность, предоставляемую сервисами RTI с помощью системы, основанной на стандарте DDS.

Поскольку для разрабатываемой среды выполнения моделей актуально проведение распределённого моделирования, особенно значимой становится задача синхронизации времени между процессорами, на которых выполняются модели компонентов PBC PB. В результате проведённого обзора алгоритмов временной синхронизации были выделены две основные схемы: консервативная и оптимистическая. На данный момент реализация CERTI RTI поддерживает лишь консервативную схему продвижения модельного времени. Однако задача моделирования PBC PB в реальном времени может быть решена более эффективно с использованием смешанной схемы продвижения времени, объединяющей преимущества консервативной и оптимистической схем. Внедрение смешанной схемы продвижения модельного времени требует разработки дополнительных средств анализа имитационной модели. Перед данными средствами будет поставлена задача эффективного распределения участников моделирования по группам и определение для них оптимальных алгоритмов временной синхронизации. Стандарт HLA IEEE-1516 2000 не является преградой для внедрения такой смешанной схемы. Таким образом, интеграция новых алгоритмов продвижения модельного времени в реализацию CERTI RTI не потребует модификации стандарта.

Обзор специализированных языков моделирования показал отсутствие необходимого языка для удобного описания HLA-совместимых моделей. Поэтому был сделан выбор в пользу создания собственного инструмента для описания моделей на HLA, при этом за основу берутся диаграммы состояний UML. Дополнительная логика, связанная с низкоуровневой реализацией компонентов или со специальными алгоритмами моделирования, может быть написана на языке C++. На основании такого графического описания, возможно, дополненного кодом на C++, порождается код HLA-совместимой модели на C++.

В ходе исследования форматов хранения трасс рассмотрены форматы OTF, OTFz TRC и TAU. Эксперименты показали, что формат TAU, практически не приспособлен для хранения данных переменного размера. Эксперименты по сравнению размеров трасс в различных форматах выявили явное преимущество формата OTFz перед остальными форматами. Эксперименты по времени поиска событий в трассе показали, что форматы OTF

и OTFz в среднем на 10-12% быстрее формата TRC. Исходя из результатов экспериментов, наиболее предпочтительным представляется применение формата OTFz.

В результате проделанной работы по созданию средства трансляции UML во временные автоматы UPPAAL удалось создать новую систему моделирования и верификации PBC PB, описанных в виде диаграмм UML. Главное достоинство разработанной системы состоит в том, что с ее помощью удалось объединить два известных, общедоступных и хорошо зарекомендовавших себя на практике программно-инструментальных средства проектирования и анализа сложных информационных систем: ArgoUML – графическое средство разработки и моделирования диаграмм UML, и UPPAAL – программно-инструментальное средство моделирования и верификации сетей временных автоматов. Благодаря этому, появилась возможность не только использовать диаграммы UML для описания устройства и поведения PBC PB, но и применять математическую модель временных автоматов и язык темпоральной логики для проверки и строгого доказательства свойств корректности и безопасности поведения проектируемых PBC PB, описанных при помощи диаграмм UML. Исследование алгоритма и средства трансляции UML во временные автоматы показало, что автоматическая трансляция диаграмм состояний UML в системы автоматов UPPAAL работает корректно. Результаты, полученные в статье [52] при помощи ручного преобразования были полностью воспроизведены при автоматической трансляции. Проведенные эксперименты показали, что созданная система эффективно применима для проверки правильности поведения сравнительно простых распределенных систем, состоящих из нескольких взаимодействующих процессов.

Вместе с тем, результаты экспериментальных исследований показали, что для эффективного применения к сложным PBC PB система нуждается в дальнейшем совершенствовании. Основной недостаток построенного двухступенчатого транслятора UML→UPPAAL состоит в том, что количество временных автоматов в сети, которая формируется на выходе транслятора, значительно превосходит число взаимодействующих процессов в исходном UML описании проектируемой PBC PB. Это объясняется тем, что многие временные автоматы в полученной сети играют лишь вспомогательную роль, обеспечивая корректность перехода от иерархического, многоуровневого описания PBC PB посредством диаграмм UML к одноуровневому, «плоскому» описанию той распределенной системы, но уже посредством сети временных автоматов, взаимодействующих друг с другом. Поскольку быстродействие UPPAAL имеет тенденцию стремительно понижаться с ростом числа временных автоматов в сети, можно предполагать, что вычислительные

возможности процедур верификации, заложенных в UPPAAL, окажутся недостаточными для проверки правильности поведения описаний PBC PB, состоящих из большого числа диаграмм UML. Для преодоления этой трудности на следующих этапах проекта планируется разработать и реализовать алгоритм оптимизации сети временных автоматов, конструируемой на выходе транслятора UML→UPPAAL. Предполагается, что этот алгоритм сможет выделять группы временных автоматов, совместное функционирование которых осуществляется по последовательным схемам передачи управления, и объединять автоматы одной группы в один последовательно работающий временной автомат. Есть основания полагать, что этот прием позволит сократить размер сети автоматов, получаемых на выходе построенного нами транслятора UML→UPPAAL, и сделает нашу систему пригодной для анализа поведения PBC PB, состоящих из большого числа процессов.

Результаты, показанные базовой реализацией системы моделирования CERTI, позволяют использовать её в данном виде лишь для полунатурного моделирования довольно простых PBC PB. Однако её производительность может быть существенно увеличена, например, с помощью изменения существующей структуры процессов распределённой системы, использования более эффективных способов передачи данных и перераспределения функциональности между глобальными и локальными компонентами RTI, а также с помощью использования смешанных алгоритмов синхронизации. В ходе работы с RTI проекта CERTI в данном дистрибутиве было выявлено десять критических ошибок. Найденные ошибки были исправлены, а предложенные изменения исходного кода согласованы с разработчиками проекта. Кроме того, одна из разработанных имитационных моделей была включена в состав набора для функционального тестирования дистрибутива CERTI. Экспериментальное сравнение с системой моделирования «Стенд ПНМ» показало преимущество системы моделирования «Стенд ПНМ». Это неудивительно, учитывая, что она была изначально ориентирована на исследование свойств PBC PB. Кроме того, система «Стенд ПНМ» является параллельной системой моделирования, исключаяющей удалённое расположение участников. В то же время отставание текущей версии RTI CERTI не является критическим и, как считают авторы настоящего отчёта, может быть преодолено способами, представленными выше.

Создание транслятора диаграммам состояний UML в исполняемые модели совместимые со стандартом HLA позволит существенно сократить время разработки HLA, а также позволит по одному описанию PBC PB проверять как логические свойства PBC PB, так и оценивать её производительность. Полностью автоматическую трансляцию из

диаграмм состояний UML в исполняемые модели совместимые со стандартом HLA на данном этапе работы не удалось получить, так как используемый для построения диаграмм состояний UML редактор ArgoUML использует для описания диаграмм состояний формат XMI. Поэтому необходимо создание средства для перевода представления XMI в формат SCXML, используемый генератором кода модели. Другим вариантом решения данной проблемы является использование другого редактора UML, в котором возможно сохранение диаграммы в формате SCXML.

В рамках работы данного этапа также были сформулированы задачи, требующие дальнейших экспериментов и исследования:

- Создание федерата, обеспечивающего трассировку событий и запись трассы в формате OTF, а также интеграция его с RTI CERTI.
- Интеграция средства визуализации трасс Vis из системы моделирования «Стенд ПНМ» с трассой в формате OTF.
- Разработка и реализация алгоритма смешанной схемы продвижения модельного времени и интеграция его в RTI CERTI.
- Разработка и реализация алгоритма оптимизации сети временных автоматов, конструируемой на выходе транслятора UML→UPPAAL.
- Проведение обзора свободно распространяемых UML-редакторов с целью выбора возможной замены ArgoUML.
- Создание средства трансляции описаний диаграммы состояний в формате XMI в представление на SCXML, используемое генератором кода HLA-совместимой модели.
- Интеграция разработанных на этом этапе средств в единую среду моделирования PBC PB.
- Разработка модели PBC PB и исследование её при помощи разработанной среды моделирования PBC PB.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Отчёт о научно-исследовательской работе «Создание прототипа интегрированной среды и методов комплексного анализа функционирования распределённых вычислительных систем реального времени (РВС РВ)» (Этап 1) // М.:, 2010. - Стр. 65
2. Замятина, Е. Б.. Современные теории имитационного моделирования: специальный курс. ПГУ, Пермь, 2007.
3. Савенков К.О., Смелянский Р.Л., Масштабирование дискретно-событийных имитационных моделей // Программирование, 2006, No. 6, стр. 14-26.
4. V.V. Balashov , A.G. Bakhmurov, V.A. Balakhanov, M.V. Chistolinov, P.E. Shestov, R.L. Smeliansky, N.V. Youshchenko. Tools for monitoring of data exchange in real-time avionics systems A Hardware-in-the-Loop Simulation Environment for Integration of Distributed Avionics Systems // Proc. 4th EUCASS European Conference for Aerospace Sciences, Saint_Petersburg, Russia, 2011. - Электрон. флеш-диск (Flash).
5. Балашов В.В., Бахмуров А.Г., Волканов Д.Ю., Смелянский Р.Л., Чистилинов М.В., Ющенко Н.В. Стенд полунатурного моделирования для разработки встроенных вычислительных систем // Методы и средства обработки информации: Третья Всероссийская научная конференция. Труды конференции. - М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова; МАКС Пресс, 2009. - С.16-25.
6. Open DDS // Официальный сайт проекта Portico. URL: <http://www.ociweb.com/> (дата обращения 13.07.2011).
7. E. Noulard, J.-Y. Rousselot. CERTI, an Open Source RTI, why and how. Toulouse, France, 2009.
8. Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules // IEEE, 2010 с. 26.
9. D'Ausbourg B., Noulard E., Siron P. Running real time distributed simulations under Linux and CERTI // In Proceedings of European Simulation Interoperability Workshop - EURO SIW 2008, 16-19 June 2008, Edimburgh, Scotland.
10. ГОСТ Р 52070–2003. Интерфейс магистральный последовательный системы электронных модулей. – Введ. 01.01.2004 – М. : Изд-во стандартов, 2001. – 23 с.
11. Martin Leucker, Christian Schallhart. A brief account of runtime verification. Journal of Logic and Algebraic Programming, Volume 78, Issue 5, The 1st Workshop on Formal

- Languages and Analysis of Contract-Oriented Software (FLACOS'07), May-June 2009, pp. 293-303.
12. Кларк Э., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. — М.: Издательство Московского центра непрерывного математического образования, 2002.
 13. Noulard E. Rousselot J.-Y. CERTI, an Open Source RTI, why and how // Spring Simulation Interoperability Workshop. San Diego, USA, 2009.
 14. Кодд Э.Ф. Реляционная модель данных для больших совместно используемых банков данных. СУБД № 1. 1995.
 15. Bieber P., Raujol D., Siron P. Security Architecture for Federated Cooperative Information Systems // in Proceedings of 16th Annual Computer Security Applications Conference, Toulouse, France, 2002.
 16. Chaudron J.B., Noulard E., Siron P. Design and modeling techniques for real-time RTI time management // In Proceedings of Spring Simulation Multiconference - SpringSim'11, 04-08 April 2011, Boston, USA.
 17. Malinga, L and Le Roux. HLA RTI performance evaluation // European Simulation Interoperability Workshop, Istanbul, Turkey, 13-16 July, 2009. P. 1-6
 18. Bréholée B., Siron P. CERTI: Evolutions of the ONERA RTI Prototype // In Proceedings of the Fall 2002 Simulation Interoperability Workshop, Orlando, USA, 2002.
 19. Stokes J. Introduction to Multithreading, Superthreading and Hyperthreading [ARS] (<http://arstechnica.com/old/content/2002/10/hyperthreading.ars>) (дата обращения: 13.07.2011).
 20. V.V. Balashov, A.G. Bakhmurov, M.V. Chistolinov, R.L. Smeliansky, D.Yu. Volkanov, N.V. Youshchenko. A Hardware-in-the-Loop Simulation Environment for Real-Time Systems Development and Architecture Evaluation // In Proc. of the Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008, Szklarska Poreba, Poland, June 26-28 2008.
 21. Object Management Group; Object Interface Systems, Inc; Real-Time Innovations, Inc; THALES, Data Distribution Service for Real-time Systems, version 1.2. 2007.
 22. Казаков Ю.П., Смелянский Р.Л. Об организации распределенного имитационного моделирования // Программирование, 1994, No. 2, стр. 45-64

23. Simulation Interoperability Standards Committee of the IEEE Computer Society IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification. 2000.
24. Fujimoto R.D. Parallel and Distributed Simulation Systems. Wiley Interscience. 2000.
25. Liu B., Yao Y., Wang H. An Efficient Algorithm in the HLA Time Management // In Proceedings of the Winter Simulation Conference, 2007.
26. Nance R.E. A history of discrete event simulation programming languages. Blacksburg, United States: Virginia Polytechnic Institute and State University, 1993.
27. Modelica - A Unified Object-Oriented Language for Physical Systems Modeling Tutorial [PDF] (<https://www.modelica.org/documents/ModelicaTutorial14.pdf>) (дата обращения: 13.07.2011).
28. James O. Henriksen SLX - THE X IS FOR EXTENSIBILITY // Wolverine Software Corporation 2111 Eisenhower Avenue, Suite 404 Alexandria, VA 22314-4679, U.S.A. 2000. [PDF] (<http://www.wolverinesoftware.com/SLX00.pdf>) (дата обращения: 13.07.2011).
29. Гома Х. UML. Проектирование систем реального времени, распределенных и параллельных приложений. Издательство ДМК, 2011 год, 704 стр.
30. D. Harel. Statecharts: A Visual Formalism for Complex Systems. Sci. Comput. Programming 8 (1987), pp. 231-274.
31. Behaviour modelling with Stateflow/Simulink Tutorial [PDF] (http://www.md.kth.se/mmk/gru/mda/mf2008/Tutorial%20-%20StateFlow_v3.pdf) (дата обращения: 13.07.2011).
32. OMG Unified Modeling Language Specification [PDF] (<http://www.omg.org/spec/UML/1.4/PDF>) (дата обращения: 13.07.2011)
33. Statechart Diagram [pdf] (<http://santos.cis.ksu.edu/771-Distribution/Reading/uml-section3.73-94.pdf>) (дата обращения: 13.07.2011)
34. Г. Буч, А. Якобсон, Дж. Рамбо. UML. Классика CS. Издание второе. Питер, 2006, 736 стр.
35. Simulink Tutorial [PDF] (http://academic.csuohio.edu/dong_1/EEEC510/material/Simulink%20Tutorial.pdf) (дата обращения: 13.07.2011)

36. Бахмулов А.Г., Чистилинов М.В. Среда моделирования многопроцессорных вычислительных систем. // Программные системы и инструменты №1. Москва: МАКС Пресс, 2000. С. 42-47.
37. Смелянский Р.Л. Анализ производительности распределенных микропроцессорных вычислительных систем на основе инварианта поведения программ. / Дисс. на соискание ученой степени доктора физико-математических наук. М.:МГУ, 1990.
38. Грибов Д.И., Смелянский Р.Л. Комплексное моделирование бортового оборудования летательного аппарата // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. - М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005. - С.59-74
39. Howard Bowman, Giorgio P. Faconti, Joost-Pieter Katoen, Diego Latella, and Mieke Massink. Automatic verification of a lip synchronisation algorithm using UPPAAL. In Bas Luttik Jan Friso Groote and Jos van Wamel, editors, In Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems. Amsterdam , The Netherlands, 1998.
40. Alexandre David and Wang Yi. Modelling and analysis of a commercial field bus protocol. In Proceedings of the 12th Euromicro Conference on Real Time Systems, pages 165–172. IEEE Computer Society, 2000.
41. Klaus Havelund, Kim G. Larsen, and Arne Skou. Formal verification of a power controller using the real-time model checker UPPAAL. 5th International AMAST Workshop on Real-Time and Probabilistic Systems, available at <http://www.UPPAAL.com>, 1999.
42. Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 2–13, December 1997.
43. Torsten K. Iversen, Kåre J. Kristoffersen, Kim G. Larsen, Morten Laursen, Rune G. Madsen, Steffen K. Mortensen, Paul Pettersson, and Chris B. Thomasen. Modelchecking real-time control programs — Verifying LEGO mindstorms systems using UPPAAL. In Proc. of 12th Euromicro Conference on Real-Time Systems, pages 147–155. IEEE Computer Society Press, June 2000.
44. Arne Skou Klaus Havelund, Kim Guldstrand Larsen. Formal verification of a power controller using the real-time model checker UPPAAL. In 5th Int. AMAST Workshop on

- Real-Time and Probabilistic Systems, volume 1601 of Lecture Notes in Computer Science, pages 277–298. Springer–Verlag, 1999.
45. D.P.L. Simons and M.I.A. Stoelinga. Mechanical verification of the IEEE 1394a root contention protocol using UPPAAL2k. Springer International Journal of Software Tools for Technology Transfer, pages 469–485, 2001.
 46. F.W. Vaandrager and A.L. de Groot. Analysis of a biphasic mark protocol with UPPAAL and PVS. Technical Report NIII-R0445, Radboud University of Nijmegen, 2004.
 47. Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. Int. Journal on Software Tools for Technology Transfer, 1(1–2):134–152, October 1997.
 48. Alexandre David, Gerd Behrmann, Kim G. Larsen, and Wang Yi. A tool architecture for the next generation of UPPAAL. In 10th Anniversary Colloquium. Formal Methods at the Cross Roads: From Panacea to Foundational Support, LNCS, 2003.
 49. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Proc. of Int. Colloquium on Algorithms, Languages, and Programming, volume 443 of LNCS, pages 322–335, 1990.
 50. Thomas A. Henzinger. Symbolic model checking for real-time systems. Information and Computation, 111:193–244, 1994.
 51. D. Harel, A. Naamad. The STATEMATE semantics of statecharts. ACM Trans. Softw. Eng. Methodol, 1996. vol. 5(4), 293-333.
 52. A. David, M. Moller Oliver, Wang Yi. Verification of UML Statechart with Real-Time Extensions / Uppsala: Department of Information Technology, Uppsala University. IT Technical Report 2003-009, 2003.
 53. ArgoUML site [HTML] (<http://argouml.tigris.org/>) (дата обращения: 13.07.2011)
 54. State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Working Draft 26 April 2011 [HTML] (<http://www.w3.org/TR/scxml/>) (дата обращения: 13.07.2011)
 55. XMI Mapping, V2.1.1, 2007 [HTML] (<http://www.omg.org/spec/XMI/2.1.1/>) (дата обращения: 13.07.2011)
 56. JGraphX User Manual [HTML] (http://www.jgraph.com/doc/mxgraph/index_javavis.html) (дата обращения: 13.07.2011)
 57. Gnosis Utilities 1.2.1 [HTML] (<http://pypi.python.org/pypi/Gnosis%20Utilities>) (дата обращения: 13.07.2011)

58. The Document Object Model API [HTML] (<http://docs.python.org/library/xml.dom.html>)
(дата обращения: 13.07.2011)
59. Cheetah Users' Guide [PDF] (http://www.cheetahtemplate.org/docs/users_guide.pdf) (дата обращения: 13.07.2011)
60. Cohen D., On Holy Wars and a Plea for Peace, [TXT]
(<http://www.ietf.org/rfc/ien/ien137.txt>) (дата обращения: 13.07.2011)
61. Condor Engineering, Inc., MIL-STD-1553 Tutorial. 2000.