Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М.В. ЛОМОНОСОВА
(ФАКУЛЬТЕТ ВМК МГУ)

УДК № госрегистрации Инв. №	УТВЕРЖДАЮ декан академик РАН _____ Е.И. Моисеев «___» _____ 2012 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)»

(итоговый, этап № 5)

Наименование этапа: «Разработка методики применения созданных методов и средств, создание экспериментального образца стенда, апробация интегрированной среды»

ТОМ I

Руководитель работ, д.ф.-м.н. _____ Смелянский Р.Л.
чл.-корр. РАН, профессор подпись, дата

Москва 2012

Обозначения и сокращения

АГЭА	Адаптивный гибридный эволюционный алгоритм
БВС	Бортовая вычислительная система
БИ	Бортовой интерфейс
БК	Бортовой канал
БПМ	Библиотека поддержки моделирования
БЦВМ	Бортовая цифровая вычислительная машина
ВД	Визуализатор диаграмм
ВН	Внесение неисправностей
ГПО	Генератор потока отказов
КБИ	Каналы бортовых интерфейсов
КБО	Комплекс бортового оборудования
МКИО	Мультиплексный канал информационного обмена
МОО	Механизм обеспечения отказоустойчивости
НИР	Научно-исследовательская работа
ОС	Операционная система
ОС РВ	Операционная система реального времени
ООП	Объектно-ориентированное программирование
ПНМ	Полунатурное моделирование
ПО	Программное обеспечение
РВС РВ	Распределённая вычислительная система реального времени
РИМ	Распределённое имитационное моделирование
РЛС	Радиолокационная система
РЧМ	Распределённая частная модель
СМ	Система моделирования
СММ КБО	Стенд математического моделирования комплексов бортового оборудования
ТЗ	Техническое задание
ЧМ	Частная модель
ЭА	Эволюционный алгоритм
ЯОМ	Язык Описания Моделей

ACSL	Язык моделирования непрерывных процессов (Advanced Continuous Simulation Language)
AFAP	Насколько возможно быстро (As Fast As Possible)
API	Интерфейс программирования приложений (Application Programming Interface)
ASCII	Американская стандартная таблица кодировки печатных символов (American Standard Code for Information Interchange)
BSD	Лицензия университета Беркли на распространение программного обеспечения (Berkley Software Distribution)
CRC	Центральный компонент RTI (Central RTI Component)
CSV	Значения, разделенные запятыми (Comma-Separated Values)
DIS	Распределенное интерактивное моделирование (Distributed Interactive Simulation)
DDR	Удвоенная скорость передачи данных (Double Data Rate)
DTEL	Метаязык для кодирования данных (Data Type Encoding Language)
FOM	Федеративная объектная модель (Federation Object Model)
GPL	Стандартная общественная лицензия (General Public License)
GPSS	Система моделирования общего назначения (General Purpose Simulation System)
GUI	Графический пользовательский интерфейс (Graphical User Interface)
HLA	Высокоуровневая архитектура (High Level Architecture)
HTA	Иерархический временной автомат (Hierarchical Timed Automata)
IPET	Метод неявного перебора путей (implicit path enumeration techniques)
LRC	Локальный компонент RTI (Local RTI Component)
LTS	Размеченная система переходов (Labeled Transition System)
MT-CERTI	Многопоточная версия системы CERTI (Multi-Threaded CERTI)
NVP	N-версионное программирование
OMT	Шаблон объектной модели федерации (Object Model Template)
OTF	Открытый формат трасс (Open Trace Format)
QoS	Качество обслуживания (Quality of Service)
RAM	Оперативная память (Random Access Memory)
RAP	Задача выбора МОО РВС РВ (Reliability Allocation Problem)
RTI	Среда имитационного моделирования (RunTime Interface)
RTIA	Процесс, являющийся частью локального компонента RTI в системе CERTI (RTI Ambassador)

RTIG	Процесс, являющийся центральным компонентом RTI в системе CERTI (RTI Gate)
	RTI Gate (RTIG), RTI Ambassador (RTIA) и libRTI
SCXML	Расширяемый язык разметки для диаграмм состояний (State Chart eXtensible Markup Language)
SLX	Расширяемый язык моделирования (Simulation Language with eXtensibility)
SOM	Имитационная объектная модель (Simulation Object Model)
SSD	Твердотельный накопитель (Solid-State Drive)
TCP	Протокол управления передачей (Transmission Control Protocol)
TCTL	Логика ветвящегося времени с таймерами (Timed Computational Tree Logic)
UDP	Протокол пользовательских дейтаграмм (User Datagram Protocol)
UML	Универсальный язык разметки (Universal Markup Language)
WAN	Глобальная компьютерная сеть (Wide Area Network)
WCET	Наихудшее время выполнения программы (Worst-case Execution Time)
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 пятого этапа НИР была разработка методики применения созданных и инструментальных средств, создание экспериментального образца стенда, апробация интегрированной среды. Основное содержание работ по пятоому этапу следующее: разработка методики совместного применения созданных методов и инструментальных средств для поддержки разработки и интеграции РВС РВ; создание экспериментального образца стенда полунатурного моделирования и интеграции РВС РВ; апробация интегрированной среды на стенде; разработка программы внедрения результатов НИР в образовательный процесс; подготовка научно-методических материалов для учебных материалов по тематике проекта объёмом 28 академических часов.

Результатом работы по пятому этапу является: итоговый отчёт о НИР. Итоговый отчёт о НИР включает в себя: описание выбора класса РВС РВ; требования к создаваемым методам и средствам; средства описания РВС РВ; описание архитектуры инструментальных средств поддержки анализа и разработки РВС РВ; описание разработанных методов и интеграция со сторонними средствами; описание исследуемых моделей РВС РВ; описание экспериментального образца стенда полунатурного моделирования и интеграции РВС РВ; методику совместного применения созданных методов и инструментальных средств для поддержки разработки и интеграции РВС РВ; описание апробации интегрированной среды на стенде; программу внедрения результатов НИР в образовательный процесс; разработанные учебные материалы.

Все задачи, поставленные в рамках пятого этапа НИР, выполнены.

Содержание

Введение

Настоящий документ представляет собой научно-технический итоговый отчет по НИР «Создание прототипа интегрированной среды и методов комплексного анализа функционирования распределённых вычислительных систем реального времени (РВС РВ)». Документ содержит отчет по пунктам 5.1-5.4 календарного плана пятого этапа, 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, а также описание основных результатов, полученных на предыдущих этапах работы [1-4].

В первой главе приводится описание выбранного класса РВС РВ.

Во второй главе описываются требования к создаваемым методам и средствам.

В третьей главе приводятся средства описания РВС РВ.

В четвёртой главе описывается архитектура инструментальных средств поддержки анализа и разработки РВС РВ.

В пятой главе приводится описание разработанных методов и интеграция со сторонними средствами.

В шестой главе описываются исследуемые модели РВС РВ.

В седьмой главе в соответствии с пунктом 5.2 календарного плана ТЗ приводится описание экспериментальный образец стенда полунатурного моделирования и интеграции РВС РВ.

В восьмой главе в соответствии с пунктом 5.1 календарного плана ТЗ описывается методика совместного применения созданных методов и инструментальных средств для поддержки разработки и интеграции РВС РВ.

В девятой главе в соответствии с пунктом 5.3 календарного плана ТЗ приводится описание апробации интегрированной среды на стенде.

В десятой главе в соответствии с пунктом 5.4 календарного плана ТЗ приводится программа внедрения результатов НИР в образовательный процесс.

В одиннадцатой главе в соответствии с пунктом 5.5 календарного плана ТЗ приводятся описание разработанных учебных материалов.

В заключении изложены основные результаты НИР.

1 Выбор класса РВС РВ

В данном разделе описывается класс вычислительных систем – распределённые вычислительные системы реального времени (РВС РВ). Разрабатываемый прототип интегрированной среды моделирования предназначен для комплексного анализа РВС РВ. В разделе 1.1 приводятся особенности РВС РВ. В разделе 1.2 описывается типовая архитектура РВС РВ. В разделе 1.3 рассматриваются особенности процесса разработки РВС РВ.

1.1 Особенности РВС РВ

Из всего многообразия вычислительных систем в этой работе будем рассматривать класс распределённых вычислительных систем реального времени (РВС РВ). Под РВС РВ будем понимать такую вычислительную систему, узлы которой распределены в пространстве, а правильность работы зависит не только от логических результатов вычислений, но и от промежутка времени, за который эти результаты были получены [5].

РВС РВ включают в себя широкий спектр систем различной степени сложности. Примерами таких систем служат бортовые системы самолетов и автоматических авиадиспетчеров, автоматизированные медицинские приборы, диспетчеры нефтехимического контроля, системы управления денежными переводами и электронной коммерции, сухопутные и морские системы военной навигации, различные системы в области космонавтики [6].

Обычно РВС РВ состоят из управляющей и управляемой систем. Управляемая система – это окружение, с которым взаимодействует компьютер. Управляющая система взаимодействует со своим окружением посредством информации об этом окружении, полученной различными датчиками. Необходимо, чтобы состояние окружения, воспринимаемое управляющей системой, совпадало с действительным состоянием окружения, иначе действия управляющей среды могут привести к пагубным последствиям. На управляющей системе в бесконечном цикле выполняются задачи взаимодействия с управляемой системой. Планировщик задач занимается управлением выполнения этих задач. Сложность построения РВС РВ сильно варьируется в зависимости от следующих характеристик [6]:

- **Длительность директивного срока**. В РВС РВ некоторые задачи имеют директивный срок и/или периодические ограничения на время их выполнения. В случаях сжатых директивных сроков, то есть относительно небольших временных интервалах между началом выполнения и окончанием задачи, от алгоритма планировщика требуется быстрая реакция.

Последнее означает, что алгоритм должен быть либо очень простым, либо должен обладать высокой степенью параллелизма.

- **Критичность директивного срока**. В соответствии с критичностью директивного срока для РВС РВ выделяют две группы задач – жёсткого и мягкого реального времени. Критичность директивного срока зависит от длины допустимого интервала времени, в течение которого задача может продолжать выполняться, несмотря на истечение директивного срока. Для задач жёсткого реального времени эта величина равна нулю, тогда как для задач мягкого реального времени величина может быть положительной. Для разных типов задач обычно используются кардинально отличающиеся методы. Во многих системах задачи жёсткого реального времени планируются в расписании заранее, что приводит к тому, что они гарантированно укладываются в директивный срок. В тоже время задачи мягкого реального времени зачастую динамически планируются при помощи алгоритмов, которые детально просчитывают временные ограничения, но они годятся лишь для систем с высокой производительностью; либо с помощью алгоритмов, сочетающих приоритет задачи с временными ограничениями.
- **Надёжность**. Для многих промышленных РВС РВ важнейшим требованием является надежность. В таких системах выделяются задачи, называемые *критическими*. Если такие задачи не укладываются в директивный срок, то выполнение РВС РВ может привести к техногенной аварии. Обычно для гарантированного выполнения критическими задачами в директивные сроки используются такие методы, как автономный анализ и схемы разработки, при которых ресурсы, необходимые данным задачам, резервируются специально под них, несмотря даже на то, что преимущественное большинство времени эти ресурсы будут простаивать. Однако стоит заметить, что в большом количестве систем все задачи со строгими временными ограничениями считаются критическими, в то время как лишь немногие из них являются таковыми.
- **Размер системы и степень взаимодействия задач**. РВС РВ сильно различаются по сложности и внутренней структуре. В подавляющем большинстве систем выполняемые приложения целиком загружаются в память, или, если есть четко выделяемые этапы работы системы, то загружаются этапы, причем, каждый этап загружается непосредственно перед началом своего выполнения. Во многих приложениях их подсистемы

слабо зависят друг от друга, а задачи взаимодействуют редко. Способность загружать систему целиком в память и ограничивать количество взаимодействий между задачами облегчает многие аспекты построения и анализа РВС РВ. При этом обычно РВС РВ выполняют определенные специализированные функции, и поэтому набор программного обеспечения в них ограничен.

- **Окружение**. Окружение РВС РВ играет огромную роль при проектировании системы. Некоторые окружения хорошо изучены, благодаря чему для них могут применяться алгоритмы построения расписания, которые гарантируют выполнение директивных сроков.

Обобщая вышесказанное, можно заметить, что РВС РВ выполняются, как правило, в рамках следующих ограничений:

- необходимость работы в режиме реального времени (выполнение директивных сроков);
- необходимость взаимодействия с окружающей средой;
- соблюдение ограничений на определённые ресурсы;
- применение определённого набора программ для каждой конкретной системы;
- чувствительность к отказам.

В соответствии с особенностями формируются специальные требования к архитектуре РВС РВ и процессу разработки РВС РВ.

1.2 Архитектура РВС РВ

Типичная РВС РВ состоит из следующих основных компонентов [7,8]:

- Вычислительная система, образованная одной или несколькими бортовыми цифровыми вычислительными машинами (БЦВМ). В составе БЦВМ функционирует программное обеспечение (ПО). Такое ПО подразделяется на функциональное программное обеспечение (набор программ, отвечающих за обработку данных для функциональных подсистем) и системное программное обеспечение (отвечает за обмен данными и планирование запуска функциональных программ согласно заданной циклограмме).
- Набор разнообразных датчиков и исполнительных устройств.
- Информационно-управляющее поле, состоящее из набора индикаторов и органов управления.

- Мультиплексный канал информационного обмена (МКИО), представляющего собой среду передачи данных, которая соединяет компоненты РВС РВ.

Исходя из анализа архитектуры некоторых классов РВС РВ, выполненных в статьях [7,9,10,11,12], в этом разделе сделаны выводы об основных тенденциях развития архитектуры РВС РВ:

- Современные РВС РВ представляют собой многомашинные вычислительные комплексы.
- На каждой из РВС РВ работает специфичный набор приложений и системных задач.
- Количество каналов связи между приборами в составе РВС РВ достигает нескольких десятков.
- Программное обеспечение работает в бесконечном цикле, в котором выделяется несколько подциклов для задач с различными приоритетами, а также задачи выполняющиеся, когда вычислители свободны.
- Среди основных типов каналов в таких системах выделяются: мультиплексный канал информационного обмена (МКИО) по ГОСТ 26765.52-87 (MILS1553), ГОСТ Р 52070-2003, ГОСТ Р 50832-95 (аналог технологии передачи данных STANAG 3910), Fibre Channel (типа FC-AE) и ARINC429 (ГОСТ 18977-79).

Схема обобщённой интегрированной РВС РВ приведён на рисунке 1. В качестве примера на рисунке 2 представлена структура РВС РВ зарубежного современного истребителя F-22 [13].

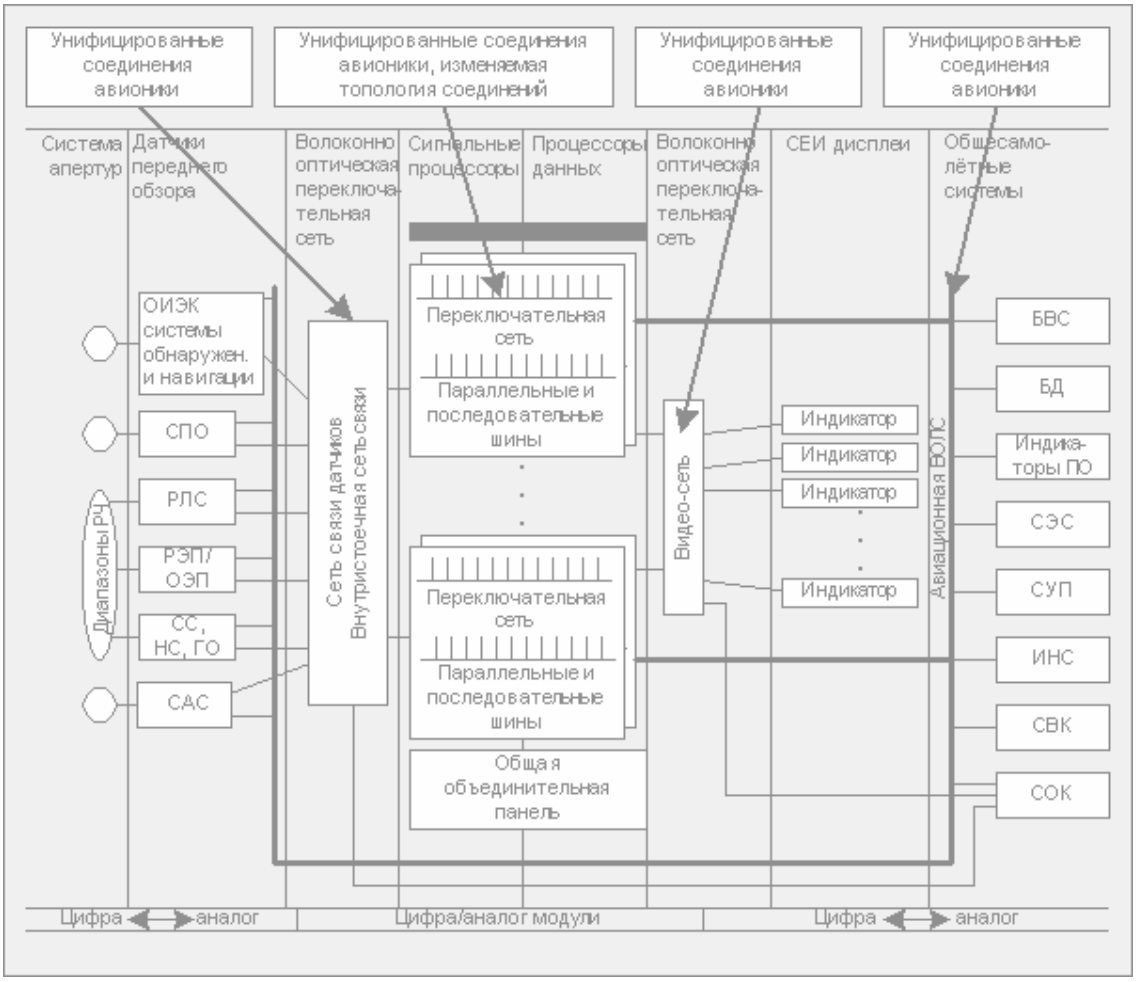

Рисунок 1. Интегрированная БВС перспективных ЛА

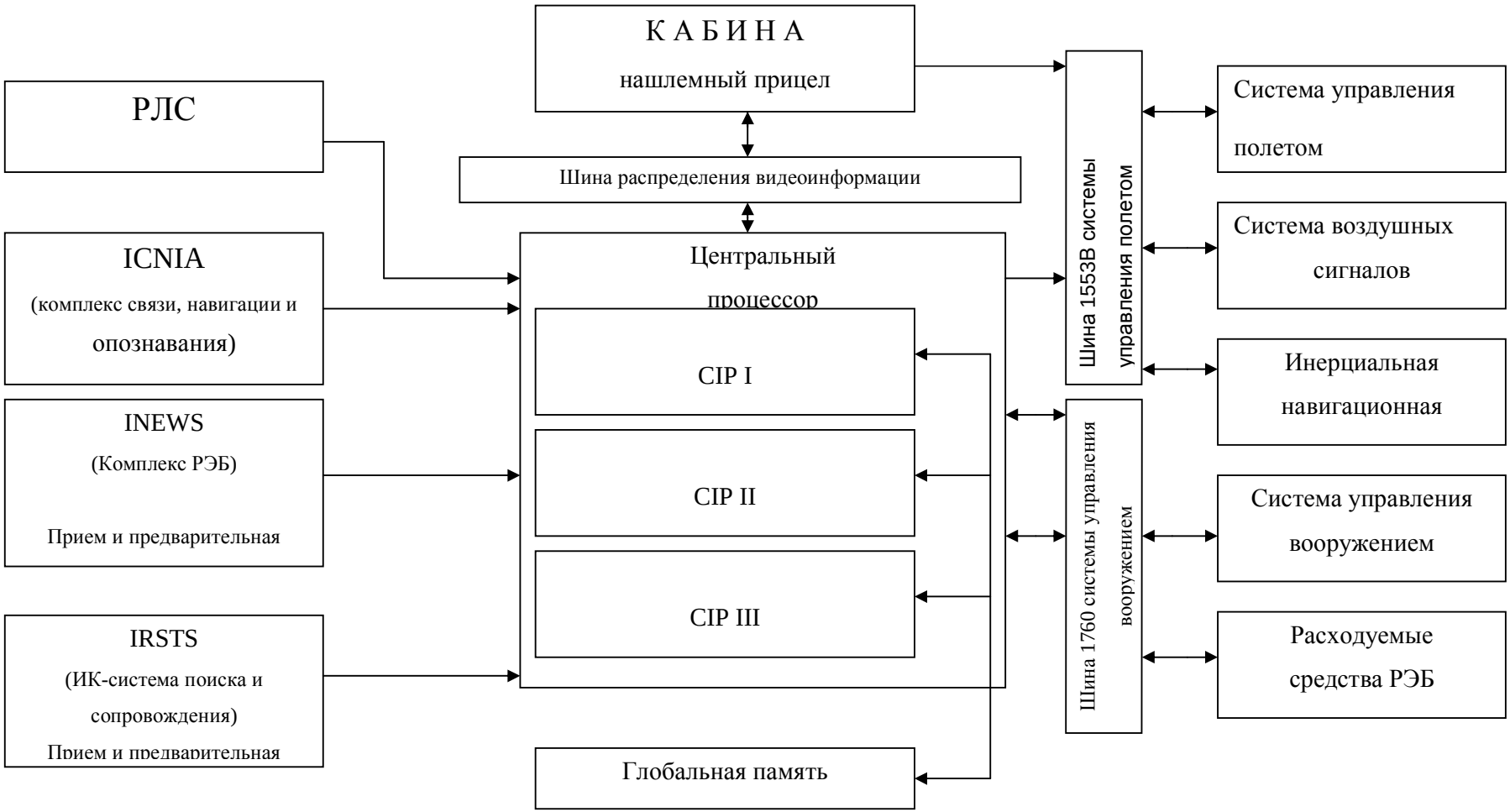

Рисунок 2. КБО самолета F-22

В состав РВС РВ самолета F-22 (Рисунок 2) входят:

- многофункциональная радиолокационная система (РЛС) AN/APG-77;
- интегрированная система связи, навигации и опознавания ICNIA (Integrated Communication, Navigation and Identification Avionics);
- интегрированный комплекс радиоэлектронной борьбы INEWS (Integrated Electronic Warfare Subsystem);
- система управления вооружением;
- система отображения информации в кабине;
- общий интегрированный процессор обработки сигналов и данных CIP (Common Integrated Processor).

Для обеспечения передачи данных и обмена информацией между элементами комплекса используются три типа шин: волоконно-оптическая высокоскоростная шина HSDB (High Speed Data Bus), цифровая шина обмена данными стандарта MILS-1553B и высокоскоростная цифровая шина управления вооружением стандарта MILS-1760. Вся аппаратура комплекса имеет модульную конструкцию и размещается в стандартных моноблоках.

Между элементами РВС РВ функции распределены следующим образом: прием и предварительная обработка сигналов проводятся непосредственно в подсистемах ICNIA, INEWS, РЛС и ИК-системе; далее сигналы поступают в процессор CIP, где производится окончательная обработка сигналов и данных, формирование отображаемой информации и целеуказания, распределение этой информации между потребителями.

Центральный процессор комплекса состоит из трех интегрированных процессоров обработки сигналов и данных CIP. Каждый процессор подключается к волоконно-оптической шине обмена данными, что обеспечивает скорость обмена информацией 400 Мбит/с, и имеет внутреннюю шину обмена данными.

Для обеспечения обмена данными между элементами подсистем разработана шина внутреннего интерфейса Pi-bus. Шина Pi-bus представляет собой линейную многоотводную синхронную шину, поддерживающую обмен сообщениями между модулями в терминале. Эта 16-разрядная шина имеет встроенные средства обнаружения ошибок и протокол обмена данными типа «ведущий-ведомый». Интерфейсный узел шины обеспечивает независимую передачу данных по двум шинам и управляет несколькими сообщениями без вмешательства процессора. Протокол обмена данными соответствует стандарту MILS -1750A. Шина содержит 29 линий обмена данными, а также линию синхронизации, работающую с тактовой частотой 12,5 МГц.

На канальном уровне шина Pi-bus имеет четыре основных типа сообщений: запись параметров, блок, передача управления и шинный интерфейс. Сообщение «запись параметра» используется для передачи параметров, содержащих от одного до трех слов, от главного устройства к подчиненному. Сообщение «блок» используется для считывания данных из подчиненного устройства в главное и для записи данных из главного в подчиненное. Сообщение «передача управления» может использоваться для непосредственной передачи управления шиной другому модулю или для начала новой фазы управления с тем же главным устройством. Сообщение «шинный интерфейс» используется для считывания или записи в регистр шинного интерфейса диапазона адресов подчиненного модуля.

В каждом процессоре CIP используется девять типов процессоров. Основными для обработки данных являются – Intel 8960, а для обработки сигналов – C31 фирмы «Texas Instruments» и 68040 фирмы «Motorola».

Между собой процессоры CIP подключены по схеме «звезда» и коммутируются посредством высокоскоростной шины обмена данными через устройства ввода-вывода (GW). В состав каждого CIP входят пять процессорных элементов обработки данных (D), процессор сигналов (PS), устройство глобальной памяти (GBM), вспомогательный процессор, работающий в дежурном режиме (L), кодирующее устройство (K), а также средства сопряжения с элементами комплекса через волоконно-оптическую шину (F) и цифровую шину стандарта 1553B. Все модули расположены в стандартном контейнере с шиной внутреннего интерфейса Pi-bus. Общее управление работой процессора и распределение программного обеспечения осуществляет сервер данных (DS). Каждый процессор CIP имеет 300 Мбайт постоянной памяти (в перспективе её планируют увеличить до 650 Мбайт), производительность процессора сигналов – 20 млрд. оп/с (планируемое наращивание до 50 млрд. оп/с) и быстродействие процессора данных – 700 млн. оп/с (планируемое наращивание до 2 млрд. оп/с).

Процессор CIP состоит из 32-х стандартных модулей SEM-E, имеет массу около 32 кг и объем около 40 куб.дм. Съемные модули имеют собственную систему жидкостного охлаждения.

1.3 Особенности процесса разработки РВС РВ

На ранних этапах разработки аппаратные и программные компоненты РВС РВ создаются параллельно. На этих этапах для разработчиков программного обеспечения доступна лишь часть прототипов аппаратных устройств, что усложняет разработку РВС РВ [14]. Далее кратко описываются проблемы, с которыми сталкиваются разработчики.

Во-первых, недоступность аппаратных устройств или их прототипов ограничивает возможности оценки временных характеристик функционирования программного обеспечения, поэтому возникает потребность для оценки этих характеристик без доступа к прототипам аппаратных устройств.

Во-вторых, стандартная ОС РВ (операционная система реального времени) включает некоторые задачи, которые реализуют логику обменов по аппаратным бортовым каналам связи, включая обработку ошибок обмена. Для активизации и тестирования этих задач без привлечения прототипов взаимодействующих устройств необходимо моделировать работу этих задач и моделировать обмен по бортовым каналам.

Также необходимо отметить, что разработка приборов, входящих в состав РВС РВ, на практике выполняется различными организациями. Готовность различных приборов к комплексированию наступает в разные моменты времени. Для соблюдения сроков комплексирования РВС РВ необходимо начинать работы по комплексированию с неполным комплектом доступных приборов.

На этапе комплексирования РВС РВ возникает ряд задач, требующих инструментальной поддержки, в том числе:

- проверка соответствия приборов РВС РВ требованиям технического задания, в том числе в части приёма и передачи данных по внешним интерфейсам;
- отработка взаимодействия между приборами РВС РВ по бортовым каналам передачи данных;
- комплексное тестирование и отладка ПО РВС РВ, в том числе ПО, выполняемого распределённо на различных приборах;
- оценка надёжности архитектуры РВС РВ, в том числе наличия резерва пропускной способности каналов передачи данных и устойчивость аппаратно-программных средств РВС РВ к сбоям при передаче данных;
- построение расписаний обмена данными по бортовым каналам, а также проверка правильности отработки этого расписания приборами в составе РВС РВ.

Ошибки при проектировании могут привести к катастрофическим последствиям. Приведём примеры ошибок и их последствий:

- ошибки в программном компоненте, отвечающем за поддержку резервирования модулей, отложили запуск Atlantis (STS-36) на три дня [15];
- программное обеспечение космического шаттла Endeavor (STS-49) округляло до нуля значения, близкие к нулю, что вызвало тем самым проблемы при стыковке с Intelstat 6 [16];
- из-за ошибки в программном обеспечении Apollo 11 получилось, что лунная гравитация отталкивает тела, а не притягивает [17];
- в январе 1990 года произошел отказ одного из переключателей системы AT&T, который из-за допущенных ошибок при проектировании сети и разработке средств устранения ошибок привел к отказу всех 114 переключателей [18];
- во время Персидской войны неправильная работа часового механизма системы Patriot привела к тому, что ракета противника поразила бараки американских солдат в Дхахране. Ракетой было убито 27 человек, 97 получили ранения различной степени тяжести. Позднее выяснилось, что отказ часового механизма был вызван различиями представления числа 0.1 в программе [19];
- ошибки при разработке программного обеспечения аэробуса A320 привели к тому, что пилотам пришлось проявить все свои навыки и умения, чтобы вывести аэробус из аномального состояния [20].

Поэтому при разработке РВС РВ необходимо использовать программно-аппаратные средства, позволяющие осуществлять комплексирование РВС РВ и решать перечисленные выше задачи поэтапно, расширяя состав интегрируемых приборов по мере их готовности в виде натурных образцов [8].

Поскольку класс РВС РВ довольно широкий, то в данной работе будет рассматриваться подмножество этого класса – системы бортовых вычислительных комплексов. Все приведённые выше примеры РВС РВ являются также системами бортовых вычислительных комплексов.

2 Требования к создаваемым методам и средствам

В данном разделе описываются требования к создаваемым методам и средствам. В разделе 2.1 приводятся требования к системе комплексного моделирования. В разделе 2.2 приводится терминология, принятая в разработке и моделировании РВС РВ.

2.1 Требования к системе моделирования

Для решения выделенных в разделе 1 задач, возникающих на этапе комплексирования РВС РВ требуется разработка инструментальных средств поддержки.

В 2002 г. НИИСУ были подготовлены проекты Государственного стандарта ГОСТ Р «Комплекс бортового оборудования. Организация проведения проектирования, испытаний и аттестации на основе использования комплексов моделирования» и проект отраслевого стандарта ОСТ В1 «Комплекс бортового оборудования ЛА. Структура стендово-имитационной среды. Технические требования». В соответствии с упомянутыми стандартами предполагается наличие трех крупных этапов моделирования:

- проектное (поисковое) моделирование;
- математическое моделирование;
- полунатурное моделирование.

На этапе поискового моделирования рассматриваются различные варианты структуры РВС РВ, варианты взаимодействия компонентов РВС РВ, форматы и устройства индикации и визуализации, расписания обмена компонентов РВС РВ по каналам передачи данных, формируются спецификации интерфейсов взаимодействия, разрабатываются высокоуровневые описания алгоритмов функционирования компонентов, создаются модели компонентов РВС РВ. На этапе математического моделирования осуществляется уточнение моделей, их отработка, интеграция моделей друг с другом и прототипами оборудования РВС РВ, оптимизация алгоритмов и их характеристик, отработка циклограмм взаимодействия и обмена, отработка информационного взаимодействия. В ходе этапа полунатурного моделирования осуществляется комплексная отработка аппаратуры с постепенной заменой моделей РВС РВ реальными устройствами, моделирующих работу аппаратуры.

Схема технологического процесса моделирования РВС РВ приведена на рисунке 3.

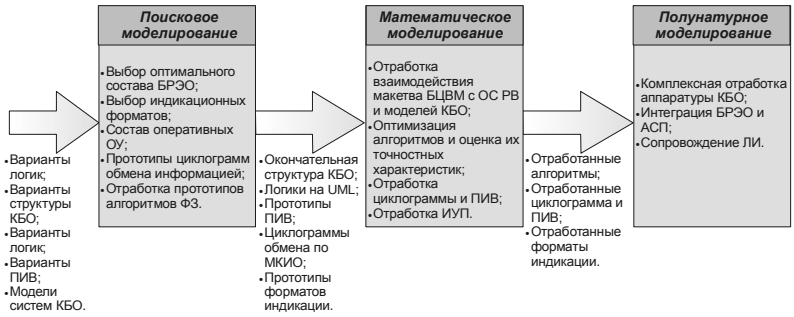

Рисунок 3. Технология моделирования РВС РВ

Проведенный анализ задач, решаемых на каждом из этапов моделирования (проектное, математическое, полунатурное), показывает, что на различных этапах для решения схожих задач целесообразно использование единых технических средств.

Идея метода комплексного моделирования состоит в следующем. Сначала создается программно-аппаратный комплекс, который мы будем называть «стенд». Этот стенд должен обеспечить с помощью методов имитационного моделирования отработку функционирования РВС РВ и отдельных его комплексов. Затем, по мере отработки функционирования комплексов РВС РВ и их реализации в виде натурных образцов, элементы стенда заменяют соответствующими натурными образцами, без изменения других элементов стенда. Так организованный процесс разработки РВС РВ позволяет минимизировать количество ошибок, выявляемых на этапах натурных испытаний.

На втором этапе данной работы [2] 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 [21]. 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 одновременного 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 предопределённому 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 натурных 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 РВС РВ 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 [22]. 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.
- ***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 [23]. 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.

2.2 Основные понятия и терминология

В данном разделе кратко представлена терминология, применяемая при разработке и моделировании РВС РВ [24].

2.2.1 Частная модель

Частная модель (ЧМ) элемента РВС РВ является имитационной моделью этого элемента и отражает следующие аспекты его функционирования:

- измерение параметров внешней среды и функционирования РВС РВ;
- вычисление выходных данных;
- работа в различных режимах;
- взаимодействие по каналам бортовых интерфейсов (КБИ) с БЦВМ и другими элементами РВС РВ, в том числе:
    - управление адаптерами КБИ;
    - формирование, отправка и приём сообщений БИ;
    - реагирование на команды, приходящие по БИ;
    - сбои в работе.

2.2.2 Параметры частной модели

Параметры ЧМ являются формой представления данных о функционировании ЧМ, которые экспериментатор может наблюдать в ходе эксперимента. У каждого параметра имеется набор свойств, задаваемых разработчиком ЧМ.

2.2.3 Интерфейс ЧМ

Интерфейсы ЧМ служат для обмена данными между ЧМ, а также между ЧМ и натурными образцами оборудования. Интерфейсы имеют набор свойств, которые определяют настройку адаптеров КБИ.

2.2.4 Сообщение

Данные, передаваемые по КБИ, представлены в форме сообщений. Таким образом, сообщения – это механизм взаимодействия между ЧМ.

2.2.5 Канал

Канал – натурный или программно моделируемый образец КБИ. Возможно создание программных моделей перспективных каналов (не имеющих реализованных натурных образцов).

2.2.6 Генератор потока отказов

Генератор потока отказов (ГПО) – это ЧМ, предназначенная для генерации сообщений, имитирующих возникновение сбоев в функционировании элементов РВС РВ. Отработка сообщений от ГПО осуществляется частной моделью элемента РВС РВ.

2.2.7 Сигналы

Сигнал – асинхронное воздействие на частную модель со стороны таймера, интерфейса или среды моделирования. Каждому сигналу в момент его возникновения сопоставляется текущее значение модельного времени.

2.2.8 События

Событием называется любое изменение состояния системы, выделяемое экспериментатором, как существенное с точки зрения логики функционирования моделируемой системы, которое может произойти во время работы стенда и которое отражается в трассе результатов эксперимента.

Событие характеризуется:

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

3 Средства описания РВС РВ

В данном разделе приводятся различные средства описания РВС РВ, используемые в данной НИР. В разделе 3.1 описаны два уровня представления единого описания РВС РВ. Раздел 3.2 содержит описание стандарта HLA, в который производится трансляции моделей, написанных на специализированном языке моделирования. В разделе 3.3 приводится обзор специализированных языков моделирования и выбирается язык диаграмм состояний UML. Раздел 3.4 содержит описание применения диаграмм состояний UML для описания моделей РВС РВ. В разделе 3.5 приводено описание РВС РВ в виде модели иерархических временных автоматов. Раздел 3.6 содержит описание РВС РВ в виде плоских временных автоматов. В разделе 3.7 приведено описание формата трассы OTF для хранения и анализа результатов моделирования РВС РВ.

3.1 Два уровня единого формата описания РВС РВ

На первом этапе [1] данной работы в качестве единого формата описания РВС РВ и их моделей предложено использовать программные компоненты, разработанные с применением стандарта HLA [25]. 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. 4). В практике моделирования встречаются два распространённых подхода:

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

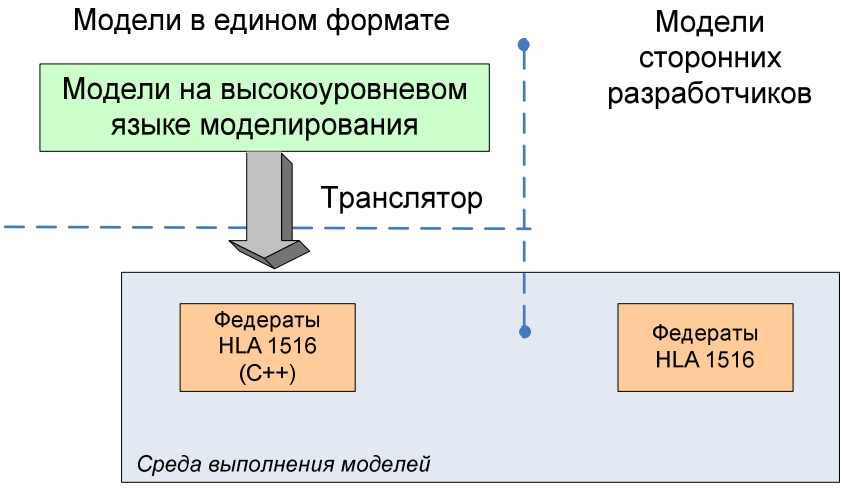

Рисунок 4. Высокоуровневое описание моделей и единый формат HLA.

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.

Поэтому было решено выбрать один из распространённых языков моделирования РВС РВ 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 описание стандарта HLA, а в разделе 3.3 приводится обзор специализированных языков описания моделей и обосновывается выбор языка.

3.2 Основные понятия стандарта HLA

В настоящее время существует единственный стандарт, разработанный для описания интерфейсов моделей для распределённого гетерогенного моделирования. Таким стандартом является High Level Architecture (HLA). Архитектура HLA была разработана по заказу Министерства обороны США для унификации и повторного использования моделей, применяемых в военных целях. В настоящее время эта архитектура стандартизована институтом IEEE (стандарт 1516) [25]. HLA представляет объектно-ориентированную систему, которая может применяться для построения моделей на различных языках, поддерживающих концепцию ООП (C++, Java и т.п.).

Корни стандарта HLA уходят к системам распределённой виртуальной реальности, которые позволяли объединять в единой модельной среде нескольких пользователей, находящихся на значительном географическом удалении друг от друга. Стандарт HLA наследует концепции, заложенные в DIS – узкоспециализированный стандарт, который

описывает архитектуру и интерфейс системы, способной обеспечить взаимодействие нескольких географически удалённых друг от друга систем моделирования [29].

В отличие от DIS, стандарт HLA позволяет объединять между собой модели, работающие на нескольких различных принципах. В частности, во время разработки первой версии стандарта в 1993 году предполагалась поддержка дискретного и дискретно-событийного моделирования с ограничениями мягкого и жёсткого реального времени. Однако поддержка жёсткого реального времени так и не была включена в спецификации вплоть до последней версии HLA IEEE 1516-2010 (Evolved), выпущенной в 2010 году.

Согласно спецификациям стандарта HLA, отдельные участники имитационного эксперимента, вне зависимости от их типа (программа, человек, аппаратное устройство), называются федератами. Совокупность федератов образует федерацию. Каждый федерат подключается к инфраструктуре RTI (Run-Time Infrastructure), которая обеспечивает их синхронизацию, выполняя, таким образом, функции среды выполнения. Фактически стандарт HLA описывает интерфейс между средой выполнения RTI и участниками моделирования (федератами) [30].

Модель, основанная на архитектуре HLA, называется федерацией и состоит из независимых федератов – основных строительных блоков. Требования HLA ограничивают возможности взаимодействия между федератами только средствами, предоставляемыми HLA RTI (Run-Time Infrastructure). Из этого следует, что всё необходимое для работы отдельно взятого федерата в составе любой модели может быть описано и документировано в терминах, определяемых стандартом HLA и не должно зависеть от конкретной реализации RTI. HLA не накладывает никаких ограничений на внутреннюю организацию федератов – каждый из них представляет из себя независимо исполняемую программу или аппаратуру, реализующую требования архитектуры.

Для взаимодействия между федератами HLA предоставляет два механизма: экземпляры объектов (objects) с атрибутами (attributes) и взаимодействия (interactions) с набором параметров (parameters). С точки зрения потоков информации, федераты задают поведение объектов через изменение их свойств-атрибутов, устанавливают зависимости между свойствами различных объектов. Объекты HLA логически отражают появление, существование и исчезновение в моделируемой среде объектов реального мира. Взаимодействия служат для представления событий, не связанных с конкретными объектами моделируемой системы, но оказывающих влияние на федераты. Весь обмен информацией в федерации осуществляется через централизованный процесс, реализующий службы RTI.

Для спецификации как интерфейсов между федератами и RTI, так и служб RTI, необходимо точное и строгое определение концепции объектной модели. Правила и

терминология, используемые для описания объектной модели федерации (federation object model FOM), описаны в документе High Level Architecture, IEEE P1516.2 [31]. Объектная модель имитационного моделирования (simulation object model - SOM) описывает существенные характеристики федерата с целью его повторного использования и другой деятельности, связанной с внутренними подробностями его функционирования. Модель SOM, как таковая, не используется RTI и ее службами. В отличие от SOM, модель FOM имеет дело с взаимодействием федератов и используется инфраструктурой RTI.

Модель FOM определяет:

- набор классов объектов, представляющих моделируемый мир в создаваемой федерации;
- набор классов взаимодействий, выбранных для представления взаимодействий между объектами моделируемого мира;
- атрибуты и параметры упомянутых классов;
- уровень детальности, на котором эти классы представляют моделируемый мир.

Любой объект должен быть экземпляром класса объектов, присутствующего в модели FOM. Классы объектов выбираются разработчиком объектной модели. Каждый класс объектов имеет связанный с ним набор атрибутов.

С точки зрения федерации, набор значений атрибутов некоторого объекта полностью определяет состояние этого объекта. Федераты имеют право связывать с объектом дополнительную информацию о состоянии, которая не подлежит передаче между федератами, это вне компетенции объектной модели федерации.

Федераты используют состояние объектов как одно из основных средств взаимодействия. В любой момент времени только один федерат должен отвечать за изменение заданного атрибута объекта. Такой федерат обязан предоставлять другим федератам новые значения этого атрибута объекта посредством служб RTI.

Федерат, поставляющий новые значения атрибута объекта, называется *обновляющим* значение этого атрибута. Федераты, получающие эти значения, называются *отражающими* этот атрибут.

В любой момент времени право на обновление значения атрибута объекта должно сохраняться за единственным федератом. Федерат, имеющий право на обновление значения атрибута объекта, называется *владеющим* этим атрибутом. Инфраструктура RTI предоставляет службы, позволяющие федератам передавать владение атрибутами объектов.

Концепция модели FOM также позволяет определять в объектной модели классы взаимодействий. Типы возможных взаимодействий и их параметры специфицируются в FOM.

Федерация - это объединение конкретной модели FOM, конкретного набора федератов и служб RTI. Федерация создается для конкретной цели на основе единообразно понимаемой объектной модели и набора федератов, которые могут связывать свою специфичную семантику с этой объектной моделью.

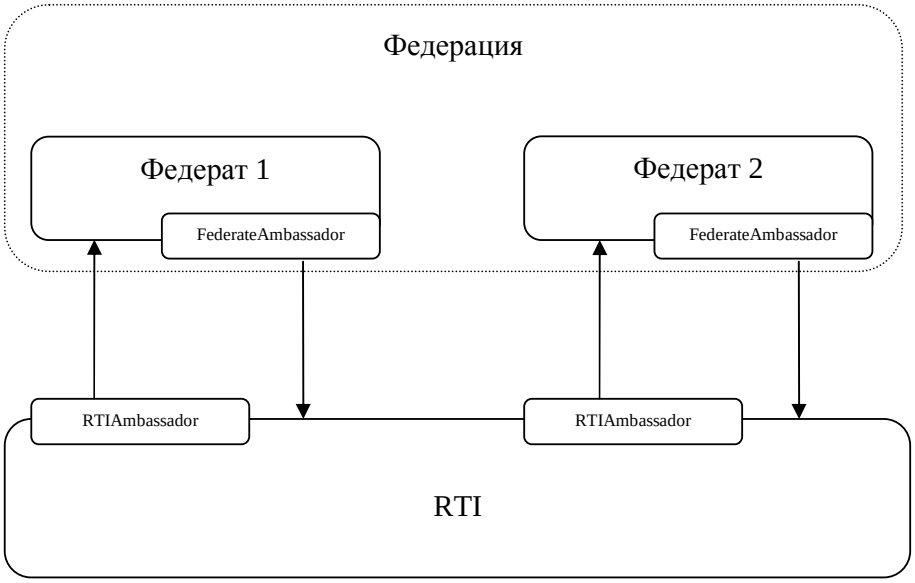

Рисунок 5. Схема взаимодействия федератов и RTI.

Далее описываются стандартизированные интерфейсы для взаимодействия федератов в федерации и приведена схема жизненного цикла федерата (см. рис.5).

Требования стандарта HLA сформулированы в виде набора сервисов, которые должны предоставлять компоненты гетерогенной имитационной модели и описания объёма предоставления данных сервисов. Сравнение упомянутых выше реализаций HLA предлагается выполнять на основе набора критериев, соответствующих данным сервиса. Ниже приводится список сервисов, выступающих в качестве критериев, с пояснениями.

3.2.1 Исполнение процесса федерации

Create Federation Execution

Служба *Create Federation Execution* создает процесс исполнения федерации и добавляет его ко множеству поддерживаемых процессов исполнения федерации. Каждый процесс исполнения федерации, созданный этой службой, должен быть независимым от остальных процессов исполнения федерации, а также между процессами исполнения федерации не

должно быть взаимодействия через инфраструктуру RTI. Параметр “описатель FED” идентифицирует данные FED, которые требуются для создания процесса исполнения федерации.

Destroy Federation Execution

Служба Destroy Federation execution удаляет процесс исполнения федерации из множества процессов исполнения федерации, поддерживаемых инфраструктурой RTI. Перед вызовом этой службы все операции федерации должны быть завершены, и все федераты должны отсоединиться.

Join Federation Execution

Служба Join Federation Execution присоединяет федерат к процессу исполнения федерации. Вызов службы Join Federation Execution должен выражать намерение участвовать в заданной федерации. Параметр «тип федерата» различает категории федератов для целей сохранения/восстановления федераций. Возвращаемый описатель федерата должен быть уникальным для всех федератов в процессе исполнения федерации.

Resign Federation Execution

Служба Resign Federation Execution запрашивает прекращение участия в федерации. Перед отсоединением должно быть принято решение о владении атрибутами объекта, принадлежащими федерату. Федерат может передать владение такими атрибутами объекта другим федератам, освободить их для приобретения прав владения позднее, или удалить объект, частью которого являются атрибуты (при условии достаточных привилегий для удаления таких объектов. Для удобства федерата служба Resign Federation Execution должна принимать аргумент, требующий от инфраструктуры RTI выполнения нуля или более из следующих действий:

- Освободить все атрибуты объекта для приобретения прав владения позднее. Это означает, что у атрибутов объекта не будет владельца (и их значения не будет обновляться), что сделает возможным приобретение прав владения;
- Удалить все объекты, для которых федерат имеет право на удаление (неявный вызов службы Delete Object Instance).

3.2.2 Работа с точками синхронизации

Register Federation Synchronization Point

Служба Register Federation Synchronization Point используется, чтобы инициировать регистрацию метки для будущей точки синхронизации. После успешной регистрации метки точки синхронизации (о чем сообщает служба Confirm Synchronization Point), инфраструктура RTI должна известить все или некоторые федераты о существовании метки

вызовом службы Announce Synchronization Point в этих федератах. Множество описателей федератов, являющееся необязательным параметром, используется федератом для указания, какие федераты в исполняемой федерации должны быть проинформированы о существовании метки. Если это множество пусто или не предоставлено, то о метке должны быть проинформированы все федераты. В противном случае, все указанные федераты должны быть членами процесса исполнения федерации. Тег пользователя позволяет связать с точкой синхронизации дополнительную информацию и должен быть сообщен вместе с меткой синхронизации. Возможно одновременное существование нескольких точек синхронизации, зарегистрированных одним или несколькими федератами. Однако, синхронизационные метки должны быть уникальными.

Confirm Synchronization Point Registration

Служба Confirm Synchronization Point Registration сообщает федерату результат запрошенной регистрации точки синхронизации. Он должен быть вызван в ответ на вызов службы Register Federation Synchronization Point. Индикатор успеха сообщает федерату либо об успешной регистрации синхронизации синхронизационной метки либо о неудаче из-за уже использованной метки или по иной причине. Попытка регистрации, закончившаяся неудачно, не должна никак влиять на процесс исполнения федерации.

Announce Synchronization Point

Служба Announce Synchronization Point информирует федерат о существовании новой точки синхронизации. После того, как метка точки синхронизации будет зарегистрирована службой Register Federation Synchronization Point , инфраструктура RTI вызывает службу Announce Synchronization Point , либо в каждом из федератов процесса исполнения федерации, либо только в федератах, указанных при регистрации метки. Федераты, проинформированные о существовании точки синхронизации с помощью вызова службы Announce Synchronization Point , образуют множество синхронизации для этой точки. Если при регистрации синхронизационной метки множество описателей федератов не было указано или было пусто, то инфраструктура RTI должна вызвать службу Announce Synchronization Point во всех федератах, присоединяющихся во время существования точки синхронизации, т.е. в промежутке от ее регистрации до вызова службы Synchronization Point Achieved всеми федератами, проинформированными о существовании этой точки синхронизации. Эти вновь присоединённые федераты становятся частью множества синхронизации этой точки. Федераты, вышедшие из процесса исполнения федерации после объявления точки синхронизации, но до синхронизации федерации в этой точке, удаляются из множества синхронизации. Тег пользователя в вызове службы Announce Synchronization

Point должен совпадать с тегом, переданным соответствующему вызову службы Register Federation Synchronization Point.

Synchronization Point Achieved

Служба Synchronization Point Achieved информирует инфраструктуру RTI о достижении федератом указанной точки синхронизации. После того, как все федератами в множестве синхронизации для какой-либо точки вызвали эту службу, RTI не имеет права вызывать Announce Synchronization Point у вновь присоединяющихся федератов.

Federation Synchronized

Служба Federation Synchronized информирует федерата о том, что все федераты из множества синхронизации для заданной точки синхронизации вызвали службу Synchronization Point Achieved для этой точки. Эта служба будет вызвана во всех федератах из множества синхронизации для этой точки, сообщая, что федераты из этого множества синхронизованы в этой точке. После синхронизации в точке (служба Federation Synchronized вызвана во всех федератах во множестве), эта точка перестаёт быть зарегистрированной, а множество синхронизации для этой точки больше не существует.

3.2.3 Сохранения состояния

Request Federation Save

Служба Request Federation Save указывает на необходимость сохранения федерации. При отсутствии необязательного аргумента «федеративное время», после вызова службы Request Federation Save инфраструктура RTI должна потребовать от всех участников процесса исполнения федерации скорейшего сохранения состояния. Если аргумент «федеративное время» присутствует, то каждому управляемому временем участнику федерации RTI указывает сохранить состояние в момент достижения логическим временем этого федерата значения, переданного службе; всем федератам, не управляемым временем, должно быть указано сохранить состояние после достижения последним из управляемых временем федератов значения логического времени, переданного службе. RTI указывает федерату о сохранении состояния с помощью вызова службы Initiate Federate Save в этом федерате. В любой момент времени может быть только один ожидающий запрос на сохранение. Новый запрос замещает любой ранее ожидавший запрос. Однако, запрос на сохранение не может возникнуть во время сохранения, то есть в промежутке между вызовом инфраструктурой RTI службы Initiate Federation Save и вызовом службы Federation Saved.

Initiate Federate Save

Служба Initiate Federate Save указывает федерату сохранить состояние. После вызова службы Initiate Federate Save федерат должен сохраниться так быстро, как только возможно. Метка, переданная инфраструктуре RTI в момент запроса сохранения службой Request Federation Save, передается федерату. Федерат должен использовать эту метку, имя исполняемой федерации, свой описатель федерата, а также тип федерата, который он передал при вызове службы Join Federation Execution, чтобы идентифицировать информацию о сохранённом состоянии. Если федерат не управляется временем, то он должен быть готов к получению службы Initiate Federate Save в любой момент времени. Если федерат управляется временем, то вызов службы Initiate Federate Save может произойти только при ожидании завершения одного из следующих служб: Time Advance Request, Time Advance Request Available, Next Event Request, Next Event Request Available, Flush Queue Request. После получения вызова службы Initiate Federate Save федерат должен немедленно прекратить передачу новой информации в федерацию. Федерат может возобновить передачу новой информации в федерацию только после получения вызова службы Initiate Federate Save.

Federate Save Begun

Служба Federate Save Begun информирует инфраструктуру RTI о начале сохранения федератом своего состояния.

Federate Save Complete

Служба Federate Save Complete сообщает инфраструктуре RTI о завершении федератом попытки сохранения. Индикатор успешности сохранения сообщает RTI об успехе или неуспехе сохранения.

Federation Saved

Служба Federation Saved сообщает федерату о завершении процесса сохранения федерации и указывает, успешно ли оно завершилось. Успешное завершение означает, что все федераты, в которых была вызвана служба Initiate Federation Save, успешно вызвали службу Federate Save Complete. Неуспешное завершение означает, что один или более федератов, в которых была вызвана служба Initiate Federation Save, неуспешно вызвали службу Federate Save с указанием неудачи, или что инфраструктура RTI обнаружила ошибку в одном или нескольких из таких федератов. Все федераты, получившие вызов службы Initiate Federation Save должны получить вызов службы Federation Saved. Если федерат, получивший вызов службы Initiate Federation Save, выходит из процесса исполнения федерации до вызова службы Federation Saved, то это должно считаться ошибкой сохранения федерации и служба Federation Saved должна быть вызвана с указанием неудачи.

3.2.4 Восстановление состояния

Request Federation Restore

Служба Request Federation Restore указывает инфраструктуре RTI начать восстановление процесса исполнения федерации. Восстановление федерации должно быть начато сразу же после проверки допустимости вызова службы Request Federation Restore. Допустимость запроса на восстановление федерации сообщается вызовом службы Confirm Federation Restoration Request.

Confirm Federation Restoration Request

Служба Confirm Federation Restoration Request указывает федерату статус запрошенного восстановления федерации. Эта служба должна быть вызвана в ответ на вызов службы Request Federation Restore. Положительное значение индикатора успеха сообщает федерату о том, что инфраструктура RTI обнаружила сохраненную информацию состояния, соответствующую указанной метке и имени процесса исполнения федерации, совпадении количество и типы присоединенных федератов совпали с количеством и типами, имевшимися на момент сохранения, и ни один иной федерат не пытается в данный момент восстановить федерацию. Если несколько федератов одновременно пытаются восстановить состояние, то один из них должен получить положительное значение индикатора успеха, а все остальные – отрицательное значение. Попытка восстановления, закончившаяся неудачно, не должна иметь никакого влияния на процесс исполнения федерации.

Federation Restore Begun

Служба Federation Restore Begun информирует федерат о начале восстановления. После получения вызова службы Federation Restore Begun федерат должен немедленно прекратить предоставление новой информации в федерацию. Федерат может продолжить предоставление новой информации в федерацию только после получения вызова службы Federation Restored.

Initiate Federate Restore

Служба Confirm Federate Restoration Request указывает федерату о возвращении к ранее сохраненному состоянию. Федерат должен выбрать подходящую информацию для восстановления на основании имени текущего процесса исполнения федерации, переданной метки сохранения федерации и переданного описателя федерата. В результате вызова этой службы, описатель федерата может принять значение, отличное от возвращенного службой Join Federation Execution.

Federate Restore Complete

Служба Federate Save Complete уведомляет инфраструктуру RTI о том, что федерат завершил попытку восстановления. Если восстановление произошло успешно, то федерат будет

находиться в состоянии, которое либо он, либо какой-то другой федерат его типа, имел в момент сохранения с указанной меткой, с тем отличием, что федерат будет ожидать вызова службы Federation Restored.

Federation Restored

Служба Federation Restored информирует федерата о завершении процесса восстановления федерации и указывает успешность или не успешность завершения. Успешность означает, что все федераты, у которых была вызвана служба Federation Restore Begun, вызвали службу Federate Restore Complete с признаком успеха. Неуспешность означает, что один или несколько федератов у которых была вызвана служба Federation Restore Begun вызвали службу Federate Restore Complete с указанием неуспеха, или что инфраструктура RTI обнаружила ошибку в одном или нескольких из таких федератов. Все федераты, получившие вызов службы Federation Restore Begun, должны получить вызов службы Federation Restored. Если федерат, получивший вызов службы Federation Restore Begun, покидает процесс исполнения федерации до вызова службы Federation Restored, это должно считаться ошибкой восстановления федерации, и служба Federation Restored должна быть вызвана с указанием неудачи.

На рис.6 представлен жизненный цикл федерата.

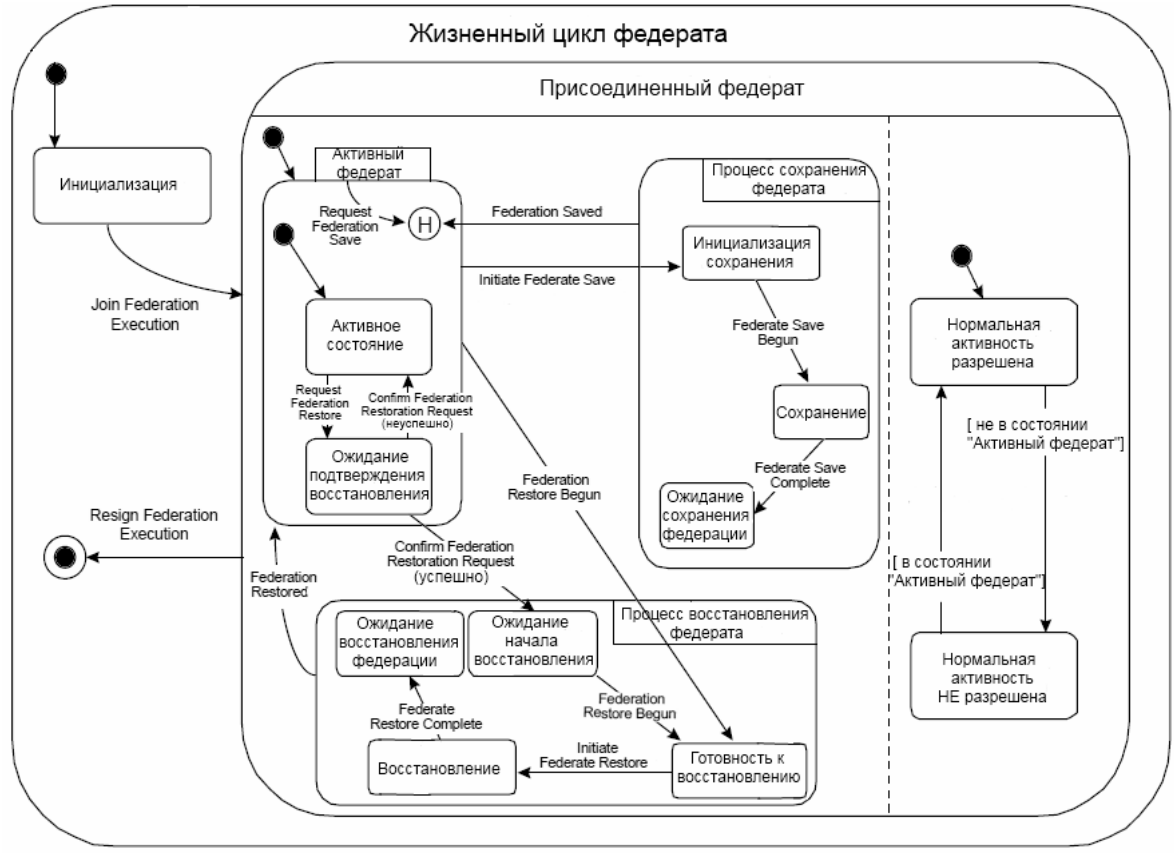

Рисунок 6. Жизненный цикл федерата.

3.3 Выбор языка описания моделей

В данном разделе рассмотрены некоторые существующие средства описания имитационных моделей и выбраны диаграммы состояний языка UML в качестве основного средства описания моделей РВС РВ. Как правило, при создании средств описания имитационных моделей разработчики используют один из следующих подходов [26]:

- создание нового специализированного языка программирования с функциями поддержки моделирования;
- написание специализированных модулей (библиотек классов, макроопределений), решающих задачи поддержки моделирования, для языка общего назначения;

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.

При использовании второго подхода описанные выше функции реализуются в виде модуля языка общего назначения, например библиотеки классов и шаблонов Си++ или Ява. 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. Такой подход, например, использовался в системе моделирования “СТЕНД” [32]. В качестве языка моделирования для этой системы использовался язык Си.

В тоже время в языках общего назначения отсутствуют или ограничены возможности контроля за соблюдением разработчиком схемы моделирования, что осложняет 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), благодаря чему снимается вероятность как умышленного, так и неумышленного нарушения ее программистом, становится возможен формальный анализ модели в рамках заданной схемы моделирования. 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.

Основная цель разрабатываемого в рамках данного НИР средства – абстрагирование пользователя (разработчика имитационных моделей) от описания интерфейсов для взаимодействия с RTI. При этом необходимо придерживаться подхода с использованием графической оболочки для построения модели, так как разработчик модели не обязан быть программистом. Исходя из этого, список требований выглядит следующим образом:

- *Графический интерфейс.* Наличие графической оболочки для построения имитационных моделей.
- *Создание HLA-совместимых моделей.* Возможность получения на выходе исходных кодов моделей с описанным интерфейсом для подключения к RTI.
- *Генерация кода по шаблонам.* Возможность генерации исходного кода модели с использованием шаблонов.
- *Свободная лицензия.* Необходимо, чтобы базовое средство распространялось по GPL лицензии.
- *Возможность верификации.* Необходимо проверять заданные функциональные свойства разрабатываемой имитационной модели, например, с использованием библиотеки UPPAL.

3.3.1 ММ-язык

Примером такого специализированного языка является язык ММ [33], разработанный для среды моделирования ДИАНА. В основу ММ-языка была положена система с сообщениями 34. Доказано, что система с сообщениями, как алгоритмическая система эквивалентна сетям Петри [35], [36]. Список алгоритмически разрешимых задач в рамках сетей Петри и их модификаций, в отличие от систем сводимых к машинам Тьюринга, наиболее полно соответствует списку задач, возникающих при анализе РВС РВ. [37] Поэтому основное преимущество использования такого языка – это использование единого описания для алгоритмического и количественного анализа РВС РВ.

ММ-язык среды моделирования ДИАНА представляет собой расширение языка Си средствами описания распределенной вычислительной системы [33]. Эти средства позволяют:

- Описывать программные и аппаратные компоненты системы, а также привязку программных компонентов к аппаратным;
- Описывать структуры сообщений, используемых при взаимодействии компонентов;
- Описывать внутреннее функционирование каждого процесса в системе и его взаимодействие с другими процессами.

В ММ-языке имеются два вида программных компонентов - последовательные процессы и распределенные программы, и два вида аппаратных компонентов - последовательные и распределенные исполнители. Процесс определяется алгоритмом его работы и взаимодействия с другими процессами. Последовательный исполнитель содержит информацию, отражающую его временные характеристики. Распределенная программа представляет собой совокупность процессов и/или других программ, распределенный исполнитель - совокупность последовательных исполнителей и/или других распределенных исполнителей. В теле последовательного процесса описывается алгоритм его работы и взаимодействия с другими процессами. Тело описывается на некотором расширении языка Си и по синтаксису напоминает описание Си-функции. В ММ-языке описываются типы компонентов, в дальнейшем используются экземпляры этих типов. Описание типа компонента состоит из заголовка и тела. Заголовок определяет вид компонента (процесс, программа, последовательный или распределенных исполнитель), имя типа компонента, список его параметров и список буферов (для аппаратных компонентов - список контактов), через которые компонент обменивается сообщениями с другими компонентами.

К недостаткам языка можно отнести плохую модульность и невозможность подключения в текст описания моделей алгоритмов функционирования на языках Си и Си++, а также внешних библиотек. Например, в ММ-языке нельзя разбить тело процесса на несколько функций.

3.3.2 ЯОМ

Недостатки ММ-языка были устранены в Языке Описания Моделей (ЯОМ) [8], который применяется в Стенде ПНМ. Язык описания моделей (ЯОМ) является расширением языка программирования Си. ЯОМ вводит ряд специальных понятий, используемых для описания свойств и логики функционирования моделируемых устройств. К таким понятиям относятся частные модели (ЧМ), режимы функционирования ЧМ, параметры ЧМ, интерфейсы, таймеры, сигналы, точки ожидания сигналов. Каждому понятию соответствуют элементы языка ЯОМ, описываемые с помощью операторов ЯОМ. Набор операторов, входящий в состав ЯОМ, отражает специфику области применения ЯОМ и предназначен для описания моделей устройств РВС РВ, привязки алгоритмов функционирования моделей ко

времени и организации их взаимодействия с учетом особенностей передачи данных по мультиплексным каналам. В тоже время непосредственное продвижение модельного времени в ЯОМ недоступно в отличие от ММ-языка.

Несмотря на свою детальную проработку, язык ЯОМ оказался слишком громоздким для практического использования. Инженерам, требуется преодолеть достаточно высокий “порог вхождения” для того, чтобы начать писать модели на ЯОМ. Для создания новой модели приходится проделывать множество рутинных процедур по созданию тел и заголовков компонентов модели.

3.3.3 Modelica

Modelica – свободно распространяемый объектно-ориентированный язык для моделирования сложных физических систем [30].

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 7.

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 8.

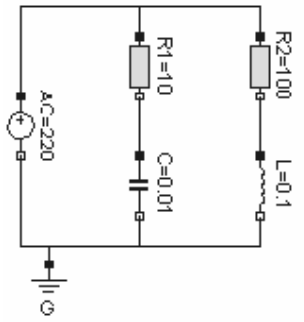

Рисунок 7. Пример графического описания модели на языке Modelica.

```
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 8. 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.

3.3.4 SLX

Язык SLX [31] 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 9 и 10 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 (GhostedTrafficLight->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 9. 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();    //importent 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")
                        {
                                GhostedObjectID = RTI_RegisterGhostedObject("TrafficLight",GhostedTrafficLight);
                                if (GhostedObjectID != -1)
                                {
                                        SLX_StateObjectPtr->ObjectsDiscovered--;
                                        print(GhostedTrafficLight->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(GhostedTrafficLight->CarLight) "Carlight after update is _______.___\n";
                        SLX_StateObjectPtr->AttributeUpdatesReceived--;
                }
                yield;
        }
}
```

Рисунок 10. Пример главного цикла модели на языке SLX.

3.3.5 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 [38].

Наиболее интересным с точки зрения моделирования встроенных систем является пакет Stateflow Simulink [39]. **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 11.

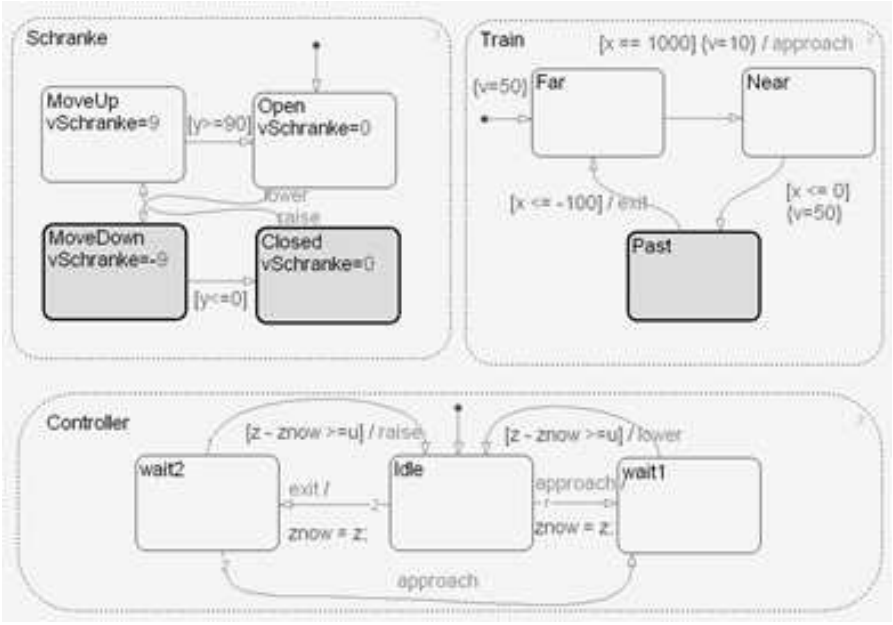

Рисунок 11. Пример описания модели на языке StateFlow Simulink.

3.3.6 Диаграммы состояний UML

Диаграммы состояний являются хорошо известным средством описания поведения систем. Они определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате влияния некоторых событий.

Из конкретного состояния в данный момент времени может быть осуществлен только один переход; таким образом, условия являются взаимно исключающими для любого события. Существует два особых состояния: вход и выход. Любое действие, связанное с событием входа, выполняется, когда объект входит в данное состояние. Событие выхода выполняется в том случае, когда объект выходит из данного состояния. Более подробно о диаграммах состояний UML в разделе 3.4.

3.3.7 Выводы

Данные языки моделирования были проанализированы на предмет соответствия требованиям (описанным в начале раздела 3.3). Результаты представлены в таблице 1.

Обзор вышеописанных языков моделирования показал несоответствие специализированных языков и сред разработки имитационных моделей предъявленным требованиям. Следовательно, необходимо разработать собственную систему построения имитационных моделей и генерации исходных кодов моделей, совместимых со стандартом HLA. Языки моделирования Modelica, Simulink Stateflow и UML удовлетворяют наибольшему количеству требований, но свободной лицензией из них обладает только язык моделирования UML. Ещё десятки лет назад Д. Харел сделал вывод об удобстве описания поведения РВС РВ с помощью диаграмм [Harel87], а похожий способ описания РВС РВ (с

несколько отличной семантикой) используется в Simulink Stateflow, что является дополнительными аргументами в пользу выбора UML. Семантика диграмм состояния UML не является жёстко зафиксированной, поэтому для моделирования РВС РВ она была зафиксирована и описывается в следующем разделе.

Таблица 1. Возможности языков моделирования.

Критерий	ММ-язык	ЯОМ	Modelica	SLX	Simulink Stateflow	UML
Графический интерфейс	-/+	-	+	-	+	+
Работа с диаграммами состояний	-	-	+	-	+	+
Создание HLA совместимых моделей	-	-	-	-	-	-
Генерация кода по шаблонам	-	-	+	-	+	-
Свободная лицензия	-	-	-	+	-	+
Возможность верификации	+	-	+	-	+	+ (*)

3.4 Применение диаграмм состояний UML для описания РВС РВ

В данном разделе приведены краткое описание универсального языка моделирования UML, особенности 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, то применение языка для описания РВС РВ требует дополнительных договорённостей. В частности, требуется определить способ описания временных аспектов поведения таких систем.

3.4.1 Язык UML

Универсальный язык моделирования UML применяется для проектирования и моделирования различных аппаратных и программных систем, в том числе и РВС РВ [40]. При помощи языка UML описываются многие аспекты систем, включая статическую структуру и их динамические характеристики.

Язык UML прост для понимания и использования. Стандарт этого языка описан таким образом, чтобы предоставить пользователю как можно большую свободу действий. Как

следствие, семантика диаграмм UML определена недостаточно для проведения полноценного анализа РВС РВ. Поэтому при использовании UML для описания моделей РВС РВ необходимо вводить дополнительные ограничения, строго определяющие допустимые конструкции диаграмм, используемые выражения и их семантику.

Стандарт UML [41,42] специфицирует большое количество диаграмм, например:

- Диаграммы использования;
- Диаграммы классов;
- Диаграммы состояний;
- Диаграммы деятельности;
- Диаграммы последовательностей;
- Диаграммы коммуникации;
- Диаграммы компонентов;
- Диаграммы размещения;
- Диаграммы объектов;
- Диаграммы внутренней структуры;
- Обзорные диаграммы взаимодействия;
- Диаграммы синхронизации;
- Диаграммы пакетов.

Многие из перечисленных диаграмм могут быть использованы для описания РВС РВ. Из всего многообразия способов представления системы в виде диаграмм UML в первую очередь интересны те диаграммы, которые описывают поведение системы. Для описания поведения систем обычно используются диаграммы последовательностей и диаграммы состояний.

Обычно на ранних этапах проектирования систем реального времени явно выделяются режимы работы компонентов этих систем, способы описания интерфейсов компонентов, а также ограничения на время пребывания в определённых режимах. По этой причине диаграмма состояний, представляющая собой автомат, то есть множество состояний, соединенных переходами, является подходящим способом описания РВС РВ. Вывод об удобстве описания поведения систем диаграммами состояний был сделан десятки лет назад Д. Харелом [43]. Также похожий способ описания, однако с отличающейся семантикой, используется в популярном коммерческом решении Simulink Stateflow [44].

В следующих подразделах приводятся используемые нами ограничения на структуру диаграмм состояний UML и описывается их функционирование.

3.4.2 Синтаксис диаграмм состояний

В данном разделе приводятся ограничения на синтаксис диаграмм состояний, которые должны быть выполнены для возможности их анализа разработанным нами средством.

Диаграммы могут содержать состояния следующих видов:

- простые состояния (simple states),
- композитные состояния (composite states),
- ссылки (submachine states),
- входы (initial states) и
- выходы (final states).

Композитное состояние состоит из одного или нескольких параллельных регионов (concurrent regions), в каждый из которых вложена диаграмма, учитывающая описанные далее ограничения. Каждая диаграмма является последовательным автоматом; при этом диаграммы, вложенные в параллельные регионы композитного состояния, представляют собой параллельно работающие компоненты.

Простое состояние представляет собой элементарное состояние компонента системы, выраженного объемлющим композитным состоянием. Простое состояние не может содержать вложенные диаграммы.

Вход и **выход** — это служебные состояния, обозначающие, соответственно, начало и завершение работы диаграммы, в которую они вложены. В каждый параллельный регион должны быть вложены ровно один вход и не более одного выхода.

Состояния диаграммы могут быть соединены **дугами**. Дуге, соединяющей состояния двух разных диаграмм D_1, D_2, равносильна следующая конструкция. Пусть P – ближайшая по вложенности диаграмма, в которую вложены D_1 и D_2. На каждом уровне вложенности между D_1 и P добавляются выходы и ведущие в них дуги так, чтобы завершить работу всех вложенных компонентов. На каждом уровне вложенности между D_2 и P добавляются входы так, чтобы инициализировать все вложенные компоненты. В диаграмме, вложенной в P, проводится дуга, необходимая для смены активности вложенных компонентов.

Пример описанного равносильного преобразования диаграммы приведен на рисунке 12. На этом и последующих рисунках

1. Si — композитные состояния без параллельных регионов,
2. bi — простые состояния,
3. вложенность состояний соответствует геометрической вложенности их изображений,
4. черный круг обозначает вход в объемлющее состояние (здесь — S3),

5. черный круг, вложенный в белый круг, обозначает выход объемлющего состояния (здесь — S2) и
6. дуги обозначены стрелками.

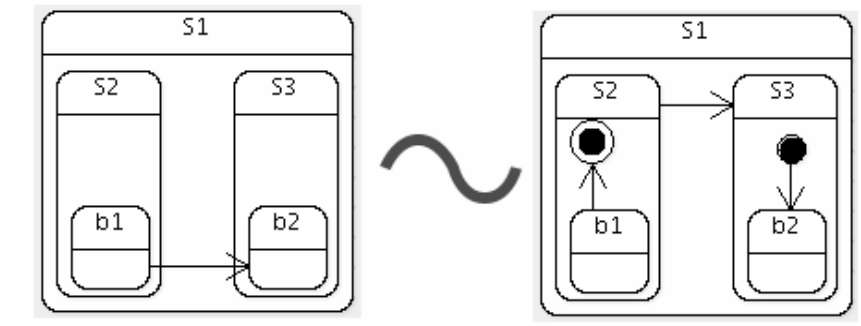

Рисунок 12 — Пример эквивалентной замены дуг

Для корректной работы на дуги описываемой модели налагаются следующие ограничения:

- модель не должна содержать дуги, соединяющие диаграммы различных параллельных регионов одного композитного состояния;
- из входа должна исходить ровно одна дуга;
- из выхода не может исходить ни одной дуги;
- во вход не должно входить ни одной дуги.

На рисунке 13 приведен пример диаграммы, содержащий входы, выходы, дуги, параллельные регионы, простые и композитные состояния. В этом примере композитное состояние S1 содержит два параллельных региона region1, region2, обозначающих два параллельно работающих компонента системы. Заметим также, что диаграммы параллельных регионов region1 и region2 эквивалентны с точностью до имен состояний.

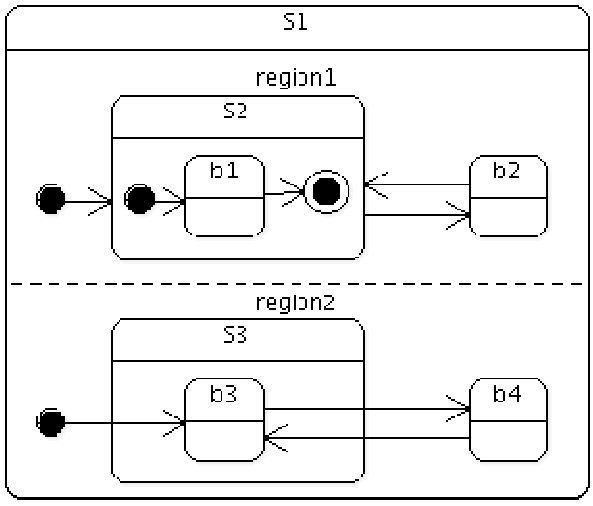

Рисунок 13 — Простой пример диаграммы

В модели есть возможность использования **данных**, а именно:

1. переменных,
2. (действительных) таймеров,
3. (широковещательных) каналов и
4. макроопределений.

Данные, локальные для композитного состояния, описываются в **блоке описаний** — комментарии, прикрепленном к этому состоянию. Таким образом, модель может содержать глобальные данные (локальные для внешнего композитного состояния модели) и данные, локальные для выбранного композитного состояния и всех вложенных в него диаграмм.

Записи в блоке описаний могут иметь следующий вид:

- int [a..b] x = c, где a, b, c — целые числа, — задание новой переменной x, принимающей целые значения в отрезке от a до b, имеющей начальное значение c;
- bool b = val, где val — либо константа true, либо константа false, — описание булевой переменной b, имеющей начальное значение val;
- clock c — задание таймера c, принимающего действительные значения и имеющего начальное значение 0;
- #define Name Expr — задание макроопределения в стиле языка C;
- signal s — задание канала s широковещательной синхронизации.

Так как все переменные, таймеры и каналы, описанные в комментарии, являются локальными для состояния, при наличии нескольких ссылок на одну диаграмму они считаются различными; при совпадении имен различных локальных переменных транслятор переименовывает последнюю встретившуюся, добавляя указание имени экземпляра подставляемой диаграммы. Все переменные, таймеры и каналы, используемые в выражениях, должны быть заданы в комментариях с учетом локальности.

Отдельно следует отметить, что для каждого состояния считается определенным таймер self.c, обозначающий время, прошедшее с начала последней активации данного состояния. При использовании таймера self.c в метках состояния и исходящих из него дуг во время трансляции заводится новый таймер; после этого все идентификаторы self.c, привязанные к состоянию, заменяются на этот таймер; кроме того, на дуги, входящие в это состояние, добавляется метка сброса этого таймера.

Метки дуг. Дуга может быть помечена предусловием, действиями, триггером приема сигнала и временным триггером. Предусловие, действия и временной триггер задаются следующими грамматиками (guard, action и ttrigger соответственно):

```
guard ::= vrelation | trelation | !guard | guard || guard | guard && guard
| (guard)
action ::= assignment_list | assignment_list !!chan
ttrigger ::= trelation | !ttrigger | ttrigger || ttrigger | ttrigger &&
ttrigger | (ttrigger)
vrelation ::= Expr rel Expr
trelation ::= timer rel Expr | timer – timer rel Expr
assignment_list ::= | assignment; assignment_list
assignment ::= var = Expr | timer = 0 | var = random()
Expr ::= var | const | Expr binop Expr | unop Expr | (Expr)
rel ::= <= | < | == | >= | >
binop ::= + | - | * | / | % | && | ||
unop ::= + | - | !
```

В приведенном описании **timer** означает имя таймера модели; **var** – имя переменной модели; **const** – целочисленную или булеву константу; **chan** – имя канала модели. Семантика арифметических и булевых выражений совпадает с семантикой выражений языка C.

Предусловие — это булево выражение над переменными и таймерами модели, которое должно быть выполнено для возможности выполнения перехода.

Действие — это набор присваиваний значений переменным и сбросов таймеров, после которого может идти посылка сигнала по широковещательному каналу связи. Присваивания совершаются непосредственно после выполнения перехода. Запись x = random() обозначает недетерминированный выбор значения переменной x.

Триггер приема сигнала с именем s обозначает прием широковещательного сигнала по каналу s. Для обеспечения корректной работы диаграммы дуге запрещено иметь одновременно метку посылки сигнала и триггер приема сигнала.

Временной триггер содержит булево выражение над таймерами модели, которое при трансляции добавляется к предусловию дуги.

Заметим, что триггер является отдельным объектом модели и может помечать более чем одну дугу. При изменении содержания триггера соответствующим образом меняются свойства всех дуг, которым присвоен этот триггер.

Помимо меток дуг, в модели также возможны **метки состояний**.

Простое состояние может быть помечено **инвариантом**, задаваемым следующей грамматикой inv:

```
inv ::= vrelation | cdtrelation | inv && inv | inv || inv | (inv)
cdtrelation ::= timer cdrel Expr | timer – timer cdrel Expr
cdrel ::= < | <=
```

Инвариант состояния является необходимым условием его активности

На рисунке 14 приведен пример диаграммы, в которой

1. определены переменные x, y, принимающие целочисленные значения из интервалов [0..2], [-5..7], инициализированные значениями 0 и 4 соответственно;
2. определена булева переменная b, инициализированная значением true;

3.определен таймер t;

4.определены каналы s1, s2;

5.присутствует макроопределение, подставляющее вместо идентификатора TO_NEW_MODE строку (y < 0);

6.предусловия ограничены квадратными скобками;

7.действия идут после косой черты;

8.состояние b3 помечено инвариантом;

9.дуга b1→b3 помечена триггером приема сигнала s2;

10.дуга b3→b1 помечена временным триггером time_trigger, записанным как after(E).

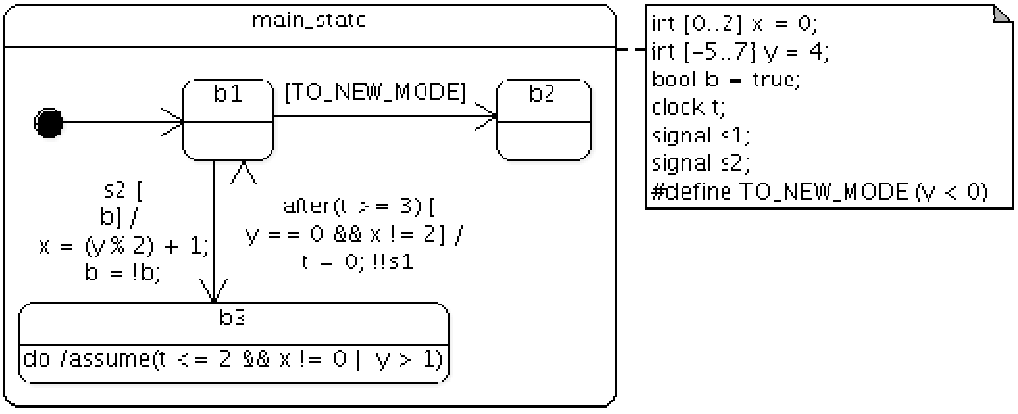

Рисунок 14 — Пример диаграммы с выражениями

На рисунке 15 изображен список элементов модели, взятый из средства ArgoUML, который иллюстрирует отображение триггеров в модели как самостоятельных объектов.

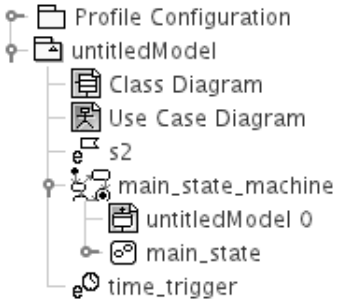

Рисунок 15 — Пример отображения триггеров в меню диаграмм

Ссылки используются для сокращения записи диаграммы. Ссылка используется в диаграмме так же, как и простое состояние. Отличие ее от простого состояния состоит в том, что в ссылке указывается диаграмма, которая будет подставлена на место ссылки во время трансляции.

Ссылки предоставляют не только возможность модульной записи диаграмм для повышения их удобочитаемости, но также и механизм шаблонов. В свойствах ссылки может

присутствовать список записей вида @param = val; каждая из таких записей задает параметру param значение val, локальное для конкретного экземпляра подставляемой диаграммы. При подстановке диаграммы все ее параметры обрабатываются так же, как и макроопределения в языке C.

На рисунках 17, 18 приведен пример ссылки reference с параметрами NUM и PARAM, имеющими значения 1 и 0 соответственно. Список объектов, приведенный на рисунке 18, показывает список объектов данной модели.

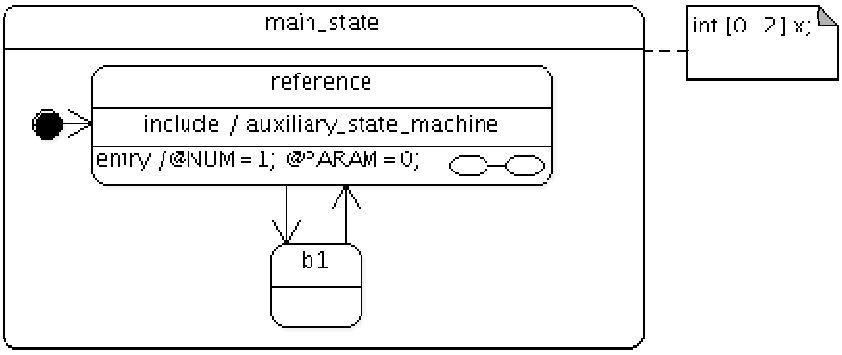

Рисунок 16 — Пример использования шаблонов (ссылок). Основная диаграмма

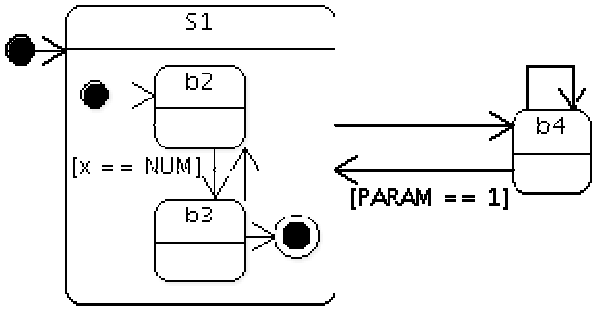

Рисунок 17 — Пример использования шаблонов (ссылок). Вспомогательная диаграмма

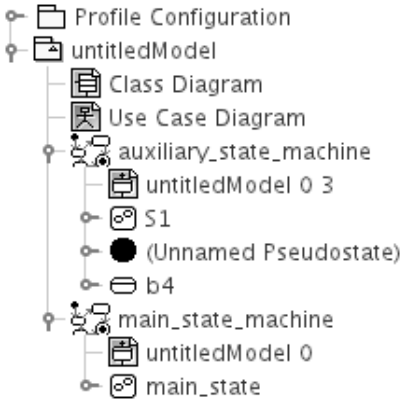

Рисунок 18 — Пример отображения ссылок в меню диаграммы

Диаграммы и группы диаграмм могут быть ``упакованы'' в **классы**. Классы используются исключительно для повышения удобочитаемости модели при наличии большого количества диаграмм.

На рисунках 19, 20 приведен пример использования классов.

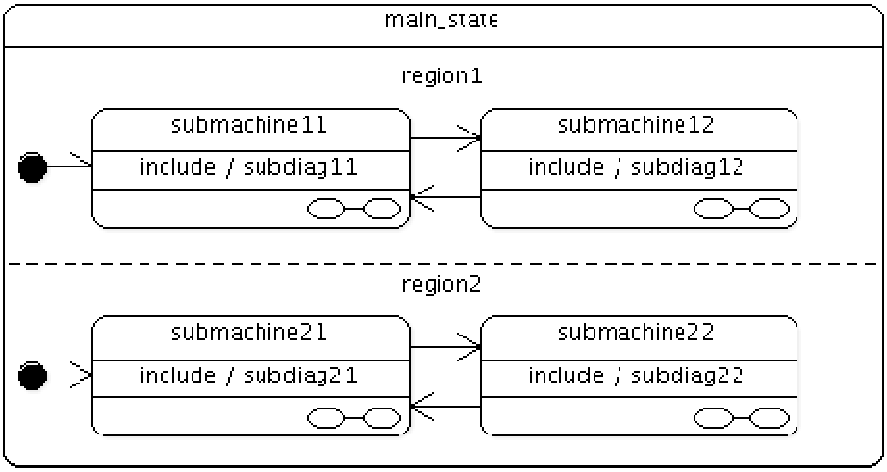

Рисунок 19 — Пример использования классов. Диаграмма

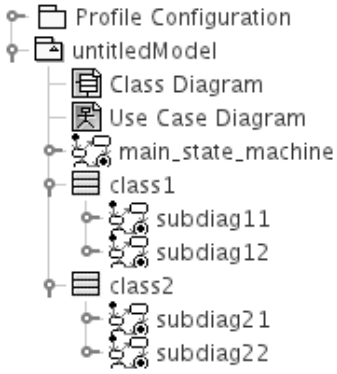

Рисунок 20 — Пример использования классов. Меню диаграммы

3.4.3 Семантика диаграмм состояний

В данном разделе приводится содержательное описание семантики диаграмм состояний во введенных нами ограничениях.

Поведение диаграммы задается сценарием ее выполнения — последовательной сменой конфигураций. Конфигурация задается множеством активных состояний и заданием конкретных значений переменных и таймеров.

Начальная конфигурация всех сценариев работы одинакова и вычисляется следующим образом. Переменные начальной конфигурации имеют значения по умолчанию, таймеры инициализируются нолями. Множество активных состояний вычисляется так: внешнее композитное состояние входит в это множество; если состояние входит в это множество, в него также входят состояния, в которые ведут дуги из вложенных входов.

Смена конфигурации происходит либо выполнением дуги (или множества дуг, если задействуется широковещательный канал), либо продвижением времени.

При продвижении времени активные состояния и значения переменных не изменяются, при этом к значениям всех таймеров прибавляется одно и то же действительное число.

При выполнении дуги $s_1 \rightarrow s_2$ без посылки или приема широковещательного сигнала, соединяющей состояния одного уровня вложенности, из множества активных состояний исключаются s_1 и все вложенные в него состояния, включаются s_2 и все состояния, достижимые через вложенные входы (как и для активных состояний начальной конфигурации); значения переменных и таймеров при этом изменяются в соответствии с действиями, которыми помечена дуга. Все исключающиеся композитные состояния должны быть готовы завершить свою работу, то есть должна существовать дуга, ведущая из вложенного в них активного состояния в выход.

При выполнении дуги $s_1 \rightarrow s_2$ с действием посылки сигнала по широковещательному каналу с определяется множество S дуг с триггером приема сигнала по этому каналу, предусловия и временные триггеры которых истинны в конфигурации; преобразование компонентов конфигурации для дуги с посылкой сообщения и каждой дуги множества S происходит аналогично предыдущему случаю.

Все конфигурации в сценарии выполнения должны удовлетворять следующему условию: инварианты всех активных состояний должны быть истинными при подстановке значений переменных и таймеров. Смена конфигураций, в результате которой истинными становятся не все инварианты, запрещена.

Семантика диаграммы состояний формализуется посредством модели иерархических временных автоматов, приведенной в разделе 3.5.

3.5 Описание РВС РВ в модели иерархических временных автоматов

Иерархические временные автоматы – это модель вычислений, предложенная в [45], которая является обобщением модели вычислений автоматов реального времени. Устройство этой модели вычислений максимально приближено к синтаксической структуре диаграмм UML, но, в отличие от диаграмм UML, для иерархических автоматов строго определено понятие вычисления. Благодаря этому модель иерархических временных автоматов можно использовать для двух целей: 1) определить на основе этой модели формальную семантику диаграмм UML, и 2) использовать эту модель для обоснования корректности алгоритмов трансляции диаграмм UML в другие формы описания РВС РВ, используемые в тех или иных средствах верификации РВС РВ.

3.5.1 Синтаксис 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

$$(S, S_0, \eta, Type, Var, Clock, BChan, PChan, Inv, T),$$

где

5. S — множество состояний,
6. $S_0 \subseteq S$ — множество инициальных состояний,
7. $\eta : S \to 2^S$ — функция вложенности состояний,
8. $Type : S \to \{and, xor, basic, entry, exit\}$ – типизация состояний,
9. Var – множество (булевых) переменных,
10. $Clock$ – множество таймеров (принимающих действительные неотрицательные значения),
11. $BChan$ – множество широковещательных каналов (broadcast),
12. $PChan$ — множество каналов типа точка-точка (handshake, peer-to-peer),
13. $Inv : S \to Invariant$ – разметка состояний инвариантами,
14. $T \subseteq S \times (Guard \times Sync \times Reset \times \{true, false\}) \times S$ — множество переходов.

Иерархический временной автомат может также содержать переменные, принимающие значения из ограниченного целочисленного интервала. При этом выражения, содержащие такие переменные, можно заменить на булевы выражения над битами их побитовой записи. Целочисленные переменные позволяют сократить размер записи иерархического временного автомата.

Элементы множеств Guard, Sync, Reset, Invariant задаются следующими грамматиками (соответственно guard, sync, reset, inv):

```
guard ::= bool | vrelation | trelation | !guard | guard || guard | guard
&& guard | (guard)
sync ::= none | chan! | chan?
reset ::= ∅ | {assignment} ∪ reset
inv ::= vrelation | cdtrelation | inv || inv | inv && inv | (inv)
vrelation ::= Expr rel Expr
trelation ::= timer rel Expr | timer – timer rel Expr
cdtrelation ::= timer cdrel Expr | timer – timer cdrel Expr
assignment ::= var = Expr | timer = 0
Expr ::= var | const | Expr binop Expr | unop Expr | (Expr)
binop ::= + | - | * | / | % | && | ||
unop ::= + | - | !
rel ::= <= | < | == | >= | >
cdrel ::= <= | <
```

В приведенной записи верно следующее: $bool \in Var$; $chan \in BChan \cup PChan$; $timer \in Clock$; var – переменная модели (в том числе целочисленная); const – целочисленная или булева константа. Семантика булевых и арифметических выражений совпадает с семантикой

выражений языка C. Грамматика reset подразумевает естественным образом определенные теоретико-множественные обозначения.

Переход (s_1, g, s, r, u, s_2) далее будет обозначаться записью $s_1 \xrightarrow{g,s,r,u} s_2$.

Состояния типов and и xor будем называть метасостояниями, состояния типа entry — входами, состояния типа exit – выходами, состояния типа basic – простыми состояниями. Если верно соотношение $s' \in \eta(s)$, будем говорить, что s' вложено в s и s объемлет s'. Если верно соотношение $s' \in \eta^*(s)$, где $\eta^*(s) = s \cup \eta(s) \cup \eta(\eta(s)) \cup \ldots$, то будем называть s' потомком s и s – предком s'. Для соотношения $s' \in \eta^+(s)$, где $\eta^+(s) = \eta(s) \cup \eta(\eta(s)) \cup \ldots$, также будем использовать понятия потомка и предка, добавляя при этом слово «существенный».

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.

1. Ограничения на структуру состояний:
    1. существует ровно одно состояние, не вложенное ни в одно другое состояние (его мы будем называть корнем);
    2. для каждого состояния найдется не более одного объемлющего состояния;
    3. состояние не может быть своим существенным потомком;
    4. только метасостояния могут быть объемлющими;
    5. в каждое метасостояние вложено некоторое другое состояние;
    6. в and-состояние могут быть вложены только входы, выходы и метасостояния.
2. Ограничения на множество инициальных состояний:
    1. корень является инициальным состоянием;

2. инициальными могут быть только простые состояния и метасостояния;
3. если некоторое состояние является инициальным, то и его объемлющее состояние также является инициальным;
4. если xor-состояние является инициальным, то ровно одно вложенное в него состояние является инициальным;
5. если and-состояние является инициальным, то все вложенные в него метасостояния являются инициальными;

3. Ограничения на встречающиеся выражения:
   1. если переменная встретилась в левой части присваивания дуги, исходящей из входа, вложенного в and-состояние, то она не встречается в левых частях остальных дуг, исходящих из этого входа;
   2. инварианты входов и выходов суть константы true;
   3. если дуга несет синхронизацию c?, $c \in BChan \cup PChan$, то она помечена пустым множеством присваиваний;
4. Ограничения на переходы:
   1. Возможны только виды переходов, обозначенные на рисунке 21; на этом рисунке кругами обозначены простые состояния, прямоугольниками – метасостояния, тонкими линиями – входы, толстыми линиями – выходы, стрелками – дуги;
   2. из входа, вложенного в xor-состояние, исходит ровно одна дуга;
   3. из входа, вложенного в and-состояние, исходит столько дуг, сколько в него вложено метасостояний, и эти дуги ведут во входы этих метасостояний, причем различные дуги ведут в различные метасостояния;
   4. все дуги, ведущие из входа или в выход, являются несрочными и помечены синхронизацией none и пустым множеством присваиваний;
   5. если дуга исходит из входа, то она также помечена предусловием true;
   6. если дуга исходит из выхода и ведет в выход, то она помечена предусловием true.

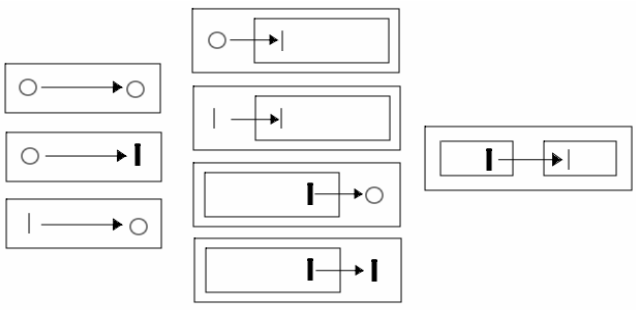

Рисунок 21 – Виды переходов иерархического временного автомата

3.5.2 Семантика иерархических временных автоматов

Конфигурацией автомата является тройка (ρ, μ, ν), где

- $\rho \subseteq S$ – множество активных состояний автомата,
- $\mu : Var \to Z$ – оценка переменных,
- $\nu : Clock \to R^+$ – оценка таймеров.

Начальная конфигурация $\mathbf{c_0}$ состоит из множества L_0 и оценок, ставящих в соответствие всем переменным и таймерам константы false и ноли соответственно.

Определим понятие входного множества EN(s) для состояния s автомата. Если s – простое состояние, то входное множество состоит из одной вершины s. Если s – вход и s’ – состояние, объемлющее для s, то EN(s) включает в себя состояние s’ и все элементы множеств EN(s’’) для состояний s’’, достижимых из s по одной исходящей дуге.

Обозначим записями μ^a, ν^a оценки переменных и таймеров, получаемые из оценок μ, ν заменой значений переменных из левых частей элементарных присваиваний множества a на значения правых частей при оценках μ, ν.

Истинность формулы f при оценках μ, ν будем обозначать записью μ, ν |= f.

Для того чтобы формализовать понятие вычисления, введем отношение $\Rightarrow$ одного шага вычисления. Для этого необходимо ввести следующие понятия. Рассмотрим множество переходов $T = \{t_1, t_2, \ldots, t_n\}$ автомата, где $t_i = s_i \xrightarrow{g_i, s_i, r_i, u_i} s_i'$. Пусть $g = g_1 \wedge g_2 \wedge \ldots \wedge g_n$, $a = a(T) = a_1 \cup a_2 \cup \ldots \cup a_n$, $\rho' = \rho(T) = \rho \setminus (\eta^*(s_1) \cup \eta^*(s_2) \cup \ldots \cup \eta^*(s_n)) \cup (EN(s'_1) \cup EN(s'_2) \cup \ldots \cup EN(s'_n))$ и I(x) – конъюнкция всех инвариантов состояний множества x.

Множество T является активным в конфигурации (ρ, μ, ν), если μ, ν |= g.

Множество T является легальным в конфигурации (ρ, μ, ν), если $s_i \in \rho$, $1 \le i \le n$:

1. $i = 1$ и c_i = none;
2. $i = 2$, $c_1 = !c$, $c_2 = ?c$, $c \in$ PChan;
3. $c_1 = !c$, $c \in$ BChan, $c_2 = c_3 = \ldots = c_n = ?c$ и в автомате нет ни одной дуги, не входящей в множество T, исходящей из одной из вершин множества ρ и помеченной предусловием g’, таким что μ, ν |= g’.

Множество T является отвечающим срочности, если либо оно содержит срочные переходы, либо в автомате не существует активных срочных переходов, исходящих из состояний множества ρ.

Пусть ex – выход автомата. Будем называть его активным, если из каждого простого состояния множества $\rho \cap \eta^*(s'')$, где s’’ – объемлющее состояние для ex, исходит активная

дуга в выход, соединенный с ех дугами через другие выходы. Множество T является готовым к завершению в конфигурации (ρ, μ, ν), если для каждой входящей в него дуги верно следующее: либо она исходит из простого состояния, либо она исходит из активного выхода.

Множество T является готовым к инициализации в конфигурации (ρ, μ, ν), если верно следующее соотношение: $\mu^a, \nu^a \models I(\rho')$.

Записью ν^{+d} обозначим следующую оценку таймеров: $\nu^{+d}(t) = \nu(t) + d$ для всех таймеров автомата.

В отношение $\Rightarrow$ входят следующие пары конфигураций:

- $(\rho, \mu, \nu) \Rightarrow (\rho, \mu, \nu^{+d})$, если верно соотношение $\mu, \nu^{+d} \models I(\rho)$;
- $(\rho, \mu, \nu) \Rightarrow (\rho', \mu^a, \nu^a)$, если существует активное легальное множество переходов T, готовое к завершению и инициализации и отвечающее срочности, для которого верны равенства $\rho' = \rho(T)$ и $a = a(T)$.

Вычислением автомата называется всякая максимальная последовательность конфигураций

$$\mathbf{c_0} \Rightarrow \mathbf{c_1} \Rightarrow \ldots \Rightarrow \mathbf{c_n} \Rightarrow \ldots .$$

Поведение (то есть семантика) автомата задается множеством его вычислений.

3.5.3 Преобразование диаграммы состояний в иерархический автомат

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

На этапе синтаксического разбора диаграммы UML проводится преобразование выражений диаграмм, отсутствующих в синтаксисе UPPAAL. Сначала производится подстановка всех макроопределений. Затем производится подстановка диаграмм на место ссылок, пока все ссылки не будут устранены. На этом этапе также посредством переименования данных предотвращается коллизия имен. Результатом подстановки макроопределений и ссылок является единственная диаграмма, выражения которой могут быть разобраны согласно описанным в разделе 3.3 грамматикам.

После подстановки макроопределений и ссылок диаграмма шаг за шагом преобразуется к виду иерархического временного автомата. Первый шаг этого преобразования – добавление входов и выходов для композитных состояний, содержащих параллельные регионы. Такие композитные состояния впоследствии становятся and-состояниями автомата. Каждая дуга, ведущая в композитное состояние, перенаправляется во вход, вложенный в это состояние. Каждая дуга, ведущая из композитного состояния,

заменяется аналогичной дугой, ведущую из выхода, вложенного в это состояние. Параллельные регионы и композитные состояния, не содержащие параллельных регионов, в дальнейшем становятся xor-состояниями автомата.

Для каждой дуги диаграммы, нарушающей ограничения автомата на типы состояний, которые могут быть соединены дугами, делается следующее. Пусть эта дуга соединяет состояния S1 и S2. Пусть S – наиболее глубоко вложенное состояние, в которое, в свою очередь, вложены состояния S1 и S2. Исходная дуга удаляется. В каждое состояние между S1 и S добавляется выход, из состояния S1 направляется дуга в наиболее вложенный выход, затем выходы соединяются между собой по убыванию уровня вложенности диаграмм. В каждое состояние между S и S2 добавляется вход, наименее вложенный выход соединяется дугой с наименее вложенным входом, входы соединяются между собой по возрастанию уровня вложенности, самый вложенный вход соединяется с состоянием S2.

После замены всех дуг указанным способом структура диаграмм совпадает со структурой автомата. Для полного соответствия необходимо явно вычислить множество инициальных состояний (этот шаг очевиден) и преобразовать выражения, присутствующие в диаграмме. Если дуга содержит временной триггер, он удаляется и дописывается в предусловие дуги. Если дуга содержит действие посылки сигнала, оно удаляется и записывается в синхронизацию дуги. Если дуга содержит триггер приема сигнала, он удаляется и записывается в синхронизацию дуги. Результатом описанной последовательности преобразований с учетом указанных в описании соответствий является иерархический временной автомат, считающийся эквивалентным исходной диаграмме состояний.

3.6 Описание РВС РВ в виде плоских временных автоматов

Одной из самых простых и в то же время довольно выразительных моделей, описывающих поведение РВС РВ, является модель (плоских) временных автоматов [46],[47],[48],[49]. РВС РВ согласно этой модели представляется в виде совокупности (сети) временных автоматов. Временной автомат – это система, определяющая множество состояний управления, способ учета временных параметров системы и механизмы синхронизации с другими временными автоматами. Состояния управления временного автомата соответствуют режимам работы элементарного компонента РВС РВ. Изменение состояний управления зависит от временных характеристик и состояний управления всех автоматов в сети. При изменении состояния управления могут выполняться различные действия. Хотя набор таких действий потенциально не ограничен, нами будет рассматриваться конкретный набор действий (отправление и прием сигнала по каналу

широковещательного типа или типа точка-точка, изменение значения переменной, сброс таймера), соответствующий действиям, выполняемым в средстве верификации UPPAAL [50].

В следующих разделах приведены синтаксис и семантика модели сети временных автоматов.

3.6.1 Модель сети временных автоматов

Обозначим символом C множество таймеров. Элементарным сравнением будем называть всякий предикат вида (x op n) или (x – y op n) , где x, y - таймеры, n - натуральное число, op - одно из арифметических отношений $<, \leq, =, \geq, >$. Предусловием называется всякая элементарная конъюнкция элементарных сравнений и их отрицаний. Множество всевозможных предусловий над множеством таймеров C обозначим записью B(C). Множество всевозможных предусловий, не содержащих отношений $=, \geq, >$, обозначим записью B'(C).

Конечный временной автомат – это система $U = (L, l_0, C, A, E, I)$, состоящая из:

- конечного множества состояний управления L,
- начального состояния управления l_0, $l_0 \in L$,
- множества таймеров C,
- множества действий A, включающего 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 если предусловие g истинно в текущих показаниях таймеров, то автомат может перейти из состояния управления l в состояние управления l'. При этом показания всех таймеров из множества C' обнуляются, 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.

Чтобы определить понятие вычисления автомата $U = (L, l_0, C, A, E, I)$, введем следующие понятия и обозначения. Оценкой таймеров будем называть всякое отображение $\nu : C \rightarrow R_{\geq 0}$, приписывающее каждому таймеру неотрицательное вещественное число в качестве значения. Оценку ν_0, приписывающую каждому таймеру число 0, назовем инициальной оценкой. Если ν - оценка таймеров и d – неотрицательное вещественное число, будем полагать $(\nu + d)(x) = \nu(x) + d$. Также будем использовать запись $\nu[C']$ для оценки,

которая равна нолю для всех таймеров множества C’ и совпадает с оценкой ν для остальных таймеров. Запись ν |= g будет использоваться для обозначения того, что для оценки таймеров ν предусловие g принимает логическое значение true.

Поведение автомата описывается системой переходов (S, s_0, →), где

7. $S = L \times R_0^{|C|}$ – множество конфигураций автомата,

11. $s_0 = (l_0, \nu_0)$ – начальная конфигурация,

12. $\rightarrow \subseteq S \times (R_{\geq 0} \cup A) \times S$ – отношение переходов, описывающее один шаг вычисления автомата.

Поясним значение системы переходов. Конфигурация автомата, то есть уникальное состояние его вычисления, определяется его текущим состоянием управления и набором значений таймеров. Любое вычисление автомата начинается в начальном состоянии управления и определяет значения всех таймеров нолями.

Переход (s_1, q, s_2) будем далее записывать как $s_1 \rightarrow_q s_2$. Отношение переходов $\rightarrow = \rightarrow_d \cup \rightarrow_a$ определяется следующими двумя правилами:

4. $(l, \nu) \rightarrow_d (l, \nu + d)$, если для любого вещественного числа d’, удовлетворяющего неравенствам $0 \leq d' \leq d$, выполняется соотношение $(\nu + d') \models I(l)$,

- $(l, \nu) \rightarrow_a (l', \nu')$, если существует такой переход $l \xrightarrow{a,g,C'} l' \in E$, для которого выполняются условия $\nu \models g$, $\nu[C'] \models I(l')$; при этом $\nu' = \nu[C']$.

Содержательно эти два правила означают следующее. Шаги вычисления автомата бывают двух типов:

- либо происходит продвижение времени, и автомат остается в прежнем состоянии управления; при этом инвариант состояния управления должен остаться истинным,
- либо происходит смена состояния управления без продвижения времени, но со сбросом времени некоторых таймеров; при этом новые показания таймеров должны удовлетворять инварианту нового состояния управления.

Вычислением системы переходов является всякая максимальная последовательность конфигураций автомата вида

$$s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n \rightarrow \ldots,$$

Вычисление системы переходов, описывающей временной автомат, определяет поведение этого временного автомата.

Несколько конечных автоматов могут вступать во взаимодействие друг с другом посредством действий синхронизации, образуя, таким образом, параллельную композицию, или сеть, конечных автоматов реального времени. Вычисления сети временных 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) $N = (U_1, U_2, \ldots, U_n)$, состоящее из автоматов $U_i = (L_i, l_{0i}, C, A, E_i, I_i)$, $1 \le i \le n$. Тогда поведение сети автоматов N задается системой переходов $(\mathbf{S}, \mathbf{s_0}, \Rightarrow)$, в которой

- $\mathbf{S} = (L_1 \times L_2 \times \ldots \times L_n) \times R_0^{|C|}$ - множество конфигураций сети,
- $\mathbf{s_0} = (l_{01}, l_{02}, \ldots, l_{0n}, \nu_0)$ – начальная конфигурация,
- $\Rightarrow \subseteq \mathbf{S} \times \mathbf{S}$ – отношение переходов сети, описывающее один шаг каждого ее вычисления.

Поясним значение системы переходов, описывающей поведение сети временных автоматов. Конфигурацией сети является совокупность состояний управления всех автоматов сети вместе со значением таймеров сети (заметим, что множество таймеров одинаково для всех автоматов сети). Начальная конфигурация складывается из начальных состояний управления автоматов сети и задания нулевых значений таймеров сети.

Отношение переходов $\Rightarrow$ описывается следующими тремя правилами:

- $(\mathbf{l}, \nu) \Rightarrow (\mathbf{l}, \nu + d)$, если для любого i, $1 \le i \le n$, справедливо отношение $(\mathbf{l}[i], \nu) \rightarrow (\mathbf{l}[i], \nu + d)$,
- $(\mathbf{l}, \nu) \Rightarrow (\mathbf{l}[\ \mathbf{l}[i] \leftarrow \mathbf{l'}[i]\ ], \nu')$, если для некоторого i, $1 \le i \le n$, справедливо отношение $(\mathbf{l}[i], \nu) \rightarrow (\mathbf{l'}[i], \nu')$, это соотношение не соответствует продвижению времени и для любого j, $1 \le j \le n$, $j \ne i$, верно соотношение $\nu' \models I_j(\mathbf{l}[j])$,
- $(\mathbf{l}, \nu) \Rightarrow (\mathbf{l}[\ \mathbf{l}[i] \leftarrow \mathbf{l'}[i], \mathbf{l}[j] \leftarrow \mathbf{l'}[j]\ ], \nu')$, если для некоторой пары i, j, $1 \le i, j \le n$, справедливы соотношения $l[i] \xrightarrow{!a, g', C'} l'[i]$, $l[j] \xrightarrow{?a, g'', C''} l'[j]$, $\nu' = \nu[C' \cup C'']$, $\nu' \models I_i(\mathbf{l'}[i])$, $\nu' \models I_j(\mathbf{l'}[j])$ и для любого k, $1 \le k \le n$, $k \ne i, j$, справедливо соотношение $\nu' \models I_k(\mathbf{l}[k])$.

Первые два правила аналогичны правилам продвижения времени и выполнения перехода для одного временного автомата. Третье правило описывает синхронизацию типа точка-точка (handshake, peer-to-peer) двух автоматов сети.

Вычислением системы переходов, описывающей сеть временных автоматов $N = (U_1, U_2, \ldots, U_n)$ является всякая максимальная последовательность конфигураций сети вида

$$\mathbf{s_0} \Rightarrow \mathbf{s_1} \Rightarrow \ldots \Rightarrow \mathbf{s_n} \Rightarrow \ldots$$

Вычисление системы переходов, описывающей сеть временных автоматов, определяет поведение этой сети.

3.6.2 Временные автоматы в системе UPPAAL

Язык описания конечных временных автоматов в системе UPPAAL разрешает использовать некоторые дополнительные средства, расширяющие минимальный набор действий, который присутствует в стандартной математической модели сети временных автоматов. Основные отличительные особенности автоматов UPPAAL таковы.

- При описании автоматов разрешается использовать
    1. целочисленные константы,
    2. переменные, принимающие значения из конечного целочисленного интервала,
    3. булевы переменные,
    4. массивы фиксированной длины, содержащие ограниченные целочисленные и булевы переменные,
    5. массивы таймеров фиксированной длины,
    6. массивы каналов фиксированной длины.

  Над переменными естественным образом 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.

1. В качестве элементарных сравнений разрешается использовать также булевы выражения, содержащие целочисленные константы и переменные, таймеры, целочисленные массивы, массивы таймеров; при этом таймеры и разности таймеров разрешается сравнивать как с константами, так и с арифметическими выражениями над переменными и массивами.
2. 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.
3. Разрешается использование широковещательных каналов связи и широковещательного обмена сообщениями по таким каналам. Семантика широковещательного обмена сообщениями в автоматах UPPAAL определяется так: если есть переход, имеющий истинное в текущей конфигурации предусловие и имеющий отправление сообщения по широковещательному каналу связи в качестве

действия, то определятся множество каналов с истинными в текущей конфигурации предусловиями, имеющие парные действия приема сообщения; вычисляется конфигурация, которая будет достигнута после смены управляющих состояний и изменения переменных и таймеров по выполнении всех переходов; если все инварианты состояний управления результирующей конфигурации истинны, то одновременно выполняются все описанные выше переходы; иначе переход объявляется заблокированным и не выполняется.

4. Разрешается использовать так называемые срочные каналы связи. Если из состояния исходит дееспособный переход, содержащий действие отправления сообщения по срочному каналу связи, то этот переход обязательно выполняется. Во избежание неоднозначности все переходы, содержащие такие действия, не имеют права использовать таймеры в своих предусловиях.
5. 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 $x \leq 0$. Кроме того, если есть возможность покинуть срочное состояние, оно должно быть покинуто раньше всех остальных «обычных» состояний.
6. Сверхсрочные, или транзитные, состояния управления налагают еще более строгие ограничения на порядок выполнения переходов в сетях автоматов. Если в состоянии вычисления **s** сети автоматов хотя бы один автомат пребывает в транзитном состоянии управления, то на следующем шаге вычисления обязан сработать хотя бы один переход, исходящий из транзитного состояния управления.

3.7 Формат трассы результатов моделирования РВС РВ

Результатом моделирования РВС РВ является трасса модели. Под трассой модели понимают последовательность записей о событиях в моделируемой РВС РВ. Процесс трассировки РВС РВ позволяет зафиксировать временные и пространственные отношения между элементами РВС РВ, позволяет реконструировать поведение РВС РВ на любом необходимом уровне абстракции, некоторые характеристики системы могут быть вычислены на основе трасс экспериментов. В этом заключаются основные преимущества трассировки РВС РВ. Однако есть и недостатки: трассы могут быть очень большими, могут оказывать

некоторое влияние на инструменты и процесс трассировки (например, буферизация событий).

Процесс трассировки должен удовлетворять двум основным требованиям:

1. Низкие накладные расходы на поддержку инструментов для проведения замеров и для генерации трассы.
2. Эффективные инструменты анализа трасс для обработки данных.

Формат трассы должен обеспечивать возможности инструментов анализа для повышения скорости доступа к данным и их обработке.

К формату трассы для РВС РВ были сформулированы следующие требования:

1. **Документированность формата** - наличие описания трассы, событий, определений и принципа их записи в трассу.
2. **Наличие реализации формата** означает, что данный формат, помимо теоретического описания, имеет реализацию на одном из языков программирования и ранее применялся для какого-либо проекта.
3. **Открытость API** означает открытость библиотек чтения и записи для данного формата.
4. **Открытость средств работы с трассами** связана со специальными средствами анализа и визуализации трасс данного формата, которые должны относиться к программному обеспечению с открытым исходным кодом (opensource).
    a. **Компактность трассы** связана с объемом памяти, необходимой для хранения трассы данного формата.
5. **Наличие специализированных запросов к трассе**, которые отсутствуют в других форматах.

3.7.1 Форматы хранения трасс

На первом этапе НИР был проведен обзор существующих форматов хранения трасс, применяемых для трассировки параллельных программ, высокопроизводительных параллельных платформ и имитационных экспериментов при моделировании РВС РВ. Было найдено 12 форматов трасс (VTF3[51], STF[52], OTF[53],[54], Paje[55], EPILOG[56], SLOG-2[57], CLOG[52], CCG[52], ALOG[52], Paraver [58], TAU[52], VD-Ray[59], TRC) и 9 средств анализа и визуализации трасс (Vampir [60], ViTE [61], TAU Performance System [62] [63], ITA 7.0 [64], KOJAK [65], JumpShot-4 [66], Vis3 [67], VD-Ray [59], Paje [55]). После детального рассмотрения форматов на соответствие сформулированным требованиям, целей создания, файловой структуры, возможностей по расширению хранения данных для экспериментального исследования были выбраны форматы TAU, TRC и OTF.

Экспериментальное исследование по размеру получаемых трасс и по возможностям поиска событий в трассах показало, что наиболее подходящим для трассировки результатов моделирования РВС РВ является открытый формат OTF, поддерживающий также возможность сжатия трасс. Ниже приводится подробное описание формата и возможностей, которые предоставляются библиотекой для работы с ним.

3.7.2 Описание формата OTF

OTF (Open Trace Format) – открытый формат трасс, разработанный в Центре высокопроизводительных вычислений университета Дрездена для работы с параллельными платформами, содержащими сотни процессоров. Три основных цели, для достижения которых создавался формат – это открытость, гибкость и производительность. Формат активно поддерживается и развивается.

Формат OTF имеет следующие особенности:

- поддерживает события: вход/выход из функции, отправка/прием сообщений, счетчики производительности аппаратного обеспечения;
- платформенная независимость;
- эффективное ACSII кодирование, поддержка Zlib сжатия;
- 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.

Для поддержки быстрого и селективного доступа к огромному количеству данных в формате OTF используется потоковая модель. OTF потоки могут содержать множество независимых процессов, при этом каждый поток принадлежит только одному потоку.

Каждый поток представляется множеством файлов, в которые загружаются определения, события, статусная информация и отдельно резюме событий. Единственный глобальный файл master файл, содержащий необходимую информацию отображения процессов на потоки.

Формат OTF имеет файловую структуру, представленную на рисунке 22.

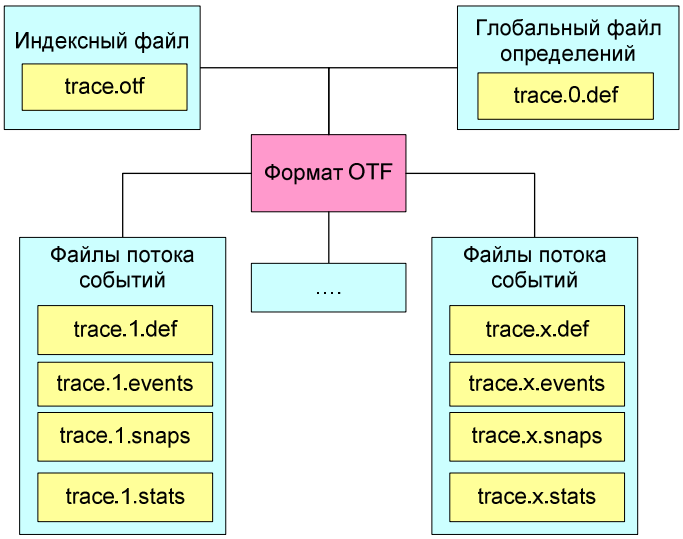

Рисунок 22. Файловая структура формата OTF.

Именование. Для именования файлов OTF трассы используется строгое соглашение. Имя каждого файла состоит из 3 частей (кроме master файла).

- Каждый файл начинается с общего префикса (названия трассы), который задается пользователем.
- Идентификатор, используемый для отображения процессов.
- Суффикс определяет тип файла (файл определений, статистики, событий).

Таким образом, при удалении, перемещении трассы важно брать все соответствующие файлы трассы. Удаление или модификация одного или нескольких файлов трассы приведет к повреждению трассы.

В состав библиотеки otflib входит утилита otfmerge для работы с трассами (объединение, преобразования), которая позволяет объединять несколько otf трасс в одну в соответствии с временными метками.

На рисунке 23 представлен высокоуровневый взгляд на архитектуру OTF, показывающий потоки и файлы, используемые компонентами генерации трасс (tracer) и анализа трасс.

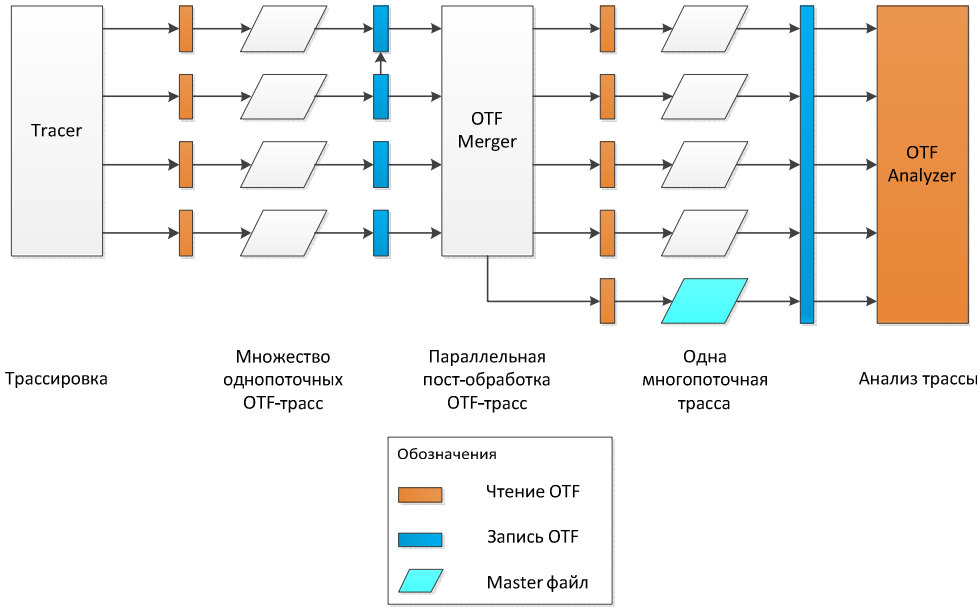

Рисунок 23. Архитектура OTF на основе потоков и файлов.

API чтения и записи трасс поддерживает три интерфейса чтения и записи на разных уровнях: на глобальном уровне (операции чтения и записи событий осуществляются одновременно с множеством всех потоков трассы), на уровне потоков (операции чтения и записи событий осуществляются с множеством процессов одного потока), на уровне файлов (т.е. фактически на уровне отдельных процессов).

В формате OTF используется специальное ASCII представление данных. ASCII кодирование позволяет уменьшить размеры для хранения малых значений, поскольку ведущие нули могут быть опущены. Все числа и переменные кодируются шестнадцатеричным представлением. Использование ASCII эффективно с точки зрения размеров получаемой трассы, возможности визуального восприятия человеком и поиску записей по временной метке. Все записи упорядочены по временной метке, поэтому применим бинарный поиск. Поисковый механизм основано на том, что записи границы могут быть быстро идентифицированы в ASCII формате.

В каждом файле OTF записи организованы в виде строк текста, в то время как детальная структура каждого типа записи определяется отдельно.

В формате OTF выделяют 4 типа записей: определения, события, снимки и статистики.

Определения. Определения могут фиксироваться как на глобальном уровне (для всех потоков), так и на уровне отдельных потоков. Определения могут объединяться в группы. В формате OTF определения используются для процессов, групп процессов, функций, групп

функций, маркеров. Перечень определений может пополняться непосредственно в процессе трассировки.

События. Записи событий составляют основную нагрузку для трассы. Каждому потоку соответствует один файл событий, в котором записи упорядочены по времени. Событиями являются вход/выход из функции, отправка/прием сообщений или замеры счетчиков. События могут объединяться в группы.

Снимки. Как правило, трассы читаются линейно с самого начала. Формат OTF предусматривает возможность получения быстрого доступа к произвольной временной метке, но для этого необходима некоторая дополнительная информация. Для того, чтобы начать чтение с произвольного места, должно быть известно состояние всех участвующих процессов. Если эта информация недоступна из чтения всех предшествующих записей, то его нужно хранить в явном виде. Для этого были разработаны записи снимков. Снимки поддерживают стек вызовов (то есть все активные вызовы функций), список ожидающих сообщений, текущие input/output действия в определенный момент времени (не включая события в этот момент), то есть своего рода контекст. На основе этой информации можно начинать читать события в этот момент времени. Снимки не генерируются сами библиотекой OTFlib, а должны быть явно добавлены. Однако, поскольку они записываются в отдельном файле, поддерживается возможность добавлять, манипулировать, заменять, удалять снимки потока не затрагивая данных о событиях.

Это было предложено для того, чтобы создавать снимки через некоторые одинаковые промежутки времени. Снимки могут быть добавлены в автоматическом режиме сразу после генерации трассы на основе некоторых требований.

Статистика. Второй класс вспомогательной информации заключается в записях статистики. Они обеспечивают обзор событий на некотором интервале времени и могут служить подсказкой для того, чтобы прочитать все события на этом интервале. Как и снимки, записи статистики хранятся в отдельном файле и могут модифицироваться, добавляться и удаляться, не затрагивая основную трассу событий. Они также должны явно создаваться и не порождаются сами библиотекой OTFlib.

3.7.3 Отображение элементов РВС РВ в трассу формата OTF

В силу определенной специфики РВС РВ и высокопроизводительных параллельных платформ необходимо произвести сопоставление основных понятий и объектов РВС РВ и понятий, используемых в формате OTF.

В соответствии с форматом: процесс в OTF можно ассоциировать с моделью компонента (узлом) РВС РВ, вход (выход) в некоторую функцию с входом (выходом) в

некоторое состояние модели (узла) в РВС РВ, метки о событиях, не связанных с обменами, в OTF с событиями в РВС РВ.

3.7.4 Дальнейшие перспективы формата OTF

В конце 2011 году в рамках проекта Score-P [68] (системы трассировки для анализа производительности параллельных приложений) университета Дрездена был разработан формат OTF2. OTF2 [69] [70] является преемником форматов OTF и Epilog и отличается новой структурой и новой реализацией. OTF2 позволяет записывать и интерактивно визуализировать трассы с 10000 процессами и с размером от 100 GB до 1 Tb. Формат OTF2 также имеет следующие особенности: хорошую производительность чтения/записи событий, высокую масштабируемость, низкие накладные расходы (дисковое пространство и время обработки), уменьшения количество файлов в формате, совместимость операции чтения для форматов OTF и Epilog. Формат имеет низкое потребление памяти по сравнению с другими форматами (24).

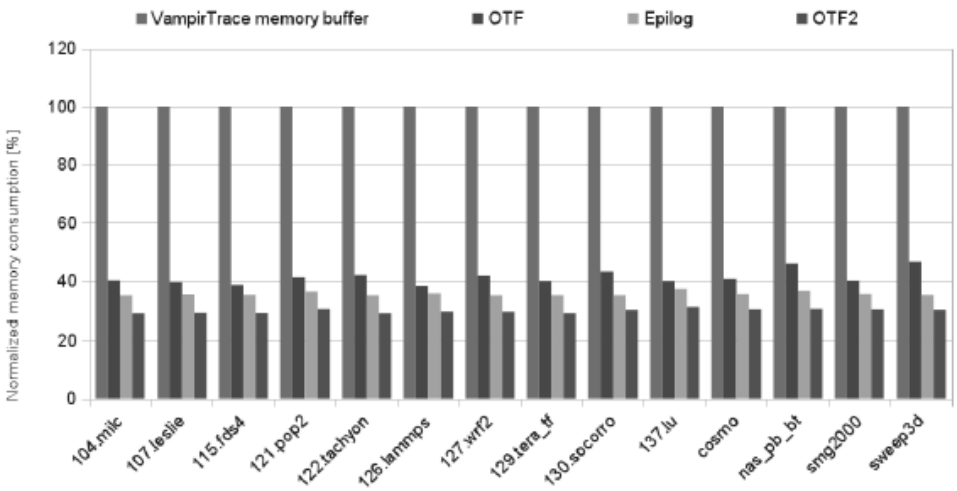

Рисунок 24. Сравнение форматов по потреблению памяти.

Поэтому, одним направлений является исследование применимости формата OTF2 для записи результатов моделирования РВС РВ. Другим направлением является вариант гибкого выбора формата трассы в зависимости от вида эксперимента и его продолжительности. Например, для длительных экспериментов или интерактивного проигрывания экспериментов целесообразно будет использовать формат OTF2.

4 Архитектура инструментальных средств поддержки анализа и разработки РВС РВ

В данном разделе описывается архитектура инструментальных средств поддержки анализа и разработки РВС РВ. В разделе 4.1 приводится общее описание архитектуры инструментальных средств поддержки анализа и разработки РВС РВ. Раздел 3.2 содержит описание редактора UML-диграмм. В разделе 4.3 приводится описание средства трансляции UML в исполняемые модели совместимые со стандартом HLA. Раздел 4.4 содержит описание среды выполнения моделей. В разделе 4.5 приведено описание средства внесения неисправностей. Раздел 4.6 содержит описание средства трассировки моделей. В разделе 4.7 приведено описание средства визуализации трассы. Раздел 4.8 содержит описание средства трансляции UML во временные автоматы. В разделе 4.9 приведено описание средства верификации моделей. Раздел 4.10 содержит описание средства оценки наихудшего времени выполнения. В разделе 4.11 приведено описание интегрированной среды разработки и анализа моделей.

4.1 Общее описание архитектуры инструментальных средств поддержки анализа и разработки РВС РВ

На основе требований, приведенных в главе 2, в рамках данной НИР была разработана среда моделирования, общая схема которой приведена на рисунке 25. На рисунке синим цветом обозначены открытые средства, используемые без модификаций, желтым – средства модифицированные, в рамках данного НИР, а зеленым – средства, полностью разработанные в рамках данной НИР.

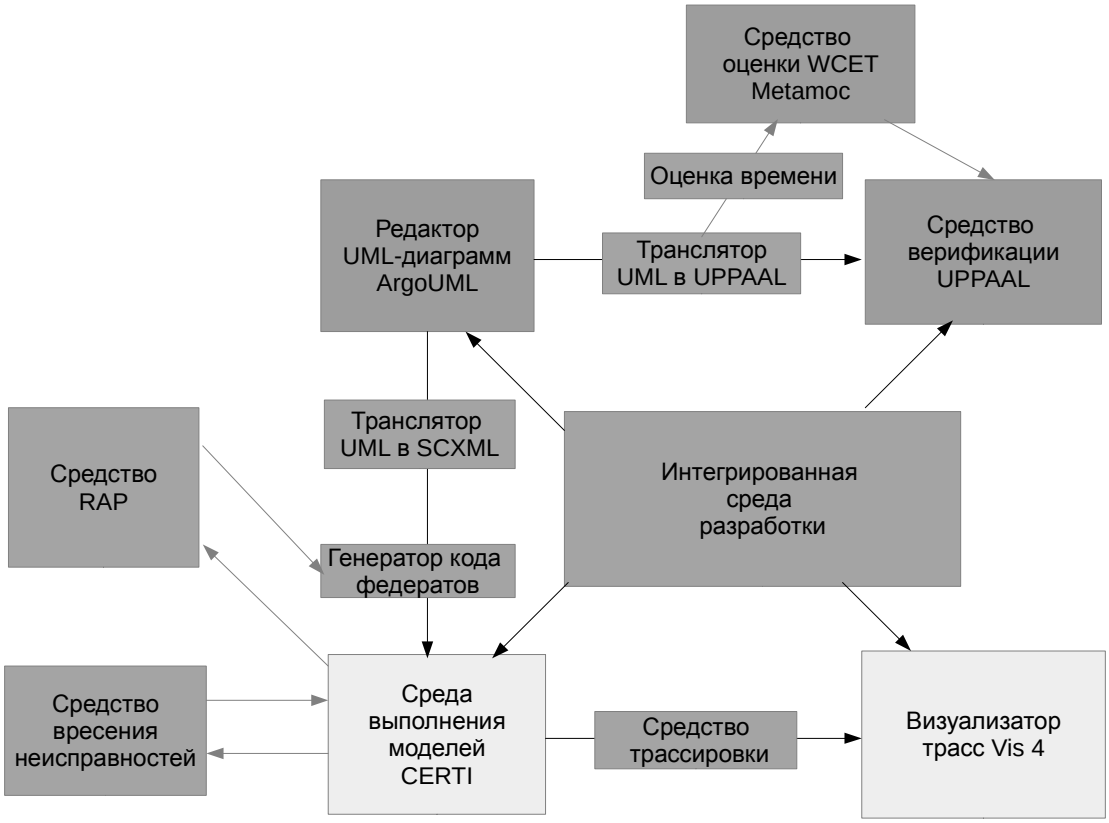

Рисунок 25. Схема системы моделирования.

В состав среды моделирования входят следующие подсистемы:

- Редактор диаграмм UML, необходимый для создания моделей РВС РВ в виде диаграмм состояний. В качестве редактора используется средство ArgoUML, подробное описание приведено в разделе 4.2.
- В средстве трансляции UML в исполняемые модели совместимые со стандартом HLA выделены две подсистемы транслятор UML в SCXML и генератор кода федератов на HLA. Эти подсистемы описаны в разделе 4.3.
- За основу среды выполнения моделей была выбрана система CERTI. В ней было исправлено несколько существенных ошибок, добавлена функциональность для поддержки моделирования РВС РВ, интеграция с натурными каналами и добавлена интеграция с библиотекой времени компиляции Proto-X, реализующей кодирование данных с использованием встроенных типов языка C++. CERTI и изменения внесённые в неё описаны в разделе 4.4.
- Средство внесения неисправностей в модель на HLA, разработанное в рамках данной НИР описано в разделе 4.5.

- Описание средства регистрации и трассировки событий моделирования, разработанного в рамках данной НИР приведено в разделе 4.6.
- За основу средства анализа и визуализации трасс взято средство анализа и визуализации трасс vis3, входящее в состав СММ КБО. Данное средство было доработано в части интеграции с форматом трассы OTF. Описание полученных в результате этих изменений средства vis4 приведено в в разделе 4.7.
- Разработанное в рамках данного НИР средство трансляции диаграмм UML в плоские временные автоматы UPPAAL описано в разделе 4.8.
- Средство верификации моделей UPPAAL описано в разделе 4.9.
- Средство оценки наихудшего времени выполнения Metamoc и средство интеграции с Metamoc описаны в разделе 4.10.
- Интегрированная среда разработки, позволяющая вызывать указанные выше средства и запускать моделирование. Также среда разработки интегрирована со средством для решения задачи выбора оптимального набора механизмов отказоустойчивости (RAP). Подробное описание среды разработки приведено в разделе 4.11.

В таблице 2 приведена краткая характеристика различных модулей программы. Всего в рамках данного проекта было написано более 700 килобайт исходного кода. Весь исходный код документирован с помощью систем doxygen (для C++) и sphinx (для Python).

Таблица 2. Возможности языков моделирования.

Модуль	**Размер (Кб)**	**Язык программирования**	**Операционная система**
Интегрированная среда разработки	32	Python	Windows, Linux
Транслятор UML в UPPAAL и SCXML	187	Python	Windows, Linux
Средства интеграции со средством оценки наихудшего времени выполнения Metamoc	5	Bash	Linux
Генератор кода федератов	196	Python, cheetah	Windows, Linux
Доработки CERTI: • Патч для среды выполнения моделей • Интерфейс между RTI и федератами • Тестовый пакет • Загрузчик • Инсталлятор	 • 130 • 56 • 31 • 6 • 5	C++, bash, Python	Linux
Визуализатор трасс Vis 4		C++	Linux
Средство трассировки		C++	Linux
Средство интеграции со средством для решения задачи выбора оптимального набора механизмов отказоустойчивости RAP	13	Python, shell	Windows, Linux
Средство внесения неисправностей	20	C++	Windows, Linux

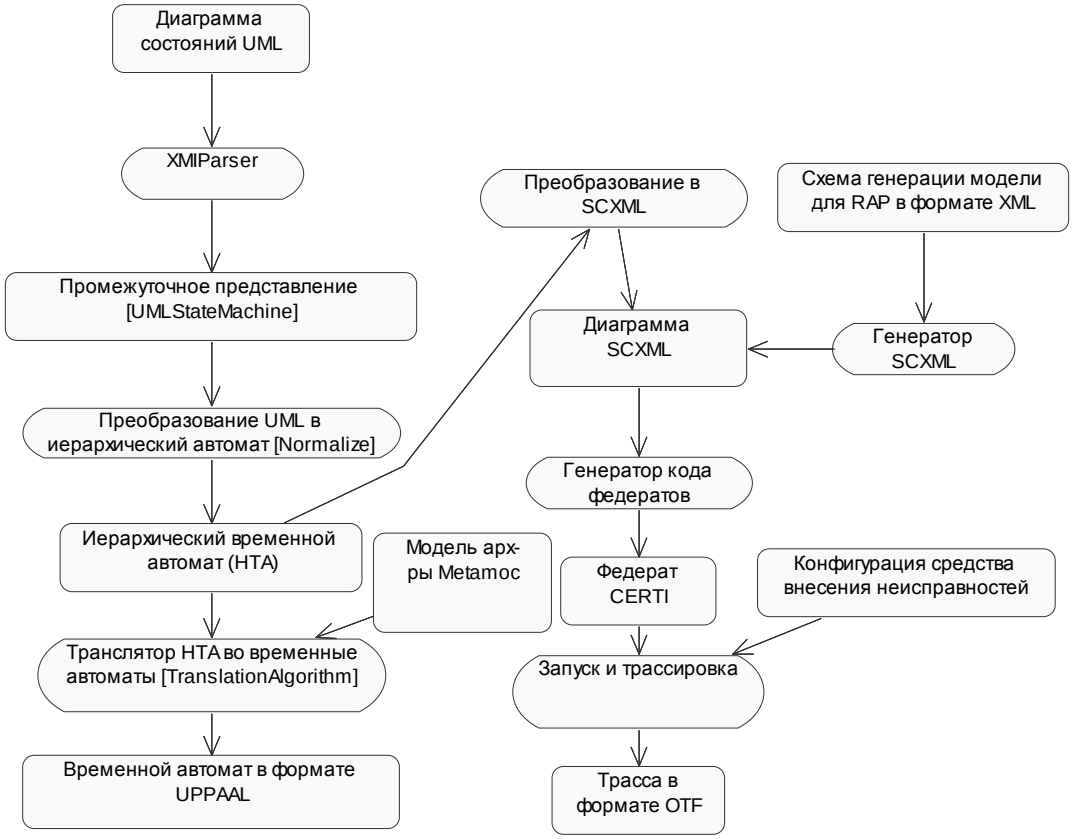

Рисунок 26. Форматы файлов, используемые в системе и их преобразования

На рисунке 26 показаны преобразования, которые происходят с файлами в процессе работы системы. Более подробно эти преобразования будут описаны в разделах 4.2 – 4.11.

4.2 Редактор UML-диаграмм

При разработке модели при помощи диаграмм состоянии UML в рамках данного проекта возникает необходимость использования графического средства разработки этих диаграмм. При разработке РВС РВ сложность внутреннего устройства их компонентов достаточно велика и элементарное неудобство и необходимость выполнения огромного числа лишних действий для создания требуемого элемента диаграммы может существенно затруднить создание необходимых моделей. Поэтому удобство и наглядность также являются одними их важнейших критериев при выборе редактора UML-диаграмм. В частности, при рассмотрении интерфейса для создания диаграмм состояний в большинстве из рассматриваемых средств, удобство и наглядность в наибольшей степени зависит от наличия/отсутствия таких возможностей как:

- Быстрое создание перехода из состояния в состояние (без использования его переноса мышью из списка доступных элементов).
- Быстрая запись сторожевых условий и триггеров прямо на переходах из состояния в состояние.
- Поддержка операции отмены совершенного действия.

При разработке большинства достаточно крупных систем возникает необходимость импорта/экспорта данных из одного используемого инструмента проектирования в другой. Касательно проектирования на языке UML это означает необходимость в обеспечении взаимодействия выбранного средства проектирования с другими, посредством стандартного формата XMI. В частности, данные в этом формате поступают на вход верификатора моделей, поэтому следующим критерием сравнения рассматриваемых средств является поддержка ими импорта/экспорта в формате XMI.

В рамках задачи сравнения средств UML-моделирования были проанализированы следующие средства с открытым исходным кодом: Papyrus [71], Moskitt [72], VioletUML [73], TinyUML [74], ArgoUML [75], Topcased [76], BOUML [77].

Сравнение проводилось в два этапа, на первом из которых все перечисленные средства исследовались на предмет наличия в них по крайней мере потенциальной возможности создания диаграмм описанного раннее вида, а на втором лишь те из перечисленных средств, что прошли отбор на первом этапе. Практически сразу были исключены из рассмотрения средства: VioletUML TinyUML и QM. Первые два оказались слишком примитивными и скорее подходят для обучения, чем для разработки моделей. Средство QM ставит своей целью создание визуального средства разработки приложений для встроенных систем на основе диаграмм состояний, причем возможна кодогенерация в C/C++ для embedded-платформ и имеется встроенная валидация моделей.

В итоге ни один из редакторов полностью не удовлетворяется всем предъявленным требованиям. В основном из-за отсутствия гибкой настройки генераторов кода, что делает невозможным написания собственных шаблонов для генерации кода федерации и федератов HLA. Стоит отметить, что полная поддержка UML 2, в результате оказалась не необходимым условием, так как данная версия стандарта не вносит существенных изменений в диаграммы состояний UML.

В результате был выбран UML редактор ArgoUML. В текущей реализации разработанного в рамках данного НИР средства моделирования использовалась версия ArgoUML 0.32.2. Определяющими факторами выбора редактора ArgoUML стали: удобство интерфейса, поддержка экспорта в XMI. Возможность кодогенерации по диаграммам состояний отошла на второй план, так как в рамках проекта реализован собственный

генератор кода, имеющий необходимую функциональность для создания федерации и федератов HLA по XMI представлению диаграмм состояний. ArgoUML полностью написан на Java. ArgoUML является открытым программным обеспечением. Распространяется под лицензией BSD. ArgoUML имеет интуитивно понятный и насыщенный пользовательский графический интерфейс (рисунок 27).

Из полезных особенностей редактора:

- Поддержка спецификаций UML 1.3, 1.4.
- Экспортирование и импортирование в формат XMI 1.0, 1.1, 1.2.
- Генерация исходного кода Java, C++, C# и PHP.
- Обратный инжиниринг из исходного кода и байткода Java.
- Автоматическую верификацию модели UML (design critics).

Нереализованные функции UML редактора:

- отсутствие функции обратного проектирования (нет возможности создавать модели из имеющегося кода на Java).
- нет импорта файлов, созданных в других пакетах для работы с UML.

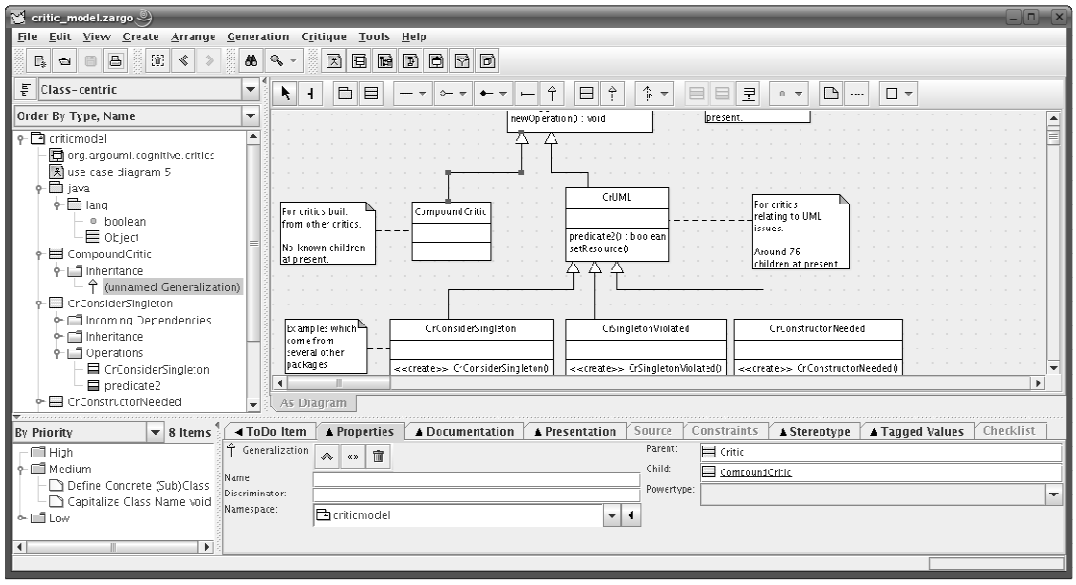

Рисунок 27. Окно редактора ArgoUML.

В разделе 8.2 описана методика использования данного редактора.

4.3 Средство трансляции UML в исполняемые модели совместимые со стандартом HLA

В данном разделе описывается основной процесс трансляции исходного кода моделей, заданных при помощи диаграмм состояний UML и представлены основные этапы генерации исходного кода моделей.

Стандарт HLA предполагает, что отдельные имитационные модели (или несколько имитационных моделей), предназначенных для использования в одном приложении, могут быть легко использованы в другом приложении, если их разработчики придерживаются концепции федератов. Исходя из стандарта HLA, необходимо для каждой имитационной модели отдельно описать несколько десятков интерфейсов для взаимодействия с Runtime Infrastructure (RTI). Под RTI понимают среду передачи данных между компонентами, входящими в модель. Следовательно, помимо затрат на построение самой модели, разработчику необходимо дополнительно заниматься описанием интерфейсов для взаимодействия с RTI.

4.3.1 Общая схема генерации кода федерата

На вход средству генерации подается диаграмма состояний языка UML и шаблон для генерации кода [78]. Далее диаграмма переводится во внутреннее представление языка Питон. После создания XML файла [79], он подаётся на вход непосредственно генератору кода по шаблону, который генерирует исходный код модели. Для генерации кода федератов (с интерфейсами для подключения к RTI) были созданы особые шаблоны [80]: отдельно для .h, .cpp и .fed файлов. На рис. 28 представлена схема работы генератора кода моделей совместимых со стандартом HLA. Примеры работы представлены в статье [81].

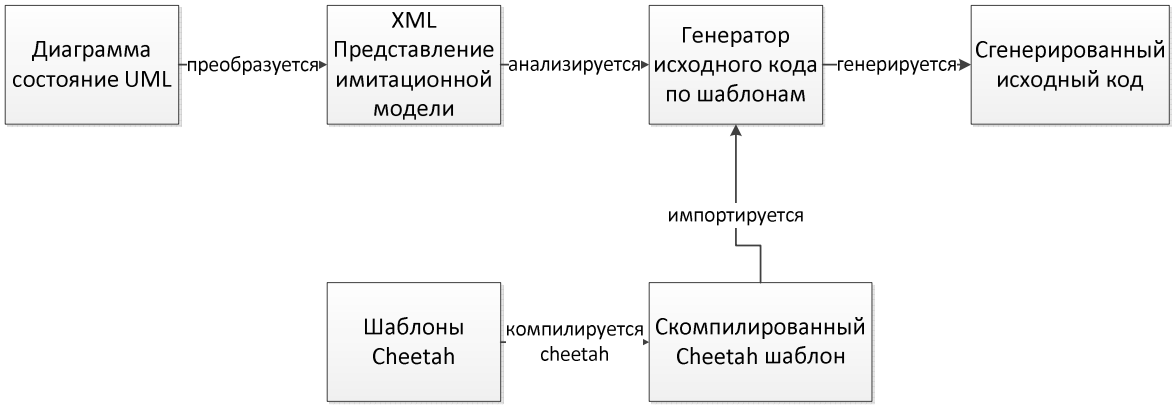

Рисунок 28. Общая схема генерации исходного кода федерата

Для генерации кода внутренней логики федерата (рис. 29) были созданы шаблоны для трансляции кода автомата, описывающие данную логику. При реализации автомата пользователь системы моделирования использует следующие примитивы:

- *Заметка (UML — Note).* Заметки могут использоваться как для пояснения, так и для задания дополнительных атрибутов объектам автомата внутренней логики.
- *Обобщение (UML — Generalization).* Соединяет объекты и заметки "UML — Note", содержащие расширенную информацию об объекте.

- *Начальное/конечное состояние (UML — State Term).* Обозначают начальные и конечные состояния. Начальное/конечное состояние не имеет названия, для начального состояния оно не требуется, для конечного необходимо добавить заметку "UML — Note" с именем состояния и связать с конечным состоянием при помощи "UML — Generalization".
- *Состояние (UML — State).* Обозначает состояние автомата. Атрибуты State:
    1. Входное действие (entry action) — действие, выполняемое при входе в состояние. Не выполняется при внутренних переходах.
    2. Внутренние действие (do action) — действие, выполняемое после внутреннего перехода.
    3. Выходное действие (exit action) — действие, выполняемое при выходе из автомата. Не выполняется при внутренних переходах.
- *Переход (UML — Transition).* Задает переход из одного состояния в другое. Атрибуты Transition:
    1. *Триггер (trigger)* — условие (событие) перехода.
    2. *Действие (action)* — действие, выполняемое при переходе.
    3. *Хранитель (guard)* — дополнительное условие перехода.

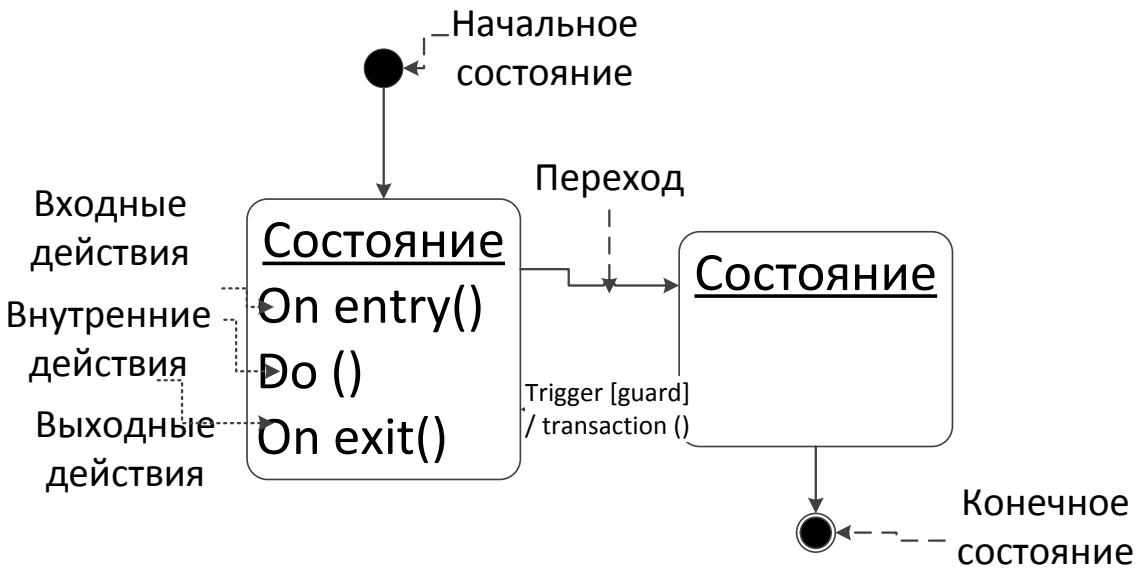

Рисунок 29. Схема автомата внутренней логики федерата

Каждое состояние автомата преобразуется в отдельный класс на языке C++. Для этого были созданы шаблоны для трансляции исходных файлов и файлов заголовков для состояний автомата (рис. 30).

Для управления переходами из одного состояния в другое используется класс *Controller*. Экземпляр данного класса создается для каждого автомата внутренней логики отдельно. Перед данным классом ставятся следующие задачи:

- Предоставлять для каждого состояния метод *InitializeSM*(). При вызове данного метода происходить инициализация состояния и выполняется внутренний код состояния.
- Предоставлять метод *triggerEvent*(). Метод позволяет менять состояния исходы из вновь поступивших входящих событий.

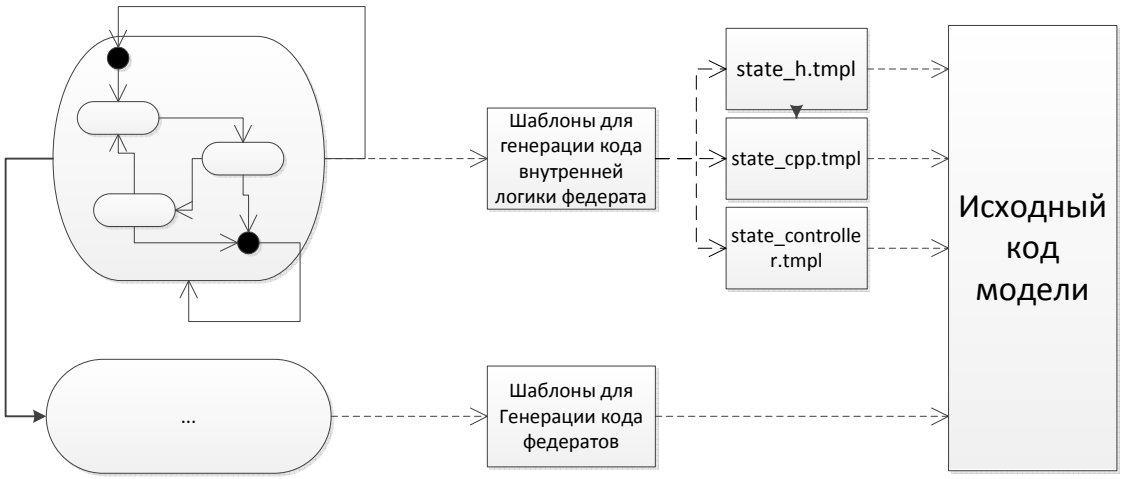

Рисунок 30. Схема работы транслятора кода федерата

При трансляции класса можно задать заголовочные файлы, которые будут подключены перед определением класса. Для этого с классом автомата связывается объект "UML — Note", первой строкой которого является строка расширенных атрибутов, в остальных — имена подключаемых файлов в угловых скобках или кавычках. Можно также добавлять код методов прямо на диаграмму. Для этого с классом связывается "UML — Note" с кодом метода.

4.3.2 Cheetah шаблоны для генерации исходных кодов модели

Полученное на предыдущем этапе SCXML представление модели подается на вход генератору исходных кодов на основе библиотеки шаблонов cheetah. Подробное описание работы с шаблонами cheetah описанной в отчетах на предыдущих этапах проекта [3,4].

Для генерации исходного кода используются следующие шаблоны (рис. 31):

- federate_hpp.tmpl, federate_cpp.tmpl– для генерации исходного кода федерата;
- generic_tickFed_h.tmpl, generic_tickFed_cpp.tmpl – для генерации исходного кода федерата синхронизации времени;
- generic_FEDfile.tmpl – для генерации файла федерации;

- launcher.tmpl – для генерации python-скрипта для запуска федерации;
- main_cpp.tmpl – для генерации main класса федерации;
- inter_class_table.hpp.tmpl – для генерации таблицы interactions;
- parameter_table.hpp.tmpl – для генерации таблицы параметров, которыми обмениваются федераты;
- simple_datatype_table.hpp.tmpl – для генерации таблицы типов параметров, которыми обмениваются федераты;
- som.hpp.tmpl – для генерации SOM схемы федерации.

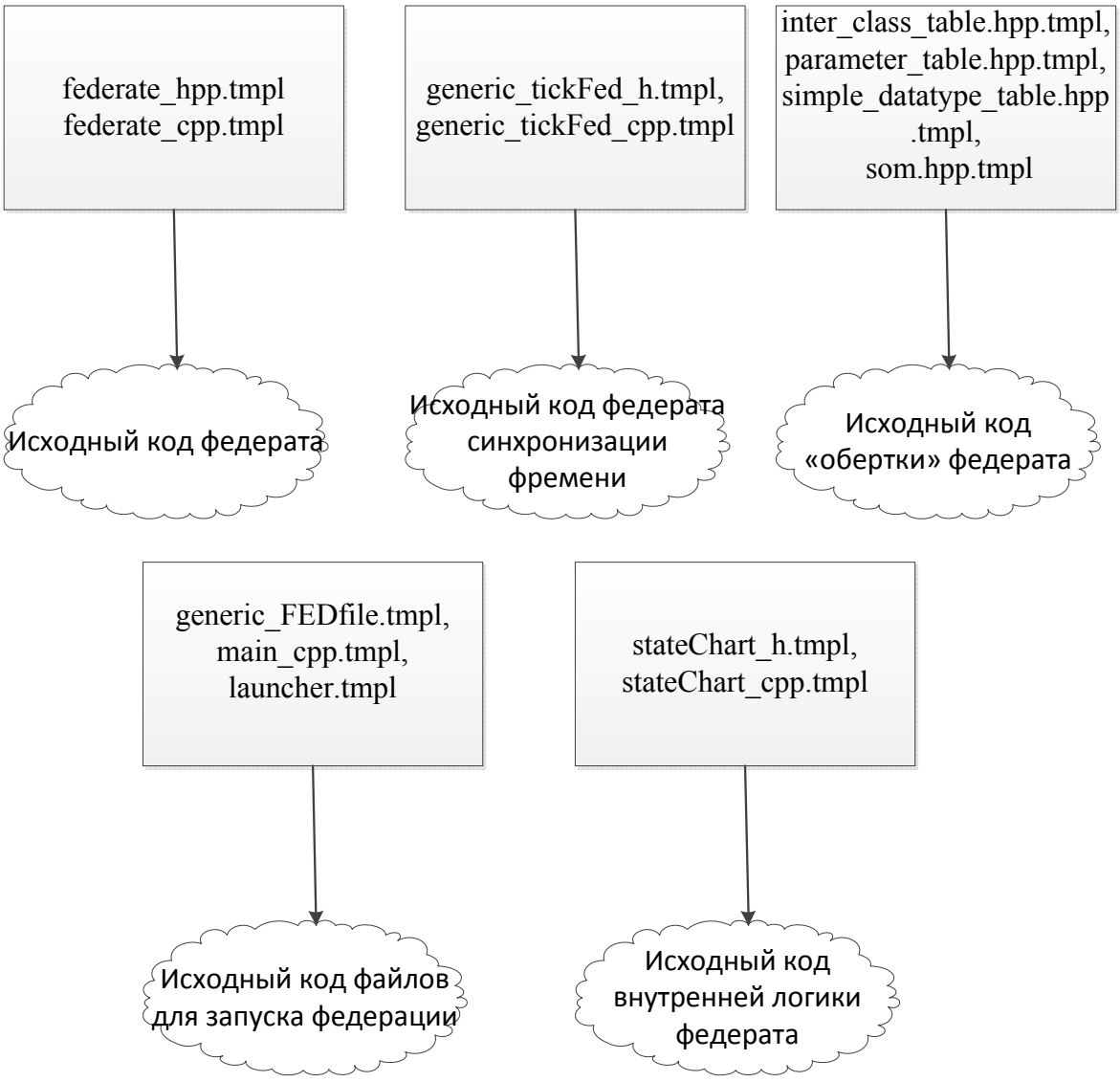

Рисунок 31. Схема шаблонов дял генерации различных частей исходных кодов федерации

Для описания внутренней логики работы каждому федерату в федерации ставится в соответствие автомат (данный автомат разрабатывается на этапе создания UML

представления модели). Для генерации исходного кода по данному автомату для генератора был создан специализированный cheetah шаблон. Подробный разбор данного шаблона представлено далее.

```
int  ${state.id}_State hart::run(void)
{
  InitializeSM();
  /* main cycle */
  while (1)
  {
    (this->*state_table[curr_state])();
    //DecrementTimer();
    /* insert you code - if needed */
  }
  return 0;
};
int  ${state.id}_State hart::make_step(void)
{
  //state_table[curr_state]();
  (this->*state_table[curr_state])();
  //DecrementTimer();
  /* insert you code - if needed */

  return 0;
};
```

Рисунок 32. Шаблон функций запуска автомата внутренней логики работы федерата

На рисунке 32 представлены описания двух базовых функций по управлению конечным автоматом внутренней логики. Функция run() предназначена для автономной работы с автоматов (для тестовых запусков федерата вне федерации), функция make_step() предназначена для запуска внутри федерации. Во втором случае бесконечный главный цикл модели описывается в шаблоне, отвечающим за генерацию кода федерата внутри федерации.

```
void ${state.id}_State□hart::InitializeSM(void)
{
  curr_state = (State_Type)0;
  #if $state.__dict__.has_key('parametr') :
    #for $parametr in $state.parametr :
  ${parametr.name} = ${parametr.initval};
    #end for
  #end if
  #if $state.__dict__.has_key('var') :
      #for $var in $state.var :
    ${var.name} = ${var.initval};
      #end for
  #end if
  //timer = 0.0;
  /* init code */
}
```

Рисунок 33. Шаблон функции инициализации запуска автомата внутренней логики работы федерата

На рисунке 33 представлено описание функции инициализации автомата внутренней логики. В этой функции устанавливаются значения всех параметров федерата, и текущие состояние автомата федерата указывает на начальное состояние.

```
void ${headState.id}_State□hart::${state.id}_func(void)
{
        #if $state.__dict__.has_key('transition') :
        #for $transition in $state.transition:
          #if $transition.__dict__.has_key('parametr')==0 :
            #if $transition.__dict__.has_key('cond') :
  if (${transition.cond})
  {
              #if $transition.__dict__.has_key('event') :
    ${transition.event};
              #end if
    curr_state = ${transition.target.upper()};
  }
            #end if
          #end if
        #end for
        #end if
}
```

Рисунок 34. Шаблон функции состояний автомата внутренней логики работы федерата

На рисунке 34 представлено описание шаблона функции для каждого состояния автомата внутренней логики. При вызове функции, отвечающей за конкретное состояний, происходит проверка возможность перехода в другое состояние (соблюдение сторожевых условий перехода) и изменения значений параметров, связанных с данным переходом. В конце каждой такой функции обязательно смена значения указателя на текущее состояние.

Вызов конечного автомата внутренней логики встроен в код федерата. Генерации исходного кода данного автомата осуществляется при помощи шаблонов stateChart_h.tmpl, stateChart_cpp.tmpl.

В данном разделе приведено описание средства трансляции UML в исполняемые модели совместимые со стандартом HLA. Описание экспериментального исследования данных средств приведено в разделе 9.1.

4.4 Среда выполнения моделей

Данный раздел работы описывает ядро разрабатываемой системы моделирования – среду выполнения имитационных моделей, отвечающую за взаимодействие отдельных участников моделирования между собой. Встроенные в среду выполнения примитивы и алгоритмы синхронизации обеспечивают возможность композиции относительно простых элементов модели. Таким образом, модель сложной исследуемой системы представляется в виде совокупности множества независимых моделей её компонентов, которые могут разрабатываться независимо друг от друга.

В процессе выполнения настоящей исследовательской работы, в частности, была реализована среда выполнения, которая может быть использована для практического решения задачи имитационного моделирования РВС РВ. В данном разделе приводится использованная при этом идея построения среды, реализующей спецификации стандарта распределённого имитационного моделирования High Level Architecture (HLA). Раздел содержит описание существующих на данный момент практических реализаций данного стандарта, анализ пригодности этих реализаций для задачи моделирования систем реального времени и обоснование выбора наиболее перспективной из них – CERTI.

Далее приводится анализ программной архитектуры системы CERTI, выделяются её преимущества и недостатки по сравнению с другими реализациями стандарта HLA. Описывается ряд доработок, которые были внесены в данную систему для того, чтобы сделать её более эффективной при решении практических задач моделирования РВС РВ. Дополнительно рассматривается несколько направлений перспективных исследований и необходимых технических работ для дальнейшего увеличения эффективности данной среды выполнения.

4.4.1 Стандартизация интерфейса среды выполнения

Устройства в составе РВС РВ нередко используются повторно: значительная часть оборудования (различные датчики, устройства отображения информации, управляющие контроллеры) устанавливается сразу в несколько различных вычислительных систем. При этом появляется закономерное желание применить такой же приём и к их моделям. Однако практическая реализации этой идеи усложняется необходимостью подгонять интерфейс модели под её новое окружение, под модели, с которыми она должна взаимодействовать. В области программной инженерии данная проблема эффективно решается с помощью сервис ориентированных архитектур, которые разрывают зависимости между отдельными компонентами системы, предоставляя им необходимый набор сервисов. Аналогичный подход применим и при построении среды выполнения: она должна предоставлять участникам моделирования набор сервисов, достаточный для описания их поведения и исключения их прямого взаимодействия друг с другом.

Иногда в процессе разработки сложных систем удобно использовать представление системы в виде множества *модулей*, набор и реализации которых определяют её поведение и свойства. Для использования аналогичного подхода при построении моделей РВС РВ, сервисы среды выполнения должны разрывать не только интерфейсные, но и логические зависимости между отдельными компонентами модели. Примером подобного интерфейса является сервис передачи данных по схеме «издатель-подписчик». Издатель публикует данные заданного типа, передавая их среде выполнения, а среда выполнения передаёт поступившие данные всем запросившим этот тип подписчикам. Таким образом, среда выполнения сохраняет анонимность участников обмена: издатель не знает, кто получил опубликованные данные, а подписчику остаётся неизвестен их источник. В ряде случаев данное свойство позволяет безболезненно изменять количество участников модели.

Проектирование интерфейса среды выполнения, сервисы которого обладали бы свойством полноты, позволяли изолировать участников моделирования друг от друга и к тому же допускали бы достаточно эффективную реализацию – нетривиальная задача. Поэтому разработчики современных сред выполнения обычно поддерживают один из известных стандартов [82],[83]. В области моделирования реального времени и, тем более, полунатурного моделирования таких стандартов на сегодняшний день не существует. Известны попытки построения соответствующих сред выполнения на основе стандарта моделирования High Level Architecture (HLA) [84]. Но использование данного стандарта для моделирования в реальном времени требует его доработки [85]:

1. HLA не предоставляет интерфейса для задания параметров качества сервиса, поэтому участники моделирования не вправе требовать от среды выполнения, например, обработки события за заданное время;
2. Стандарт не предусматривает механизмов управления ресурсами операционной системы, поэтому она может, например, неожиданно откачать из оперативной памяти страницы процессов моделирования или дать другим существующим задачам больший приоритет в использовании процессорного времени;
3. HLA поддерживает только две политики QoS для обмена данными: надежный и ненадежный обмен (обычно реализованных с помощью протоколов TCP и UDP), чего недостаточно для моделирования в реальном времени.

Тем не менее, скорость развития и динамика распространения HLA на смежные области моделирования говорят о большей целесообразности его доработки для реального времени, чем о необходимости разработки нового интерфейса среды выполнения с нуля. Поэтому для проведения имитационных экспериментов в настоящей работе используется среда выполнения, реализующая спецификации стандарта HLA. Подробнее об этом стандарте можно прочитать в разделе 3.2.

4.4.2 Выбор среды выполнения моделей

На данный момент существует множество готовых реализаций стандарта HLA, что даёт возможность использовать существующие наработки, не реализуя необходимую RTI с нуля. В качестве критериев оценки применимости конкретных реализаций RTI для целей настоящего проекта выступали следующие:

1. Версия и полнота поддерживаемого стандарта HLA;
2. Доступность описания программной архитектуры и принципов реализации RTI;
3. Доступность исходного кода и возможность его использования;
4. Поддержка средства со стороны его разработчиков;
5. Использование средства для моделирования систем реального времени.

Большая часть найденных реализаций RTI (таблица 3) представляют собой коммерческие продукты с закрытым исходным кодом [86],[87],[88],[89]. Их описание принципов их работы не даёт необходимый уровень детальности или недоступно. Наравне с коммерческими системами было найдено так же несколько реализаций с открытым исходным кодом. Реализация RTI ARTIS GAIA привлекает встроенными в неё механизмами балансировки программных моделей между инструментальными машинами, но лицензия данной среды выполнения не допускает свободного использования её кода (хотя заявлено, что в будущем этот проект станет полностью открытым) [90]. Разработка реализации

EODiSP была заморожена в 2006 году, что делает её рассмотрение нецелесообразным [91]. Реализация RTI Portico разработана с использованием языка программирования Java, который плохо приспособлен к работе в режиме реального времени, требуемом для решения имитационных задач данного проекта [92].

Таблица 3. Реализации RTI.

Название RTI	Разработчик	Версия стандарта HLA	Тип лицензии
ARTIS GAIA	University of Bologna	DMSO 1.3	Открытый код
CERTI	ONERA	DMSO 1.3, IEEE 1516 2000	GPLv2
EODiSP	P&P Software	IEEE 1516 2000	GPL
MAK	MAK Technologies	DMSO 1.3, IEEE 1516 2000, HLA Evolved	Коммерческая
NCWare	Nextel	IEEE 1516 2000	Коммерческая
Portico	Portico	DMSO 1.3, IEEE 1516 2000	CDDL
pRTI	Pitch Technologies	DMSO 1.3, IEEE 1516 2000, HLA Evolved	Коммерческая
RTI NG Pro	Raytheon	DMSO 1.3, IEEE 1516 2000	Коммерческая

Таким образом, наиболее перспективной реализацией RTI является распределённая система моделирования с открытым исходным кодом CERTI, вокруг которой уже долгое время функционирует сообщество энтузиастов, продолжающих разрабатывать новые и совершенствовать существующие её подсистемы . Система CERTI реализована на языке C++, поддерживает основные ОС (Windows and Linux, Solaris, FreeBSD) и компиляторы (gcc, MSVS, Sun Studio, MinGW) [93].

4.4.3 Архитектура среды выполнения CERTI

Любая реализация RTI является распределённой программной системой и обеспечивает множество подключённых к ней участников моделирования стандартным интерфейсом HLA, скрывая при этом детали их взаимодействия, включая сетевое. Эта цель достигается благодаря построению RTI, как совокупности удалённых компонентов, соответствующих участникам моделирования. Таким компоненты работают локально на той же машине, что и федерат, и обычно называются *локальными компонентами RTI* (Local RTI Component, RLC) [94].

Инфраструктура RTI постоянно должна поддерживать согласованность отдельных компонентов имитационной модели с помощью встроенных механизмов синхронизации.

Поэтому эффективность синхронизации оказывает огромное влияние на общие показатели производительности системы. Полностью распределённая архитектура RTI предполагает равнозначность и самодостаточность её локальных компонентов LRC, а её реализация требует использования сложных распределённых алгоритмов согласования. Поэтому разработчики обычно отказываются от полностью распределённой архитектуры и вводят понятие *центрального компонента RTI* (Central RTI Component, CRC), который хранит разделяемые данных выполняемой модели и производит синхронизацию участников моделирования локально. Обе модели, как централизованная, так и децентрализованная, имеют свои сильные и слабые стороны, и их взвешенное и обоснованное совмещение является краеугольным камнем любой эффективной реализации RTI [93].

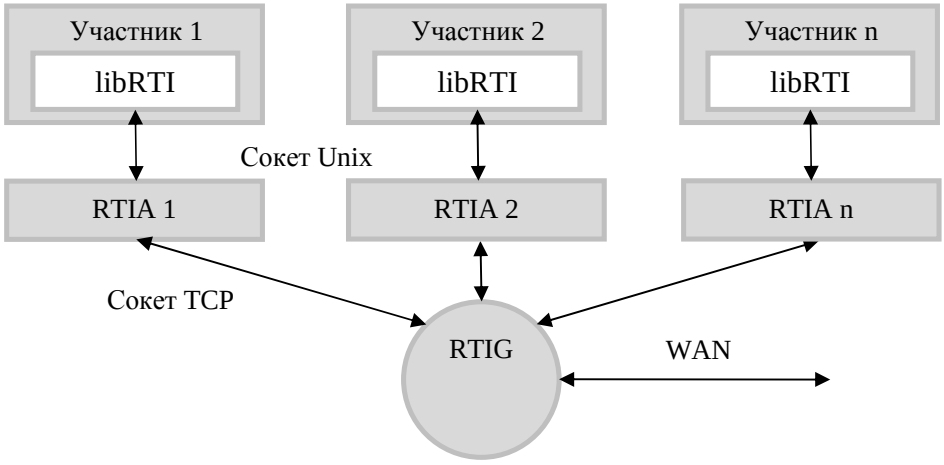

Рисунок 35. Архитектура CERTI RTI.

В общем виде архитектура CERTI RTI может быть представлена в виде совокупности трёх компонентов [95]: RTI Gate (RTIG), RTI Ambassador (RTIA) и libRTI (Рисунок 35). Процесс RTIG может выполняться на отдельном вычислительном узле, к которому не подключён ни один участник моделирования, и является центральным компонентом CRC системы CERTI. Процесс RTIA, напротив, выполняется на том же узле, что и участник моделирования. Таким образом, число запущенных процессов RTIA во время выполнения модели равно числу участников моделирования. Взаимодействие между процессами RTIG и RTIA происходит через сокет TCP.

В то время как процессы RTIG и RTIA формируют «внутренности» CERTI RTI, библиотека времени выполнения libRTI непосредственно реализует интерфейс стандарта HLA. Библиотека libRTI линкуется с процессом участника моделирования и отвечает за его соединение с соответствующим процессом RTIA через канал передачи данных

операционной системы (Unix Socket). Таким образом, локальный компонент LRC системы CERTI состоит из процесса RTIA и библиотеки libRTI.

Процессы RTIA никогда не взаимодействуют друг с другом напрямую – весь обмен данными идёт через процесс RTIG. Таким образом, система CERTI имеет полностью централизованную архитектуру, и её центральный компонент CRC единолично управляет выполнением имитационной модели. Процесс RTIG реализует большую часть сервисов и служб RTI, в то время как единственная цель процесса RTIA и библиотеки libRTI – создание удобной коммуникационной инфраструктуры между центральным процессом RTIG и процессами федератов. Другими словами, LRC системы CERTI служит, главным образом, каналом передачи данных между её CRC и участниками моделирования.

Компонент LRC [94] системы CERTI состоит из библиотеки libRTI и процесса RTIA, соединённых сокетом Unix. Хотя библиотека libRTI и линкуется с процессом федерата, обеспечивая его интерфейсом к службам и сервисам HLA, эта библиотека не реализует логику инфраструктуры RTI. Вместо этого библиотека просто перенаправляет получаемые от федерата запросы присоединённому к ней процессу RTIA. Более точно, при каждом вызове одного из сервисов RTI библиотека пересылает процессу RTIA сообщение с идентификатором вызванного метода и набором поступивших при этом аргументов. Процесс RTIA обрабатывает поступившее сообщение и отвечает федерату.

Таким образом, библиотека libRTI является интерфейсной частью LRC системы CERTI, а процесс RTIA реализует внутреннюю логику приложения. Благодаря такому разделению компонента LRC на независимые модули, возможные изменения спецификаций стандарта HLA не способны повлиять на логику компонента LRC, и соответствующие изменения инфраструктуры моделирования потребуют минимальных трудозатрат. На данный момент система CERTI использует описанную возможность для одновременной поддержки сразу двух версий стандарта HLA: более старого DMSO 1.3 и более нового IEEE 1516 2000 [95].

Другим преимуществом двух-компонентного LRC перед монолитным является его большая надежность и защищённость. При двухкомпонентной организации библиотека libRTI и процесс RTIA выполняются независимо друг от друга и проверяют каждое поступающее к ним сообщение. Поэтому федерат никак не может прочитать или изменить внутренние данные инфраструктуры моделирования RTI, кроме как через стандартные интерфейсы HLA. По этой же причине отказ внутри федерата сам по себе не может стать причиной отказа всей системы моделирования.

К сожалению, гибкость модульного компонента LRC снижает общие показатели производительности системы. Рассмотрим передачу сообщений между федератами с точки

зрения процессов CERTI. Федерат-отправитель вызывает соответствующий метод библиотеки libRTI. Библиотека отправляет их процессу RTIA в виде сообщения через сокет Unix. Процесс RTIA анализирует сообщение и переправляет полученные данные процессу RTIG сообщением через сокет TCP. Затем RTIG передаёт данные федерату-получателю, используя аналогичную цепочку процессов в обратном порядке. При этом важно отметить, что передача каждого сообщения через сокет требует предварительной сериализации передаваемых данных и их последующей распаковки. Таким образом, передача одного сообщения между федератами приводит к передаче четырёх сообщений и восьми конвертаций формата данных. Первый федерат отправляет сообщение своему RTIA, RTIA передаёт его процессу RTIG, далее цепочка повторяется в обратном порядке. Конвертация данных требуется при каждой передаче.

Описанный процесс обмена сообщениями между компонентами инфраструктуры RTI приводит к неоправданно большому времени обработки события. В то же время, данный показатель является ключевым для систем моделирования реального времени и, в частности, систем полунатурного моделирования. Он определяет минимальный размер допустимого директивного интервала и фактически задаёт диапазон полунатурных задач, которые способна решать конкретная система моделирования. Данное замечание делает крайне перспективной любую модификацию системы CERTI, которая была бы способна уменьшить время обработки события и не требовала бы при этом чрезмерных трудозатрат. Несколько таких модификаций рассматриваются в последующих секциях данного раздела.

4.4.4 Изменение модели потоков управления CERTI

Одним из способов повышения общей эффективности системы CERTI может служить объединение процесса RTIA и процесса федерата в один единственный процесс, при котором процесс RTIA выполнялся бы в контексте процесса федерата как отдельный поток управления. При такой организации процессов RTIA и libRTI будут выполняться в общем контексте, поэтому информация между ними может передаваться как обычный указатель. Тем самым исключается необходимость механизма сообщений, приводящего к дополнительной конвертации и излишнему копированию данных, что позволяет снизить расходы на передачу данных между федератами почти вдвое.

Аналогичных результатов можно достичь и с помощью разделяемой памяти, однако организация взаимодействия на уровне потоков позволяет использовать более эффективные механизмы синхронизации, чем существующие на уровне взаимодействия полновесных процессов. В частности, потоки способны более эффективно использовать доступные вычислительные ресурсы.

Раздел стандарта HLA, описывающий интерфейс RTI и правила её взаимодействия с подключёнными федератами, вводит понятия *прямого* и *обратного вызова* [96]. Прямой вызов происходит при обращении федерата к RTI, то есть во время использования этим федератом одним из сервисов инфраструктуры RTI. Обратный вызов, наоборот, происходит при обращении инфраструктуры RTI к одному из методов подключённого к ней федерата. Например, один из методов федерата вызывается при получении им сообщения или при его уведомлении об успешности предшествующего прямого вызова.

Разработчики стандарта HLA стремились создать такую среду выполнения, которая бы накладывала минимальные ограничения на федератов, которые к ней подключатся. В частности, предполагалось, что федераты могут использовать любое количество потоков управления. Согласно этой логике если федерат использует единственный поток, то странно заставлять его заботиться о состояниях гонки, которые могут возникнуть во время обратного вызова, когда внутренний поток RTI может попытаться получить доступ к данным федерата. Поэтому инфраструктура RTI никогда не обращается к федерату, пока он сам это обращение не разрешит.

В итоге разработчики HLA предусмотрели два принципиально различных способа получения обратных вызовов: обратиться к методу RTI, который совершит однократный обратный вызов внутри себя, или же разрешить асинхронные вызовы. В последнем случае разработчик федерата берёт ответственность за корректное выполнение программы на себя. При этом инфраструктура RTI может вызывать один из методов федерата в одном из собственных потоков управления в произвольный момент времени.

Прямые и обратные вызовы на практике могут быть реализованы несколькими различными способами в зависимости от модели потоков управления, лежащей в основе конкретной RTI. Текущая версия RTI CERTI использует в своей реализации только полновесные процессы, поэтому вопросы выбора подходящей модели потоков управления для неё некорректны. Однако данный вопрос приобретает актуальность в свете того, что предложенная модификация CERTI предполагает замену полновесного процесса RTIA отдельным потоком управления в контексте процесса федерата.

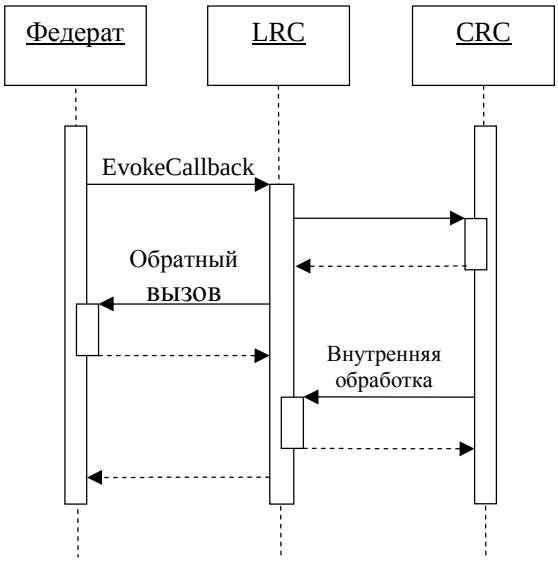

Рисунок 36. Схема работы асинхронной модели управления.

Существует три различных модели потоков управления среды выполнения RTI: однопоточная, асинхронная и многопоточная [94]. Наиболее подходящей из них для целей данного проекта с точки зрения эффективности модели и требуемых для её реализации затрат является асинхронная модель потоков, применяющаяся, в частности, в RTI от компании Pitch – pRTI [97]. Асинхронная реализация (рисунок 36) RTI использует один или несколько внутренних потоков управления, которые самостоятельно обеспечивают взаимодействие с удалёнными компонентами инфраструктуры. Поэтому разработчику федерата не нужно заботиться о *постоянном* и *своевременном* выделении процессорного времени на нужды RTI, что критично в случае выполнения RTI и федерата в соответствии с однопоточной моделью потоков, которая требует дополнительного планирования вычислений в рамках каждого федерата.

Кроме того, асинхронная модель потоков похожа на модель процессов системы CERTI: федерат отвечает за прямые и обратные вызовы, а процесс RTIA – за корректную работу инфраструктуры RTI. Таким образом, асинхронная модель требует минимального количества изменений в существующем коде, поэтому она была выбрана в качестве основы для реализации прототипа доработанной среды выполнения.

4.4.5 Реализация MT-CERTI

В рамках настоящей работы был создан прототип многопоточной версии системы моделирования CERTI – Multi-Threaded CERTI (MT-CERTI). Внесённые в оригинальный код системы изменения были оформлены в виде патча, и предоставлены разработчикам. В

момент написания настоящего отчёта производится согласование модификаций и обсуждается возможность их внесения в основную ветку разработки.

Изменение поточной модели CERTI потребовало изменений не только в исходном коде системы моделирования, но и её структуре. Результатом проведённых доработок стала переработка библиотеки libRTI. Новая версия библиотеки, как и раньше, способна линковаться с федератом. Но вместо дополнительного процесса, она запускает RTI как дополнительный поток управления внутри его собственного процесса. При этом любое взаимодействие федерата и RTIA реализуется на уровне потоков управления libRTI.

Стоит отметить, что многокомпонентная процессная архитектура оригинальной системы CERTI обладает некоторыми преимуществами по сравнению с разработанной на данном этапе версией с многопоточной архитектурой. Например, при совместном моделировании устройств конкурирующих организаций инфраструктура RTI, к которой подключены модели разрабатываемых устройств, может стать источником получения закрытых данных, и привести к нечестному конкурентному преимуществу [98]. Поэтому изменение поведения CERTI инициируется поднятием дополнительного флага сборки RTIA_USE_THREAD. Ложное или пустое значение флага соответствует многопроцессной версии системы, истинное – многопоточной её версии.

Для работы с потоками управления была использована сторонняя библиотека boost_thread с открытым исходным кодом и свободной лицензией [99]. Данная библиотека написана на языке C++ и фактически, является стандартным решением для кроссплатформенной организации работы с потоками. На момент написания отчёта boost_thread поддерживает более широкий диапазон систем, чем CERTI (Windows, Linux, MacOS, MinGW, и так далее). Поэтому код, написанный с её помощью, не сужает первоначальной области применения данной системы моделирования. Таким образом, дополнительные технические трудности, связанные с использованием boost_thread, сводятся к необходимости её линковки к оригинальным библиотекам системы CERTI.

В традиционной версии системы CERTI обмен данными между процессом RTIA и соответствующим ему федератом происходит через локальный сокет Unix и выполняются в синхронном режиме: процесс RTIA записывает в канал своё сообщение только в ответ на сообщение процесса федерата. На уровне модели потоков управления аналогичная функциональность может быть организована с использованием одной разделяемой переменной. Действительно, каждый из потоков может записывать в неё указатель на своё сообщение и ждать, пока там не появится указатель на сообщение от другого процесса. Однако описанная организация взаимодействия не позволяет реализовать полностью асинхронную модель, в которой процесс федерата и соответствующий ему процесс RTIA

могли бы посылать друг другу сообщения одновременно. Кроме того, при данной организации ни один из участников взаимодействия не может послать другому сразу несколько сообщений, не дожидаясь получения его ответов. Тем более, ни один из компонентов LRC не может работать в несколько потоков.

Чтобы не ограничивать потенциал разработанной библиотеки, взаимодействие её потоков управления было организовано с использованием двух независимых очередей, каждая из которых служит промежуточным буфером для указателей на сообщения одного из потоков (рисунок 37). В перспективе такой подход позволит использовать не только асинхронную, но и многопоточную модель, которая может быть более эффективной в случае должной поддержки со стороны федерата.

Таким образом, каждый поток записывает указатели на свои сообщения в одну очередь и проверяет другую очередь, чтобы получить указатели на сообщения от другого потока. Целостность каждой из очередей защищается отдельным семафором, поэтому в каждый момент времени доступ к очереди может получить лишь один поток. Если во время обращения к очереди одного из потоков, она уже используется, то он блокируется и ожидает освобождения очереди. Стоит заметить, что при чтении данных такой подход может быть менее эффективным, чем неблокирующая проверка. Данный вопрос должен быть исследован дополнительно.

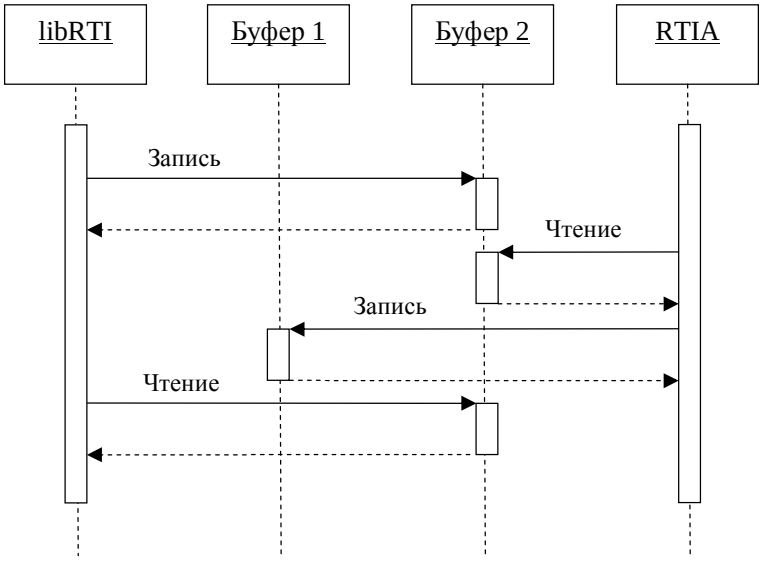

Рисунок 37. Обмен данными через две очереди.

Проблемным местом многопоточной реализации MT-CERTI является вопрос эффективности работы потока RTIA, в обязанности которого входит ожидание сообщений от удалённых компонентов RTI и от других потоков федерата одновременно. Такое поведение может быть организовано с помощью *активного* или *пассивного* ожидания. При активном

ожидании поток находится в вечном цикле, поочерёдно проверяя все возможные источники входящих сообщений, что расходует процессорное время. При пассивном ожидании поток просит операционную систему оповестить его о приходе сообщения и засыпает, пока не получит уведомление. Данный подход имеет более сложную реализацию, но эффективнее при обработке потока редких сообщений.

Так как процесс RTIA получает сообщения достаточно редко, то базовая версия CERTI использует пассивное ожидание: он ждёт сообщения от RTIG с одной стороны и от libRTI с другой на уровне процессов. В новой реализации RTIA получает сообщения от libRTI через отдельную очередь на уровне потоков, в то время как продолжает получать сообщения RTIG на уровне процессов. Операционные системы обычно не предоставляют средств организации пассивного ожидания сразу на двух уровнях взаимодействия. В результате выборка сообщений при текущей логике системы моделирования может быть реализована лишь с использованием активного ожидания. Более эффективное пассивное ожидание требует создания ещё одного потока управления, единственной целью которого станет прослушивание сетевого соединения и отображение получаемых от других процессов сообщений на уровень потоков. Реализация пассивного ожидания может дать существенный прирост производительности, и соответствующее предложение должно быть дополнительно рассмотрено в дальнейшем.

Ещё одним важным аспектом реализации многопоточной версии системы CERTI является эффективность работы с памятью. При использовании независимых процессов все данные необходимо было передавать в виде строки, что неизбежно приводило к копированию памяти. Использование потоков позволило передавать сообщения простой передачей указателя. Такой подход требует использования динамической памяти, выделение и освобождение которой приводит к дополнительным накладным расходам. Однако, эти затраты можно уменьшить с помощью *кольцевого буфера* [100].

Кольцевой буфер создаётся в процессе инициализации инфраструктуры системы моделирования. При этом под него выделяется заданный объём динамической памяти. Во время создания нового сообщения под него отдаётся часть памяти буфера, реального выделения памяти при этом не происходит. Аналогично, во время удаления сообщения часть памяти буфера помечается как свободная, реальное же её освобождение происходит только при удалении буфера.

Неэффективность использования памяти так же проявляется в разных форматах сообщений между библиотекой libRTI и процессом RTIA, и между процессом RTIA и процессом RTIG. Если бы эти сообщения имели единый общий формат, то процесс RTIA мог бы передавать полученные от федерата сообщения напрямую процессу RTIG. При этом не

требовалась бы лишнее перекодирование формата данных. Соответствующее исправление кода может привести к существенному приросту производительности, что делает перспективной дальнейшую работу в данном направлении.

4.4.6 Доработка интерфейсов синхронизации

Корректная сериализация данных является ключевой проблемой при обеспечении взаимодействия между отдельными федератами. Например, список трудностей, связанных с сериализацией данных модели, включает в себя [101]:

1. Ошибки программирования;
2. Ошибочная интерпретация спецификаций сериализации;
3. Зависимость сериализации от среды (встроенная в язык система типов данных, низкоуровневые опции компиляции и сборки программы, используемая архитектура процессора и так далее).

Ошибки сериализации могут приводить к «падениям» и «зависаниям» распределённой модели, противоречивым или вводящим в заблуждение результатам экспериментов, потерям данных. Стандарт HLA версии IEEE 1516-2000 включает в себя шаблон объектной модели OMT (Object Model Template), представляющий собой набор правил кодирования для всех типов данных, использование которых допускает этот стандарт [83]. Правила должны были прояснить, как правильно кодировать и декодировать модельные данные, и привести к уменьшению числа ошибок, связанных с некорректной реализацией обмена пользовательскими данными через RTI. Однако на практике данные меры оказались неэффективны. Разработчики имитационной модели вынуждены были самостоятельно реализовывать преобразования своих типов данных, и регулярно допускали ошибки.

Разработчики следующей версии стандарта HLA, HLA 1516-2010 (Evolved), пошли дальше и включили в его спецификации дополнительные программный интерфейс (API) кодировки, называемый «Encoding Helpers» [101]. Его использование упрощает процесс построения корректных функций сериализации для заданных типов данных. Однако, по мнению многих разработчиков, предложенное средство излишне обобщено и, как следствие, неудобно для практического применения [102]. Оно оторвано от системы типов данных языка программирования, используемого для создания модели, и вынуждает разработчиков моделей конструировать их самостоятельно. Таким образом, предложенный интерфейс не решает проблемы кодировки до конца.

Поэтому было предпринято несколько попыток построения более удобных интерфейсов кодирования для частных случаев использования HLA. Например, библиотека времени компиляции с открытым исходным кодом Proto-X реализует кодирование данных с

использованием встроенных типов языка C++ [102]. Данная библиотека предоставляет метаязык для кодирования данных DTEL (Data Type Encoding Language), операнды которого выражаются с помощью шаблонов C++. При компиляции выражения DTEL автоматически генерируют функции перекодирования типов данных встроенных в язык C++ в структуры, определённые стандартом HLA и обратно. Такой подход позволяет обнаруживать ошибки сериализации данных ещё в процессе написания программы, тем самым, снижая трудовые и финансовые затраты на отладку модели.

В ходе работы по интеграции среды выполнения имитационных моделей и системы автоматической генерации исходного кода её компонентов была разработана программная прослойка, призванная облегчить построение федератов-моделей компонентов PBC PB. В частности, прослойка использует библиотеку Proto-X.

Во время генерации модели из UML диаграммы автоматически определяется набор классов-обёрток для модельных данных, описанных на метаязыке шаблонов C++. Обёртки содержат встроенные методы для конвертирования встроенных структур языка и их производных в формат, определённый спецификациями стандарта HLA. При этом автоматически решается множество проблем конвертирования: порядок байт в представлении целых чисел (endianity), выравнивание данных по машинным словам (alignment), вложенность типов данных друг в друга (nesting) и так далее.

Для получения и модификации значений данных классы-обёртки предоставляют соответствующие методы сеттеры и геттеры, которые работают на уровне встроенных типов языка. Таким образом, обёртки вполне соответствуют роли основного хранилища данных и не требуют поддержания своей дополнительной копии с более удобным интерфейсом. При этом, однако, существует проблема дополнительного копирования данных при их чтении. Впрочем, авторы планируют избежать дополнительного копирования в последующих версиях библиотеки.

Публикация, подписка и передача взаимодействий также осуществляется через встроенные методы класса-обёртки, и их использование не вызывает трудностей. Однако, текущая внутренняя реализаций данных методов жёстко привязана к интерфейсам стандарта версии HLA DMSO 1.3, несовместимым с более новой версией IEEE 1516-2000, которая используется средой выполнения и генератором исходного кода моделей. В результате в исходный код Proto-X пришлось внести ряд правок: заменить методы RTI, которыми пользуется библиотека, и внутренние структуры данных HLA, передаваемые в качестве параметров при вызове данных методов.

Концепция автоматического построения функций кодировки данных, привязанных к системе типов языка, показала свою практическую эффективность. Она исключает ошибки,

связанные с написанием сложного, громоздкого, низкоуровнего кода. Классы-обёртки предоставляют удобный способ хранения данных и высокоуровневый интерфейс для проведения манипуляций с ними, упрощая как процесс ручной разработки, так и механизмы автоматической генерации кода.

На данный момент библиотека Proto-X применима лишь для конвертирования данных в типы HLA. Однако нетрудно видеть, что концепция легко расширяема и на другие форматы. Перспективным направлением исследования представляется разработка аналогичного средства для конвертирования данных в форматы натурных каналов передачи данных, используемых в РВС РВ, и автоматизированной их обработки. Примерами таких каналов являются ARINC 429, MIL STD – 1553B, Fibre Channel.

4.4.7 Каскадная архитектура CERTI

Выраженная централизованная архитектура CERTI приводит к чрезмерной загрузке её компонента CRC во время выполнения сложных имитационных моделей, что является серьёзным архитектурным недостатком. Существует множество реализаций RTI, которые совмещают централизованную и децентрализованную архитектуру более равномерно, что позволяет им достичь лучших показателей производительности [103]. Однако смешение этих подходов в том же смысле приведёт к коренной переделки CERTI и потере всех преимуществ, которые даёт её модульная архитектура сегодня. Настоящий раздел рассматривает альтернативный способ уменьшения нагрузки CRC – *каскадную архитектуру* инфраструктуры RTI.

Несмотря на то, что каждый федерат соответствует конкретному компоненту моделируемой системы, уровень их абстракции может существенно различаться и не отражать её логической структуры. Например, модель бортовой системы может состоять из одного федерата, соответствующего множеству второстепенных систем, и множество федератов, моделирующих поведение наиболее важной подсистемы. При этом последние имеют меньший уровень абстракции, так как соответствуют более мелким компонентам системы. Лишь совокупность этих федератов формирует логическую подсистему того же уровня абстракции, что и остальные участники моделирования. Поэтому агрегированные в эти совокупности федераты зависят друг от друга, они логически связаны между собой.

Во время выполнения имитационной модели, группа агрегированных федератов выделяется на фоне остальных участников моделирования высокой интенсивностью взаимодействий. Практика проведения экспериментов показывает, что агрегированные федераты взаимодействуют друг с другом гораздо чаще, чем с внешними по отношению к этой группе федератами. Таким образом, происходит естественная кластеризация всех

федератов модели на множество *агрегированных групп*, внутри которых происходит большая часть обменов данными.

Ввиду того, что система CERTI имеет ярко выраженную централизованную архитектуру, любые взаимодействия федератов происходят с помощью её компонента CRC, вне зависимости от логической структуры модели. В результате, если число обменов данными между участниками моделирования *достаточно* велико, центральный компонент CRC перегружается и становится узким местом системы, замедляя скорость моделирования.

В тоже время, выполнение конкретного федерата на самом деле не зависит от обмена данными внутри агрегированной группы, к которой он не принадлежит. Поэтому трафик может быть эффективно разделён в соответствии с логической структурой исследуемой системы с помощью набора вспомогательных компонентов CRC, каждый из которых соответствует одной из агрегированных групп. С одной стороны, эти CRC будут самостоятельно контролировать сопоставленную им группу федератов, выполняя при этом часть функций основного CRC и, тем самым, снимая с него часть нагрузки. С другой стороны, с точки зрения основного CRC, вспомогательные CRC будут выглядеть как обычные федераты, которые выполняются согласно правилам HLA, но генерируют поток трафика, соответствующий целой группе федератов. Описанные вспомогательные CRC могут быть реализованы как новый интерфейс к локальному компоненту LRC, что не повлечёт за собой существенного увеличения сложности CERTI.

Рассмотрим несколько естественных расширений описанной идеи. Во-первых, на практике федераты могут объединяться в группы сразу по нескольким признакам, не связанным с логической структурой моделируемой системы: хорошим критерием может служить, например, неравномерное распределение интенсивности обменов данными. Во-вторых, один и тот же приём можно использовать несколько раз. Агрегированная группа может быть, в свою очередь, разбита на несколько подгрупп, тем самым сформировав новый *каскад* управления моделью. Именно поэтому описанная архитектура описывается в настоящей работе как *каскадная* (рисунок 38).

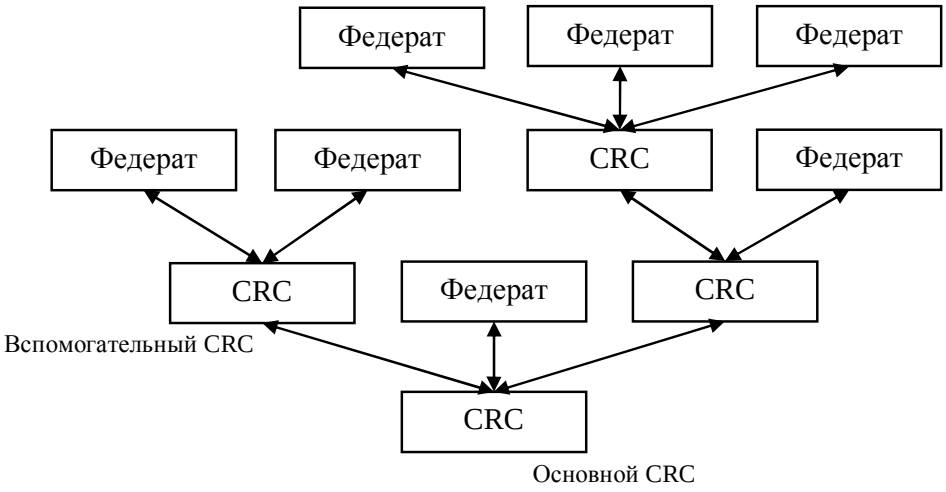

Рисунок 38. Каскадная архитектура

Построение системы моделирования на основе каскадной архитектуры на практике даёт сразу несколько преимуществ. Прежде всего, данный подход позволяет решить проблему чрезмерной загрузки компонента CRC и повышает масштабируемость системы. В самом деле, вспомогательные CRC самостоятельно обрабатывают внутренние обмены данных между федератами соответствующей им группы и, тем самым, снимают часть обязанностей с основного CRC. Каждый вспомогательный CRC может выполняться на отдельном вычислительном узле, при этом загрузка автоматически перераспределяется по множеству доступных ресурсов и не требует дублирования данных или использования сложных алгоритмов согласования состояния распределённой системы.

Во-вторых, агрегация позволяет уменьшить потери на синхронизацию систем. Внутренние взаимодействия агрегированной группы происходят через вспомогательный компонент CRC, который не учитывает множество зависимостей от других участников моделирования, и поэтому может выполняться более эффективно, чем основной CRC. В то же время, внешние взаимодействия агрегированных федератов реализуются описанной схемой с использованием и вспомогательного, и основного компонента CRC, и поэтому менее эффективны. Однако если группа федератов выбрана *удачно*, синхронизационные потери уменьшаются соответствующим образом.

В-третьих, агрегация позволяет провести точную настройку RTI под конкретную имитационную модель. Например, федераты могут быть кластеризованы в соответствии с набором сервисов RTI, которые они используют. При этом неиспользуемые сервисы RTI могут быть безболезненно удалены из вспомогательных CRC, тем самым уменьшая их сложность и увеличивая эффективность. Далее, оставшиеся сервисы CRC, в свою очередь, могут быть подстроены под требования подключённых к ним федератов. Например, каждый

из вспомогательных компонентов CRC может использовать свой собственный алгоритм продвижения модельного времени. В случае полунатурного моделирования основной CRC всегда должен использовать только консервативную схему продвижения времени, однако вспомогательный CRC, к которому не подключено оборудование, может использовать внутри себя и обычно более эффективную оптимистическую схему.

Наконец, агрегация даёт возможность увеличения эффективности взаимодействия федератов, выполняющихся на единственном узле. Централизованная архитектура CERTI не учитывает относительное расположение федератов. Даже в случае, если они запущены на одном вычислительном узле, каждый обмен данными происходит через процесс RTIG. При этом RTI использует два сетевых сообщения, чтобы передать данные между двумя процессами, выполняющимися на одном узле, что приводит к значительному падению производительности системы. В то же время агрегация всех федератов, выполняющихся на одном узле, позволяет реализовать внутренний обмен данными между ними без использования сетевых взаимодействий. Таким образом, концепция теоретически позволяет достичь эффективности децентрализованных peer-to-peer систем без внесения каких-либо изменений в логику работы CERTI.

Слабой стороной рассмотренной каскадной архитектуры является индетерменизм структуры RTI и её зависимость от конкретной имитационной модели. Построение каскадной инфраструктуры, соответствующей эффективному разделению федератов на группы, требует разработки автоматических статических и динамических анализаторов выполняемой модели. Тем не менее, полученный прирост производительности системы моделирования, по всей видимости, стоит затраченных усилий, и каскадная архитектура выглядит достаточно перспективно.

4.4.8 Выводы

В данном разделе было приведено обоснование выбора базовой реализации среды выполнения CERTI, на основе которой строилась среда выполнения для решения задачи моделирования РВС РВ. Было произведён анализ ограничений текущей версии данной системы и описан прототип её модификации, устраняющий обозначенные ограничения. Была затронута проблема кодирования модельных данных, связанная с несоответствием системы типов, встроенных в язык программирования, и структур данных, описанных спецификациями стандарта HLA. Описана библиотека времени компиляции Proto-X, позволившая эффективно решить затронутую проблему, и метод её интеграции со средой выполнения и, менее подробно, подсистемой генерации кода моделей.

Кроме того, в разделе было обозначено несколько перспективных направлений дальнейших научных исследований и соответствующих им технических задач:

1. Оптимизация системы CERTI. Асинхронная модель потоков может уступать в производительности более сложной многопоточной модели. Используемое внутри процесса RTIA активное ожидание вероятно менее эффективно, чем пассивное ожидание. Для передачи данных между процессом RTIA и процессом федерата нужно попробовать использовать неблокирующий кольцевой буфер. Система передачи сообщений между компонентами CERTI предполагает изменение их формата внутри RTIA, что снижает производительность системы;
2. Задачи подстройки существующего интерфейса RTI под задачу моделирования РВС РВ. Необходимо изучить возможности для более плотной интеграции работы с натурными каналами передачи данных. Перспективной выглядит разработка средства, аналогичного Proto-X для кодирования встроенных типов языка программирования в структуры, соответствующие спецификация этих каналов.
3. Задачи оптимизации среды выполнения под конкретную задачу моделирования. Перспективным представляется практическое исследование предложенной каскадной архитектуры, которая позволяет увеличить производительность среды выполнения за счёт перераспределения нагрузки между логическими компонентами системы.

4.5 Средство внесения неисправностей

4.5.1 Метод внесения неисправностей

Одним из методов тестирования РВС РВ является внесение неисправностей. Главной идеей метода внесения неисправностей является внесение неисправностей в компоненты системы с целью анализа, каким образом влияет та или иная неисправность на систему, приводит ли она к ошибке или нет, а также способна ли система обработать эту ошибку. [104]. Таким образом, оценивается способность системы обнаруживать и устранять те или иные ошибки[105].

В качестве метода внесения неисправностей, реализованного в данном средстве был выбран подход предполагающий добавление нескольких федератов, выступающих в роли перехватчиков сообщений. Добавление нескольких перехватчиков служит для того, чтобы распределять нагрузку между перехватчиками. Главным преимуществом этого метода является отсутствие необходимости изменения среды выполнения. Также придется вносить незначительные изменения в исходных участниках моделирования. Федераты-перехватчики

выступают в роли посредников, через которых проходят все сообщения, которые могут либо пересылаться получателю, либо предварительно редактироваться (Рисунок 39).

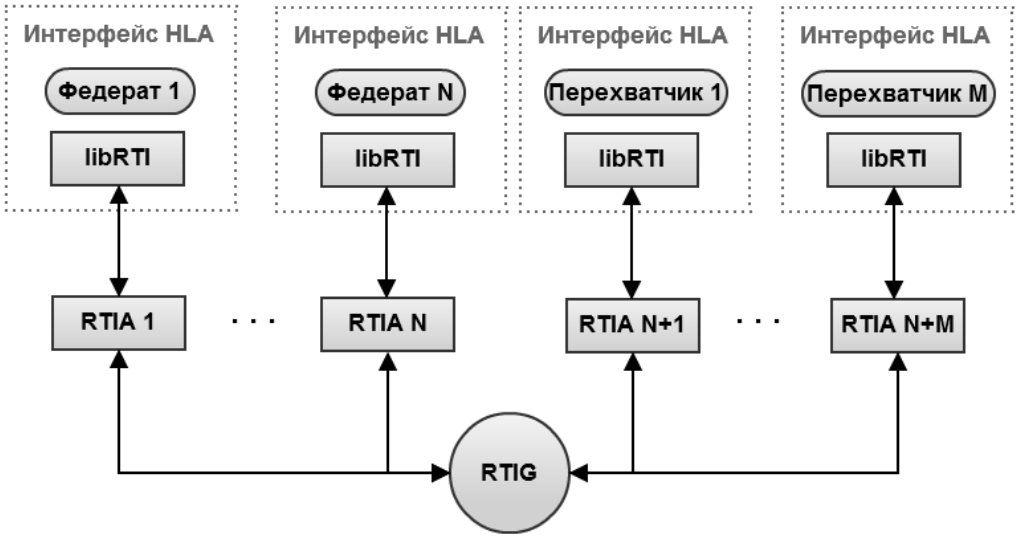

Рисунок 39 - Внесение неисправностей на уровне федератов

Используя данный метод, необходимо вносить незначительные изменения в каждый федерат и создавать группу федерат-перехватчиков.

4.5.2 Схема работы метода

Главной задачей данного метода является создание федерата-перехватчика, играющего роль посредника в передачи сообщений. Для начала необходимо создать шаблон федерата, который в зависимости от тестируемой модели будет заполняться необходимой информацией о других федератах.

Было принято решение о реализации упрощенной схемы ВН, то есть при M = 1(1 федерат-перехватчик). В этом случае сообщения от всех участников моделирования поступают сначала к федерату-перехватчику, а затем отправляются по назначению.

Чтобы федерат-перехватчик имел возможность редактировать все сообщения, необходимо чтобы все сообщения изначально направлялись ему. Рассмотрим пример: пусть Федерат 1 хочет передать сообщение типа T1. Федерат 2 может принимать сообщения типа T1, значит исходное сообщение предназначается для Федерата 2 (Рисунок 40).

Для того, чтобы Перехватчик мог иметь возможность принимать сообщение T1, он подписывается на этот тип сообщений. Чтобы исходное сообщение приходило к Федерату 2 только после обработки его в Перехватчике, необходимо поменять в Федерате 2 тип принимаемых сообщений на T2. Получается что исходное сообщение имеет тип T1 и

предназначается для Перехватчика. Далее перехватчик обрабатывает сообщения(вносит необходимые изменения), меняет его тип на T2 и отправляет Федерату 2.

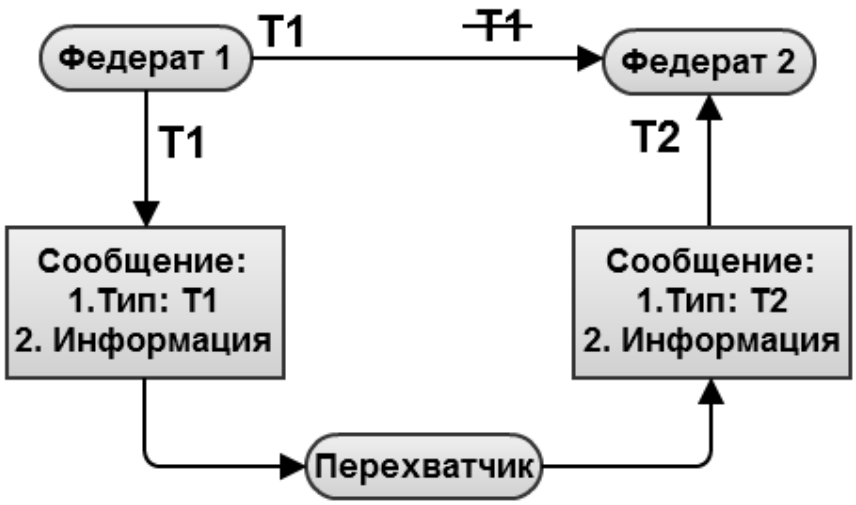

Рисунок 40 - Пересылка сообщений

Этот алгоритм применим к моделям любого объема, однако, после тестирования, было принято решение о его модификации. Предположим, что идет тестирование взаимодействия двух федератов, состоящих в федерации с большим числом федератов. Выполнив предложенный алгоритм, федерат-перехватчик подпишется на сообщения всех типов, а значит, любое сообщение сначала придет к перехватчику и только затем к получателю. Таким образом, число передаваемых сообщений удвоится, что сильно повлияет на скорость работы CERTI, использующей централизованную архитектуру. Поэтому было принято решение о том, что пользователь заранее указывает типы сообщений, на которые должен подписаться федерат.

Таким образом, для реализации схемы с федератом-перехватчиком необходимо не только сгенерировать сам Перехватчик, но и произвести замену типов сообщений таким образом, чтобы все сообщения сначала шли Перехватчику и затем шли получателям.

4.5.3 Архитектура средства

Как было сказано выше, все сообщения, передаваемые в исполняемой модели, сначала приходят федерату-перехватчику. Для удобства пользователей был сделан графический интерфейс перехватчика с использованием кроссплатформенной библиотеки wxWidgets[106]. Процесс внесения неисправностей может проходить как в автоматическом режиме, так и с возможностью перехода в ручной режим.

В ручном режиме внесения неисправностей пользователю показывается каждое пришедшее сообщение и предлагается внести свои корректировки. В этом режиме предполагается что пользователь будет обрабатывать каждое сообщение, что не всегда удобно и нужно. В этом случае невозможно тестирование моделей в режиме реального времени. Для таких случаев был создан автоматический режим внесения неисправностей.

В автоматическом режиме внесения неисправностей пользователю предлагается загрузить XML файл со сценарием неисправностей. XML-сценарий содержит набор типов сообщений и описания действий для их модификации. Средство ВН позволяет менять режим внесения неисправностей во время выполнения (Рисунок 41).

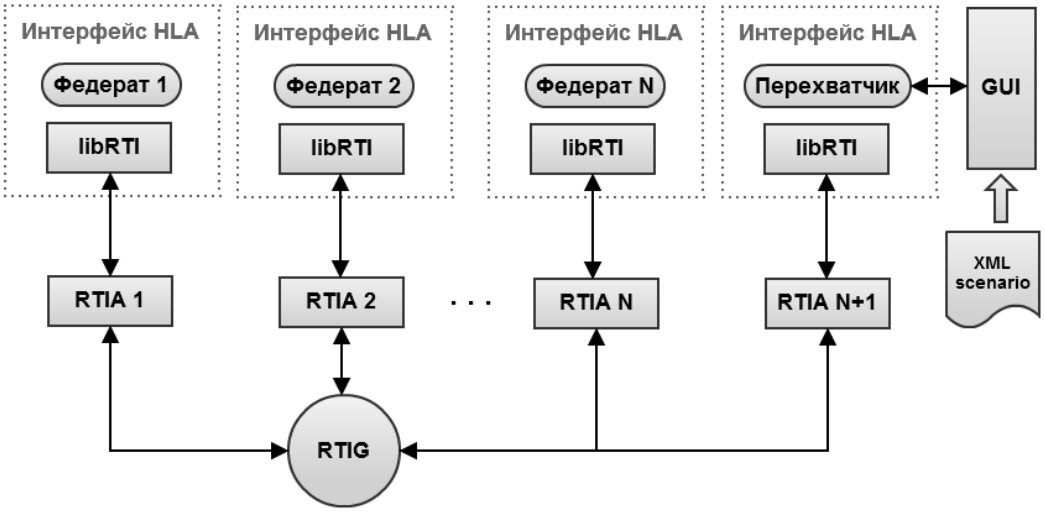

Рисунок 41 - Архитектура средства

Все сообщения, полученные перехватчиком, передаются средству обработки сообщений, которое имеет графический интерфейс, написанный средствами wxWidgets. Перед тестированием модели следует запустить сначала графическую составляющую и загрузить в нее сценарий ВН. Для работы со сценариями используется класс wxXMLDocument, являющийся частью wxWidgets (Рисунок 42).

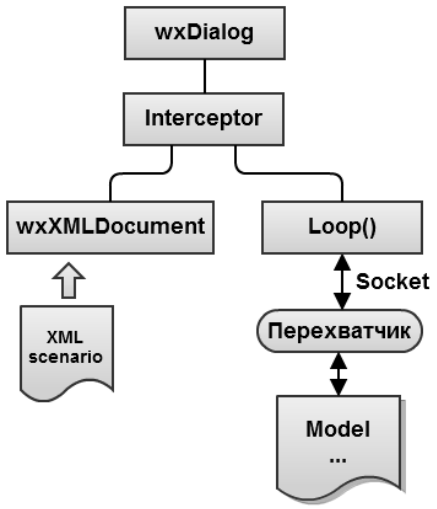

Рисунок 42 - Структура средства

После запуска тестируемой модели, средство получает, обрабатывает согласно сценарию и передает измененные данные федерату-перехватчику (Рисунок 43).

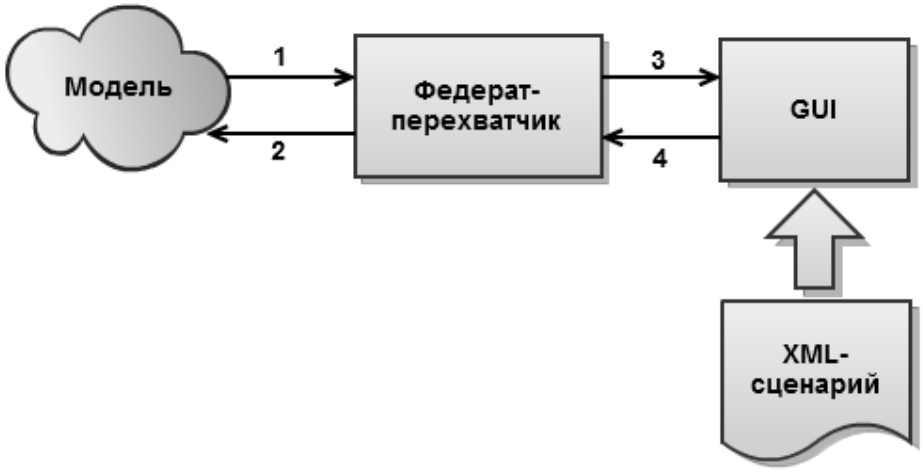

Рисунок 43 – Схема работы средства

4.5.4 Выводы

В данном разделе описан метод внесения неисправностей и описывается архитектура средства, реализующего данный метод. Такое средство позволит вносить неисправности в компоненты системы с целью анализа, каким образом влияет та или иная неисправность на систему.

4.6 Средство трассировки моделей

Средства регистрации и трассировки событий моделирования предназначены для фиксации изменений состояния и параметров моделей компонентов РВС РВ и обменов сообщениями между ними в трассе имитационного эксперимента. Для решения задачи формирования трассы необходимо:

- определить перечень компонентов модели РВС РВ, которые будут трассироваться;
- определить перечень событий для регистрации в трассе;
- определить перечень параметров и состояний компонентов, которые необходимо регистрировать;
- непосредственно сформировать трассу заданного формата на основе последовательности событий.

При разработке средства визуализации важное значение имеют первые три задачи, непосредственно определяющие объекты для визуализации и анализа.

В данной работе РВС РВ рассматривается как набор взаимодействующих иерархически организованных *компонентов*. В качестве компонентов могут быть: частные модели (ЧМ), распределённые частные модели (РЧМ), интерфейсы частных моделей, каналы бортовых интерфейсов (БИ), «искусственные» компоненты, введённые для повышения наглядности отображения.

Событием называется любое изменение, существенное с точки зрения логики функционирования моделируемой РВС РВ, которое отражается в трассе результатов эксперимента. Каждое событие характеризуется типом, временем возникновения, местом возникновения и набором дополнительных атрибутов в зависимости от типа события.

Состояние компонента обозначает неизменный режим работы компонента в течение некоторого промежутка времени и характеризуется типом, временем начала и продолжительностью.

В рамках НИР качестве основной технологии для распределенного имитационного моделирования в реальном времени была выбрана технология HLA RTI, в качестве реализации HLA RTI, принятой за основу разрабатываемой среды моделирования РВС РВ – среда распределенного моделирования CERTI.

Для выбор схемы трассировки моделей и её реализации в среде распределенного моделирования РВС РВ на основе CERTI необходимо:

- Рассмотреть особенности архитектуры CERTI.
- Рассмотреть существующие общие подходы к трассировке.
- Сформулировать требования к схеме трассировки.

- Рассмотреть возможные реализации схем трассировки применительно к средам моделирования на основе HLA RTI.
- Выбрать схему трассировки и реализовать её в виде программного средства.

4.6.1 Особенности архитектуры CERTI RTI с точки зрения трассировки моделей

Архитектура CERTI RTI представлена на рисунке 44.

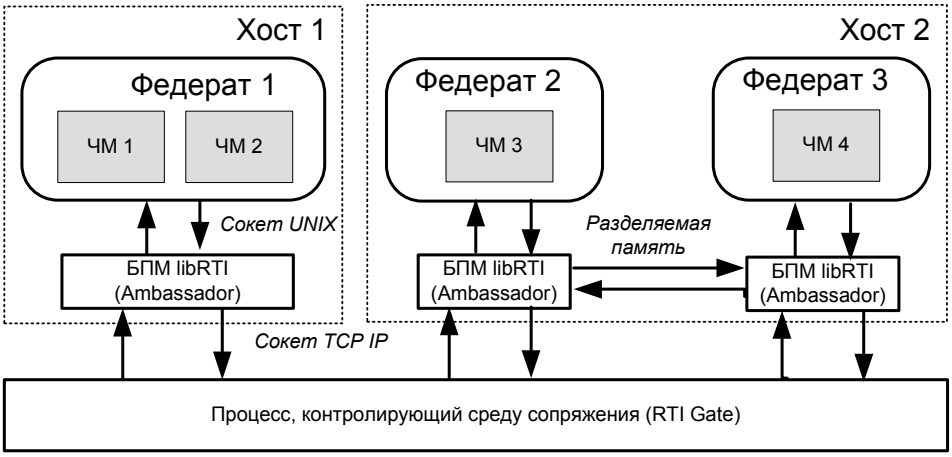

Рисунок 44. Архитектура среды моделирования на основе CERTI RTI

RTI обеспечивает набор общих и независимых сервисов, успешно отделяет реализацию модели, управление выполнением модели и управление взаимодействием моделей. Шесть типов сервисов управления, как показано на рисунке 45, обеспечивают связь и координацию сервисов в федератах. 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 подключенными федератами. Таким образом, в CERTI используется централизованная архитектура, и часто синхронизация распределённой имитационной модели сводится к последовательному выполнению запросов федератов на единственном вычислительном узле.

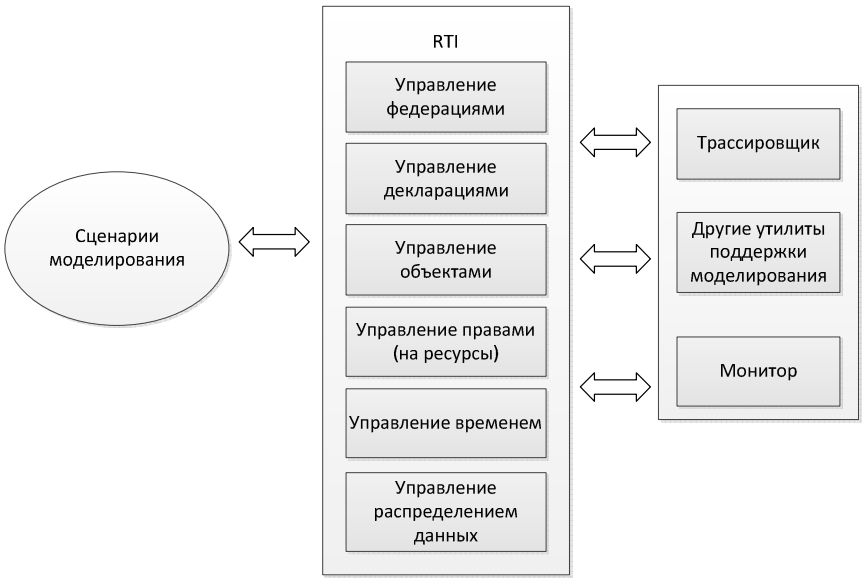

Рисунок45. Функциональное представление моделирования на основе HLA RTI

В терминах HLA отдельная модель компонента рассматривается как «федерат». Группа федератов, которые намереваются взаимодействовать друг с другом, образуют систему моделирования, называемую «федерацией». В общем HLA определяется тремя компонентами: спецификацией интерфейса, набором правил и шаблоном объектной модели (OMT).

HLA интерфейс определяет правила взаимодействия федератов через набор общих интерфейсов. Спецификация интерфейса устанавливает интерфейс между федератами и RTI, который имеет имя, данное в программной реализации спецификации интерфейса.

HLA правила определяют набор соответствующих правил для обеспечения надлежащего взаимодействия федератов во время выполнения федерации, которая управляет поведением всей федерации и федератов.

HLA OMT обеспечивает стандартный фреймворк для записи информации, включенной в требуемую HLA модель объекта для каждой федерации и федератов. Основные компоненты OMT - объектная модель федерации (FOM) и имитационная объектная модель (SOM). FOM – это спецификация на уровне федераций, описывающая всю информацию (объекты, атрибуты, связи и взаимодействия), которая может быть разделяемой федератами, принимающими участие в отдельных имитационных экспериментах. SOM – это спецификация на уровне федератов, описывающая объекты, атрибуты и взаимодействия в отдельных федератах, которые доступны для других федератов.

Нужно отметить, что одним из преимуществ CERTI является более эффективное взаимодействие между моделями посредством разделяемой памяти в том случае, если

федераты, которым принадлежат модели, располагаются на одном хосте. К достоинствам архитектуры CERTI можно отнести её простоту.

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.

4.6.2 Общие подходы к сбору трасс

Существующие механизмы сбора данных (в виде трассы) о выполнении имитационных моделей компонентов РВС РВ можно разделить на две категории: централизованный сбор и распределенный сбор.

Централизованный сбор данных - это централизованный подход, который предполагает единую точку сбора данных имитационного эксперимента. Этот подход требует, чтобы сборщик (коллектор) захватывал все необходимые данные в одной точке сети. Главными достоинствами централизованного подхода являются присущая простота и отсутствие необходимости сортировки данных по временной метке. Также достоинством является гибкость и простота управления процессом трассировки из единой точки сбора данных. Главным недостатком является скопление большого объема трафика в одной точке сети, что может быть послужить причиной заторов трафика в сети и, таким образом, сбор данных может быть узким местом всей системы моделирования, особенно в масштабах HLA-систем моделирования на основе WAN. В результате использования централизованного подхода по окончании процесса моделирования получается одна трасса.

Распределенный сбор данных предполагает наличие нескольких точек сбора данных. Каждый сборщик (коллектор) отвечает за сбор части данных имитационного эксперимента. Главным достоинством распределенного сбора является предотвращение в одной точке сети чрезмерного трафика, связанного с трассировкой имитационного эксперимента. Основной недостаток подхода заключается в том, что необходима дополнительная обработка данных с целью их сортировки при анализе всего эксперимента. Если требуется формирование единой базы данных или трассы всего эксперимента, то все распределенные данные должны быть перемещены по сети в одно место и упорядочены по временной метке, поэтому особое значение уделяется времени выполнения этой операции, чтобы избежать перегрузки сети и ее компонентов. В результате использования распределенного подхода по окончании процесса моделирования получается множество трасс, находящихся в разных точках сети.

Описанные общие подходы к процессу трассировки могут иметь различную реализацию. Ниже будут рассмотрены следующие схемы трассировки:

- Централизованные схемы:
  - Схема на основе прослушивания сетевого трафика.
  - Схема «Федерат-логгер».
- Распределенные схемы:
  - Сбор трасс для отдельных федератов
    - Схема на основе встраивания функций трассировки в федерат.
    - Схема «RTI-интерфейс логгер».
  - Сбор трасс для отдельных хостов
    - Схема на основе общей памяти в рамках хоста.
  - Сбор трасс федератов с разных хостов
    - Схема трассировки на основе федератов-логгеров
    - Схема трассировки на основе многоагентной системы.

4.6.3 Требования к схеме трассировки

Критерии выбора схемы трассировки:

- Минимальное влияние на производительность разрабатываемой среды моделирования.
- Гибкость управления процессом трассировки. До начала имитационного эксперимента пользователю должны быть доступны возможности настройки перечня трассируемых компонентов РВС РВ, перечня регистрируемых событий и их параметров, фильтрации данных.
- Возможность реализации подхода в CERTI.

4.6.4 Централизованная схема трассировки на основе прослушивания сетевого трафика

Схема трассировки на основе прослушивания сетевого трафика была предложена в [107] и основана на использовании библиотеки поддержки моделирования, удовлетворяющей требованиям HLA RTI. Она заключается в том, что дополнительный процесс прослушивает сетевой интерфейс, по которому осуществляется передача пакетов HLA RTI. Пакеты разбираются, на их основании формируется трасса (рис. 46). Хотя такой способ и представляется универсальным, он требует настройки на конкретную библиотеку, так как реализация механизма в рамках HLA не определена. Другим недостатком такого подхода оказывается неспособность перехватывать события в том случае, когда объекты

работают в рамках одной федерации и библиотека использует общую память для ускорения обмена.

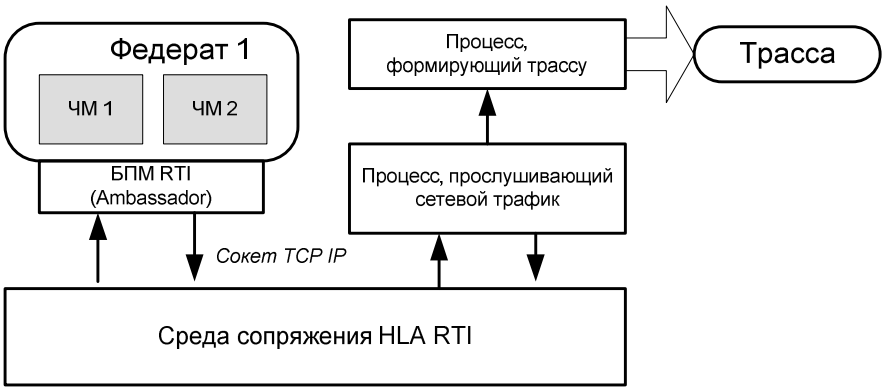

Рисунок 46. Схема формирования трассы на основе прослушивания сетевого канала

4.6.5 Централизованная схема трассировки «Федерат-логгер»

Схема трассировки «Федерат-логгер» заключается в создании отдельного федерата (федерат 2 на рисунке 47), занимающегося исключительно сбором информации о событиях в среде моделирования и их записью в трассу. Данная схема является централизованной, поэтому ей присущи достоинства и недостатки подобного типа схем.

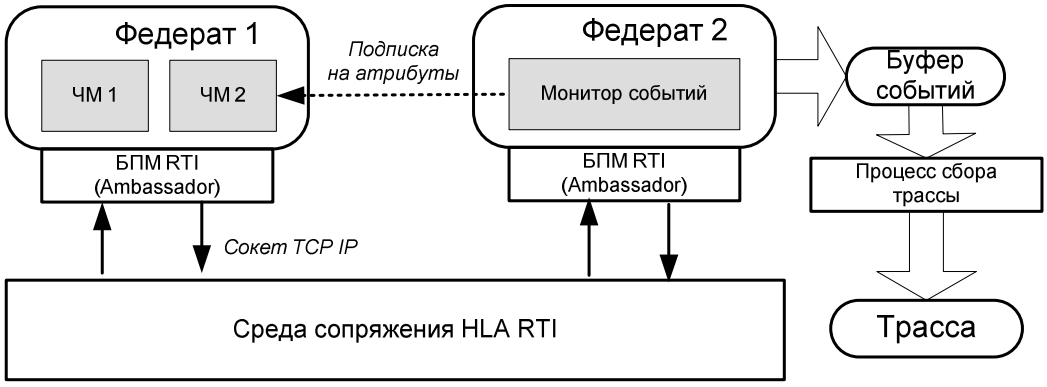

Рисунок 47. Схема сбора трассы с помощью федерата-логгера

Если в федерации только один федерата-логгера, то сетевой трафик может резко возрасти, потому что ему придется подписаться на все данные из других федератов. Другой недостаток федерата-логгера связан с временем прибытия данных в него: нет показаний времени, когда данные были получены федератом, подписавшимся на эти данные. Таким образом, становится трудным вычисление задержек. Для того, чтобы временные метки были адекватными, логгер-федерат должен располагаться в каждом удаленном узле и время прибытия данных в такой узел должно записываться локальным логгером-федератом. Подобная схема реализована в MAK HLA.

4.6.6 Распределённая схема трассировки на основе встраивания функций сбора трассы в ПО федерата

В данной распределенной схеме трассировки отдельная трасса собирается для каждого федерата. Для реализации данной схемы в каждый федерат необходимо добавить дополнительный программный код, осуществляющий сбор и формирование трассы по мере поступления событий. Основной недостаток подхода заключается в том, что код, который должен быть добавлен в федерат, будет специфичным для федерата или федерации и не может повторно использоваться в других федератах или федерациях.

4.6.7 Распределенные схемы трассировки на основе RTI-Interface logger

Взаимодействие федерата со средой сопряжения HLA RTI в CERTI осуществляется посредством БПМ libRTI (Ambassador). Таким образом, этот RTI-интерфейс собирает все данные, которые пересылаются от федерата к RTI и все данные от RTI к федерату [108]. Следовательно, функции трассировки можно внедрить в БПМ libRTI. Таким образом, в данной схеме трасса собирается для каждого отдельного федерата, процесс трассировки осуществляется в точках взаимодействия федератов и RTI.

4.6.8 Распределённая схема трассировки на основе общей памяти

В Стенде ПНМ реализуется данная распределенная схема трассировки на основе общей памяти и на каждом хосте собирается своя отдельная трасса. На каждом хосте события заносятся основным процессом моделирования в область общей памяти. Специальный вспомогательный процесс считывает из общей памяти события, передаёт их средству оперативного управления и периодически сохраняет события в трассу. Общая схема трассировки на основе общей памяти изображена на рисунке 48.

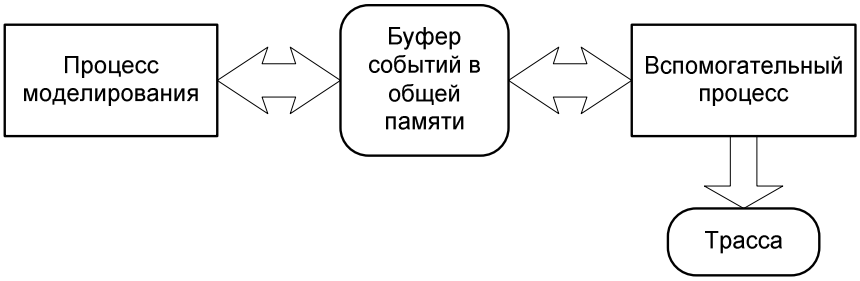

Рисунок 48 - Схема трассировки на основе общей памяти, реализованная в Стенде ПНМ

Поскольку CERTI поддерживает эффективное взаимодействие федератов, расположенных на одном хосте, через разделяемую память, то такая схема может использоваться на каждом хосте среды моделирования.

4.6.9 Распределённая схема трассировки на основе федератов-логгеров

Данная схема является развитием централизованной схемы «федерат-логгер», описанной выше и заключается в том, что для решения проблемы временных меток «федераты-логгеры» устанавливаются в удаленных узлах, осуществляющих моделирование, которые могут включать в себя несколько хостов.

4.6.10 Схема трассировки на основе многоагентной системы

С целью повышения надежности и производительности среды моделирования на основе HLA RTI в работе [109] предлагается распределенная схема трассировки на основе многоагентной системы сбора данных.

Многоагентная система состоит из множества взаимодействующих агентов-логгеров. Каждый логгер включает в себя:

- Коммуникационный модуль, отвечающий за связь с моделями и другими агентами-логгерами.
- Совместный модуль принятия решений.
- Модуль обработки данных, получаемых от моделей.
- Модуль записи данных в трассу

Каждый логгер загружает данные в соответствующий сервер баз данных.

Логгеры каким-то образом располагаются на хостах сети. Каждый логгер может трассировать одну или несколько моделей, расположенных на разных хостах. Причем модели для трассировки могут определяться по некоторому эвристическому алгоритму в зависимости от количества событий, выдаваемых каждой моделью. Набор трассируемых моделей для каждого логгера в процессе работы может динамически изменяться в зависимости от загрузки сети и количества выдаваемых моделями событий. К недостаткам данной схемы можно отнести сложность её реализации.

4.6.11 Выводы

В ходе анализа различных схем трассировки были рассмотрены их особенности реализации, достоинства и недостатки. Нужно отметить, что централизованные схемы больше подходят для ситуаций, когда модели не генерируют большое количество событий, и как следствие сетевого трафика, связанного с трассировкой, и когда моделирование ведется на не очень удаленных узлах сети. При этом собирается единая трасса с упорядоченными по временным меткам событиями. Наиболее предпочтительными для реализации в распределенной среде моделирования РВС РВ являются распределенные схемы трассировки.

В рамках НИР была выбрана и реализована централизованная схема «Федерат-логгер» с сбором трассы в отдельном федерате-трассировщике. Нужно отметить, что перспективными является подходы на основе многоагентной системы трассировки, на основе общей памяти и множества федератов-логгеров, однако данные схемы несколько сложнее с точки зрения реализации.

4.7 Средство анализа и визуализации трассы

В ходе проведённых экспериментальных исследований было установлено, что для задачи анализа поведения имитационных моделей РВС РВ наиболее приемлемым является открытый формат OTF (Open Trace Format) с возможностью сжатия трассы [110]. Исследование существующих средств анализа и визуализации трасс показало наличие лишь двух средств, поддерживающих данный формат:

- Vampir 7.3 - проприетарное средство с достаточно широкими возможностями анализа трасс в формате OTF.
- ViTE 1.2 (Visual Trace Explorer) – развивающееся средство с открытыми исходными кодами, поддерживающее три формата трасс (TAU, OTF и Paje), но имеющее значительное количество недостатков с точки зрения его функциональных возможностей для работы с трассой и её визуализацией.

Таким образом, в настоящее время не существует хорошего, с точки зрения функциональности, открытого программного средства для анализа и визуализации трасс в формате OTF. Поэтому в рамках НИР актуальной стала задача разработки такого программного средства.

Для решения поставленной задачи было предложено взять за основу одно из существующих открытых и доступных программных средств, работающих с трассами в других форматах, и внедрить в него поддержку формата OTF. В качестве такого средства было выбрано средство Vis3, предназначенное для работы с трассами в формате TRC. Визуализатор временных диаграмм Vis3 и формат TRC были разработаны в лаборатории вычислительных комплексов для хранения результатов имитационного моделирования функционирования вычислительных систем, их анализа и визуализации в рамках среды моделирования ДИАНА [111], стенда полунатурного моделирования (стенд ПНМ) и стенда математического моделирования КБО (СММ КБО) [8]. Главным недостатком формата TRC оказалась плохая расширяемость, гибкость и сложная структура. Средство визуализации и анализа трасс Vis3 хорошо себя зарекомендовало в течение нескольких лет работы в рамках этих стендов, поэтому в рамках НИР было принято решение по развитию Vis3 и разработке средства Vis4 с поддержкой формата OTF.

4.7.1 Окружение Vis3

Средство визуализации и анализа трасс Vis3 является одним из модулей стендов ПНМ и СММ КБО, поэтому тесно взаимосвязано с другими модулями. Диаграмма зависимостей Vis3 от других модулей стенда и внешних библиотек представлена на рисунке 49.

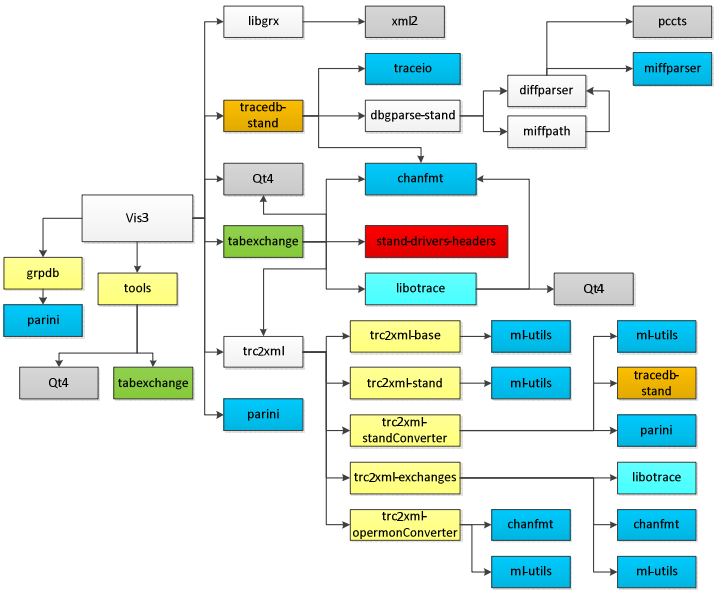

Рисунок 49. Диаграмма зависимости Vis3 от других модулей стенда

Из рисунка 49 видно, что Vis3 взаимодействует с 21-м модулем стенда и требует наличия 3-х внешних библиотек (pccts, xml2, Qt4). Таким образом, одна из проблем Vis3 – сильная связность с другими модулями стенда. Поскольку поддержка формата TRC не требуется в рамках НИР, в Vis4 была удалена поддержка формата TRC из-за сильной зависимости с модулями стенда и добавлена поддержка только формата OTF.

4.7.2 Основные классы Vis4

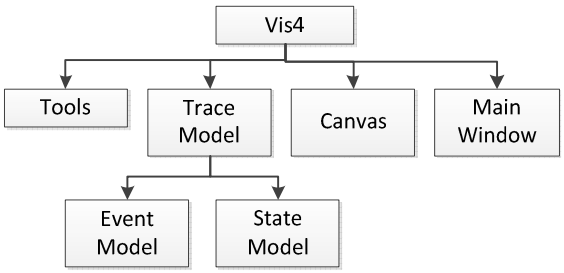

Рисунок 50. Основные классы и модули Vis4

На рисунке 50 представлены основные модули Vis4: Trace_model отвечает за чтение трассы и ее представление, классы Event_model и State_model определяют события и состояния РВС РВ, canvas отвечает за визуальное отображение трассы, MainWindow реализует пользовательский интерфейс Vis4, модуль Tools содержит набор инструментов (в том числе инструментов анализа) для работы с трассой. Методы классов MainWindow, Trace_model и Canvas представлены на рисунке 51.

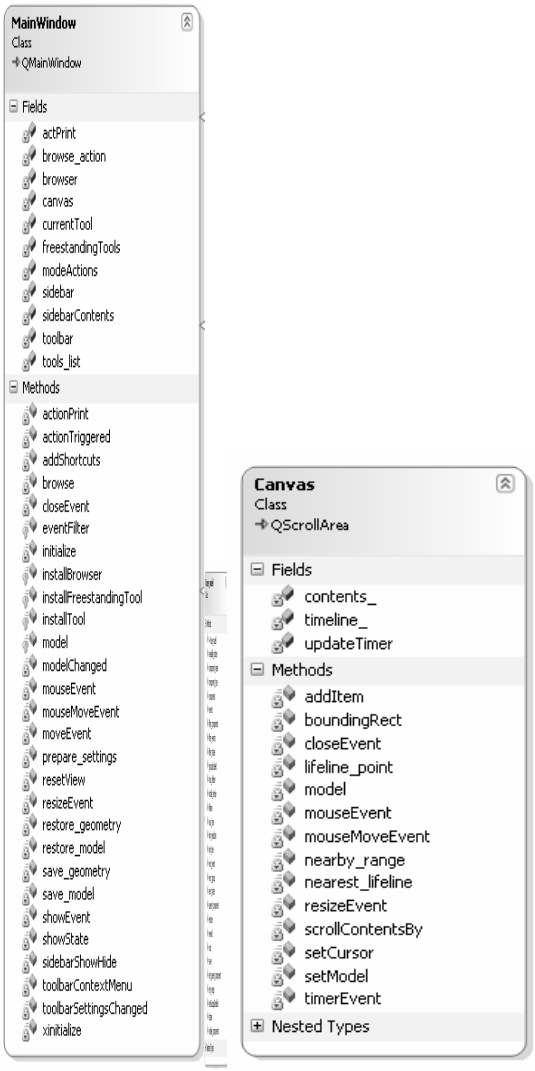

Рис. 51 Методы классов MainWindow, Trace_model и Canvas.

Класс MainWindow, определяющий общий вид окна временной диаграммы Vis4, содержит панель инструментов (toolbar), область показа диаграммы (canvas), и область для инструментов (sidebar). Классу MainWindow после объявления передается объект класса Trace_model, который описывает начальный вид трассы. После этого создается область показа диаграммы, набор доступных инструментов, и дальнейшее взаимодействие с пользователем определяется инструментами.

Класс Trace_model – базовый класс, на основе которого для каждого проекта создается дочерний класс, определяющий, откуда брать трассу. Набор инструментов для работы с трассой определяется виртуальным методом initialize. Доступ к трассе осуществляется через абстрактный интерфейс класса Trace_model. Основная функция класса – последовательно выдавать события и состояния, которые необходимо отображать в визуализаторе.

За визуализацию трассы в Vis4 отвечает класс Canvas. Он хранит текущий показываемый объект класса Trace_model, и показывает его в виде временной диаграммы. Класс отвечает за визуализацию событий, состояний и обменов сообщениями (в виде стрелочек). Кроме этого, он предоставляет интерфейс, через который инструменты могут добавлять графические объекты поверх диаграммы, например, выделить найденное событие.

4.7.3 Инструменты для работы с трассой

Инструменты для работы с трассой в Vis4 можно разделить на два типа: "управляемые" и независимые. Управляемые элементы отображаются в правой части приложения (sidebar), и в каждый момент времени отображается только один инструмент. На панели инструментов (toolbar) в верхней части Vis4 есть набор кнопок, позволяющих переключать управляемые инструменты.

Независимые инструменты - это кнопки в отдельной области панели инструментов (toolbar) сверху. Обработка нажатия на подобные кнопки задаются автором инструмента, например, может появляться дополнительное окно с графиками.

Взаимодействие между инструментами и визуализатором происходит следующим образом. Объект класса Canvas хранит текущий объект класса Trace_model. Все инструменты имеют к нему доступ. Если инструмент не изменяет текущий вид трассы (например «измеритель расстояний»), то он просто считает трассу, возможно добавляя к Canvas дополнительные графические элементы. Если инструмент в результате работы изменяет текущий вид трассы, то он создает новый объект класса Trace_model и передает его в Canvas. При этом генерируется сигнал, который ловится всеми инструментами и приводит к обновлению всех элементов графического интерфейса. Например, если изменяется глобальный фильтр событий, то инструмент «поиск событий» делает глобально скрытые события недоступными для поиска.

4.7.4 Механизмы расширения Vis4

Одним из достоинств Vis4 является наличие механизмов расширения, при помощи которых архитектура средства может быть расширена для новых проектов и работы средства

с другими форматами трасс. Таким образом, для внедрения нового формата трасс в Vis4 необходимо:

1. определить новый класс, реализующий интерфейс Trace_model. Интерфейс достаточен для показа временной диаграммы, но может быть недостаточен для реализации специфичных возможностей, например показа параметров или поиска по параметрам. В этом случае, в унаследованный класс могут быть добавлены иные методы, необходимые для конкретного проекта.
2. Расширить следующие классы с помощью наследования:
   - класс MainWindow, определяющий общий вид окна временной диаграммы, и переопределить в нём метод initialize.
   - класс Event_model, определяющий информацию о событии, и переопределить в нём метод detailWidget.
   - Класс State_model, определяющий информацию о состоянии компонента РВС РВ.
   - класс Tool, реализующий «управляемый инструмент», и переопределить в нём методы activate и deactivate.

Унаследованные класс, как правило, должны использовать интерфейс класса Canvas для реализации графических элементов. В каждый из унаследованных классов могут быть добавлены дополнительные методы, используемые специфичными инструментами или учитывающие особенности формата трассы.

4.7.5 Система сборки для Vis4

Для автоматизации сборки проектов традиционно используют системы управления сборкой, такие как make на Unix подобных системах и nmake для компиляторов Microsoft. Для сборки Vis3, как одного из модулей стенда СММ КБО, использовалась система управления сборкой Boost.build. Однако следует отметить, что для сборки остальных модулей стенда, от которых зависит Vis3, используется система cvslvk, разработанная в лаборатории вычислительных комплексов в 1999. В настоящее время cvslvk устарела, не поддерживается и используется только для работы с уже существующими проектами, поэтому её использование в рамках НИР нецелесообразно. Таким образом, возникла проблема выбора системы управления сборкой для Vis4. К системе сборки предъявлялись следующие требования:

- Открытость и доступность.
- Кроссплатформенность.
- Поддерживаемость.

- Используемость.
- Простота написания скриптов для сборки.
- Способность кеширования собираемых файлов для ускорения сборки.
- Поддержка языков C, C++.
- Поддержка конфигураций сборки.
- Автоматический поиск зависимостей в системе.

Для выбора системы управления сборкой Vis4 был проведен сравнительный анализ наиболее известных и используемых в последнее время систем (Таблица 4).

Таблица 4. Сравнительный анализ систем управления сборкой

Критерий	**make**	**Boost.Build**	**CMake**	**Scons**
сведения о разработчике (правообладателе)	Free Software Foundation	David Abrahams and Vladimir Prus	Компания Kitware	Steven Knight
Открытость (Лицензия)	Лицензия GNU GPL версии 3.	Лицензия Boost Software License версия 1	Лицензия BSD	Лицензия MIT
Используемость и апробированность средства в эксплуатации	Известно более 30 лет, широко используется для сборки ПО как с открытым, так и с закрытым кодом	Известно с 2002 г.	Известно не менее 6 лет. Успешно используется для сборки таких крупных проектов как KDE, MySQL, **CERTI**	Известен с 2008 года. Успешно используется для сборки таких проектов как:VMWare, Google Chrome, Blender и др.
Кросс-платформенность	да	да	да	да
1. Языковые возможности				
1.1. Переменные	Строковые, списки строк	Строковые, списки строк	Строковые, списки строк	Строковые, списки строк
1.2. Управляющие структуры	if в теле файла управления сборкой, if и foreach в функциях	if, foreach	if, foreach	if, foreach
1.3. Средства задания зависимостей	Явные правила для конкретных файлов, неявные правила для шаблонов имен файлов	Гибкие средства задания зависимостей, в том числе управляемое пользователем связывание цели с подцелью	Набор правил для предопределённых целей, окончательный список зависимостей формируется после генерации файла управления сборкой; есть конструкция для задания определяемых пользователем типов целей.	Гибкие средства задания зависимостей

Продолжение таблицы 4

Критерий	make	Boost.Build	CMake	Scons
1.4. Средства задания действий	Последовательность команд на языке shell	Императивный язык программирования	Набор предопределённых действий.	Сценарии на языке Python
2. Возможность параллельного выполнения задач сборки	Есть	есть	Есть	Есть
3. Оценка удобства поддержки сложных проектов	Малая степень удобства	Большая степень удобства	Большая степень удобства	Большая степень удобства
4. Сборка проекта с иерархией подпроектов	Есть поддержка рекурсии по каталогам. Но для практической работы нужно предварительное конфигурирование.	Автоматическое распространение параметров сборки по подцелям; удобство ссылок на подпроекты	Поддерживается, автоматический поиск CMakeList	Поддерживается
5. Поддержка различных конфигураций сборки	Для реальной работы нужно предварительное конфигурирование Make-файлов дополнительными средствами (GNU autoconf)	Поддерживается	Поддерживается	Поддерживается

На основе проведённого анализа для Vis4 было решено использовать систему управления сборкой CMake [112] и рекомендовано использовать её для других программных средств в рамках НИР.

4.7.6 Пользовательский интерфейс Vis4

Средство визуализации и анализа трасс Vis4 представлено на 52 и содержит область инструментов для работы с трассой (сверху), область просмотра информации о трассе и её отдельных элементах (слева) и рабочую область (справа), отображающую трассу.

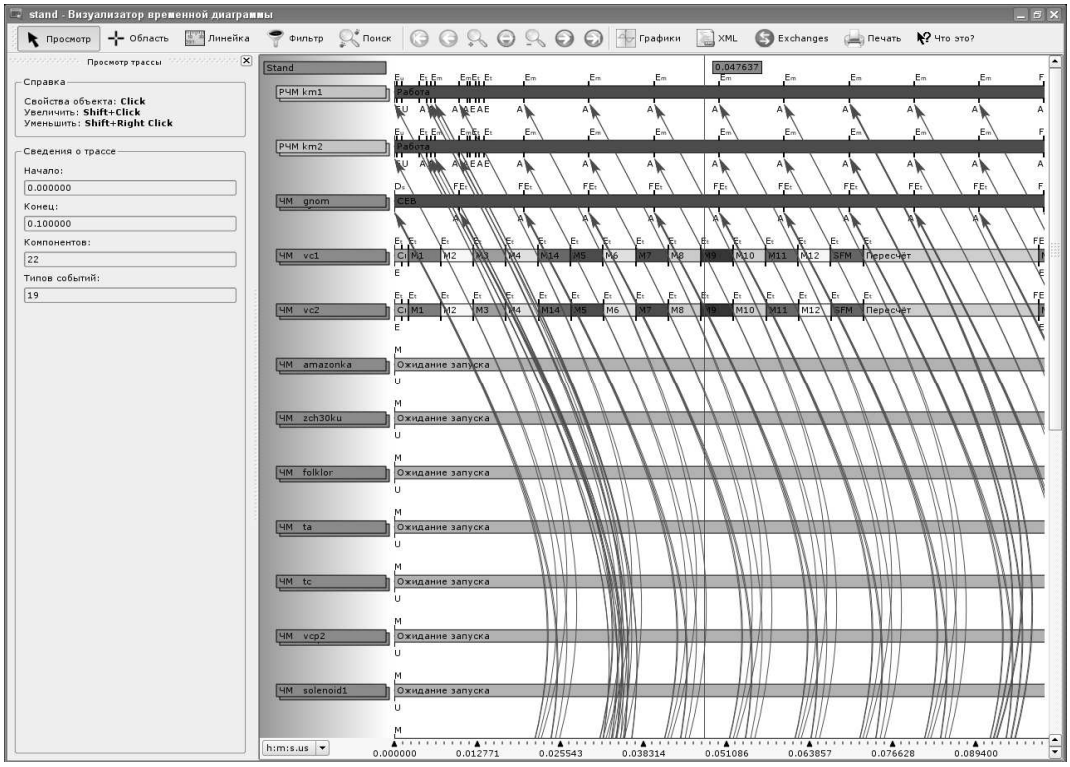

Рисунок 52. Средство визуализации и анализа трасс Vis4

4.7.7 Функциональные возможности Vis4

Vis4 поддерживает следующие функциональные возможности для работы с трассой:

- Наглядное отображение иерархической структуры моделируемой РВС РВ.
- Визуализация линий жизни каждого компонента, изменений состояний компонентов с длительностью пребывания компонента в конкретном состоянии.
- Визуализация обменов между компонентами.
- Визуализация изменения параметров компонентов в виде графиков.
- Отображение атрибутов событий и состояний.
- Масштабирование и навигация по трассе.
- Поиск событий и состояний.
- Возможность применения фильтров отображения по компонентам, событиям и состояниям.

4.7.8 Перспективные направления развития средства Vis4

В перспективе предполагается продолжить доработку Vis4, и возможны следующие направления развития:

- Добавление дополнительных алгоритмов анализа трасс и их сравнения.
- Добавление возможности «проигрывания трассы».
- Развитие средства для использования трассы в on-line режиме.

- Внедрение поддержки других открытых форматов и OTF2 в частности.

4.7.9 Выводы и результаты работы

В рамках задачи разработки средства анализа и визуализации трасс результатов моделирования РВС РВ были получены следующие результаты:

- Проведен анализ существующих средств анализа и визуализации трасс, опробованы на практике средства Vampir 7.3 и Vite для работы с трассами в формате OTF, в результате за основу выбрано средство Vis3.
- Проведен анализ архитектуры средства Vis3 и её возможностей.
- Проведен обзор систем управления сборкой проектов и в качестве единой для всех программных средств НИР рекомендовано использовать систему CMake.
- На основе средства Vis3, входящего в состав СММ КБО Разработано программное средство Vis4 для анализа и визуализации трасс в формате OTF.

4.8 Средство трансляции диаграмм состояний UML в автоматы UPPAAL

Для того чтобы модель можно было проверифицировать при помощи средства верификации UPPAAL в рамках данной НИР был разработан транслятор диагармм состояний в автоматы UPPAAL. Этот транслятор работает в два этапа. Сначала проверяется корректность диаграмм состояний, а затем происходит многошаговая трансляция диаграммы состояний в сеть плоских временных автоматов UPPAAL.

4.8.1 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).

4.8.2 Структура транслятора

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

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

Общая схема работы транслятора приведена на рисунке 53. Программа выполняет следующие действия:

- Библиотека xmiparser осуществляет синтаксический разбор поданного на вход xmi-файла, преобразуя его в структуры из пакета UMLStateMachine. В процессе используются анализаторы, генерируемые antlr.
- Диаграмма, заданная пользователем, преобразуется во временной автомат по алгоритму, описанному в разделе 5.1. Это действие выполняет функция Normalize из одноименного модуля.
- Преобразованная диаграмма подается на вход функции main_block из модуля TranslationAlgorithm. Выполняется преобразование по алгоритму, описанному в разделе 5.1.
- Полученный в результате предыдущего пункта объект типа TimedAutomaton из пакета UPPAAL экспортируется в строку в формате XML.
- Полученная строка в формате XML подается на вход библиотеке pyUPPAAL, которая приводит диаграмму в удобочитаемый вид и сохраняет в файл.

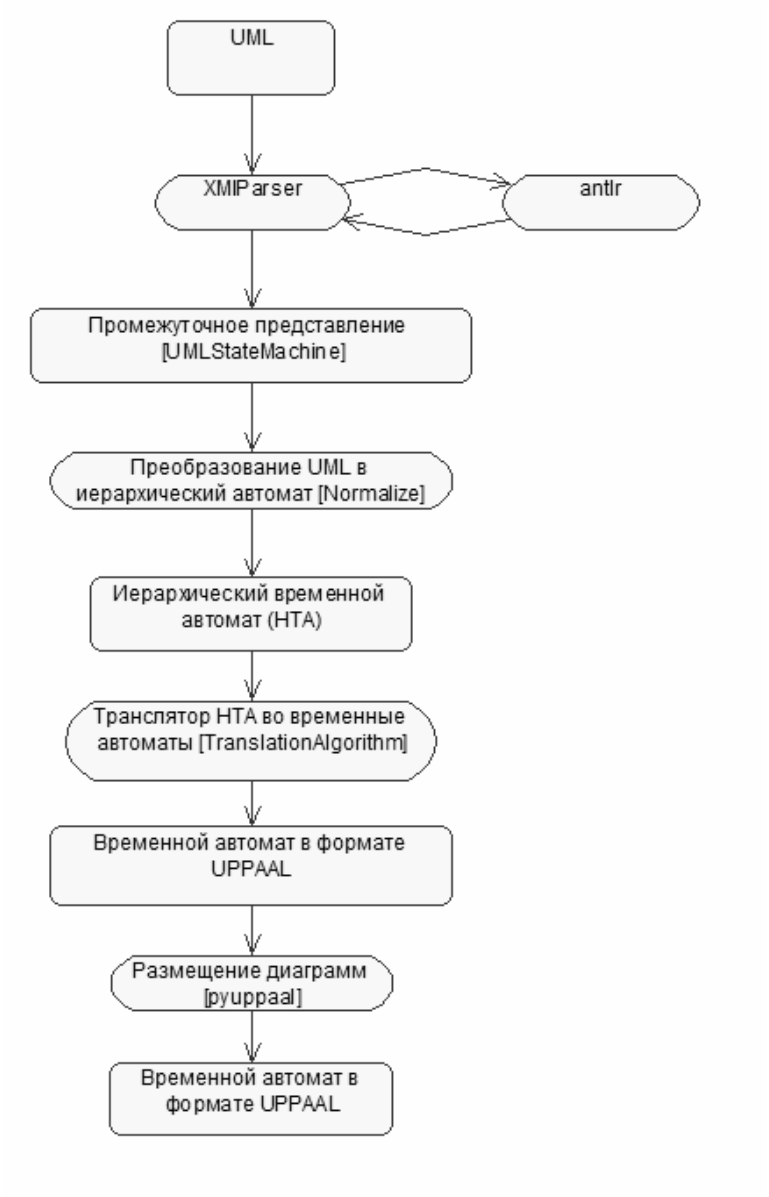

Рисунок 53. Схема работы транслятора

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

python uml2ta.py <xmi-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.

На рисунке 54 приведена диаграмма классов пакета UMLStateMachine.

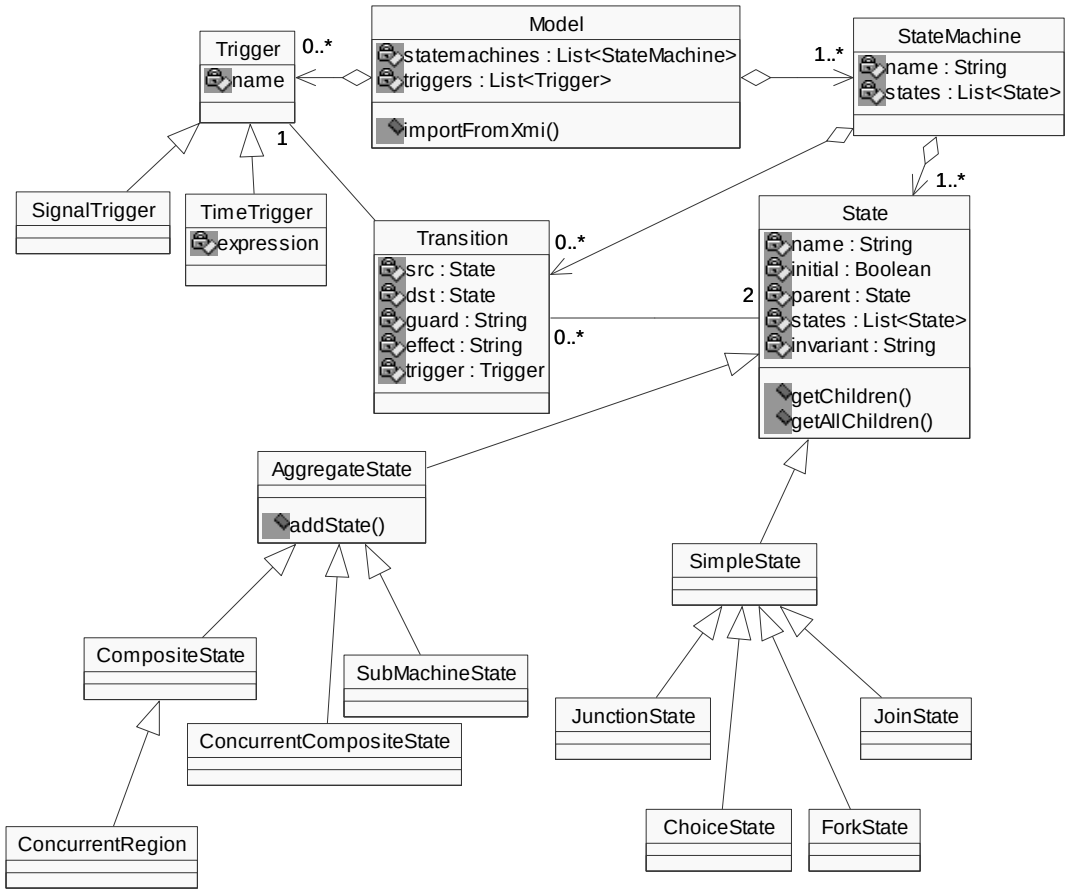

Рисунок 45. Диаграмма классов пакета UMLStateMachine

С точки зрения транслятора, 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 Xor-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.

На рисунке 55 приведена диаграмма классов пакета UPPAAL.

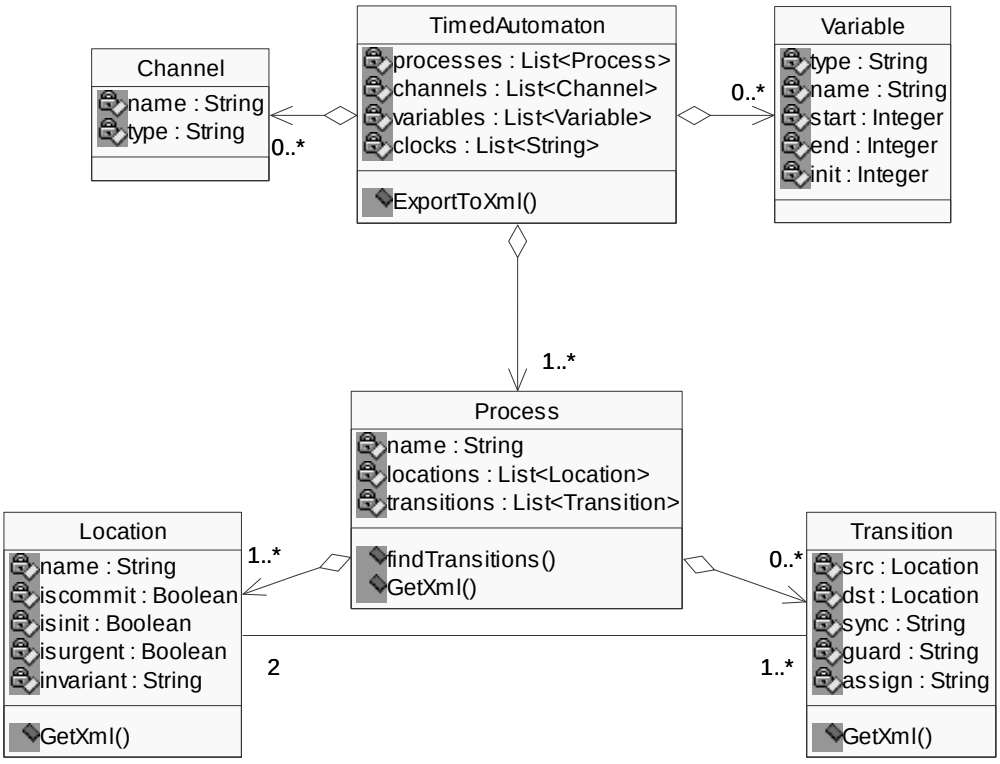

Рисунок 55. Диаграмма классов пакета UPPAAL

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.

В данном разделе описано средство трансляции диаграмм состояний UML в системы временных автоматов UPPAAL. Методика использования данного средства приведена в разделах 8.4 и 8.5. Эксперименты с этим средством описаны в разделах 9.5, 9.6 и 9.7.

4.9 Средство верификации модели

Как правило, средства верификации работают с моделями, описанными на специализированных языках, предназначенных для удобной обработки алгоритмами верификации. Выбранный нами формат описания РВС РВ — диаграммы состояний UML — предоставляет возможность быстро создавать удобочитаемые модели РВС РВ, однако на данный момент не существует эффективных средств верификации диаграмм данного формата.

В качестве средства верификации РВС РВ нами используется UPPAAL [113]. Система UPPAAL хорошо зарекомендовала себя во многих научно-исследовательских и промышленных проектах, связанных с анализом поведения РВС РВ [114],[115],[116],[117],[118],[119],[120],[121], она находится в открытом доступе, имеет удобный и развитый интерфейс. Система UPPAAL работает с математическими моделями РВС РС, представленными в виде сетей конечных временных автоматов, и спецификациями их поведения, выраженными формулами темпоральной логики TCTL.

Экспериментальное исследование средства верификации в рамках разработанной нами системы (в частности, примеры проверяемых системой темпоральных свойств) приведено в разделе 9.6.

4.9.1 Описание средства UPPAAL

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. [122],[123]. 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. [124]. 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 [125]. 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 TCTL (Timed 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/.

Основная возможность, предоставляемая средством UPPAAL, — автоматическая проверка темпоральных свойств входной модели — сети временных автоматов. Устройство этой модели подробно обсуждается в разделе 3.6. Устройство формул логики TCTL подробно обсуждается в следующем подразделе.

Помимо ответа на вопрос, выполняется ли данная формула логики TCTL в данной сети временных автоматов, UPPAAL предоставляет некоторые возможности симуляции работы сети и анализа ошибочных трасс. Симуляция состоит в пошаговом построении вычисления сети временных автоматов; при наличии возможности выполнения нескольких переходов или выбора между выполнением перехода и продвижением времени окончательное решение система может принимать как случайным образом, так и руководствуясь параметрами, заданными пользователем. При нахождении ошибочной трассы она тут же может быть воспроизведена симуляцией.

Чтобы воспользоваться программно-инструментальным средством UPPAAL для верификации РВС РВ, описанных в виде диаграмм состояний UML, нами был разработан алгоритм трансляции диаграмм в модель сети временных автоматов UPPAAL. Устройство диаграмм состояний подробно обсуждается в разделе 3.4, описание алгоритма приведено в разделах 4.8 и 5.1.

4.9.2 Структура 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 сверхсрочное) и инвариант состояния. Дуги графа соответствует 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

4.9.3 Темпоральная логика TCTL

Самая главная процедура, которую способна выполнять система верификации UPPAAL (и ради которой была создана вся система) – это процедура проверки выполнимости формул темпоральной логики TCTL на бесконечных моделях, представляющих собой множество всевозможных вычислений сетей конечных временных автоматов. Формулы логики TCTL используются в качестве спецификаций свойств поведения автоматов. Процедура верификации системы UPPAAL для заданной сети автоматов и заданной темпоральной формулы проверяет, выполняется ли эта формула на модели, образованной множеством всевозможных вычислений этой сети. 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 сравнение (сравнение таймера или разности таймеров с константой), а также формулы вида A.l, где A — автомат сети и l — состояние управления автомата. Из атомарных формул при помощи обычных булевых связок 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<>, E[], E<>, -->. Выполнимость формул каждого из перечисленных видов определяется следующим образом. Для каждого состояния вычисления s = (**l**, v) заданной сети N = (U_1, U_2,

…, U_n) и для каждого элементарного сравнения g это отношение выполняется в состоянии s (для обозначения этого факта используется обозначение s |= g), если v |= g. Атомарная формула U_i.l выполняется в состоянии вычисления s = (**l**, v) тогда и только тогда, когда l = **l**[i]. 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 |= A[] φ тогда и только тогда, когда в каждом состоянии всякого вычисления сети N, которое начинается в состоянии s, выполняется формула φ;
- s |= E[] φ тогда и только тогда, когда в каждом состоянии хотя бы одного вычисления сети N, которое начинается в состоянии s, выполняется формула φ;
- s |= A<> φ тогда и только тогда, когда хотя бы в одном состоянии всякого вычисления сети N, которое начинается в состоянии s, выполняется формула φ;
- s |= E<> φ тогда и только тогда, когда хотя бы в одном состоянии хотя бы одного вычисления сети N, которое начинается в состоянии s, выполняется формула φ;
- s |= φ --> ψ тогда и только тогда, когда в каждом вычислении сети N, 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<> φ. 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[] φ и E[] φ. 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<> φ и φ --> ψ.

4.10 Средство оценки наихудшего времени выполнения Metamoc

В качестве средства оценки наихудшего времени выполнения используется средство Metamoc, которое в процессе анализа оценки WCET использует верификатор UPPAAL. Описание работы средства можно найти в [126].

Средство поддерживает программы, написанные на подмножестве языке программирования Си, и поддерживает следующие структуры языка:

- целочисленные переменные
- массивы целочисленных данных
- указатели
- арифметика над целочисленными данными и указателями
- условные операторы

4.10.1 Общая схема работы средства

Общая схема работы средства изображена на рисунке 56 и заключается в следующей последовательности шагов:

1. Построение объектного файла по исходному коду программы.

На данном этапе используется кросс-компилятор GCC для целевой архитектуры для получения исполнимого кода программы. Текущей целевой архитектурой является процессора ARMT9.

2. Получение целевого кода вычислителя по объектному файлу.

Производится дизассемблирование объектного файла с помощью утилиты objdump для целевой архитектуры для получения ассемблерного кода программы.

3. Преобразование кода программы в модель для верификации.

На данном этапе используется специальный преобразователь Arm-to-uppaal целевого кода в модель для верификации с помощью инструмента UPPAAL.

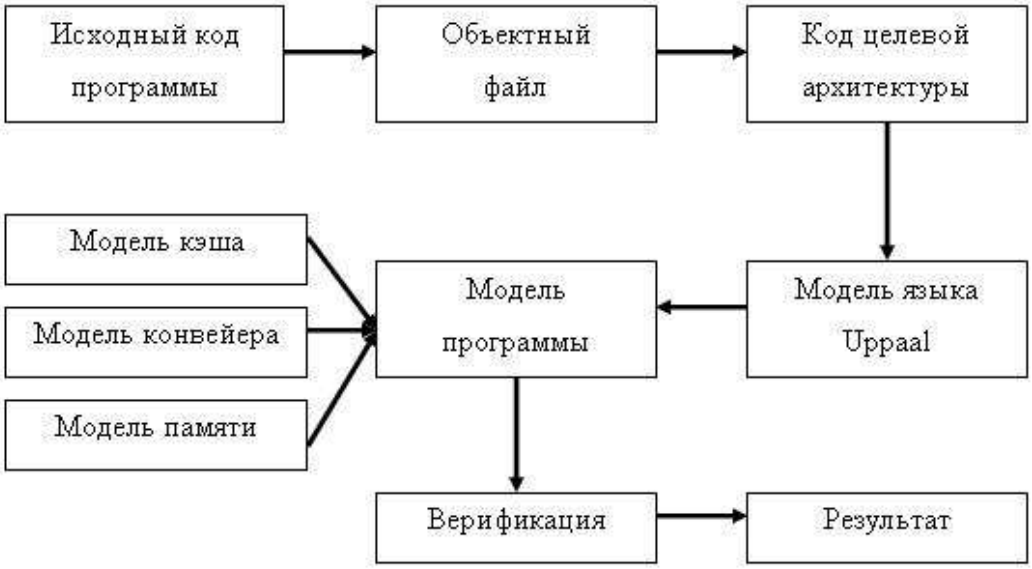

Рисунок 56. Схема работы средства.

4. Присоединение модели вычислителя к модели программы.

Описание модели компонентов целевого вычислителя, такие как модель кэша, конвейера и памяти, соединяются с построенной моделью программы. Обзор существующих моделей вычислителей приводится в следующем разделе.

5. Запуск верификации с поиском наихудшего времени выполнения программы.

На данном этапе используется специальная возможность верификатора UPPAAL – возможность поиска наибольшего значения переменной, при котором выполняется заданное свойство (выполняется с помощью операции SUP инструмента UPPAAL). Производится поиск наибольшего значения переменной, задающей время выполнения программы.

Найденное значение переменной является наихудшим временем выполнения программы.

4.10.2 Описание используемой модели вычислителя

Используемая модель вычислителя состоит из следующих компонентов:

- кэш
- конвейер
- основная память

Конвейер представляет собой набор последовательно работающих функциональных устройств, каждый из которых выполняет некоторый этап выполнения инструкции программы. Каждый из функциональных устройств моделируется в виде диаграммы, которая состоит из нескольких состояний, таких как ожидание, начало выполнения соответствующего этапа, конец выполнения этапа, передача операции следующей части конвейера или выдача результата. В процессе моделирования программы каждая инструкция подается на вход конвейеру и ожидает завершения его работы с вычислением времени выполнения данной инструкции. На некоторых этапах работы конвейера (таких как чтение или запись значения переменной) происходит обращение к другим моделям компонент вычислителя, таким как кэш и память. Схема модели пятиступенчатого конвейера изображена на рисунке 56.

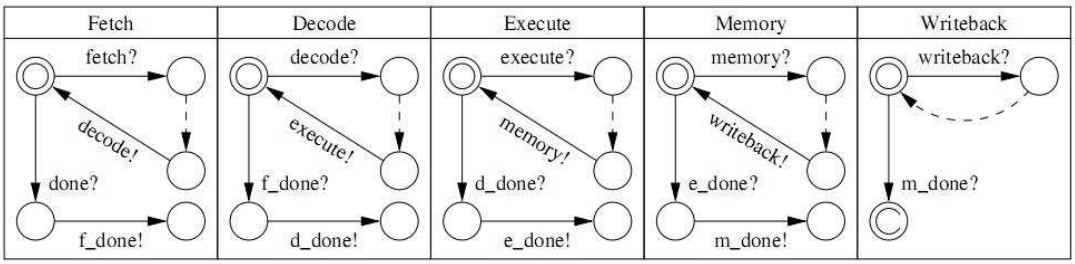

Рисунок 56. NTA-модель пятиступенчатого конвейера

В средстве моделируются как кэш данных, так и кэш инструкций. Модель каждого кэша представляется в виде множества ячеек, в которых хранятся переменные. На этапе чтения/записи значения переменной конвейер обращается к кэшу для выявления наличия в нем переменной. При попадании моделируется чтение/запись значения и вычисляется время доступа (например, один такт работы процессора). При промахе производится обращение к модели памяти, и время доступа к переменной существенно увеличивается (например, до 30 тактов). Значение после промаха записывается в кэш.

В работе [126] показаны результаты проведения экспериментов, показывающие существенное повышение точности оценки наихудшего времени выполнения при использовании описанной модели вычислителя. При этом существует специальный преобразователь arm-to-uppaal, преобразующий код целевой архитектуры в модель языка UPPAAL, которая в совокупности с моделями архитектурных компонент описывает модель временного поведения программ.

4.10.3 Архитектура средства

Средство оценки наихудшего времени выполнения программ Metamoc состоит из следующих компонент:

- Дизассемблер целевого процессора
- Статический анализатор кода
- Преобразователь
- Верификатор

Дизассемблер целевого процессора используется для построения ассемблерного кода из объектного файла. В средстве Metamoc в качестве данного компонента используется утилита objdump для целевого процессора.

Статический анализатор кода используется для разметки кода целевого процессора, выделения линейных участков; опционально данная компонента может производить поиск и удаление недостижимых участков кода.

Преобразователь производит построение модели для ее дальнейшего анализа верификатором. Модель строится по размеченному ассемблерному коду программы, результатом работы является модель программы на языке, испольщуемом верификатором. В средстве используется преобразователь arm-to-uppaal, который производит построение модели на языке UPPAAL.

Верификатор используется для запуска процесса верификации модели программы с целью получения наихудшего времени выполнения. В средстве используется верификатор UPPAAL.

Архитектура срества изображена на рисунке 57.

Общая схема анализа времени выполнения линейных участков производится в следующей последовательности шагов:

- На вход подается объектный файл программы.
- Производится дизассемблирование объектного файла с помощью утилиты objdump для целевого процессора.
- Производится разметка кода, выделение линейных участков.
- Производится построение модели на языке UPPAAL с помощью преобразователя arm-to-uppaal
- Модель программы на языке UPPAAL объединяется с моделями компонент вычислителя.

Полученная модель подается на вход верификатору UPPAAL, который вычисляет время выполнения линейных участков.

Подробное описание метода оценки наихудшего времени выполнения, используемого в средстве Metamoc приведена в разделе 5.5, а схема интеграция средства оценки наихудшего времени выполнения Metamoc со средой моделирования описана в разделе 5.6.

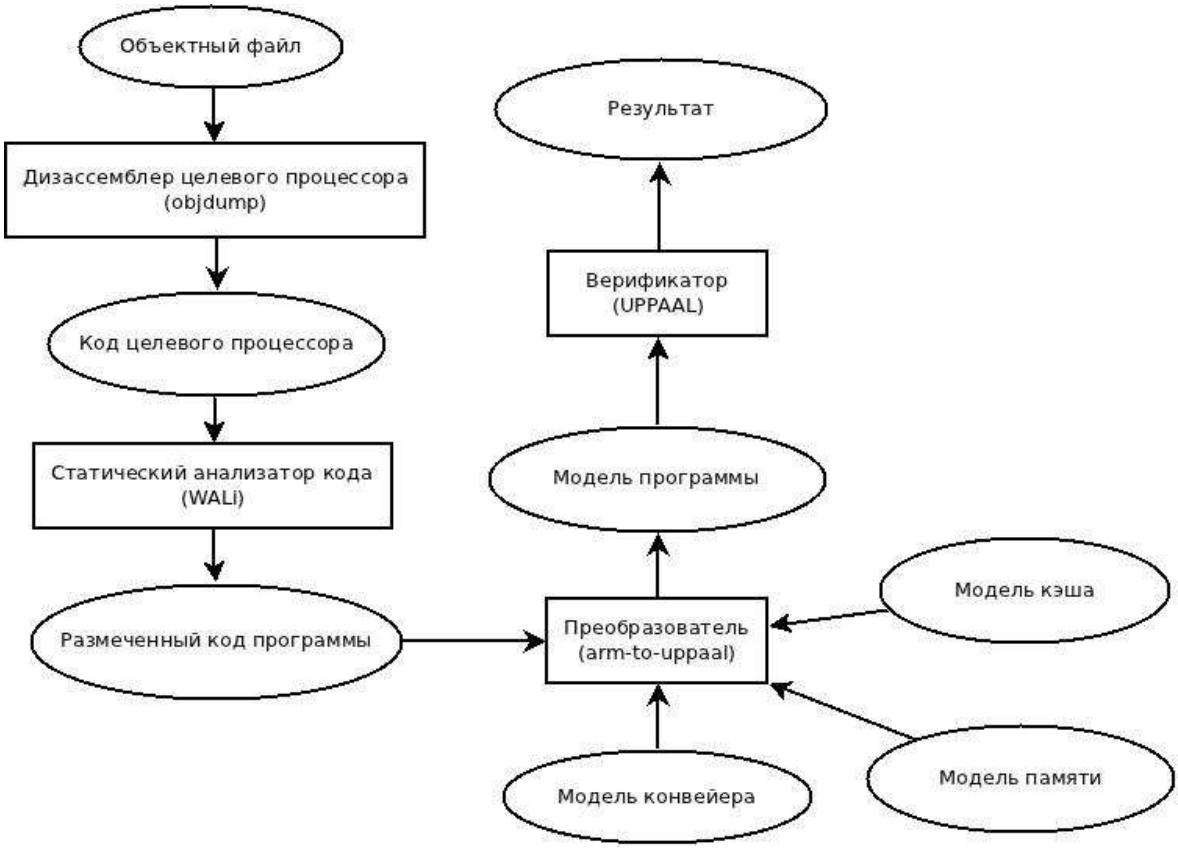

Рисунок 57. Архитектура средства Metamoc

4.11 Интегрированная среда разработки и анализа моделей

Интегрированная среда разработки и анализа моделей объединяет под единым управлением все программные средства разработанные или используемые в данном проекте.

Главное окно программы (рисунок 58), помимо меню и панели инструментов, содержит три основных области: слева находится список файлов, с которыми идет работа, справа сверху – информация об открытых в данный момент файлах, справа снизу – область, куда выводятся диагностические сообщения и логи работы системы.

Основная сущность, с которой работает программа – проекты. В файловой системе проект представляет собой древовидную структуру файлов и директорий, вложенную в одну общую директорию, в которой также находится файл с расширением .dyana, описывающий структуру проекта. В левой части рабочего окна в виде дерева визуализируется структура

проекта, полностью совпадающая со структурой каталогов в файловой системе. У каждого типа объектов, с которыми работает программа, есть своя иконка в дереве. Типы объектов следующие:

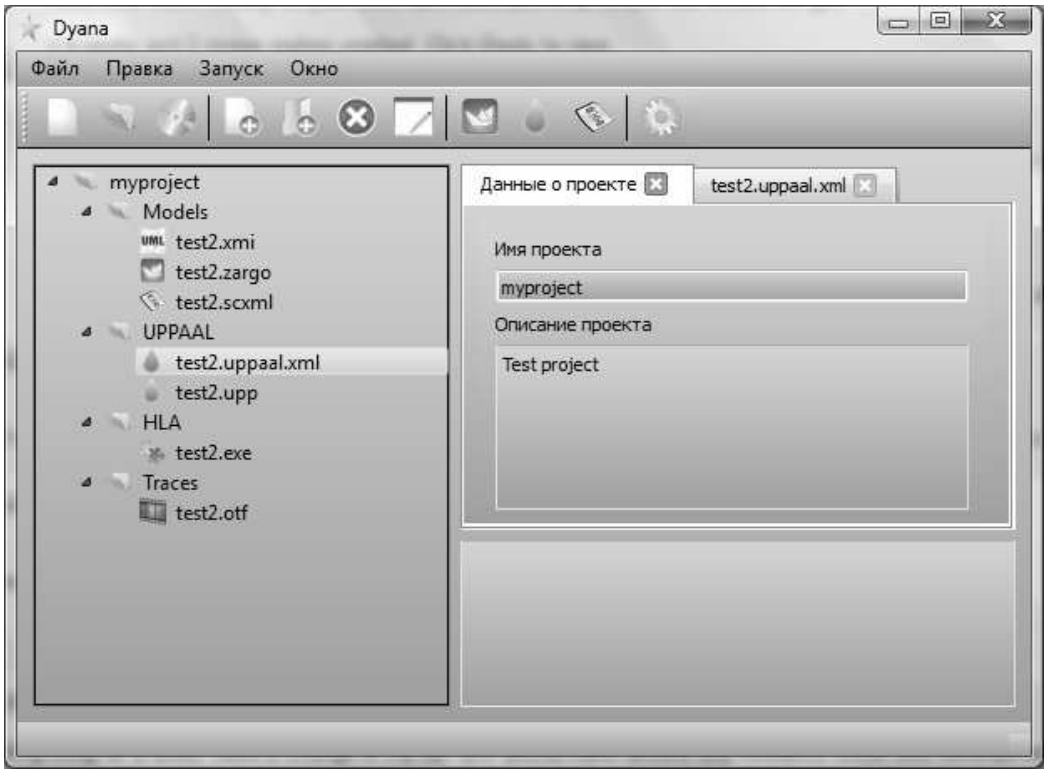

Рисунок 58. Главное окно программы

Папки – для объединения файлов в группы.

Модели на UML – файлы с расширениями .uml и .zargo, предназначенные для обработки в ArgoUML.

Модели на XMI – файлы с расширением .xmi, экспортируемые из ArgoUML.

Модели на SCXML – файлы с расширением .SCXML, создаваемые в ручную или генерируемые на основе XMI.

Файлы UPPAAL – системы автоматов в файлах с расширением .uppaal.xml, служебная информация для анализа трасс UPPAAL в файлах с расширением .upp, свойства верификатора в файлах с расширением .q

Модели HLA – файлы launcher.py, запускающие прогон в CERTI

Файлы трасс – трассы в формате OTF.

Прочие файлы – исходные коды и исполняемые файлы моделей CERTI, служебные файлы транслятора в UPPAAL, файлы со свойствами для верификатора, произвольные файлы, для которых в системе не предусмотрено специальной обработки.

Для каждого типа файлов существует свой диалог с настройками, который появляется в правой части окна в виде вкладки при двойном клике на файл в дереве.

В главном меню доступны следующие действия:

- File – New Project (Ctrl+N, также кнопка на панели инструментов). Создает новый проект в указанной директории. В этой директории создаются пустые папки Models, HLA, UPPAAL, Traces, а также конфигурационный файл проекта с расширением .dyana и названием таким же, как у выбранной директории.
- File – Open Project (Ctrl+O, также кнопка на панели инструментов). Загружает проект из заданного файла с расширением .dyana.
- File – Save Project (Ctrl+S, также кнопка на панели инструментов). Сохраняет состояние проекта в конфигурационный файл.
- Список из не более 10 последних открытых проектов. Клик по элементу списка, а также Alt+номер, открывает соответствующий проект.
- File – Exit (Ctrl+X). Завершает работу программы.
- Edit – Add File (Ctrl+A, также кнопка на панели инструментов и пункт в контекстном меню списка файлов). Запускает стандартный диалог выбора файла, после чего файл добавляется в выделенную директорию. Физически файл копируется в директорию проекта. Тип файла определяется автоматически по расширению.
- Edit – Add Folder (также кнопка на панели инструментов и пункт в контекстном меню списка файлов). Добавляет папку с выбранным названием
- Edit – Delete (Del, также кнопка на панели инструментов и пункт в контекстном меню списка файлов). Удаляет выбранный элемент в дереве и все элементы, вложенные в него. Файл также удаляется из файловой системы, поэтому данная операция необратима.
- Edit – Change Type (Ctrl+E, также кнопка на панели инструментов и пункт в контекстном меню списка файлов). Меняет тип, приписанный к файлу. Следует использовать, если у файла нестандартное расширение.
- Launch – ArgoUML (F2, также кнопка на панели инструментов). Запускает ArgoUML без входного файла.
- Launch – UPPAAL (F3, также кнопка на панели инструментов). Запускает UPPAAL без входного файла.
- Launch – SCXMLGui (F4, также кнопка на панели инструментов). Запускает SCXMLGui без входного файла.

- Window – Settings (F10, также кнопка на панели инструментов). Открывает окно настроек. В этом окне можно указать пути к используемым внешним программам.

Далее подробно рассмотрим окна для каждого типа файлов.

4.11.1 Окно для файлов UML

На рисунке 59 приведено окно для работы с файлами UML. Основную часть окна занимает виджет, в котором можно просмотреть содержимое текущего файла. В текущей версии редактирование файла вручную не допускается.

Кнопка внизу запускает ArgoUML и сразу открывает в нем файл, соответствующий данной вкладке.

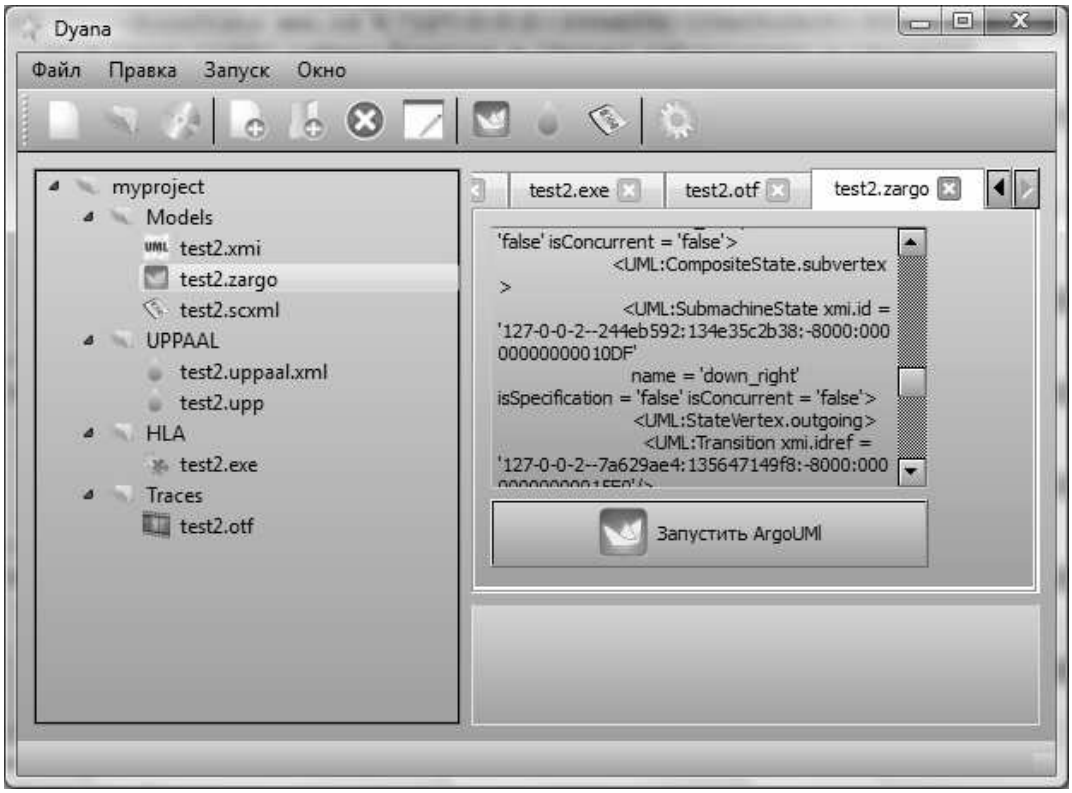

Рисунок 59. Окно для файлов UML

4.11.2 Окно для файлов XMI

На рисунке 60 приведено окно для работы с файлами XMI. При открытии XMI-файла он предварительно анализируется, чтобы определить, какие автоматы в нем заданы.

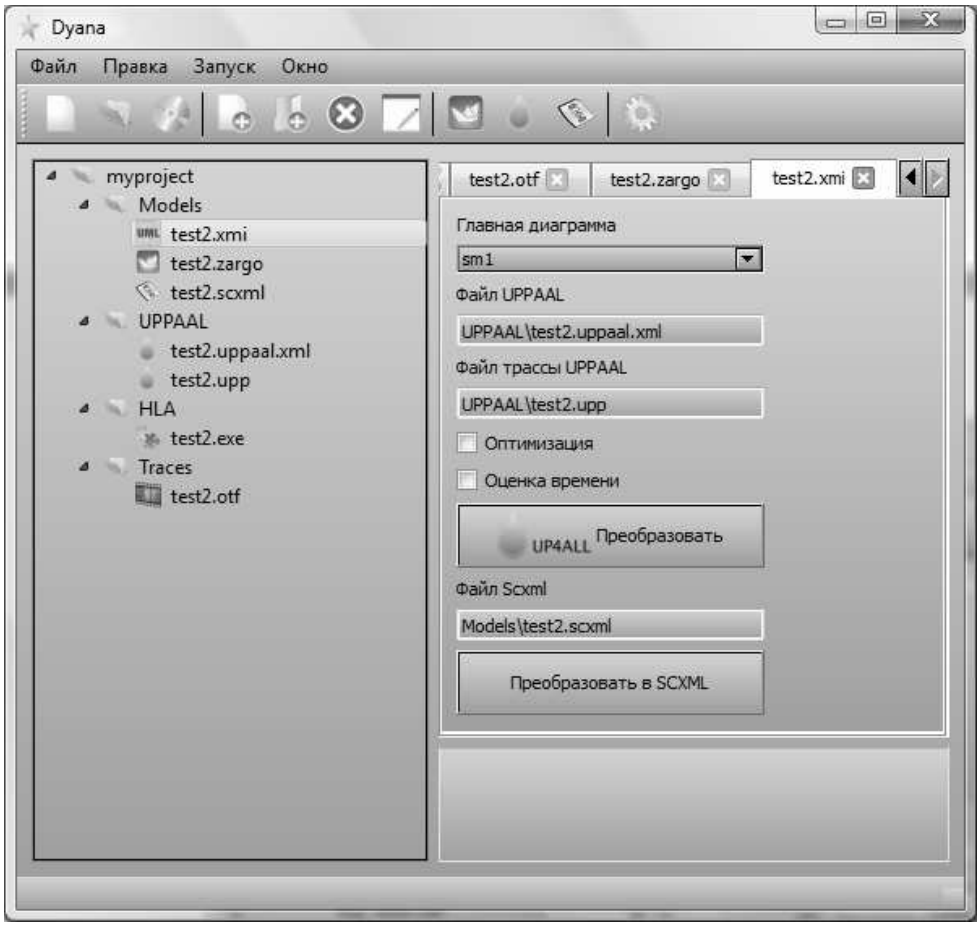

Рисунок 60. Окно для файлов XMI

Список позволяет выбрать, какой из автоматов следует транслировать в UPPAAL. Две редактируемые строки задают путь, по которому следует записать результаты трансляции в UPPAAL. Путь задается относительно корня дерева файлов проекта. По умолчанию файлы имеют то же имя, что и XMI-файл и расширения .uppaal.xml и .upp для файла UPPAAL и вспомогательных данных для анализа трассы соответственно. По умолчанию файлы помещаются в директорию «UPPAAL». Галочки задают, необходимо ли оптимизировать автомат и оценивать время выполнения соответственно.

4.11.3 Окно для файлов SCXML

На рисунке 61 приведено окно для работы с моделями в формате SCXML. Возможны три основных действия. Во-первых, можно просто открыть файл для просмотра и редактирования в редакторе SCXML GUI. Во-вторых, можно сгенерировать код модели HLA, задав путь, по которому будут сохранены файлы модели. Путь интерпретируется как имя директории, в которую будут помещены все файлы, созданные кодогенератором. Файл launcher.py имеет тип «модель HLA» и через него запускается CERTI, остальные файлы относятся к типу «прочие файлы». В-третьих, можно конвертировать модель в систему временных автоматов UPPAAL, аналогично XMI-файлу.

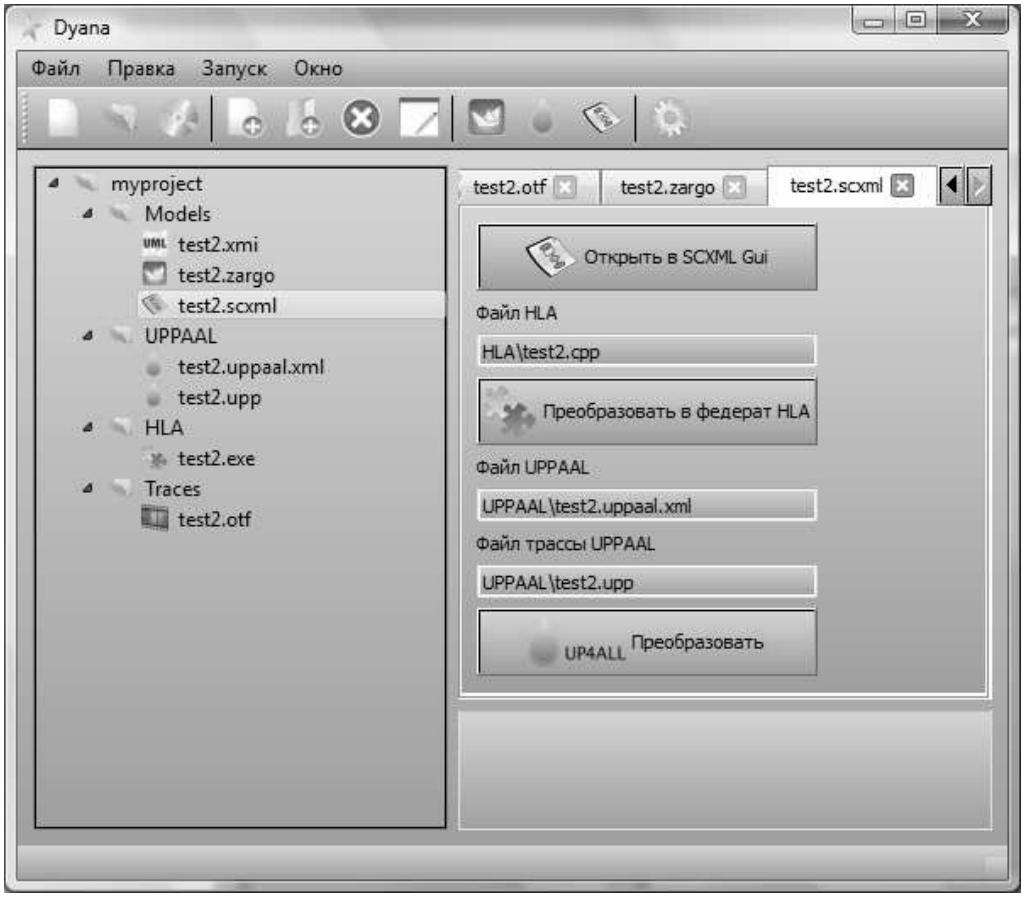

Рисунок 61. Окно для файлов моделей SCXML

4.11.4 Окно для файлов UPPAAL

На рисунке 62 приведено окно для работы с файлами UPPAAL. Кнопка «запустить UPPAAL» запускает GUI средства UPPAAL и открывает в нем выбранный файл. Можно также сразу загрузить файл со свойствами для верификации.

Кнопка «Преобразовать в трассу UML» конвертирует трассу UPPAAL в трассу на соответствующей модели UML. Необходимо выбрать трассу из списка всех загруженных в проект файлов с расширением .xtr, служебный файл с расширением .upp, а также указать имя файла с результирующей трассой.

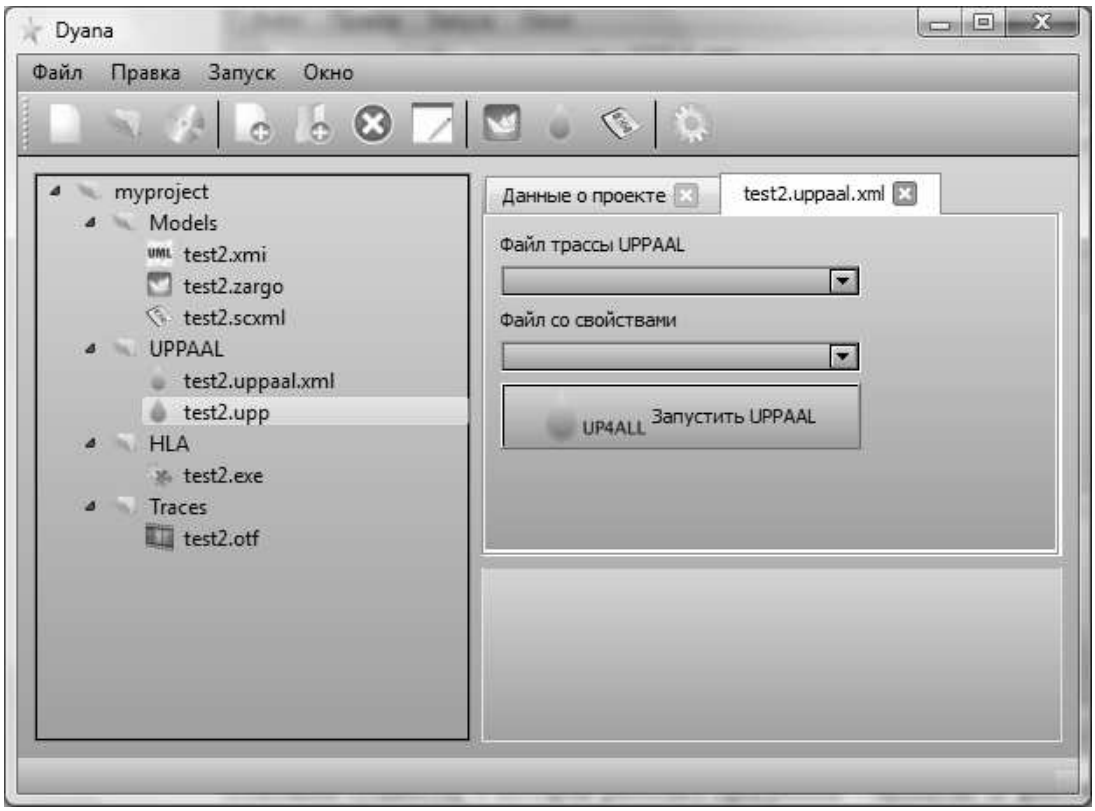

Рисунок 62. Окно для файлов UPPAAL

4.11.5 Окно для файлов HLA

На рисунке 63 приведено окно для запуска экспериментов в CERTI. Большую часть окна занимает виджет, в котором можно просмотреть содержимое файла launcher.py в текстовом виде.

В строке задается путь, по которому будет создан файл с трассой в формате OTF.

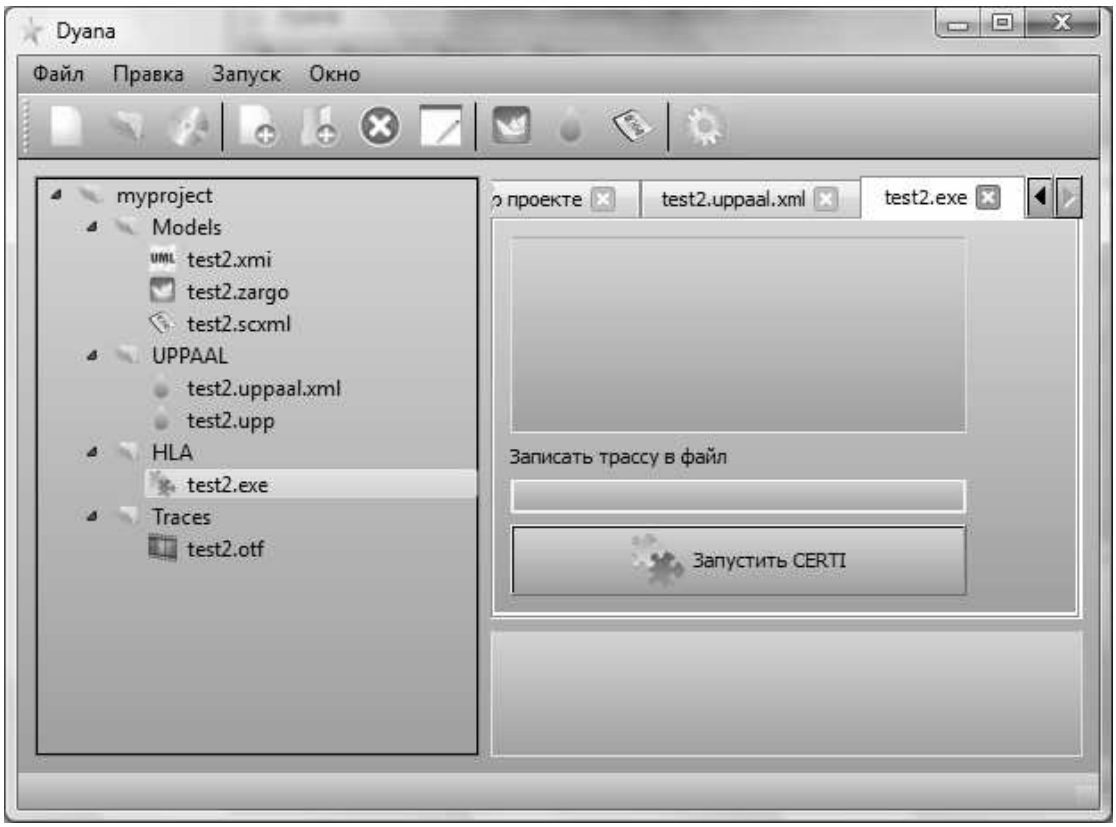

Рисунок 63. Окно для файлов CERTI

4.11.6 Окно для файлов трасс

На рисунке 64 приведено окно для работы с трассами в формате OTF. Большую часть окна занимает виджет, в котором можно просмотреть содержимое трассы в текстовом виде. Ручное редактирование трасс не допускается из соображений безопасности.

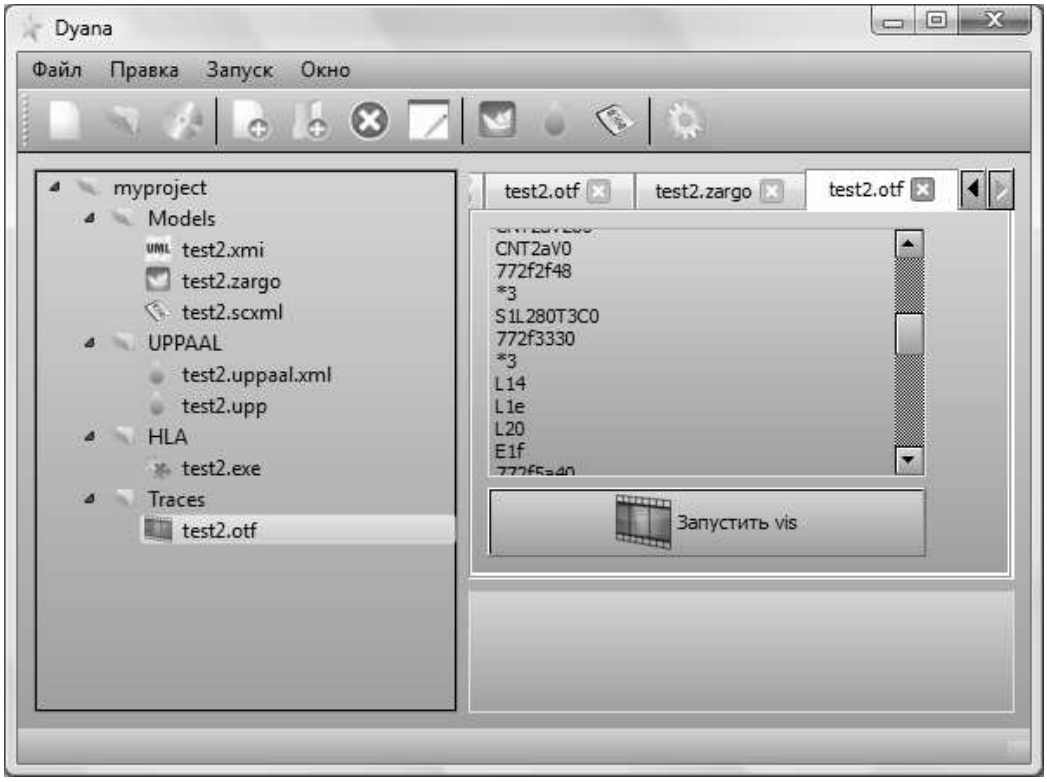

Рисунок 64. Окно для файлов трасс

Кнопка внизу запускает vis и сразу открывает в нем соответствующую трассу.

В данном разделе была описана интегрированная среда моделирования и верификации РВС РВ. Подробное описание сценариев работы с данным средством приведены в разделах 8.2 – 8.6 и 9.9.

5 Описание разработанных методов и способов интеграции со сторонними средствами

В данном разделе описываются разработанные методы и способы интеграции со сторонними средствами. В разделе 5.1 приводится описание алгоритма трансляции иерархических временных автоматов в сети плоских временных автоматов. Раздел 5.2 содержит доказательство корректности алгоритма трансляции UML-диаграмм в сеть плоских временных автоматов. В разделе 5.3 приводится описания способов минимизации временных автоматов. Раздел 5.4 содержит описание алгоритма восстановления параметров модели по контрпримеру в среде верификации UPPAAL. В разделе 5.5 приводится описание метода оценки наихудшего времени выполнения, основанный на верификации программ на моделях. Раздел 5.6 содержит описание интеграции с методом оценки наихудшего времени выполнения. В разделе 5.7 содержится описание методов решения задачи выбора механизмов обеспечения отказоустойчивости РВС РВ. Раздел 5.8 содержит описание интеграции со средствами планирования расписаний и синтеза архитектур.

5.1 Алгоритм трансляции иерархических временных автоматов в сети плоских временных автоматов

В данном разделе приводится алгоритм, не зависящий от средств написания UML-диаграмм и переводящий математическую модель иерархических временных автоматов в модель сети временных автоматов, используемой в средстве верификации UPPAAL. За основу взят алгоритм, предложенный в [127].

Соответствие между диаграмми состояний UML и иерархическими временными автоматами подробно описано в разделе 3.5. Для простоты записи алгоритма полагается, что иерархический временной автомат получен из диаграммы состояний UML согласно соответствию, описанному в разделе 3.5; в частности, полагается, что в каждое метасостояние вложен ровно один вход и ровно один выход. Способ обработки этой и других опущенных деталей можно посмотреть в [127].

В следующих подразделах приведены содержательное и формальное описания алгоритма трансляции иерархических временных автоматов в сеть плоских временных автоматов.

5.1.1 Содержательное описание алгоритма

Во избежание неоднозначности, возникающей из-за совпадения терминов в описании диаграммы UML и сети, в дальнейшем предполагается следующее: под диаграммой

понимается иерархический автомат, под автоматом понимается временной автомат сети; под дугой понимается переход автомата; под вершиной понимается состояние автомата.

Для единообразия обозначения переходов в диаграммах и автоматах условимся изображать переход в диаграмме из состояния s_1 в состояние s_2 записью $s_1 \xrightarrow{G,C,R} s_2$, где

- G - предусловие перехода,
- C - это либо запись $ch!$, если переход содержит действие отправления сигнала синхронизации !!ch, либо запись $ch?$, если переход содержит триггер приема сигнала синхронизации ??ch, либо служебное слово $none$, если переход выполняет внутреннее действие,
- R - это либо то действие присваивания, которое приписано указанному переходу, либо запись $\varnothing$, если переход не изменяет значения переменных и показания таймеров.

Аналогичным образом, дуга $s_1 \xrightarrow{g,a} s_2$ во временном автомате будет обозначаться записью $s_1 \xrightarrow{G,C,R} s_2$, где

- G - предусловие g данной дуги,
- C - это либо действие a, если a - это действие синхронизации, либо запись $none$ в противном случае,
- R - это либо действие a, если a - это действие присваивания (включая сброс таймеров), либо запись $\varnothing$ в противном случае.

Алгоритм трансляции диаграмм в сети автоматов представляет собой рекурсивную процедуру, которая, руководствуясь функцией вложенности состояний η, строит для каждого метасостояния s вспомогательный автомат P_s, моделирующий активацию (прохождение через вход) и деактивацию (прохождение через выход) композитного состояния s. Кроме того, вводится еще один вспомогательный автомат, обеспечивающий инициализацию построенной сети автоматов.

В начале работы процедуры трансляции каждый автомат P_s содержит только инициальную вершину $idle$; в этой вершине автомат P_s пребывает, пока метасостояние s не является активным.

Обработка and-состояний.

Если s — and-состояние, то в автомат P_s добавляются вершина $active$, обозначающая активность данного состояния, и по одной транзитной вершине $c_1, c_2, \ldots, c_n$ для каждого метасостояния, вложенного в s. Вершина $active$ соединяется с вершиной $idle$

дугой, несущей синхронизацию $exit_c?$. Далее в автомат P_s добавляется дуга $idle \xrightarrow{true,enter_s?,\varnothing} c_2$, а также дуги $c_i \xrightarrow{true,enter_{bi}?,\varnothing} c_{i+1}$, где $1 \le i \le n$, $c_{n+1} = active$, а символы bi - это имена всех метасостояний, вложенных в S.

Обработка xor-состояний.

Если S — xor-состояние, то его обработка происходит следующим образом. Для каждого простого состояния и каждого метасостояния b, вложенного в S, в автомате P_s заводится вершина b_{active}, обозначающая активность состояния b и помеченная тем же инвариантом, что и состояние b. Если состояние b является метасостоянием, то дополнительно заводится транзитная вершина b_{aux}, соединяющаяся с состоянием b_{active} дугой, помеченной действием синхронизации $enter_b!$.

Каждый переход $b_1 \xrightarrow{G,C,R} b_2$, где b_1, b_2 - состояния, вложенные в композитное состояние S, порождает дугу $u_1 \xrightarrow{G,C,R} u_2$ в автомате P_s, где:

- $u_1 = idle$, если b_1 - вход;
- $u_2 = idle$, если b_2 - выход и $u_1 = b_{1\_active}$, если при этом b_1 - метасостояние
- $u_1 = b_{1\_active}$, если b_1 - простое состояние;
- $u_2 = b_{2\_active}$, если b_2 - простое состояние;
- $u_2 = b_{2\_aux}$, если b_2 - метасостояние.

Если b_1 является метасостоянием и b_2 - либо простым состоянием, либо метасостоянием, то вместо одной дуги порождается множество дополнительных вершин и дуг, обеспечивающих корректную деактивацию компоненты b_1. Эти вершины и дуги добавляются после того, как завершится трансляция метасостояния b_1.

Подробно этот случай рассмотрен далее.

Деактивация состояний.

Для обеспечения корректной деактивации при совершении перехода из метасостояния в сеть добавляются служебные булевы переменные, вершины и дуги.

Для каждого простого состояния b_2, из которого есть переход в выход, в сети заводится глобальная *деактивационная* булева переменная x_b, обозначающая готовность выполнить этот переход и тем самым деактивировать объемлющее состояние. Ко всем дугам,

входящим в вершину b_{active}, добавляется присваивание $x_b := true$, ко всем дугам, исходящим из вершины b_{active}, - присваивание $x_b := false$.

Рассмотрим переход $b \xrightarrow{G,C,R} b'$, где состояния b и b' вложены в xor-состояние s. В автомате P_s данному переходу ставится в соответствие множество последовательностей транзитных вершин, соединенных дугами по порядку. Каждая последовательность служит для деактивации состояния b посредством одновременного выхода из некоторого множества простых состояний с последующей деактивацией объемлющих метасостояний. Первая дуга каждой последовательности помечена предусловием $G \wedge G'$, где G' - конъюнкция элементарных условий вида $x_b := true$ для деактивационных переменных x_b, отвечающих описанным простым состояниям. Последняя дуга последовательности помечена действием R. Кроме того, дуги последовательности помечены действиями синхронизации вида $exit_b?$ для всех деактивируемых метасостояний. Опишем добавляемые последовательности вершин и дуг более строго.

Рассмотрим дерево вложенности состояний T, корнем которого является состояние b, а листьями — простые состояния и псевдосостояния. Исключим из дерева вложенности все псевдосостояния. Также исключим все простые состояния, из которых не исходит дуг в выход, и все композитные состояния, ставшие после этого листьями (т.е. все состояния, заведомо не участвующие в деактивации состояния b).

Выделим все множества листьев, одновременный переход из которых в выходы деактивирует рассматриваемое метасостояние b. Для этого опишем понятия выходного и деактивационного множеств. Рассмотрим некоторое множество L листьев дерева T. Будем называть *активными для множества* L вершины дерева T, лежащие на простых путях, соединяющих листья из множества L с корнем дерева. Множество L назовем *выходным*, если выполнены следующие условия:

- если and-состояние является активным для множества L, то все его потомки в дереве T также активны для L;
- если xor-состояние активно для множества L, то ровно один его потомок активен для L.

Множество вершин дерева, активных для выходного множества L и не вляющихся простыми состояниями, назовем *деактивационным множеством* и обозначим записью D_L. Индукцией по глубине дерева можно показать, что семейство выходных множеств - это в точности совокупность всех тех множеств простых состояний, которые обладают

следующим свойством: деактивации всех состояний любого из выходных множеств достаточно для деактивации рассматриваемого метасостояния b. В свою очередь, D_L - это множество тех метасостояний, деактивация которых будет последовательно осуществляться по мере того как все простые состояния из множества L будут утрачивать активность.

Для каждого выходного множества L делается следующее. В автомат P_s для каждого состояния b_i из множества D_L вводится вершина c_i. Введенные таким образом вершины $c_1, c_2, \ldots, c_n$ соединяются дугами. В автомат P_s добавляется дуга $b_{active} \xrightarrow{G \wedge G', C, \varnothing} c_1$, где G' - конъюнкция элементарных условий вида $x_b = true$ для всех деактивационных переменных выходного множества L. Кроме того, добавляются дуги $c_i \xrightarrow{true, exit_{b\_i}!, \varnothing} c_{i+1}$, где $c_{n+1} = b'_{active}$, если b' - простое состояние, и $c_{n+1} = b'_{aux}$, если b' - метасостояние, и b_i суть все элементы множества D_L. Затем к дуге, исходящей из вершины c_m, добавляется действие R.

Обеспечение инициализации сети.

Для корректной инициализации сети в нее добавляются служебный автомат $Launch$ и служебные дуги в уже построенные автоматы. Автомат $Launch$ представляет собой множество последовательно соединенных вершин, среди которых все, кроме последней, являются транзитными. Число транзитных вершин в автомате $Launch$ совпадает с числом начальных метасостояний. Дуги, исходящие из транзитных вершин, помечены специальными действиями синхронизации вида $init_s!$ для каждого начального метасостояния S.

В автомат, соответствующий начальному and-состоянию S, добавляется дуга из вершины $idle$ в вершину $active$. В автомат, соответствующий начальному xor-состоянию S, добавляется дуга из вершины $idle$ в вершину b_{active}, если из начального состояния, вложенного в S, ведет дуга в простое состояние b, или в вершину b_{aux}, если из данного начального состояния ведет дуга в метасостояние. Во всех указанных случаях добавленная дуга несет действие синхронизации $init_s?$.

5.1.2 Формальное описание алгоритма

В данном подразделе приведено формальное описание алгоритма трансляции иерархических временных автоматов в сеть плоских временных автоматов в виде псевдокода в стиле языков C и Pascal с привлечением операций над множествами.

Алгоритм принимает на вход корректный иерархический временной автомат

$$HTA = (S, S0, \eta, Type, Var, Clock, BChan, PChan, Inv, T).$$

В процессе работы алгоритмом формируется система плоских временных автоматов

$$M = (A, Var, Clock, BChan, PChan, Urg),$$

где

- Var, Clock – множество переменных и таймеров сети соответственно,
- BChan, PChan – множество каналов сети типа точка-точка и широковещательных соответственно,
- $Urg : BChan \cup PChan \rightarrow \{0, 1\}$ – разметка каналов флагами срочности.
- A – вектор процессов сети.

Компоненты сети в псевдокоде считаются полями объекта M. Также считается, что процесс имеет следующие поля:

- L – множество вершин,
- $Type : L \rightarrow \{o, c\}$ – разметка вершин типами (обычные, транзитные),
- $Inv : L \rightarrow Invariant$ – разметка вершин инвариантами,
- $T \subseteq L \times Guard \times Sync \times Reset \times L$ – множество переходов процесса,
- l0 – инициальная вершина.

Все разметки хранятся в виде множества пар (вершина-значение). Псевдокод алгоритма трансляции приведен далее.

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

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

//Добавление канала типа точка-точка
procedure add_pchan(name, t, urg)
begin
  M.PChan := M.PChan ∪ { name };
  M.Urg := M.Urg ∪ { (name, urg) };
end;
```

```
//Добавление широковещательного канала
procedure add_bchan(name, t, urg)
begin
  M.BChan := M.BChan ∪ { name };
  M.Urg := M.Urg ∪ { (name, urg) };
end;

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

// Добавление перехода
procedure add_transition(var P, source, target, g, syn, r)
begin
  P.T := P.T ∪ { (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)
begin
  if c in PChan then
    t := p;
  elseif c in BChan then
    t := b;
  else
    t := ⊥;
  fi
  switch (urg, syn, t)
  of
  (false, none, ⊥),
    add_transition(P, source, target, g, none, r);
  (false, ch!, p):
    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?, p):
    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, 'urg[ch]'?, r);
  (true, none, ⊥),
    add_transition(P, source, target, g, 'Hurry'!, r);
  (true, ch!, p):
    add_transition(P, source, target, g, 'urg[ch]'!, r);
    add_transition(P, source, target, g, urg_sync(ch!), r);
  (true, ch!, b):
    add_transition(P, source, target, g, 'urg[ch]'!, r);
  (true, ch?, p):
    add_transition(P, source, target, g, 'urg[ch]'?, 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, 'urg[ch]'?, r);
  fo
end;

// Добавление процесса Hurry
procedure add_hurry()
begin
```

```
  add_pchan('Hurry', 1);
  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(root)
begin
  S0 := S0 ∩ { B in S | XOR(B) or AND(B) };
  let S0 = { B_i | 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_pchan('init[B_i]', 0);
    add_transition(P, L_i, L_(i+1), true, 'init[B_i]'!, {});
  od
end;

// Построение множества деревьев выходов
function make_tree_set(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(v_i) | i = 1..m;
    if AND(η^-1(B)) then
      forall (s_1, s_2, ..., s_m) such that s_i in temp_set_i, i = 1..m
      do
        result := result ∪ { {B} ∪ ∪{i=1..m} s_i };
      od
    fi
    if XOR(η^-1(B)) then
      forall s such that
        exists i : i = 1..m and s in temp_set_i
      do
        result := result ∪ { {B} ∪ s };
      od
    fi
  fi
  return result;
end;

// Добавление синхронизаций для корректного выхода из вложенных метасостояний
procedure add_exit_cascades(s, var P)
begin
  forall EX in η(η(s)) such that exists (EX, (g, syn, r, urg), B) in T : BASIC(B) or ENTRY(B)
  do
    Tree_set := make_tree_set(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 ∪ { 'exit_ready[E]' | 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
        add_sync_transition(P,
```

```
                        'active[s,η^-1(EX)]',
                        'exit_cascade[t,k,1]',
                        g and or{EX in t_leaves}('exit_ready[EX] = 1'),
                        syn,
                        {},
                        urg,);
        for i := 1..m-1
        do
          add_transition(P,
                       'exit_cascade[t,k,i]',
                       'exit_cascade[t,k,i+1]',
                       true,
                       'exit[η^-1(v_i)]'!,
                       {});
        od
        if BASIC(B) then
          add_transition(P, 'exit_cascade[t,k,p]', 'active[s,B]', true, 'exit[η^-1(v_m)]'!, r);
        fi
        if ENTRY(B) then
          add_transition(P, 'exit_cascade[t,k,p]', 'aux[s,η^-1(B),B]', true, 'exit[η^-1(v_m)]'!, r);
        fi
      od
    od
  od
end;

// Добавление предусловий для корректного выхода из метасостояний
procedure add_exit_assignments(s, var P)
begin
  forall EX in η(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 ('active[s,B]', (g, syn, r), v) in P.T such that
        not ( exists u :  v = 'active[s,u]' and BASIC(u) and u in In_set )
      do
        Vars := Vars ∪ { 'exit_ready[EX]' };
        r := r ∪ 'exit_ready[EX] := 0';
      od
      forall (v, (g, syn, r), 'active[s,B]') in P.T such that
        not ( exists u :  v = 'active[s,u]' and BASIC(u) and u in In_set )
      do
        Vars := Vars ∪ { 'exit_ready[EX]' };
        r := r ∪ 'exit_ready[EX] := 1';
      od
    od
  od
end;

// Обработка состояния типа Xor
procedure process_xor(s, var P)
begin
  forall B in η(s) such that XOR(B) or AND(B) or BASIC(B)
  do
    add_location(P, 'active[s,B]', o, Inv(B));
    if B in S0 then
      add_transition('idle[s]', 'active[s,B]', true, 'init[s]'?, {});
    fi
    if XOR(B) or AND(B) then
      forall E in η(B) such that ENTRY(E)
      do
        add_pchan('enter[s,B,E]', 0);
        add_location(P, 'aux[s,B,E]', c, true);
        add_transition(P, 'aux[s,B,E]', 'active[s,B]', true, 'enter[s,B,E]'!, {});
      od
    fi
  od
  forall (B_1, (g, syn, r, urg), B_2) in T such that B_1, B_2 in η(s) and BASIC(B_1) and BASIC(B_2)
  do
    add_sync_transition(P, 'active[s,B_1]', 'active[s,B_2]', g, syn, r, urg);
  od
```

```
  forall (B, (g, syn, r, urg), E) in T such that B, η^-1(E) in η(s) and BASIC(B) and ENTRY(E)
  do
    add_sync_transition(P, 'active[s,B]', 'aux[s,η^-1(E),E]', g, syn, r, urg);
  od
  forall (E, (true, none, r, false), B) in T such that E, B in η(s) and ENTRY(E) and BASIC(B)
  do
    add_pchan('enter[η^-1(s),s,E]', 0);
    add_transition(P, 'idle[s]', 'active[s,B]', true, 'enter[η^-1(s),s,E]'?, r);
  od
  forall (E_1, (true, none, r, false), E_2) in T such that
    E_1, η^-1(E_2) in η(s) and ENTRY(E_1) and ENTRY(E_2)
  do
    add_pchan('enter[η^-1(s),s,E_1]', 0);
    add_transition(P, 'idle[s]', 'aux[s,η^-1(E_2),E_2]', true, 'enter[η^-1(s),s,E_1]'?, r);
  od
  forall (B, (g, none, {}, false), EX) in T such that B, EX in η(s) and BASIC(B) and EXIT(EX)
  do
    add_pchan('exit[s]', 0);
    add_transition(P, 'active[s,B]', 'idle[s]', 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 η(η(s)) and EX_2 in η(s)
  do
    add_pchan('exit[s]', 0);
    add_transition(P, 'active[s,η^-1(EX_1)]', 'idle[s]', true, 'exit[s]'?, {});
  od
  add_exit_cascades(s, P);
  add_exit_assignments(s, P);
end;

// Обработка состояния типа And
procedure process_and(s, var P)
begin
  let η(s) ∩ {B in S | BASIC(B) or AND(B) or XOR(B)} = { B_i | i = 1..n };
  add_location(P, 'active[s]', o, Inv(s));
  if s in S0 then
    add_transition('idle[s]', 'active[s]', true, 'init[s]'?, {});
  fi
  forall E in η(s) such that ENTRY(E)
  do
    let (E, (g, s, r, u), E_i) in T such that E_i in η(B_i) and i = 1..n;
    for i := 1..n
    do
      add_location(P, 'enter_loc[B_i,E]', c, true);
    od
    add_pchan('enter[η^-1(s),s,E]', 0);
    add_transition(P, 'idle[s]', 'enter_loc[B_1,E]', true, 'enter[η^-1(s),s,E]'?, {});
    for i := 1..n-1
    do
      add_transition(P, 'enter_loc[B_i,E]', 'enter_loc[B_(i+1),E]', true, 'enter[s,B_i,E_i]'!, {});
    od
    add_transition(P,
                   'enter_loc[B_n,E]',
                   'active[s]',
                   true,
                   'enter[s,B_n,E_n]'!,
                   ∪{(E, (g, syn, r), B) in T}(r));
  od
  add_pchan(M, 'exit[s]', c);
  add_transition(P, 'active[s]', 'idle[s]', true, 'exit[s]'?, {});
end;

// Входная точка программы
begin
  let root = root_s such that root_s in S and η^-1(root_s) = {};
  M.A := {};
  M.Var := Var;
  M.Clock := Clock;
  M.BChan := {};
  M.PChan := {};
  forall ch in PChan
  do
```

```
    add_pchan(ch, 0);
    add_pchan('urg[ch]', 1);
    add_pchan('recv[ch]', 1);
    add_pchan('send[ch]', 1);
  od
  forall ch in BChan
  do
    add_bchan(ch, 0);
    add_bchan('urg[ch]', 1);
  od
  add_global_kickoff(root);
  add_hurry();
  To_process := {};
  To_process.push_back(root);
  while not_empty(To_process)
  do
    s := pop_front(To_process);
    forall B such that B in η(s) and (XOR(B) or AND(B))
    do
      push(To_process, B);
    od
    P := empty_process;
    add_location(P, 'idle[s]', o, true);
    set_init(P, 'idle[s]');
    if XOR(s) then process_xor(s, P);
    if AND(s) then process_and(s, P);
    A.push_back(P);
  od
end.
```

5.1.3 Оценка сложности результирующей сети

При оценке сложности описания РВС РВ, как правило, не достаточно учитывать только полученное в описании число состояний управления. Это связано с тем, что конфигурация системы определяется не только ими, но и показаниями таймеров. Таким образом, система, содержащая всего один таймер, принимающий действительные значения, потенциально содержит континуум различных конфигураций. Кроме того, немалая часть совокупных состояний управления компонентов системы может оказаться недостижимой из-за специфики системы, связанной с синхронизацией компонентов и преобразованием данных, представляющим собой неявное преобразование состояний управления, часто в весьма ограниченном относительно потенциальных возможностей диапазоне. Однако и число явных состояний управления не следует игнорировать при оценке сложности: время автоматического анализа системы зависит и от размера представления этой системы.

Нами были рассмотрены оценки сложности сети временных автоматов, получаемой в результате трансляции согласно описанному алгоритму, представляющие собой отношение объектов исходного иерархического временного автомата и результирующей сети, имеющих близкую семантику, а именно:

- таймеров,
- каналов,
- переменных,
- состояний управления,

- переходов и
- отношение числа процессов сети к числу состояний исходного автомата.

Анализ алгоритма позволяет получить следующие оценки представленных характеристик.

Множество таймеров сети, очевидно, совпадает с множеством таймеров исходного автомата.

Каналы результирующей сети можно подразделить на три класса: транслированные широковещательные каналы, транслированные каналы типа точка-точка и служебные каналы. Сеть содержит в два раза больше каналов первого класса и в четыре раза больше каналов второго класса, чем исходный автомат. Служебный канал создается для каждого входа и каждого выхода, содержащегося в исходном иерархическом автомате. Отсюда может быть получена верхняя оценка NC = O(N + C) числа каналов сети; здесь N – число состояний управления иерархического автомата, C – число содержащихся в нем каналов.

Переменные сети состоят из переменных автомата и дополнительных служебных переменных, добавляемых для обеспечения инициализации сети. При этом служебные переменные сети заводятся лишь для некоторых выходов иерархического автомата. Верхняя оценка числа переменных сети, таким образом, имеет вид NV = V + O(N), где V – число переменных исходного автомата.

Несложные рассуждения о структуре процессов, порождаемых метасостояниями автомата, приводят к следующим оценкам числа состояний управления (NS), числа переходов (NT) и числа процессов (NP) результирующей сети:

- NS = O(E * N) + S_DEACT,
- NT = O(T + E * N) + S_DEACT,
- NP = O(N).

Здесь E – максимум по всем метасостояниям автомата числа непосредственно вложенных в метасостояние входов; T – число переходов автомата; S_DEACT – число состояний и переходов, добавляемых в сеть для обеспечения деинициализации компонентов системы. Текущее описание алгоритма приводит к оценке S_DEACT = $O(2^N)$ в худшем случае, что явно показывает наличие экспоненциального взрыва числа состояний управления. Одно из возможных направлений исследования алгоритма состоит в преодолении экспоненциального взрыва числа управляющих состояний, то есть в обеспечении более «компактной» деактивации компонентов.

Стоит отметить, что для всех иерархических автоматов, получаемых из диаграмм состояний согласно описанию, приведенному в разделе 3.5, выполняется равенство E = 1, что приводит к оценкам, линейным относительно числа состояний исходного иерархического временного

автомата, если не считать слагаемого S_DEACT. Более того, константы, подразумеваемые записью O(.) в линейных оценках, в этом случае оказываются невысокими, а именно не превосходящими числа 4.

5.2 Корректность алгоритма трансляции иерархических временных автоматов в плоские временные автоматы

Чтобы обосновать корректность алгоритма трансляции автоматов в сети, достаточно для некоторого класса темпоральных формул, допустимых в рамках средства UPPAAL, показать их равновыполнимость на множествах вычислений автомата и соответствующей ему сети. Для этого введем понятие размеченной системы переходов, удобное для описания поведения автомата и сети. Затем выделим класс формул, проверка которых обеспечивается средством UPPAAL, и покажем, что оценка истинности этих формул одинакова для систем переходов автомата и соответствующей ему сети. Для этого воспользуемся отношением эквивалентности по прореживанию [128] и убедимся в том, что системы переходов, описывающие поведение автомата и сети, получаемой в результате работы алгоритма, эквивалентны.

Поведение автомата и сети может быть описано с помощью размеченных систем переходов.

Размеченная система переходов (LTS) над множеством логических переменных A - это система $M = (S, s_0, L, R)$, где

15. S - некоторое множество состояний, содержащее инициальное состояние s_0 ;
16. $L : S \to 2^A$ - разметка состояний;
17. $R \subseteq S \times S$ - отношение переходов.

Путем в LTS M называется максимальная последовательность состояний $tr = s_0, s_1, \ldots, s_n, \ldots$, для которой верно соотношение $s_i R s_{i+1}$ для всех i, $i \geq 0$.

LTS $M^T = (S^T, s^T_0, L^T, R^T)$, описывающая поведение системы T (автомата или сети), строится над множеством элементарных ограничений данных и таймеров автомата. Заметим, что множество таймеров автомата и сети совпадают и множество переменных автомата вложено в множество переменных сети.

При этом S^T есть множество конфигураций автомата (или сети) T, s^T_0 - начальная конфигурация T, каждая конфигурация помечена множеством ограничений, истинных в ней, и переход sRs’ допустим в том и только в том случае, если s’ может быть получена из s согласно одному из правил в описании семантики T. В этом случае множество вычислений системы T совпадает с множеством путей LTS M^T.

Как уже было отмечено, в качестве языка запроса в UPPAAL используется подмножество формул логики TCTL (timed computational tree logic), которое порождается грамматикой:

```
Φ ::= AG φ | AF φ | EG φ | EF φ | φ --> φ;
φ ::= p | ¬ φ | φ ∨ φ,
```

где p - элементарное ограничение данных и таймеров или специальный предикат deadlock. Формула указанного вида называется допустимой, если она не содержит символа deadlock или имеет вид: AG ¬deadlock, AF deadlock, EG ¬deadlock, EF deadlock, φ --> deadlock, где φ не содержит предикат deadlock.

Выполнимость формулы Φ на LTS M (обозначение M |= Φ) определена обычным для логики CTL* образом [129]. Воспользовавшись соотношениями

- s |= deadlock ⇔ ¬∃s’ : (sRs’);
- M |= EG φ ⇔ M |≠ AF ¬φ;
- M |= EF φ ⇔ M |≠ AG ¬φ;
- M |= φ --> ψ ⇔ M |= AG(φ -> Fψ),

легко заметить, что рассматриваемый фрагмент TCTL тесно взаимосвязан с темпоральной логикой LTL(-X): каковы бы ни были LTS M_1 и M_2, если на них выполняется одно и то же множество формул LTL(-X), то на LTS M_1 и M_2 также выполняется одно и то же множество допустимых формул TCTL. Это наблюдение играет ключевую роль в обосновании корректности предложенного алгоритма трансляции.

Обозначим записью Trans(H) сеть временных автоматов, получаемую в результате применения описанного алгоритма трансляции к иерархическому автомату H.

Корректность алгоритма трансляции подразумевает справедливость соотношения M^H |= Φ ⇔ $M^{Trans(H)}$ |= Φ для любого иерархического автомата H и любой допустимой формулы Φ. Для обоснования корректности введем на множестве путей LTS отношение эквивалентности по прореживанию.

Рассмотрим пару путей tr = s_0, s_1, …, s_n, … и tr’ = $s’_0$, $s’_1$, …, $s’_n$, … в LTS M = (S, s_0, L, R) и M’ = (S’, $s’_0$, L’, R’) соответственно, причем либо оба пути конечные, либо обе бесконечные. Тогда пути tr, tr’*эквивалентны по прореживанию*, если найдутся такие возрастающие последовательности целых чисел α = i_0, i_1, … и β = j_0, j_1, … одинаковой длины, что:

13. $i_0 = j_0 = 0$;
14. если пути tr и tr’ конечны и имеют длины n и m, то последовательности α и β также конечны и завершаются числами n+1 и m+1 соответственно;

15. для любых i, j, k, $1 \le k \le |\alpha| = |\beta|$, $i_k \le i < i_{k+1}$, $j_k \le j < j_{k+1}$, справедливо равенство $L(s_i) = L'(s'_j)$.

Иначе говоря, пути эквивалентны по прореживанию, если их можно разбить на блоки так, чтобы значение функции разметки на блоках с одинаковыми номерами было одинаково.

Две LTS M_1 и M_2 назовем эквивалентными (и запишем этот факт как $M_1 \sim M_2$), если для любого пути в LTS M_i найдется эквивалентный ему по прореживанию путь в LTS M_{3-i}, $i \in \{1, 2\}$. Как известно [129], для любых эквивалентных LTS M_1 и M_2 и любой формулы **Ф** логики LTL(-X) верно соотношение $M_1 \models \Phi \Leftrightarrow M_2 \models \Phi$.

Это соотношение верно и для допустимых формул, содержащих предикат deadlock, т.к. все такие формулы можно переписать в виде сохраняемых эквивалентностью по прореживанию требований конечности или бесконечности путей. Из проведенных рассуждений вытекает следующая теорема.

Теорема 1. Для любых эквивалентных LTS M_1 и M_2 и любой допустимой формулы **Ф** верно соотношение $M_1 \models \Phi \Leftrightarrow M_2 \models \Phi$.

Для завершения обоснования корректности алгоритма трансляции осталось показать, что LTS M^H и $M^{Trans(H)}$ эквивалентны для любого автомата H.

Рассмотрим иерархический автомат $H=(S,S_0,\eta,Type,Var,Clock,BChan,PChan,Inv,T)$ и соответствующую ему сеть $Trans(H) = (Var \cup Var_{add}, Clock, BChan, PChan, \mathbf{A})$. Значения всех переменных множества Var_{add} однозначно определяются значениями переменных множества Var, таймеров множества Clock и вектором активных вершин сети.

Поэтому для удобства переменные Var_{add} в записи конфигураций будем опускать. Сопоставим конфигурации $c = (\rho, \mu, \nu)$ иерархического автомата H конфигурацию $\xi(c) = (\mathbf{l}, \mu, \nu)$ сети и построим для произвольной трассы $tr = s_0, s_1, \ldots, s_n, \ldots$ иерархического автомата H эквивалентную по прореживанию трассу $tr' = s'_0, s'_1, \ldots, s'_n, \ldots$ сети автоматов Trans(H).

Из описания процесса Launch, приоритета в выходе из вершин типа c и правила для перехода с сигналом точка-точка в описании семантики сети видно, что любая трасса сети начинается одной и той же последовательностью конфигураций $s'_1, s'_2, \ldots, s'_k$ с неизменными значениями переменных и таймеров, т.е. $s'_k = \xi(s_1)$.

Предположим, что по префиксу $s_1, s_2, .., s_p$ трассы tr уже построен префикс $pr' = s'_1, s'_2, \ldots, s'_k$ трассы tr', для которого $s'_k = \xi(s_p)$, и его конфигурации можно разбить на p блоков так, что значение функций разметки на i-м блоке и в конфигурации s_i совпадают, $1 \le i \le p$.

Если |tr| = p, то в конфигурации s_p иерархического автомата H нет активных переходов. Но из описания процессов сети Trans(H) следует, что построенный префикс pr’ нельзя продолжить, и эквивалентная по прореживанию трасса tr’ = pr’ построена.

Если конфигурация s_{p+1} получается из s_p продвижением времени на константу d, то к построенному префиксу pr’ также может быть добавлена конфигурация s'_{k+1}, получающаяся из s'_k продвижением времени на константу d. Если s_{p+1} получается из s_p применением одного активного перехода $e = l'_1 \xrightarrow{g,r} l'_2$ без синхронизации, то к построенному префиксу pr’ можно добавить последовательность конфигураций, отвечающую следующим переходам в сети:

- деинициализация метасостояний, внутренних для l'_1, если l'_1 - метасостояние (первая конфигурация - по правилу обычного перехода, остальные - по правилу перехода с сигналом точка-точка);
- дуга, порождаемая переходом e, если l'_1 простое состояние (по правилу обычного перехода); затем конфигурации, получаемые при инициализации входного дерева состояния l'_2, если l'_2 - метасостояние (по правилу перехода с сигналом точка-точка).

Если s_{p+1} получается из s_p применением совокупности активных переходов, среди которых есть переход $e = l'_1 \xrightarrow{g,c!,r} l'_2$, префикс pr’ достраивается аналогичным образом с следующим уточнением: прежде всего последовательно происходит деинициализация метасостояний для каждого из переходов, затем - инициализация.

В любом случае к построенному префиксу добавляются конфигурации $s''_1, s''_2, \ldots, s''_m, s''_{m+1}, \ldots, s''_q$, для которых верно следующее: значения функций разметки на конфигурациях $s''_1, s''_2, \ldots, s''_m$ и s_p совпадают, и значения функций разметки на конфигурациях $s''_{m+1}, \ldots, s''_q$ и s_{p+1} также совпадают.

Аналогичным образом по трассе tr’ сети автоматов Trans(H) строится эквивалентная ей по прореживанию трасса tr иерархического автомата H. Здесь важно учесть, что в любой трассе сети автоматов бесконечно часто встречаются конфигурации, являющиеся ξ-образами конфигураций иерархического автомата.

Проведенные рассуждения служат обоснованиями следующих теорем.

Теорема 2. Для любого иерархического автомата H верно соотношение $M^H \sim M^{Trans(H)}$.

Теорема 3. Для любой допустимой формулы Φ и любого иерархического автомата H верно соотношение $M^H \models \Phi \Leftrightarrow M^{Trans(H)} \models \Phi$.

Таким образом, алгоритм трансляции преобразует всякую диаграмму UML (вернее, соответствующий этой диаграмме иерархический автомат) H в такую сеть конечных временных автоматов Trans(H), что для любого запроса, представленного допустимой формулой TCTL, этот запрос выполняется в модели вычислений диаграмм UML в том и только том случае, когда запрос выполняется в соответствующей сети конечных временных автоматов. Это означает, что средство верификации моделей PBC PB UPPAAL может быть использовано для корректной проверки поведения диаграмм UML при использовании предложенного нами алгоритма трансляции.

5.3 Минимизация временных автоматов

Система верификации, разработанная в рамках настоящего проекта, позволяет сводить проверку свойств вычислений распределенных систем, описание которых представлено UML-диаграммами, к задаче анализа поведения сетей временных автоматов, для решения которой применяется программно-инструментальное средство верификации UPPAAL.

Состояние вычисления временного автомата характеризуется двумя компонентами – состоянием управления автомата и конечным набором вещественных чисел, представляющих собой значения (показания) таймеров. Таким образом, в отличие от дискретного конечного автомата, временной автомат допускает бесконечное (вообще говоря, континуальное) множество состояний вычисления. Для того чтобы эффективно проверить свойства вычислений временных автоматов, множество всех состояний вычисления этих автоматов разбивается на конечное семейство регионов; каждый регион описывается двумя составляющими – состоянием управления автомата и конечной системой неравенств вида $t \le c$, $c \le t$ и $t' \le t'' + c$, задающих выпуклый многогранник (полиэдр) в многомерном вещественном пространстве показаний таймеров. На множестве регионов определяется отношение переходов, соответствующее отношению переходов анализируемого временного автомата, и в результате этого образуется конечная система размеченных переходов, которая полностью отражает всевозможные вычисления этого автомата. Именно для этой системы переходов проводится проверка темпоральных свойств временного автомата.

Этот метод верификации вычислительных систем реального времени был предложен в серии работ, посвященных вопросам повышения эффективности алгоритмов верификации конечных автоматов реального времени. В этих работах было показано, что число регионов в системе переходов, соответствующей временному автомату, может оказаться экспоненциально зависящим от количества таймеров, используемых автоматом. В связи с

этим возникает задача минимизации временных автоматов: сократить размер описания временного автомата и/или соответствующей ему системы переходов при сохранении множества вычислений, порождаемых исходным временным автоматом. Существует два основных подхода к решению этой задачи. Первый из них оставляет без изменения описание исходного автомата, но осуществляет минимизацию пространства регионов в системе переходов по ходу ее построения. Второй подход проводит оптимизацию самого временного автомата подобно тому, как это осуществляется для дискретных конечных автоматов. В данном разделе отчета рассмотрен первый из указанных подходов и описан один из возможных алгоритмов минимизации системы переходов (графа регионов), соответствующих временному автомату.

Воспользуемся определением конечного временного автомата, приведенным в предшествующих разделах отчета, для решения задачи минимизации пространства регионов в системе переходов.

Рассмотрим конечное множество вещественных переменных $X = (x_1, x_2, \ldots, x_n)$, которые будем называть *таймерами*. *Элементарным линейным ограничением* называется всякое неравенство одного из следующих видов $x \le c$, $x < c$, $c \le x$, $c < x$, $x \le x'+c$, $x < x'+c$ где x, x’ - таймеры, а c - целое число. Решением элементарного линейного ограничения является множество наборов вещественных чисел $<r_1, r_2, \ldots, r_n>$, удовлетворяющих соответствующему неравенству. Очевидно, что решением всякого линейного ограничения является некоторое полупространство n-мерного вещественного пространства. Любое конечное множество элементарных линейных ограничений называется *линейным ограничением*. Решением линейного ограничения является пересечение решений неравенств, составляющих это ограничение. Это множество, являющееся выпуклым многогранником (возможно, пустым), будем называть *временной зоной*, или просто *зоной*. Семейство всех возможных зон обозначим записью Z(n).

Временной автомат описывается четверкой $A = (S, X, s_{init}, T)$, где

- S - конечное множество *состояний управления*,
- X - конечное множество таймеров,
- s_{init} - начальное состояние управления,
- $T \subseteq S \times 2^X \times Z(n) \times S$ - конечное отношение переходов.

Каждая четверка <s, Y, z, s’> из множества T означает, что из состояния управления s в состояние управления s’ возможен переход в том случае, когда показания таймеров образуют набор, принадлежащий зоне z. При этом по окончании перехода значения всех таймеров из множества Y полагаются равными нулю (сброс таймеров). Переходы вида <s, Y, z, s’> будем обозначать записью $s \xrightarrow{Y,z} s'$.

Вычислением автомата $A = (S, X, s_{init}, T)$ называется последовательность пар

$$(s_0, \mathbf{x}_0) \Rightarrow (s_1, \mathbf{x}_1) \Rightarrow \ldots \Rightarrow (s_k, \mathbf{x}_k) \Rightarrow (s_{k+1}, \mathbf{x}_{k+1}) \Rightarrow \ldots,$$

удовлетворяющая следующим условиям: $s_0 = s_{init}$, $x_0 = <0, 0, \ldots, 0>$ и для любого k, $k \geq 0$, верно одно из двух:

- $s_k = s_{k+1}$ и $\mathbf{x}_{k+1} = \mathbf{x}_k + <\delta, \delta, \ldots, \delta>$ для некоторого вещественного числа δ, $\delta > 0$ (продвижение времени);
- существует такой переход $s_k \xrightarrow{Y,z} s_{k+1}$ из множества T, что $\mathbf{x}_k \in z$ и набор $\mathbf{x}_{k+1}$ отличается от набора $\mathbf{x}_k$ только тем, что все показания всех таймеров из множества Y в наборе $\mathbf{x}_{k+1}$ равны 0 (срабатывание перехода).

Для эффективной верификации свойств вычислений конечного автомата было введено понятие региона и на основании этого понятия введена система размеченных переходов (граф регионов).

Регионом называется всякое множество F, $F \subseteq S \times R^n$. Регион $\{<s,\mathbf{x}> \mid \mathbf{x} \in Z\}$, где Z - некоторая зона, обозначается записью (s, Z). На множестве регионов вводится следующее *отношение регионального продвижения* $\rightarrow$. Пусть F, F' - регионы и $<s, \mathbf{x}> \in F$. Тогда отношение продвижения $<s, \mathbf{x}> \rightarrow_F F'$ имеет место в том случае, когда выполняется одно из следующих двух условий:

- существует такое вещественное число δ, $\delta > 0$, для которого выполняются два включения $<s, \mathbf{x} + \delta> \in F$ и $\{<s, \mathbf{x} + \delta'> \mid 0 \leq \delta' \leq \delta\} \subseteq F \cup F'$ (продвижение времени),
- существует такое вещественное число δ, $\delta \geq 0$, и такой набор $<s', \mathbf{x} + \delta> \in F'$, для которых выполняются два включения $\{<s, \mathbf{x} + \delta'> \mid 0 \leq \delta' \leq \delta\} \subseteq F$ и $<s, \mathbf{x}> \Rightarrow <s', \mathbf{x} + \delta>$ (продвижение управления).

Разбиение пространства $S \times R^n$ на регионы называется *стабильным* в том случае, когда для любой пары регионов F, F' и для любого набора $<s, \mathbf{x}> \in F$, если выполняется отношение регионального продвижения $<s, \mathbf{x}> \rightarrow_F F'$, то отношение регионального продвижения того же типа $<s', \mathbf{x}'> \rightarrow_F F'$ выполняется и для любого другого набора $<s', \mathbf{x}'> \in F$.

Графом регионов, соответствующим временному автомату A и начальному разбиению $Q_0 = \{<s, R^n> \mid s \in S\}$ пространства $S \times R^n$ называется граф $GR(A, Q_0)$, вершинами которого являются регионы, образующие некоторое стабильное разбиение Q пространства $S \times R^n$, удовлетворяющее следующему требованию: для любого региона F, $F \in Q$, существует регион F', $F' \in Q_0$, для которого верно включение $F \subseteq F'$. В графе $GR(A, Q_0)$ из региона F_1 в

регион F_2 ведет дуга в том и только том случае, когда для некоторого набора $<s, \mathbf{x}> \in F$ выполняется отношение регионального продвижения $<s, \mathbf{x}> \rightarrow_{F1} F_2$. Ранее в нескольких работах, посвященных вопросам применения временных автоматов для верификации РВС РВ, было установлено, что для любого временного автомата A и любого начального разбиения Q_0 пространства $S \times R^n$ существует конечный граф регионов $GR(A, Q_0)$. Задача минимизации системы переходов, соответствующих временному автомату A состоит в построении графа регионов как можно меньшего размера.

Рассмотрим следующую схему алгоритма, который применяется для решения задачи минимизации числа состояний в системе переходов (недетерминированном конечном автомате).

Пусть задана некоторая система переходов $B = (S, s_0, \rightarrow)$ с множеством состояний S, начальным состоянием s_0 и отношением переходов $\rightarrow \subseteq S \times S$. Для состояния S и множества состояний X запись $s \rightarrow X$ будет обозначать выполнимость отношения переходов $s \rightarrow s'$ для некоторого состояния s', $s' \in X$. Для произвольного разбиения ρ множества состояний S на классы состояний класс X, $X \in \rho$, называется стабильным, если для любого состояния S из класса X выполнимость отношения $s \rightarrow Y$ влечет выполнимость отношения $s' \rightarrow Y$ для любого состояния s' из класса X. Разбиение ρ называется бисимуляцией, если каждый класс этого разбиения стабилен.

Известно, что выполнимость формул темпоральных логик ветвящегося времени (и в том числе логики TCTL) инвариантна относительно отношения бисимуляции. Это означает, что при решении задачи минимизации систем переходов достаточно для заданной системы переходов вычислить максимальное отношение бисимуляции этой системы и затем построить минимальную систему переходов состояниями которой являются классы эквивалентности пространства состояний по максимальному отношению бисимуляции. Таким образом, задача минимизации системы переходов сводится к задаче вычисления максимального отношения бисимуляции в пространстве состояний этой системы. Последняя задача имеет следующую простую схему решения.

Рассмотрим систему переходов $B = (S, s_0, \rightarrow)$, для которой введем следующие три функции:

- функция split(X, ρ) вычисляет для класса X разбиения ρ минимальное расщепление класса X на стабильные относительно разбиения ρ подклассы;
- функция succ(X, ρ) вычисляет множество $\{Y \mid \exists x(x \in X \wedge x \rightarrow Y)\}$ всех классов, хотя бы одно из состояний которых достижимо из некоторого состояния класса X;

- функция pred(X, ρ) вычисляет множество $\{Y \mid \exists y(y \in Y \wedge y \rightarrow X)\}$ всех классов, хотя бы из одного состояния которых достижимо некоторое состояния класса X.

Тогда алгоритм, вычисляющий максимальную бисимуляцию ρ, являющуюся сужением заданного начального разбиения ρ_0, имеет следующий вид (здесь α - множество классов, достижимых из класса, содержащего начальное состояние системы переходов, а σ - множество классов текущего разбиения ρ, которые стабильны относительно ρ) :

```
ρ := ρ₀; α := {[s₀]ρ}; σ := ∅;
while α != σ do
      let x ∈ α \ σ;
      if (split(X, ρ) = X)
      then {σ = σ ∪ {X}; α := α ∪ succ(X, ρ)}
      else
      {
            α := α ∪ succ(X, ρ);
            if ∃Y (Y ∈ α ∧ s₀ ∈ Y) then α := α ∪ Y;
            σ := σ \ pred(X, ρ); ρ := (ρ \ {X}) ∪ α;
      }
      fi
od
```

Применение данного алгоритма к решению задачи минимизации графа регионов требует уточнения функций split(X, ρ), succ(X, ρ) и pred(X, ρ), поскольку в отличие от конечных систем переходов граф регионов допускает бесконечное множество регионов. Для того чтобы преодолеет эту трудность, предлагается ввести ряд специальных операций на множестве регионов.

Для произвольной пары зон Z_1, Z_2 обозначим записью

- $Z_1 \backslash Z_2$ такое множество попарно непересекающихся зон, что объединение $\{Z_2\} \cup Z_1 \backslash Z_2$ образует разбиение зоны Z_1. Тогда положим $Z_1 \cup Z_2 = \{Z_1 \cap Z_2\} \cup Z_1 \backslash Z_2 \cup Z_2 \backslash Z_1$;
- $Z_1 \Uparrow Z_2$ множество векторов (наборов показаний таймеров) **x**, которые для некоторого положительного вещественного числа δ удовлетворяют двум условиям:
  - $\mathbf{x} + <\delta, \delta, \ldots, \delta> \in Z_2$,
  - для любого δ', $0 \le \delta' \le \delta$, верно включение $\mathbf{x} + <\delta', \delta', \ldots, \delta'> \in Z_1$.

Тогда в терминах этих операций функции split(X, ρ), succ(X, ρ) и pred(X, ρ) могут быть определены следующим образом.

Вначале введем функцию расщепления для пары регионов:

- split((s, Z), (s', Z')) = (s, Z) $\cup \cup_t$ (s, Z$\Uparrow$(Z $\cap$ z $\cap$ a^{-1}(Z'))) для всякого состояния управления s ≠ s' (здесь $\cup_t$ – объединение по всем переходам вида $s \xrightarrow{z,a} s'$);
- split((s, Z), (s, Z')) = (s, Z) $\cup$ (s, Z$\Uparrow$Z') $\cup \cup_t$ (s, Z$\Uparrow$(Z $\cap$ z $\cap$ a^{-1}(Z'))).

На основании этой функции можно определить функцию расщепления региона относительно заданного разбиения ρ:

split((s, Z), ρ) = $\cup_{(s', Z') \in \rho}$ split((s, Z), (s', Z')).

Функции вычисления предшествующих и последующих регионов определяются так:

- pred((s, Z), ρ) = {(s, Z') | Z'$\Uparrow$Z ≠ ∅} $\cup_t$ {(s', Z') ∈ ρ | a(Z'∩z) ∩ Z ≠ ∅};
- succ((s, Z), ρ) = {(s, Z') | Z$\Uparrow$Z' ≠ ∅} $\cup_t$ {(s', Z') ∈ ρ | a(Z∩z) ∩ Z' ≠ ∅}.

Таким образом, из описанной выше общей схемы вычисления максимальной бисимуляции в конечной системе переходов получаем алгоритм минимизации системы переходов (графа регионов), соответствующей временному автомату. Эффективность этого алгоритма существенно зависит от эффективности реализации процедур, вычисляющих функции split(X, ρ), succ(X, ρ) и pred(X, ρ), которые, в свою очередь, зависят от методов вычисления операции $Z_1 \backslash Z_2$ и $Z_1 \Uparrow Z_2$. Исследование этого вопроса планируется провести на следующем этапе выполнения проекта.

В данном разделе отчета описан алгоритм минимизации системы переходов для простейшего временного автомата. Однако в системе UPPAAL в качестве моделей систем реального времени используются сети взаимодействующих временных автоматов, в которых используются операции синхронизации. Описанный выше метод минимизации может быть применен и для сетей временных автоматов. В этом случае регионами будут являться наборы вида $<s_1, s_2, \ldots, s_n, \nu_1, \nu_2, \ldots, \nu_k, Z>$, где $s_1, s_2, \ldots, s_n$ - состояния управления отдельных временных автоматов, образующих рассматриваемую сеть, $\nu_1, \nu_2, \ldots, \nu_k$ - значения предметных переменных, которыми оперируют временные автоматы сети, а Z - зона. Для сетей временных автоматов соответствующим образом модифицируется система переходов (граф регионов).

5.4 Алгоритм восстановления параметров модели по контрпримеру в UPPAAL

5.4.1 Введение

Задача построения и анализа контрпримеров неизбежно возникает при верификации моделей вычислительных систем. В том случае, если некоторое проверяемое свойство поведения системы не выполняется, необходимо установить причину этого явления. Для

этого необходимо отыскать хотя бы одно из тех вычислений системы, которые не удовлетворяют проверяемому свойству. Большинство средств верификации, включая UPPAAL, снабжено процедурами построения контрпримеров. Однако описания этих контпримеров проводятся на языке входной модели средства верификации. В случае использования UPPAAL таким языком является язык сетей временных автоматов. Поскольку в разрабатываемой нами системе проектирования РВС РВ модели представляются в виде UML диаграмм, которые транслируются в сети временных автоматов, возникла необходимость в отображении трасс вычислений в сетях временных автоматов в трассы вычислений в UML диаграммах, то есть необходимо по трассе, полученной в средстве верификации, восстановить последовательность шагов исходной программы. Поскольку трансляция из UML в UPPAAL делается автоматически, преобразование трасс также следует сделать автоматическим. Для этого необходимо сделать следующее:

- Проанализировать файл трассы, сохраняемый из UPPAAL
- Установить соответствие между состояниями временного автомата и состояниями UML-диаграммы
- Восстановить значения переменных и таймеров на каждом шаге трассы

5.4.2 Формат трасс UPPAAL

Файлы трасс UPPAAL предназначены прежде всего для визуализации трассы в пошаговом режиме в графическом интерфейсе UPPAAL, и изначально не предусматривается их читаемость человеком. Тем не менее, формат трасс текстовый, и его синтаксис возможно установить. Ниже приведено общее описание формата трассы, полученное в результате обратной инженерии.

Файл трассы UPPAAL представляет собой текстовый файл, в котором каждый элемент, описанный ниже, расположен в отдельной строке. Разделителем служат строки со знаком точки.

Описание формата:

Общие положения.

- Все состояния каждого автомата UPPAAL нумеруются с нуля в порядке их появления в xml-файле.
- Все переходы нумеруются с нуля в порядке их появления в xml.
- Все процессы (автоматы) нумеруются с нуля в порядке их появления в xml.
- Все переменные нумеруются с нуля в порядке их появления в xml.
- Все таймеры нумеруются с **единицы** в порядке их появления в xml. Значение 0 зарезервировано под глобальный таймер.

- Трасса содержит состояния, значения таймеров и переменных и переходы (именно в таком порядке).

Формат описания состояний.

- После каждого состояния стоит одна точка в отдельной строке.
- После блока со значениями таймеров стоят две точки в отдельных строках (длина блока таймеров может меняться, поэтому необходим отдельный признак конца).
- После блока со значениями переменных стоит одна точка в отдельной строке.
- После каждого перехода стоит одна точка в отдельной строке.
- В конце трассы стоит точка в отдельной строке.
- В начале трассы записано начальное состояние и значения переменных и таймеров.
- Дальше записаны блоки, сначала очередное состояние, затем значения таймеров и переменных, затем сделанный переход.
- Состояние записывается как набор номеров активных состояний во всех процессах, каждый номер в отдельной строке.

Формат описания переходов.

- Переход записывается как пара чисел: номер процесса и номер перехода, в одной строке, эти два числа разделены пробелом.
- Если при переходе задействуются сразу несколько ребер (при посылке и приеме сигналов), то каждый обозначается описанной выше парой чисел в отдельной строке.

Формат описания переменных и таймеров.

- Значения переменных записываются подряд, в каждой строчке – значение соответствующей переменной, каждый раз для всех переменных.
- Значения таймеров записываются блоками по три числа, разделенными строкой, содержащей точку (блок таймеров заканчивается двумя точками). Три числа в блоке имеют следующие значения: номер первого таймера, номер второго таймера, верхняя граница их разницы, умноженная на 2.

Общая схема файла трассы:

<Начальное состояние>
.
<Второе состояние>
.
<Переход из начального во второе состояние>
.
<Третье состояние>
<…>

Схема описания состояния:

<Номер состояния 1-го автомата>
<Номер состояния 2-го автомата>
<…>
<Номер состояния последнего автомата>
.
<Выражение над таймерами>
.
<Выражение над таймерами>
.
<…>
.
<Значение 1-й переменной>
<…>
<Значение последней переменной>

5.4.3 Преобразование имен состояний

Так как перед экспортом иерархического временного автомата в UPPAAL создается внутреннее представление для временных автоматов, получить по номерам имена состояний, переменных и таймеров несложно.

В автоматах UPPAAL каждому простому состоянию UML соответствует одно несрочное состояние. Помимо этого в UPPAAL есть множество служебных промежуточных состояний, которые не имеют прямых соответствий в UML. Однако в UML нас интересуют только простые состояния, в которых модель может находиться некоторое время. Составные состояния интереса не представляют, поскольку модель всегда находится в одном из вложенных в них простых состояний, либо в процессе входа или выхода, который считается мгновенным и с точки зрения поведения программы нас не интересует.

Таким образом, необходимо для каждого простого состояния UML указать соответствующее ему состояние UPPAAL. Согласно алгоритму трансляции, простое состояние s преобразуется в состояние с именем вида <s>_active_in_<P>. В процессе трансляции в UPPAAL можно при каждом создании такого состояния можно записывать этот факт в специальный лог.

Однако здесь возникает проблема с переименованием: к моменту трансляции в UPPAAL название простого состояния s вовсе не обязательно будет таким же, как в изначальном UML. Это может случиться из-за переименования состояний при преобразовании в HTA, которое делается во избежание совпадения имен состояний из разных вложенных автоматов. В результате требуется при каждом переименовании состояний записывать этот факт в лог.

Применяя записанные в лог переименования в обратном порядке, можно получить исходное название состояния. Если отбросить служебные состояния UPPAAL, в итоге для каждого состояния в трассе UPPAAL получается состояние диаграммы UML. Это по сути и есть трасса для UML-диаграммы, но в ней не хватает значений переменных и таймеров, которые необходимо знать для анализа трассы.

5.4.4 Вычисление значений таймеров

Как было сказано выше, значения таймеров в трассе UPPAAL не хранятся в явном виде, вместо этого там записаны неравенства, связывающие различные таймеры. Это связано с тем, что в самой системе UPPAAL значения таймеров показываются в таком виде. Тем не менее, для анализа трассы было бы полезно знать абсолютные значения таймеров. В особенности это было бы важно для проверки, соблюдаются ли в модели директивные сроки, заданные абсолютными константами.

В общем случае задача поиска решения системы линейных неравенств достаточно сложна, однако в данном случае рассматривается упрощенная задача. Все неравенства, как следует из описания формата трассы, имеют вид x_i-x_j<=c_k. Кроме того, система обладает следующим свойством: если разбить все таймеры на две непересекающиеся группы, то для любого такого разбиения обязательно найдется неравенство, в котором есть таймер из первой и второй группы. Такую систему можно решить алгоритмом описанным ниже.

Часть неравенств содержит «нулевой» таймер, обозначающий глобальное время. Фактически неравенство вида x_i-x_o<=c означает, что абсолютное значение таймера x_i не превышает c.

В результате изначально имеется ряд неравенств, задающих нижние и верхние границы некоторых таймеров. Далее эти неравенства складываются с остальными, в результате чего получаются неравенства для других таймеров, и так пока не будут найдены все значения. Можно представить алгоритм следующим псевдокодом.

```
function SolveTimers(inequalities):
    values = {} // Список нижних и верхних границ таймеров
    // Обработать все неравенства, в которых присутствует только один таймер
    for (x_i-x_j<=c_k) in inequalities:
        if i == 0:
            values = values ∪ ("x_j >= -c_k / 2")
        else if j == 0:
            values = values ∪ ("x_i <= c_k / 2")
    // В цикле складываем неравенства друг с другом,
    // получая границы на все остальные таймеры
    while True:
        leng = |values|
        for (x_i-x_j<=c_k) in inequalities:
            for (x_l <= d) in values:
                if l == j:
                    values = values ∪ ("x_i <= d + c_k/2")
            for (x_l >= d) in values:
```

```
            if l == i:
                values = values ∪ (“x_j >= d - c_k/2”)
    if |values| == leng: // Если не добавилось новых неравенств - выходим
        break
return values
```

Следует обратить внимание на то, что знак объединения в данном алгоритме должен работать так, чтобы не допускать добавления во множество неравенств values не только повторяющихся неравенств, но и более широких неравенств. Например, если уже есть неравенство $x < 5$, то $x < 10$ добавлять не нужно.

5.4.5 Заключение

В данном разделе был описан алгоритм построения трассы переходов UML по трассе UPPAAL. Алгоритм позволяет разобрать файл в формате трасс UPPAAL и, используя специально собранную при трансляции информацию, восстановить соответствующие события в терминах диаграммы UML. Также рассчитываются значения переменных и таймеров.

Описанный метод построения трассы был реализован в рамках средства трансляции UML в UPPAAL. Программа берет на вход трассу UPPAAL и файл со вспомогательной информацией. Примеры работы транслятора приведены в разделе 9.7.

5.5 Метод оценки наихудшего времени выполнения

5.5.1 Введение

Во многих ситуациях вычислительным системам предъявляются строгие требования ко времени их реакции на внешние события. В случаях, когда запаздывание реакции даже на доли секунды может привести к существенным потерям, применяются системы реального времени. При этом возникает задача проверки, отвечает ли система предъявляемым ко времени реакции требованиям.

Задача оценки наихудшего времени выполнения (WCET) решается для определения, удовлетворяет ли система требованиям ограничений работы по времени. В равной степени оценка WCET важна и на этапе разработки системы, как метод априорного анализа кода и выявления несоответствий до начала тестирования системы.

При моделировании РВС РВ разработчику модели некоторые действия компонента моделируемой системы удобней описывать на алгоритмическом языке программирования, чем при помощи UML диаграмм. Поэтому некоторые простые состояния UML диаграмм, помечаются комментариями, содержащими код на языке C++. Знание этого кода позволяет,

оценить время его выполнения и, таким образом, вычислить максимальное время пребывания системы в том или ином состоянии.

Согласно [130], невозможно найти оценку наихудшего времени выполнения произвольной программы. Как известно, в общем случае невозможно даже выявить завершимость задачи. Однако программы, функционирующие в РВС РВ, используют ограниченный набор конструкций языков программирования, гарантирующих, что программа закончит выполнение. Во-первых, эти программы в основном последовательные. Во-вторых, эти программы обрабатывают целочисленные данные. В-третьих, для этих программ явно задаются ограничения на число итераций циклов. В-четвёртых, запрещается рекурсивный вызов процедур для программ, функционирующих в РВС РВ.

На рисунке 65 изображено типичное распределение времен выполнения программы в зависимости от различных наборов входных данных. Внутренний интервал показывает распределение, которое можно получить при профилировке. Реальное распределение времен выполнения ограничено действительными наилучшим (BCET) и наихудшим (WCET) временами выполнения программы. Для программ с нетривиальным потоком управления, данные значения практически невозможно получить с помощью измерений. Оценки наихудшего времени выполнения, получаемые современными методами, превышают реальное WCET. Задачей оценки наихудшего времени выполнения является нахождение наиболее точного значения, которое при этом не ниже реального WCET.

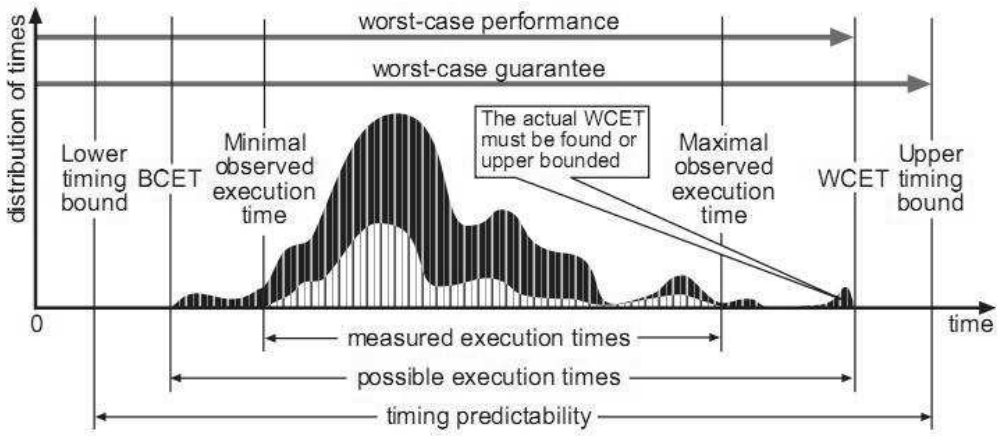

Рисунок 65. Распределение времени выполнения задачи в зависимости от различных входных данных

5.5.2 Метод, основанный на верификации программ на моделях

Для реализации анализатора оценки наихудшего времени выполнения на этапе 4 [4] был выбран статический метод, основанный на верификации программ на моделях.

Статические методы (такие как методы, описанные в работах [130],[131],[132],[133]) основаны на аналитическом исследовании программы без её выполнения на целевом оборудовании.

Типичная схема анализа программы по статическому методу включает в себя следующую последовательность шагов (рисунок 66):

- дизассемблирование программы (в случае наличия исполняемого файла и недоступности исходного кода);
- построение графа потока управления (CFG);
- анализ потока управления (Control-Flow Analysis);
- анализ поведения процессора (Processor-Behavior Analysis);
- вычисление оценки наихудшего времени выполнения (estimate calculation).

Рисунок 66. Типичная схема анализа программы статическим методом

В процессе анализа программы выполняются следующие операции:

1. Дизассемблирование программы.

Данный этап используется для получения исходного или ассемблерного кода программы в случае его отсутствия.

2. Построение графа потока управления.

На данном этапе производится анализ кода и построение графа потока управления программы (граф, у которого вершины – линейные участки, ребра – переходы между ними). В процессе анализа код программы разбивается на линейные участки, которые являются вершинами графа. Ребра формируются из переходов между линейными участками.

3. Анализ потока управления.

Производится анализ кода и графа потока управления программы и выявление таких свойств, как количество итераций циклов, границы рекурсий, наличие недостижимых линейных участков (участки кода, выполнение которых никогда не происходит).

После обхода графа к узлам и ребрам графа потока управления сопоставляется дополнительная информация, полученная в результате анализа. Например, к ребрам,

соответствующим циклическим переходам, может быть сопоставлена информация о количестве проходов по ним.

4. Анализ поведения процессора.

На данном этапе происходит вычисление времени выполнения каждого линейного участка. При этом учитывается влияние архитектурных компонент: кэшей, конвейера, предсказателя переходов. При наличии любой из перечисленных компонент время выполнения каждой инструкции зависит от истории выполнения программы, а именно от последовательности ранее выполненных инструкций, переходов, обращений к памяти.

Для вычисления времени выполнения линейных участков используются такие подходы, как эмуляция выполнения и абстрактная интерпретация. Описание упомянутых подходов можно найти в работе [130].

5. Вычисление оценки наихудшего времени выполнения.

Производится проход по графу и вычисление наихудшего времени работы программы с учетом данных о времени выполнения и о выполнимости линейных участков. Существуют следующие подходы оценки наихудшего времени выполнения:

- метод, основанный на представлении программы в виде дерева (structure-based/tree-based)
- метод полного перебора путей (path-based)
- метод неявного перебора путей (implicit path enumeration techniques — IPET)
- метод, основанный на верификации программ на моделях(model checking)

Метод, основанный на верификации программ на моделях заключается в описании модели поведения программы и проверки выполнимости программы за определенный временной промежуток с помощью верификации. Модель программы может быть представлена, например, в виде диаграмм переходов, как, например, в [134], или в виде конъюнктивной нормальной формы, как описано в [135].

Использование верификации програм для оценки наихудшего времени выполнения подробно описано в [136]. Схема вычисление оценки наихудшего времени выполнения методом, основанным на верификации программ, выглядит следующим образом:

- построение модели программы с добавлением описания временных свойств её поведения
- формирование свойства проверки выполнимости программы за время, большее некоторой оценки
- итеративное уточнение построенной оценки путем запуска верификатора и проверки выполнимости данного свойства до нахождения точной нижней грани оценки, при которой свойство перестает выполняться

- полученная оценка считается наихудшим временем выполнения программы

Итеративное уточнение оценки может производиться по различным схемам. Один из способов итеративного уточнения оценки заключается в выполнении следующей последовательности шагов:

1. Сформировать оценку *estimateWcet = 0.*
2. Сформировать условие проверки выполнимости программы за время, большее *estimateWcet.*
3. Запустить верификацию программы с проверкой выполнимости поставленного условия.
4. Если поставленное условие не выполняется, *estimateWcet* считается наихудшим временем выполнения программы.
5. Если поставленное условие выполняется, получить из контрпримера, выдаваемого верификатором, новое значение *estimateWcet* (то есть значение, при котором программа работает дольше исходной оценки), и перейти на шаг 2.

В некоторых верификаторах (таких как UPPAAL) имеется возможность запустить верификацию в режиме автоматического поиска наименьшего (или наибольшего) значения некоторой переменной, при которой заданное свойство перестает выполняться. Данную возможность можно использовать вместо предложенной итеративной схемы.

Статический метод, основанный на верификации программ на моделях, был выбран для вычисления наихудшего времени работы программы в связи со следующими причинами:

- при оценке статическими методами не требуется запуск программы на целевом вычислителе, в отличие от других методов
- при использовании верификации программ на моделях имеется возможность анализировать программы с недетерминированным поведением, в то же время не требуется полностью перебирать все пространство путей выполнения (как, например, в методе полного перебора)

Преимуществом данного метода является то, что имеется возможность анализировать недетерминированное выполнение программы. Недостатком является вычислительная сложность, так как на каждой итерации производится запуск верификатора, который должен произвести поиск в полном пространстве состояний программы для проверки выполнимости поставленного свойства.

5.5.3 Описание выбранной модели процессора

В первых работах, посвященных оценке наихудшего времени выполнения программ (таких как [137]), предполагалось, что время выполнения отдельных участков кода не зависит от контекста выполнения. При таком предположении время выполнения двух участков кода можно вычислить, просуммировав времена выполнения этих участков.

В современных вычислителях, содержащих в себе такие компоненты, как кэш и конвейер, независимость от контекста уже не имеет места. Время выполнения отдельной инструкции может существенно варьироваться в зависимости от состояния процессора. В общем случае для определения времени выполнения текущей инструкции необходимо знать историю выполнения программы непосредственно до выполнения данной инструкции. При этом необходимо учитывать влияние таких компонент вычислителя, как память, кэш, конвейер и предсказатель переходов. Например, время считывания значения переменной, находящейся в кэше, на порядок меньше, чем время считывания переменной из оперативной памяти. Анализ попадания/промаха при поиске переменной в кэше может существенно улучшить точность оценки наихудшего времени выполнения. Влияние компонент на время выполнения описано в работе [130].

Анализ времени выполнения инструкций и линейных участков в процессе оценки наихудшего времени выполнения статическими методами проводится на этапе анализа поведения процессора. Время выполнения вычисляется исходя из модели вычислителя и истории выполнения программы. При этом модель вычислителя состоит из набора моделей компонент, влияющих на время выполнения. Точность полученных результатов времени выполнения инструкций и линейных участков зависят от точности модели вычислителя.

Для анализа наихудшего времени выполнения был выбран метод представления вычислителя в виде плоского временного автомата. Данная модель используется для верификации с помощью инструмента UPPAAL[113]. В работе [138] описан пример использования данного подхода при моделировании архитектуры вычислителя ARM9T. Каждая из компонент модели, таких как кэш, конвейер и память, моделируются в виде сети автоматов. В процессе анализа строится модель программы в виде NTA, и к ней подключаются модели компонент вычислителя. Полученная модель подается на вход верификатору UPPAAL и производится верификация с проверкой временных свойств программы с учетом влияния поведения вычислителя.

Основными причинами выбора данной модели процессора являются следующие особенности:

- Существуют готовые описания компонент процессора ARM9T в виде сетей автоматов. Модели входят в состав средства Metamoc [126].

- Модели написаны на языке верификатора UPPAAL.

В данном разделе приводится описание выбранного метода оценки наихудшего времени выполнения, основанного на верификации программ на моделях и выбранной модели представления вычислителя в виде плоского временного автомата. Описанные в этом разделе метод оценки наихудшего времени работы программы и модель вычислителя используются в средстве Metamoc, которое используется в рамках среды моделирования и описано в разделе 4.9.

5.6 Интеграция среды моделирования со средствами оценки WCET

5.6.1 Встраивание средств оценки WCET в транслятор UML в UPPAAL

Синтаксис UML допускает использование комментариев для пометки некоторых состояний диаграмм. Эти комментарии могут содержать важную информацию о поведении проектируемой системы, но обработка комментариев должна осуществляться процедурами, выходящими за пределы обычных средств моделирования.

В частности, простые состояния, в которых выполняются содержательные операции, могут быть помечены комментариями, содержащими код на языке C++. Знание этого кода позволяет, например, оценить время его выполнения и, таким образом, указать ограничение на время пребывания системы в том или ином состоянии.

Для оценки времени выполнения программного кода может быть использовано программно-инструментальное средство, использующее один из алгоритмов WCET, которое позволяет по заданному программному коду и модели аппаратуры, выполняющей этот код, получить верхнюю оценку времени выполнения кода. Оценка времени выполнения программного кода, которым помечены простые состояния UML диаграмм, проводится на этапе преобразования UML в HTA. Для каждого состояния вводится дополнительный таймер t . Показания этого таймера сбрасываются на каждом переходе, ведущем в любое простое состояние UML диаграммы. При этом верхняя оценка времени выполнения кода преобразуются в ограничения вида $t \leq c$, которые добавляются к инвариантам, приписанным простым состояниям. После этого комментарий состояния удаляется, а вместо него добавляются таймер, инвариант и таймаут, как показано на рисунке 67. Здесь E – это оценка времени выполнения, полученная из средства WCET.

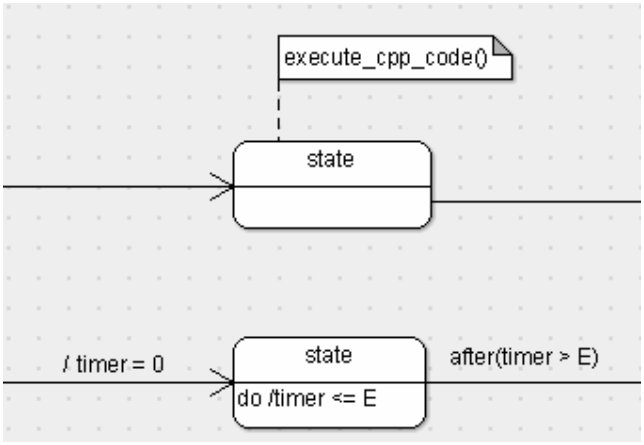

Рисунок 67. Добавление таймера, инварианта и таймаута в модель по результатам оценки наихудшего времени выполнения программы

5.6.2 Запуск анализатора WCET

Для запуска анализатора оценки максимального времени выполнения линейного участка используется специальный скрипт count_wcet_basic_block, которому на вход подается файл с описанием кода линейного участка на языке C++ и который возвращает наихудшее время выполнения линейного участка в тактах работы процессора.

При вызове анализатора из транслятора UPPAAL по умолчанию происходит оценка времени выполнения с использованием модели ARM9TDMI. Можно настраивать параметры модели вычислителя, указывая следующие параметры:

- модель конвейера; в настоящее время доступны модели для архитектур arm7, avr, arm9 со следующими характеристиками:
  - процессор ARM920T семейства arm9, использующим процессорное ядро ARM9TDMI, пятиступенчатый конвейер, раздельные кэши данных и инструкций, MMU и шинный интерфейс для соединения с основной памятью; описание процессора можно найти в работе [139]
  - процессор семейства arm7 с процессорным ядром ARM7TDMI, отличающийся от процессора ARM920T использованием трехступенчатого конвейера; описание процессора можно найти в работе [140]
  - процессор ATMEL AVR 8-bit, с двухступенчатым конвейером; описание процессора можно найти в работе [141]
- модель кэша; в этой модели можно настраивать такие параметры, как размер блока кэша, максимальное количества блоков, поведение кэша при промахе и попадании

- модель памяти; эта модель позволяет настраивать такие параметры, как время доступа к памяти, размер блока памяти, количество блоков памяти.

Перечисленные параметры указываются с помощью задания соответствующих ключей при вызове анализатора оценки максимального времени выполнения. Пример работы данного средства приведен в рамках описания методики работы с транслятором UML в UPPAAL в разделе 8.5.

5.7 Методы решения задачи выбора механизмов обеспечения отказоустойчивости РВС РВ

Все РВС РВ имеют четыре фундаментальные характеристики: функциональность, производительность, надежность и стоимость. В ходе разработки РВС РВ возникает задача построения системы, имеющей заданную функциональность (описываемую в спецификациях) и максимальную надежность при ограничениях на производительность и стоимость. Надежность РВС РВ может быть повышена посредством использования механизмов обеспечения отказоустойчивости (МОО), однако это снижает производительность и увеличивает стоимость системы.

В процессе построения конфигурации РВС РВ необходимо оценивать ее производительность, то есть время выполнения программ на заданных устройствах. Произвести такую оценку аналитически невозможно, поэтому необходимо обращаться к процедуре имитационного моделирования.

В данном разделе описано несколько методов решения задачи выбора МОО РВС РВ. Описание схемы интеграции программного средства, решающего эту задачу, со средой имитационного моделирования приведено в разделе 5.8, методика использования средства – в разделе 8.6, экспериментальное исследование – в разделе 9.8.

5.7.1 Неформальная постановка задачи

Структура РВС РВ задана в виде набора модулей и связей между ними. Каждый модуль состоит из аппаратного компонента, программного компонента и, возможно, МОО. Известны наборы вариантов устройств и программ, которые могут присутствовать в модуле в качестве соответственно аппаратного и программного компонентов. Для каждого варианта устройства или программы известны надежность и цена. Для каждой пары (устройство, программа) известно время выполнения программы на данном устройстве. Также известен объем выходных данных программного компонента каждого модуля. Для каждого модуля

известны модули, от которых необходимо получать входные данные и которым нужно передавать выходные данные.

Необходимо для каждого модуля системы выбрать такой МОО и варианты аппаратных и программных компонентов, при которых надёжность максимальна при ограничениях на стоимость и время выполнения программ.

Под надежностью системы будем понимать вероятность ее безотказной работы в течение некоторого заданного промежутка времени.

Приведем описание рассматриваемых механизмов обеспечения отказоустойчивости (МОО) [142].

1) N-версионное программирование (NVP). Данный МОО включает модуль принятия решения (голосователь) и N независимо разработанных версий программы (N – нечетное). Все N версий работают параллельно, результат их работы обрабатывает голосователь. Тот результат, который выдает большая часть версий, принимается за финальный. В данном случае рассматривается два варианта NVP, в каждом из которых N=3. В NVP/0/1 все версии программы работают на одном аппаратном компоненте, в то время как в NVP/1/1 каждая версия программы работает на отдельном аппаратном компоненте. Таким образом NVP/0/1 допускает одну неисправность в программных версиях, а NVP/1/1 – одну неисправность либо в программных версиях, либо в аппаратных.

2) Восстановление блоками (RB/1/1). Данный МОО включает в себя модуль принятия решений (контрольный тест) и минимум две программных версии. Когда первая из версий завершает свою работу, то результат её работы тестируется контрольным тестом. Если результат теста оказался неудачным, то процесс откатывается на начало работы и запускается на выполнение следующая версия. Процесс продолжается пока результат одной из версий не будет принят, либо результат работы всех версий не будет отклонён. В данной постановке задачи используются два аппаратных компонента, на каждом из которых существует две разные версии программы.

5.7.2 Формальная постановка задачи

Для формального описания структуры РВС РВ введём следующие обозначения:

- n – количество модулей РВС РВ;
- U_i – i-ый модуль системы;
- p_i, q_i – количество доступных версий соответственно аппаратного и программного компонентов в модуле U_i;
- H_{ij} – j-ая версия аппаратного компонента модуля U_i, $j \in [1, p_i]$;

- S_{ij} – j-ая версия программного компонента модуля U_i, $j \in [1, q_i]$;
- FT_i – множество доступных МОО для i-ого модуля;
- $FT_i \subseteq \{None, NVP/0/1, NVP/1/1, RB/1/1\}$, *«None»* обозначает отсутствие МОО;
- $F_i \in FT_i$ – МОО, используемый в модуле U_i;
- $H_i^{F_i}$ – мультимножество версий H_{ij}, выбранных для аппаратного компонента модуля U_i;
- $S_i^{F_i}$ – множество версий S_{ij}, выбранных для программного компонента модуля U_i.

Конфигурацию *System* РВС РВ можно представить в виде ориентированного графа без циклов, в котором каждой вершине соответствует модуль U_i, а множество ребер содержит ребро (U_i, U_j) тогда и только тогда, когда выходные данные программного компонента модуля U_i являются входными данными для программного компонента модуля U_j. В таком случае будем считать, что модуль U_i непосредственно зависит по данным от модуля U_j.

Конфигурация i-ого модуля однозначно определяется тройкой $\{H_i^{F_i}, S_i^{F_i}, F_i\}$.

- R_{ij}^{hw}, C_{ij}^{hw} – надежность и стоимость j-ой версии аппаратного компонента модуля U_i;
- R_{ij}^{sw}, C_{ij}^{sw} – надежность и стоимость j-ой версии программного компонента модуля U_i;
- x_{ij}, y_{ij} – количество экземпляров H_{ij} и S_{ij} в модуле U_i.

Стоимость системы можно вычислить по формуле:

$$C_{System} = \sum_{i=1}^{n} \left(\sum_{j=1}^{p_i} x_{ij} \cdot C_{ij}^{hw} + \sum_{j=1}^{q_i} y_{ij} \cdot C_{ij}^{sw} \right) = \sum_{i=1}^{n} \sum_{j=1}^{p_i} x_{ij} \cdot C_{ij}^{hw} + \sum_{i=1}^{n} \sum_{j=1}^{q_i} y_{ij} \cdot C_{ij}^{sw}.$$

- P_{rv} – вероятность отказа между двумя версиями программного компонента;
- P_{all} – вероятность одновременного отказа всех версий программного компонента;
- P_d – вероятность отказа схемы голосования.

Надежность i-ого модуля можно вычислить, используя R_{ij}^{hw}, R_{ij}^{sw}, P_{rv}, P_{all}, P_d. Формулы для этого приведены в [142]. Например, если в модуле U_i используется МОО RB/1/1, выбраны версии k_1, k_2 аппаратного компонента и версии l_1, l_2 программного компонента, то надежность этого модуля вычисляется по формуле:

$$R_i = 1-(P_{rv}+(1-P_{rv})\cdot P_d+(1-P_{rv})\cdot(1-P_d)\cdot P_{all}+(1-P_{rv})\cdot(1-P_d)\cdot(1-P_{all})\cdot$$
$$\cdot(1-R_{ik_1}^{hw})\cdot(1-R_{ik_2}^{hw})+(1-P_{rv})\cdot(1-P_d)\cdot(1-P_{all})\cdot(1-(1-R_{ik_1}^{hw})\cdot(1-R_{ik_2}^{hw}))\cdot$$
$$\cdot(1-R_{il_1}^{sw})\cdot(1-R_{il_2}^{sw}))$$

Надежность всей системы равна:

$$R_{System}=\prod_{i=1}^{n}R_i .$$

- D_i – директивное время выполнения программного компонента S_i модуля U_i;
- T_i – время выполнения программного компонента S_i модуля U_i; оценивается с помощью имитационного моделирования; будем считать, что работа программного компонента модуля завершена, если его выходные данные полностью переданы всем зависящим от него модулям.

В данной постановке задачи для каждого модуля U_i время выполнения программного компонента для каждой пары версий (H_{ij_1}, S_{ij_2}), $j_1 \in [1, p_i]$, $j_2 \in [1, q_i]$ считается известным. Также известны времена работы голосователей, контрольных тестов и времена откатов к начальным состояниям в схеме RB/1/1. Кроме того, известен объем данных, передаваемых между программными компонентами, и скорость передачи данных.

- $C_{System}^{\max}$ – максимальная допустимая стоимость конфигурации РВС РВ;
- *Systems* – множество всевозможных конфигураций РВС РВ.

В рамках введённых обозначений задачу сбалансированного выбора МОО РВС РВ можно сформулировать следующим образом:

Дано:

1. n;
2. $p_i, q_i, \forall i \in [1,n]$;
3. R_{ij}^{hw}, C_{ij}^{hw}, $\forall i \in [1,n], \forall j \in [1, p_i]$;
4. R_{ij}^{sw}, C_{ij}^{sw}, $\forall i \in [1,n], \forall j \in [1, q_i]$;
5. FT_i, $\forall i \in [1,n]$;
6. P_{rv}, P_{all}, P_d;
7. D_i, $\forall i \in [1,n]$;
8. $C_{system}^{\max}$.

Требуется определить:

Конфигурацию $System_{best} \in Systems$, такую что:

$$R_{System_{best}} = \max_{Systems} R_{system}$$

при этом должны выполняться условия:

$$C_{System_{best}} \le C_{system}^{\max},$$

$$T_i \le D_i, \ \forall i \in [1, n]$$

то есть стоимость системы не должна превышать максимально допустимой, а время работы каждого модуля не должно превышать индивидуального директивного срока для этого модуля.

Данную задачу можно записать как задачу дискретной оптимизации:

$$\begin{cases} System_{best} \in Systems \\ R_{System_{best}} = \max_{Systems} R_{system} \\ C_{System_{best}} \le C_{system}^{\max} \\ \mathrm{T_i} \le \mathrm{D_i}, \forall \mathrm{i} \in [1, \mathrm{n}] \end{cases}.$$

5.7.3 Алгоритмы решения задачи

В [143] был предложен метод решения поставленной задачи, состоящий из следующих шагов: построение модели РВС РВ в общем виде; поиск оптимальной конфигурации РВС РВ с помощью одного из алгоритмов оптимизации. В качестве алгоритма использовался адаптивный гибридный эволюционный алгоритм (АГЭА).

Модель РВС РВ строится на основе графа, представляющего внутреннюю структуру системы.

На текущем этапе работы, кроме АГЭА, были реализованы следующие оптимизационные алгоритмы: классический ЭА, два островных ЭА, алгоритм имитации отжига.

Каждое возможное решение кодируется в виде строки, состоящей из блоков, которые соответствуют модулям РВС РВ. Каждый блок представляет собой тройку вида <*H, S, F*>. *H* - номер конфигурации аппаратной составляющей модуля, *S* – номер конфигурации программной составляющей, а *F* – номер МОО, используемого в данном модуле. По строке такого вида может быть вычислена целевая функция – надёжность системы, которая характеризует качество решения, и однозначно восстановлена соответствующая конфигурация РВС РВ.

Далее подробно описаны схемы работы всех исследованных алгоритмов.

Адаптивный гибридный эволюционный алгоритм.

1. Генерация случайным образом популяции решений.
2. Оценка популяции: вычисление целевой функции и проверка ограничений стоимости и времени; фиксация лучшего на текущий момент решения, вычисление среднего значения целевой функции для всей популяции.
3. Отбор особей для скрещивания в отдельную промежуточную популяцию: популяция сортируется, и отбираются лучшие N_{sel} % особей по значению целевой функции.
4. Операция скрещивания (одноточечное скрещивание).
5. Формирование новой популяции: в новую популяцию берется некоторый процент лучших особей от текущей популяции, остальная часть популяции формируется из лучших особей промежуточной популяции, полученной после операции скрещивания.
6. Операция мутации (модификация одноточечной мутации).
7. Повторение п.2.
8. Проверка критерия останова: если он выполнен, то переход к п.10, если нет, то переход к п.9. Критерием останова в данном случае является выполнение алгоритмом априорно заданного числа итераций без улучшения целевой функции.
9. Блок нечеткой логики осуществляет автоматическую подстройку алгоритма и переход к п.3.
10. Завершение алгоритма, вывод наилучшей конфигурации.

Так как при жестких ограничениях на стоимость и время выполнения программ большое количество решений ограничениям не удовлетворяет, то необходимо уменьшить вероятность попадания таких решений в текущую популяцию. Для этого авторами было предложено использование штрафных функций, позволяющих искусственно уменьшить значение целевой функции для решений, не удовлетворяющих ограничениям. Значение штрафной функции для каждого решения представляет собой коэффициент, который равен 1, если решение удовлетворяет ограничениям, и принадлежит интервалу (0;1), иначе. В качестве целевой функции берется значение надежности решения, умноженное на этот коэффициент. В данной работе штрафной коэффициент обратно пропорционален стоимости (времени) в случае невыполнения ограничений:

$$E_{time}^{i} = \begin{cases} 1, & T_i < D_i \\ \dfrac{D_i}{T_i}, & иначе \end{cases}, \quad R_i^* = R_i \cdot E_{time}^{i}$$

$$E_{cost} = \begin{cases} 1, & C_{system} < C_{system}^{\max} \\ \dfrac{C_{system}^{\max}}{C_{system}}, & иначе \end{cases}, \quad R_{system}^* = R_{system} \cdot E_{cost}, \quad R_{system}^* = \prod_{i=1}^{n} R_i^*$$

Опишем подробно блок нечеткой логики. Он нужен для того, чтобы в автоматическом режиме корректировать настройки ЭА, управлять степенью влияния операций селекции, скрещивания и мутации на эволюционный процесс согласно некоторым правилам в зависимости от результатов работы алгоритма, получаемых на каждой итерации.

Ключевыми параметрами для оценки популяции являются среднее значение целевой функции текущей популяции и лучшее значение целевой функции текущей популяции.

Введём следующие обозначения:

– R_1^{av} и R_0^{av} – средние значения целевой функции в текущей и предыдущей популяциях;

– $R_1^{\max}$ и $R_0^{\max}$ – лучшие значения целевой функции в текущей и предыдущей популяциях.

Изменяемые параметры АГЭА:

- N_{sel} – процент от популяции лучших особей, которые затем будут скрещиваться;
- P_{cross}– вероятность скрещивания;
- N_{nmut} – процент лучших особей текущей популяции, которые не мутируют;
- P_{mut} – вероятность мутации;

Параметры N_{nmut} и P_{mut} имеют три значения (большое, среднее и малое). Параметры N_{sel} и P_{cross} имеют два значения (большое и малое).

В зависимости от изменений параметров R^{av} и $R^{\max}$ в процессе работы АГЭА происходит переключение значений параметров алгоритма в соответствии с таблицей 5.

Таблица 5. Правила работы блока нечеткой логики

	$R_0^{av} < R_1^{av}$	$R_0^{av} \approx R_1^{av}$ (~3%)	$R_0^{av} > R_1^{av}$
$R_0^{\max} < R_1^{\max}$	Малый P_{mut} Большой N_{nmut} Большой N_{sel} Большой P_{cross}	Средний P_{mut} Средний N_{nmut} Большой N_{sel} Большой P_{cross}	Большой P_{mut} Малый N_{nmut} Большой N_{sel} Большой P_{cross}
$R_0^{\max} \approx R_1^{\max}$ (~3%)	Малый P_{mut} Большой N_{nmut} Малый N_{sel} Малый P_{cross}	Средний P_{mut} Средний N_{nmut} Малый N_{sel} Малый P_{cross}	Большой P_{mut} Малый N_{nmut} Малый N_{sel} Малый P_{cross}

Такой блок нечеткой логики имеет малую сложность по сравнению с вычислением целевых функций и проверкой ограничений для всех особей популяции, но при этом позволяет производить автоматическую перенастройку эволюционного алгоритма в процессе его работы.

Классический эволюционный алгоритм.

1. Генерация случайным образом популяции решений.
2. Вычисление целевой функции и проверка ограничений для каждого решения из популяции. Фиксация лучшего на текущий момент решения.
3. Выполнение оператора селекции.
4. Выполнение оператора скрещивания.
5. Выполнение оператора мутации.
6. Повторение п.2.
7. Проверка критерия останова: если критерий не достигнут, то перейти к шагу 3, иначе завершить работу алгоритма.

Операции селекции, скрещивания и мутации, критерий останова и штрафные функции используются такие же, как и в АГЭА.

Островные эволюционные алгоритмы.

Островной ЭА представляет собой несколько ЭА, работающих параллельно каждый на своей подпопуляции и периодически обменивающихся решениями. Операция обмена решениями называется миграцией и определяется следующим образом: несколько лучших решений (количество определяется в настройках алгоритма) каждые несколько шагов

(количество определяется в настройках алгоритма) алгоритма мигрируют в соседнюю подпопуляцию и замещают там худшие решения. Предполагается, что все ЭА пронумерованы и для каждого из них, кроме последнего, соседним является следующий по порядку алгоритм; для последнего ЭА соседним является первый алгоритм.

В ходе данной работы были реализованы два островных ЭА: в первом параллельно работают классические ЭА, во втором – адаптивные гибридные.

Алгоритм имитации отжига.

Рассмотрим модификацию алгоритма имитации отжига, предложенную в [144].

Пусть X – множество двоек вида (решение, вероятность), X – множество доступных решений. T – текущая температура.

Изначально X пусто, T=0.

Предложенный алгоритм состоит из следующих шагов:

1. Случайным образом генерируется начальное решение $System_1$, для него вычисляется надежность ($R(System_1)$) и проверяются ограничения на стоимость и время. Данные действия повторяются до тех пор, пока сгенерированное решение не будет удовлетворять всем ограничениям.
2. $System_1$ становится текущим решением $System_{cur}$. $System_1$ с вероятностью 1 добавляется в X.
3. В качестве рабочего решения $System_w$ берется копия решения $System_{cur}$.
4. К $System_w$ применятся следующая операция мутации: произвольно выбирается модуль и его конфигурация заменяется на конфигурацию, сгенерированную случайным образом.
5. Вычисляется надежность $R(System_w)$ полученного решения и сравнивается с надежностью $R(System_{cur})$; проверяются ограничения на стоимость и время для $System_w$. Возможны три случая:
   - $R(System_w) > R(System_{cur})$ и $System_w$ удовлетворяет ограничениям. Тогда $System_w$ становится текущим решением $System_{cur}$, вектор X очищается, $System_w$ с вероятностью 1 добавляется в X.
   - $R(System_w) > R(System_{cur})$ и $System_w$ не удовлетворяет ограничениям. Тогда $System_w$ добавляется в X с вероятностью $\frac{1}{v+1}$, где v – количество элементов в X, не считая $System_w$.

- $R(System_w) \leq R(System_{cur})$. $System_w$ добавляется в X с вероятностью $\frac{1}{v+1} \cdot \exp\left(\frac{MaxTemperature \cdot (R(System_w) - R(System_{cur}))}{MaxTemperature + 1 - T}\right)$, где v – количество элементов в X, не считая $System_w$.

6. Проверка критерия останова. Если критерий не выполнен, переход к п.7. Алгоритм завершает работу, если выполнено хотя бы одно из двух условий: либо текущая температура T стала равна максимальной допустимой температуре *MaxTemperature*, либо размер вектора X превысил некоторое заданное число n (это значит, что прошло n итераций алгоритма без улучшения целевой функции). В случае завершения работы алгоритма за искомое решение берется $System_{cur}$.
7. T увеличивается на 1. Из элементов X выбирается новое рабочее решение $System_w$. Выбор происходит случайным образом в соответствии с вероятностями решений из X.
8. Переход к п.4

В данном разделе была поставлена задача выбора МОО РВС РВ и описаны предложенные алгоритмы. Экспериментальное исследование данных алгоритмов приведено в разделе 9.8.

5.8 Интеграция со средствами синтеза архитектур и построения расписаний

Средства синтеза архитектур и построения расписаний решают задачи построения архитектур и расписаний, оптимальных по некоторому критерию и удовлетворяющих определенным ограничениям. Зачастую в ходе работы синтеза архитектур и построения расписаний требуется проверить, удовлетворяет ли определенная архитектура или расписание некоторым требованиям, однако сделать это аналитически невозможно. В таких случаях обращаются к процедуре проведения имитационных экспериментов. Типичным примером ограничения, проверка выполнимости которого часто проводится с помощью имитационного моделирования, является ограничение на время выполнения программ. Примером задачи синтеза архитектуры является задача выбора механизмов обеспечения отказоустойчивости для РВС РВ, описанная в разделе 5.7.

Таким образом, возникает задача интеграции синтеза архитектур и построения расписаний со средой имитационного моделирования. Так как большинство алгоритмов синтеза архитектур и построения расписаний требуют многократных проведений

имитационных экспериментов, то взаимодействие средств планирования со средой моделирования должно быть полностью автоматизированным. Кроме того, разрабатываемая схема интеграции не должна требовать от авторов синтеза архитектур и построения расписаний знания принципов работы и внутренней структуры среды моделирования.

5.8.1 Формальное определение расписания

Формальное определение расписания было построено на основе определений, рассматриваемых в работах [145] и [146].

Аппаратная часть РВС РВ, на которой выполняется расписание, представляет собой множество процессоров, соединенных средой передачи данных. Время передачи единицы данных от одного процессора к другому одинаково для всех процессоров и считается заданным.

Программа, для которой строится расписание, состоит из конечного множества заданий (подпрограмм). Для каждого из заданий известен объем результата, получаемого на выходе. Некоторые из заданий получают на вход выходные данные других заданий. Таким образом, программу можно представить в виде графа потока данных, представляющего собой ориентированный граф без циклов. Каждой вершине графа соответствует задание программы; из i-ой вершины идет ребро в j-ую вершину тогда и только тогда, когда выходные данные i-ого задания являются входными данными для j-ого задания. i-ое задание зависит по данным от j-ого задания тогда и только тогда, когда в графе программы существует путь из j-ой вершины в i-ую, непосредственно зависит – из j-ой вершины в i-ую идет ребро.

Будем считать, что для программы задано расписание, если у каждого из заданий есть привязка – однозначно определено, на каком процессоре оно выполняется, и задан порядок – для каждого процессора известно, в какой очередности выполняются задания.

Определим понятие расписания формально. Пусть:

V – множество всех заданий программы;

M – множество процессоров в ВС.

Тогда под расписанием будем понимать множество S троек (v,m,n), $v \in V$, $m \in M$, $n \in \mathbb{N}$, таких, что:

$$\forall v \in V \quad \exists! \, s_j = (v_j, m_j, n_j) \in S : v = v_j,$$

$$\forall s_i = (v_i, m_i, n_i) \in S, \forall s_j = (v_j, m_j, n_j) \in S : (s_i \neq s_j, m_i = m_j) \Rightarrow n_i \neq n_j.$$

Элемент расписания $s_i = (v_i, m_i, n_i) \in S$ непосредственно зависит от элемента $s_j = (v_j, m_j, n_j) \in S$, если либо v_i непосредственно зависит по данным от v_j, либо $m_i = m_j$ и

$n_i=n_j+1$. Расписание может быть представлено в виде ориентированного графа такого, что вершинам графа соответствуют элементы расписания, и из *i*-ой вершины идет дуга *j*-ую вершину тогда и только тогда, когда элемент s_j непосредственно зависит от элемента s_i. Элемент s_i зависит от элемента s_j, если из *j*-ой вершины существует путь в *i*-ую.

Будем говорить, что расписание *S* является корректным, если оно удовлетворяет свойству ацикличности, то есть не существует набора элементов расписания s_1, ..., s_n , такого что s_i зависит от s_{i-1} для всех $i \in [2,n]$ и s_n зависит от s_1 [146].

Для каждой пары (процессор, задание) известно время выполнения данного задания на данном процессоре без учета времени получения и отправки данных. Для каждого задания задан директивный срок выполнения. Задание считается выполненным, если его выходные данные переданы всем непосредственно зависящим от него заданиям. Каждое задание должно быть выполнено не позже своего директивного срока.

Для каждого элемента расписания время начала работы равно $\max_{s_i}\{0,T_i\}$, где s_i – элементы расписания, от которых данный элемент непосредственно зависит, а T_i – их времена завершения.

5.8.2 Формат представления расписаний

Для того чтобы не обязывать авторов средств синтеза архитектур и построения расписаний самостоятельно реализовывать построение модели, выполнимой в среде CERTI, необходимо ввести единый формат представления расписаний и разработать отдельное программное средство, строящее модель системы по представленному в этом формате расписанию. Данный формат не должен зависеть от особенностей реализации алгоритмов синтеза архитектур и построения расписаний и среды выполнения моделей.

В качестве формата представления расписаний был выбран формат, основанный на XML. При описании расписания используются следующие теги:

<system> – корневой элемент в описании системы. Имеет атрибут rt, принимающий значение 1, если рассматриваемая система является системой жесткого реального времени, и 0 – иначе. Содержит внутри себя теги <processor>.

<processor> – описание процессора. Имеет атрибут id – уникальное имя процессора. Содержит внутри себя теги <task>.

<task> – описание задания. Имеет атрибуты num – порядковый номер задания в расписании; id – уникальное имя задания; time – время выполнения задания на процессоре, к которому привязано данное задание; dirtime – директивный срок выполнения задания; datavol – объем выходных данных. Внутри тега <task> могут содержаться теги <prev> и <next>.

<prev> – задание (атрибут id – имя), от которого текущее задание непосредственно зависит по данным.

<next> – задание (атрибут id – имя), которое зависит по данным от текущего задания.

Средства построения должны по внутреннему представлению расписания генерировать файл описанного формата. Расписание, формально определенное в предыдущем разделе, можно представить в указанном формате.

Ниже приведен пример файла, описывающего расписание:

```
<system rt="0">
	<processor id="Processor_1">
		<task id="task_1" num="1" time="5" dirtime="15" datavol="1" >
			<next id="task_4"></next>
		</task>
		<task id="task_4" num="2" time="10" dirtime="50" datavol="2">
			<prev id="task_1"></prev>
			<prev id="task_2"></prev>
			<prev id="task_3"></prev>
		</task>
	</processor>
	<processor id="Processor_2">
		<task id="task_2" num="1" time="6" dirtime="25" datavol="3">
			<next id="task_4"></next>
		</task>
		<task id="task_3" num="2" time="8" dirtime="35" datavol="4">
			<next id="task_4"></next>
		</task>
	</processor>
</system>
```

5.8.3 Модель РВС РВ

По описанному в формате XML расписанию строится модель РВС РВ, представляющая собой диаграмму состояний в формате SCXML [147]. Далее по ней будет сгенерирован код федератов на C++. Данная модель описывает выполнение расписания задач на процессорах с учётом задержек на выполнение задач и обмен данными.

Опишем общую логику построения модели в виде диаграммы состояний.

Всей вычислительной системе соответствует AND-состояние *system*, включающее в себя составные состояния, соответствующие процессорам.

Каждое состояние, соответствующее процессору, представляет собой последовательный автомат, моделирующий выполнение заданий, привязанных к заданному

процессору. i-ому заданию соответствует последовательность состояний, начинающаяся состоянием *task_<i>_entry* и завершающаяся состоянием *task_<i>_exit*. $\forall i \in [1, n-1]$ существует переход из состояния *task_<i>_exit* в состояние *task_<i+1>_entry*. *task_1_entry* – начальное состояние автомата.

Рассмотрим последовательность состояний, соответствующих i-ому заданию (префикс *task_<i>_* будет опущен).

entry – начальное состояние.

Переход в **waiting.**

waiting – ожидание входных данных.

Переход в **working**:

- условие: для всех j *task_j_ready* == true, j – номера всех заданий других процессоров, от которых необходимо получать данные;
- действия: *current_time* = max{ *task_j_task_i_sending_end* } (значение счетчика времени становится равным времени получения последней порции входных данных).

working – выполнение задания.

Переход в **time_exceeded:**

- условие: (*current_time + time > dir_time) && (hard_rt)* (превышение директивного времени для систем жесткого реального времени);
- действия: *task_i_ time_exceeded = true;*

Переход в **sending:**

- условие: (*current_time + time <= dir_time)* || *(!hard_rt)*
- действия: *task_i_time_exceeded=current_time+time>dir_time; current_time=time+current_time; task_i_task_j_sending_ready=false;* (*task_j* – непосредственно зависящие от *task_i* задания на других процессорах)

time_exceeded – нарушен директивный срок для системы жесткого реального времени.

Конечное состояние, переходов нет.

sending – состояние перед передачей данных.

Переход в **task_i_task_j_sending:**

- условие: *task_i_task_j_sending_ready*==false (*task_j* – непосредственно зависящие от *task_i* задания на других процессорах)

Переход в **end**:

- условие: для всех i *task_i_task_j_sending_ready*==true;
- действия: *task_i_ready*=true;

task_i_task_j_sending – передача данных от i-ой задачи к j-ой (передачи данных между заданиями одного процессора не моделируются).

Переход в **sending:**

- действие: task*_i_task_j_sending_ready*=true; *current_time=data_vol+current_time;* (время передачи пропорционально объему данных)

end – завершение работы задания.

Переход в **time_exceeded:**

- условие: (*current_time > dir_time) && (hard_rt)* (превышение директивного времени для систем жесткого реального времени);

Переход в **exit :**

- условие: (*current_time <= dir_time)* || *(!hard_rt)*

exit – выход.

Переход в **task_<i+1>_entry**, либо переходов нет.

Взаимодействие между автоматами, соответствующими процессорам, осуществляется с помощью глобальных переменных. В терминах среды моделирования CERTI при изменении значения глобальной переменой изменивший ее федерат должен выполнить вызов RTI send interaction с параметром-значением переменной, а все федераты, использующие эту переменную, – receive interaction. В SCXML-модели такое взаимодействие отображается как переход между состояниями параллельных регионов, соответствующих процессорам. Поле «event» такого перехода содержит имя глобальной переменной. При генерации кода федерата переходы такого вида рассматриваются как указания на то, что при нахождении автомата, задающего логику работы процессора в некотором состоянии, необходимо выполнить вызов RTI send (receive) interaction.

На рисунке 68 приведен фрагмент файла в формате SCXML, соответствующий одному заданию, а на рисунке 69 - его визуализация в редакторе SCXML-gui:

```
<state id="task_2_entry">
  <transition cond="1" target="task_2_waiting" />
  </state>
<state id="task_2_waiting">
  <transition cond="1" target="task_2_working" />
  </state>
<state id="task_2_task_4_sending">
  <transition cond="1" event="c2=task_2_trans+task_2_task_4_sending_start;
  IRT_task_2_task_4_sending_end=c2;task_2_task_4_sending_ready=true;"
target="task_2_sending" />
  </state>
<state id="task_2_sending">
  <transition cond="!task_2_task_4_sending_ready" event="task_2_task_4_sending_start=c2;"
  target="task_2_task_4_sending" />
  <transition cond="task_2_task_4_sending_ready" event="IRT_task_2_ready=true;
  task_2_time_ex=task_2_dirTime<c2;" target="task_2_end" />
  </state>
<state id="task_2_working">
  <transition cond="(c2>task_2_dirTime)&&hard_rt" target="task_2_time_exceeded" />
  <transition cond="!hard_rt||task_2_dirTime>=task_2_time+c2"
event="c2=task_2_time+c2;task_2_time_ex=(c2>task_2_time);task_2_task_4_sending_ready=false;"
target="task_2_sending" />
  </state>
<state id="task_2_end">
  <transition cond="task_2_time_ex&&hard_rt" target="task_2_time_exceeded" />
  <transition cond="!task_2_time_ex||!hard_rt" target="task_2_exit" />
<transition cond="1" event="IRT_task_2_task_4_sending_end" target="task_4_waiting">
  <parametr name="IRT_task_2_task_4_sending_end" />
  </transition>
<transition cond="1" event="IRT_task_2_ready" target="task_4_waiting">
  <parametr name="IRT_task_2_ready" />
  </transition>
  </state>
  <state id="task_2_time_exceeded" />
<state id="task_2_exit">
  <transition cond="1" target="task_3_entry" />
  </state>
```

Рисунок 68. Фрагмент SCXML-файла, описывающий выполнение одного задания

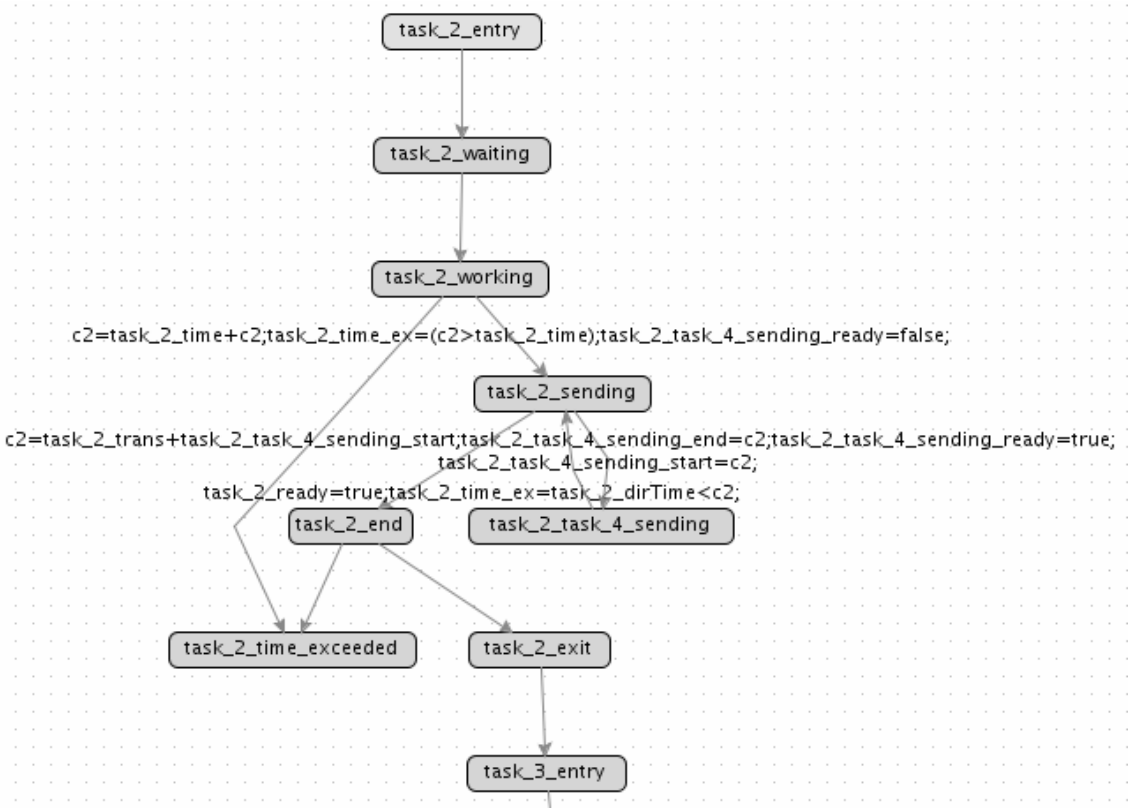

Рисунок 69. Визуализация фрагмента SCXML-файла, описывающего выполнение одного задания

5.8.4 Программная реализация

Общая схема интеграции средства синтеза архитектур и построения расписаний и среды выполнения моделей представлена на рисунке 70.

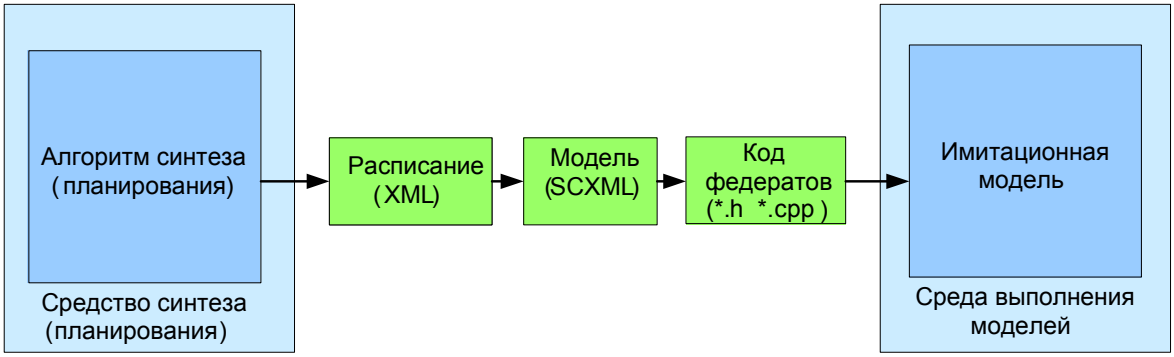

Рисунок 70. Схема интеграции средства синтеза архитектур и построения расписаний со средой выполнения моделей

Средство автоматической генерации модели РВС по заданному в формате XML файлу представляет собой программу на языке Python. В ходе выполнения программы содержимое входного файла представляется в виде модели DOM (Document Object Model) и по нему

строится модель DOM для целевого SCXML-файла. Считается, что заданное во входном файле расписание корректно. Основной функцией данной программы является функция createTask, строящая последовательность состояний, соответствующую одному заданию. Функция createTask вызывается циклически для всех заданий определенного процессора, и сгенерированные ею последовательности состояний объединяются в один последовательный автомат.

Рассмотрим саму схему интеграции. Когда в ходе работы средства построения требуется точное вычисление времен выполнения заданий расписания, происходит вызов функции, строящей XML-файл с расписанием в формате, описанном в разделе 5.8.2. Затем вызывается shell-скрипт, выполняющий последовательно следующие действия:

1. Запуск программы, строящей по XML-файлу с расписанием SCXML-файл с моделью РВС РВ.
2. Запуск программы, генерирующей по SCXML-файлу с моделью РВС исходный код федератов на C++, удовлетворяющий стандарту HLA. Для осуществления полностью автоматизированного запуска имитационных экспериментов данная программа была модифицирована так, что она также генерирует файлы CMakeLists.txt и launcher.py, использующиеся на следующих этапах работы скрипта. Содержимое данных файлов зависит от структуры модели РВС, поэтому их нельзя задать заранее. Подробно генерация кода федератов описана в разделе 4.3.
3. Запуск утилиты cmake с файлом CMakeLists.txt в качестве входного параметра. При этом происходит генерация Makefile-а для сборки исполняемых файлов федератов.
4. Сборка исполняемых файлов каждого федерата с помощью утилиты make.
5. Запуск скрипта launcher.py, осуществляющего автоматический запуск имитационного эксперимента.
6. Запуск скрипта, осуществляющего поиск в выведенном федератами тексте значений переменных, соответствующих искомым временам работы программных компонентов. Найденные значения записываются в определенный файл.

Далее продолжает работу средство синтеза архитектур и построения расписаний. Из сгенерированного на 6 этапе работы скрипта файла считываются искомые значения времен.

Описанная в данном разделе схема позволяет в ходе работы средства синтеза архитектур и построения расписаний проводить имитационные эксперименты и использовать их результаты. При этом средство синтеза архитектур и построения

расписанийдолжно лишь в нужный момент генерировать xml-файл с расписанием определенного в разделе 5.8.1 формата и вызывать скрипт, запускающий всю цепочку средств интеграции. Пример средства синтеза архитектур, для которого была апробирована данная схема, приведен в разделе 8.6, результаты экспериментов с этим средством – в разделе 9.9.

6 Исследуемые модели РВС РВ

В данном разделе описаны модели, разработанные в рамках данного НИР. Эти модели включают себя, как самые простые модели для тестирования разработанной среды моедлирования, так и модель бортовой многопроцессорной РВС РВ. Раздел 6.1 содержит описание тестовых моделей “Лавина” и “Пинг-Понг”. В разделе 6.2 приводится описание модели системы управления уличным движением на перекрёстке. Раздел 6.3 содержит описание модели поведения бортовых компьютеров автомобилей. В разделе 6.4 описана модель бортовой многопроцессорной РВС РВ.

6.1 Тестовые модели «Лавина» и «Пинг-Понг»

В качестве моделей для тестирования среды моделирования использовались модели «Лавина» и «Пинг-Понг», описанные в статье [148].

На рисунке 71 приведена диаграмма UML для модели «Лавина».

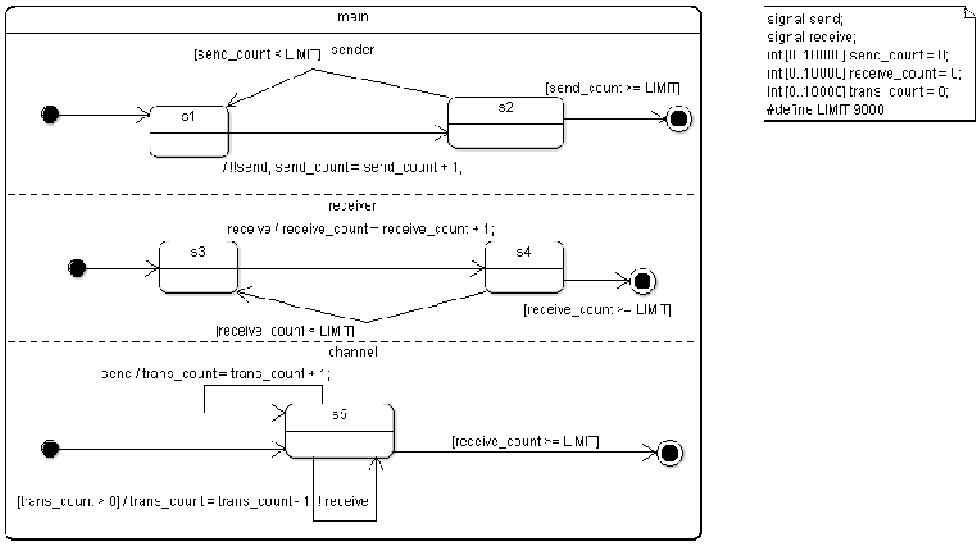

Рисунок 71. Диаграмма UML для модели "Лавина"

Модель содержит отправителя (sender) и получателя (receiver). Отправитель пересылает сообщения, пока из количество не достигнет предела (LIMIT), не ожидая ответа. Получатель обрабатывает сообщения.

На рисунке 72 приведена диаграмма UML для модели «Пинг-Понг».

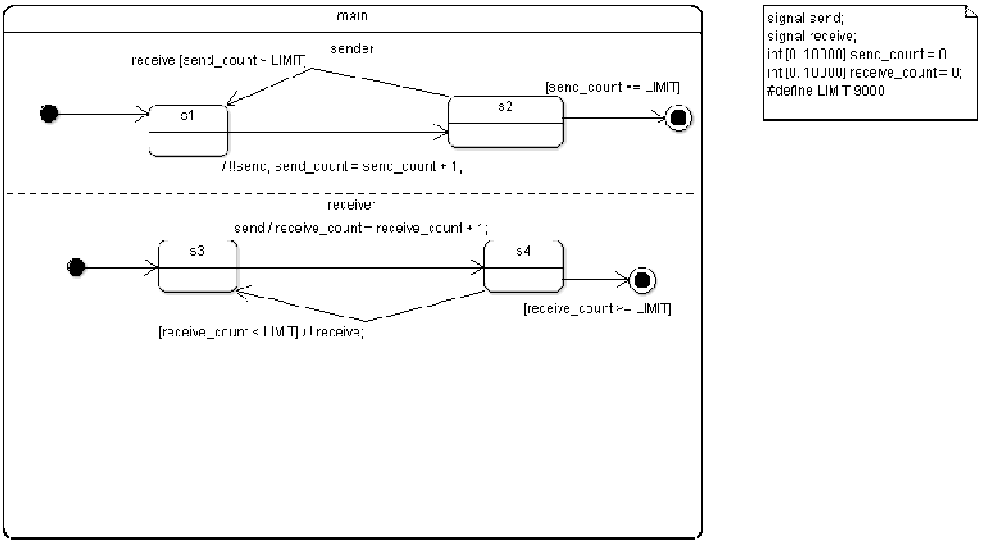

Рисунок 72. Диаграмма UML для модели "Пинг-Понг"

Модель содержит отправителя (sender) и получателя (receiver). Отправитель пересылает сообщения, пока из количество не достигнет предела (LIMIT), ожидая ответа на каждое. Получатель обрабатывает сообщения и посылает отправителю ответ.

6.2 Модель регулируемого перекрестка

В качестве одного из простых, но очень наглядных примеров РВС РВ нами была рассмотрена система управления уличным движением на перекрестке. Эта система призвана осуществлять правильное управление сигналами двух светофоров на перекрестке проспекта и улицы. В обычных условиях светофоры согласованно изменяют цвет сигнала так, чтобы периодически открывать движение по обеим магистралям. 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. В обычном режиме работы сигналы светофора чередуются строго периодически, поочередно открывая движение по обеим магистралям: зеленый свет горит на протяжении 45 условных единиц времени (у.е.в.) времени, желтый – на протяжении 5 у.е.в. Промежуток между циклами светофоров, во время которого они оба красные, составляет 1 у.е.в. 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» (appr); 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.

6.2.1 Описание диаграмм

Модель управляющей информационной системы реального времени, описывающая регулируемый перекресток, состоит из двух параллельно работающих компонентов — «скорой помощи» (ambulance) и перекрестка с двумя светофорами (crossroads) (рис. 73). Перекресток в свою очередь состоит из светофора на улице (street_light), светофора на проспекте (avenue_light) и управляющего устройства, обеспечивающего корректную работу светофоров (controller) (рис. 74); эти компоненты также работают параллельно. Остальные диаграммы приведены на рисунках 75-78.

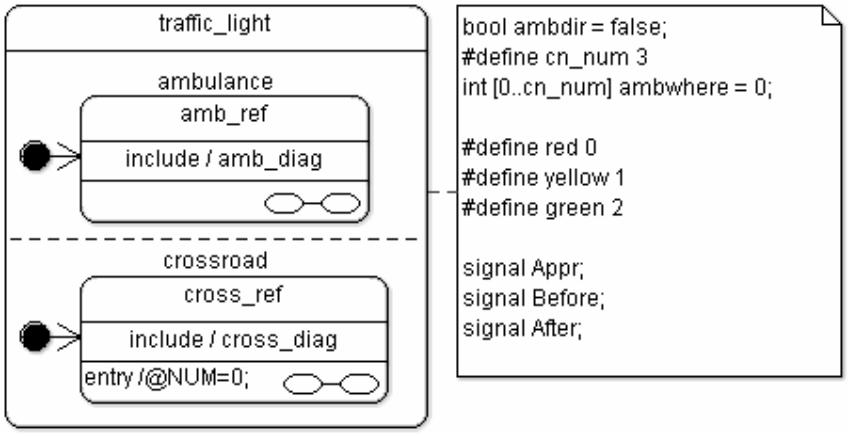

Рисунок 73. Главная диаграмма модели

Модель может быть расширена без особых усилий до следующей: вместо одного перекрестка рассматривается любое число (cn_num) идущих подряд перекрестков (то есть проспект пересекает несколько улиц). Для корректной работы также необходимо добавить нужное число параллельных регионов в диаграмму traffic_light, разместить в каждом из

параллельных регионов ссылку на диаграмму cross_diag и присвоить параметру NUM ссылки порядковый номер улицы (от 0 до (cn_num – 1)).

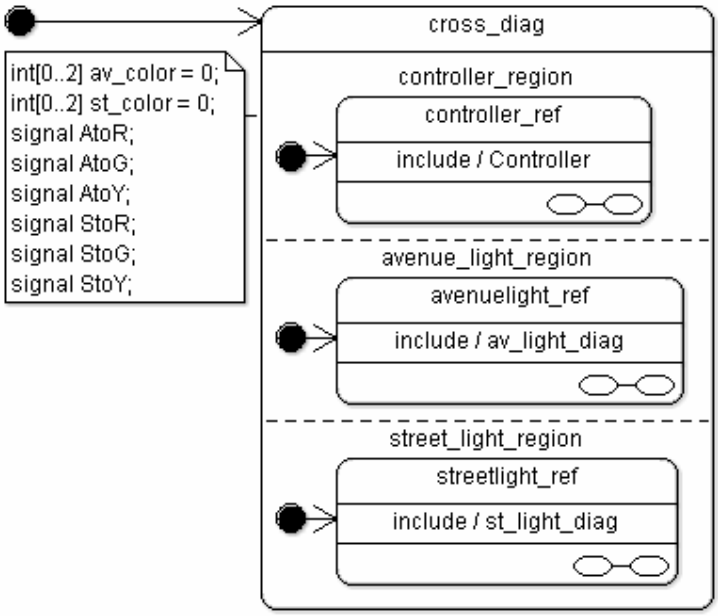

Рисунок 74. Диаграмма перекрестка

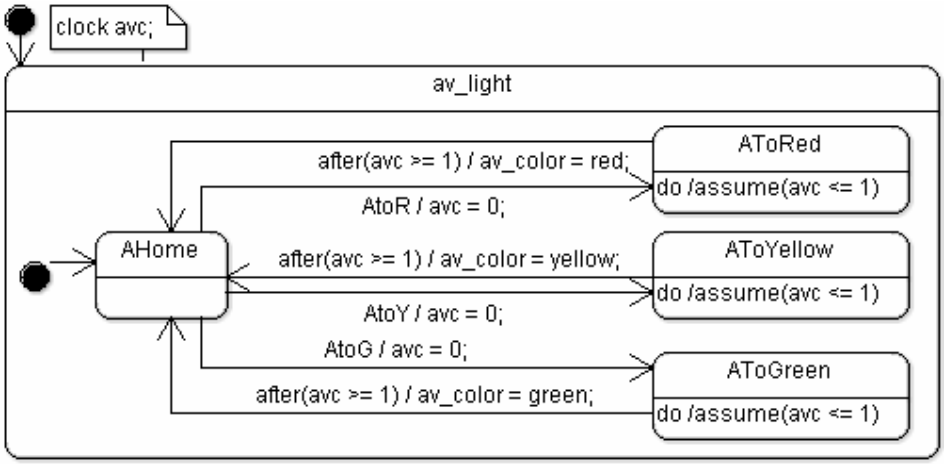

Рисунок 75 – Диаграмма светофора на проспекте

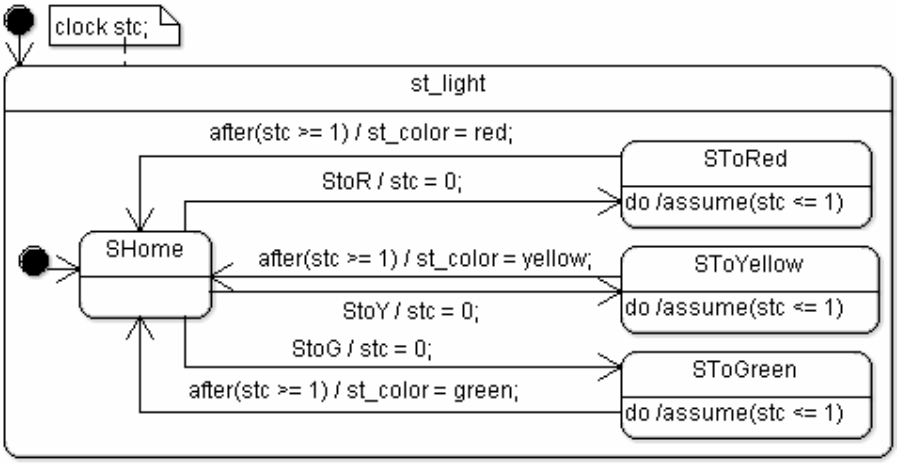

Рисунок 76. Диаграмма светофора на улице

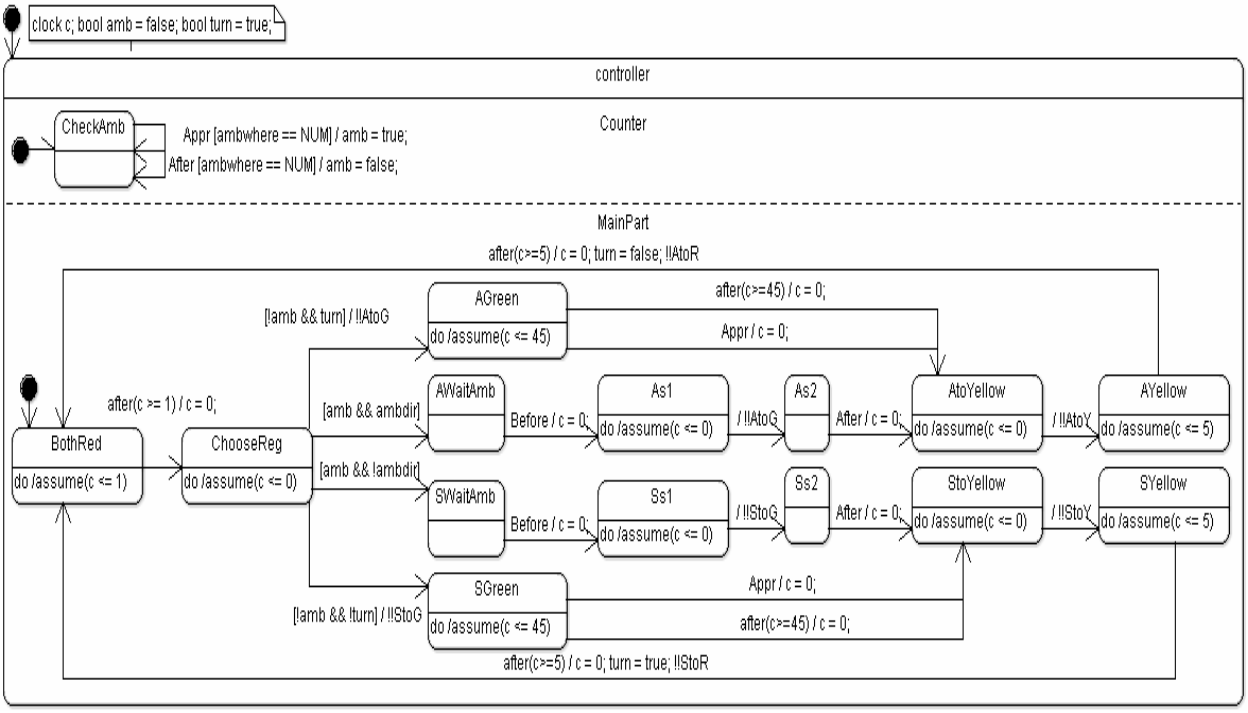

Рисунок77. Диаграмма управляющего устройства светофора

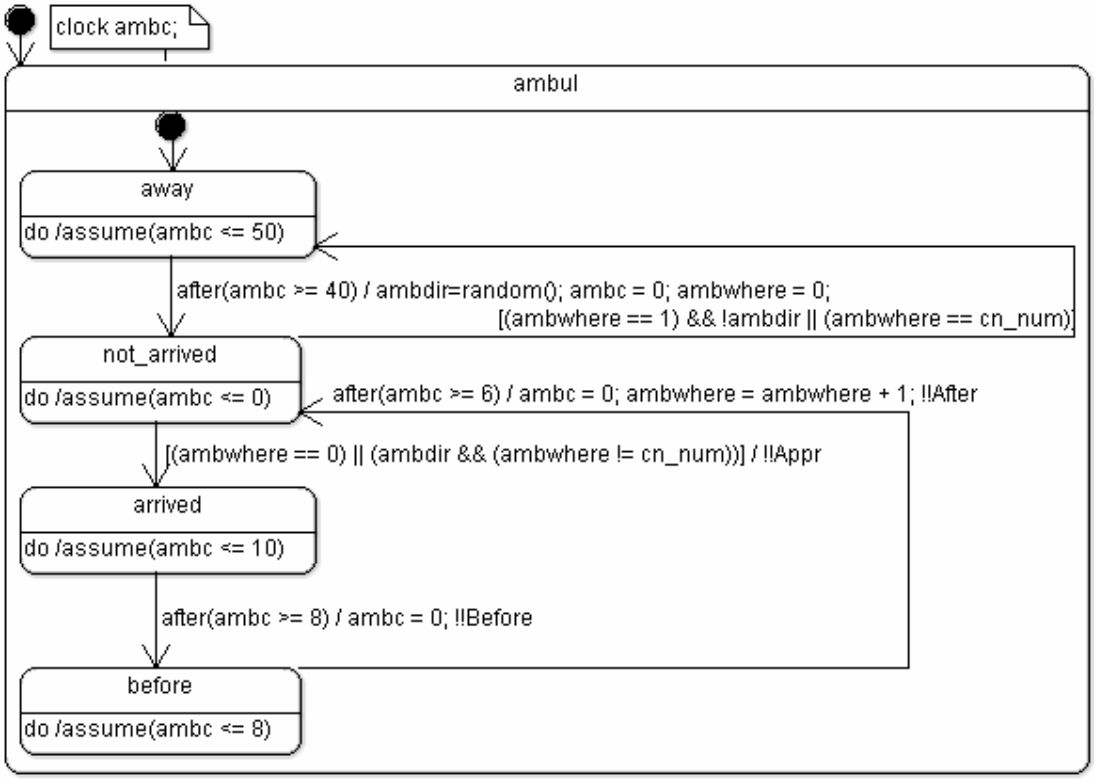

Рисунок 78. Диаграмма «скорой помощи»

В таблице 6 приведены данные модели. Все булевы переменные инициализируются значением true, все целочисленные переменные – нолями.

Таблица 6– Описание данных модели регулируемого перекрестка

Имя	**Тип**	**Локально для диаграммы**	**Содержательное описание**
ambdir	bool	traffic_light	движется ли скорая помощь по проспекту?
cn_num	макрос	traffic_light	число улиц, пересекающих проспект
ambwhere	int	traffic_light	номер улицы, на которой должна появиться «скорая помощь»
red	макрос	traffic_light	красный цвет светофора
yellow	макрос	traffic_light	желтый цвет светофора
green	макрос	traffic_light	зеленый цвет светофора
Appr	сигнал	traffic_light	«скорая» появилась на перекрестке
Before	сигнал	traffic_light	«скорая» подъезжает к перекрестку
After	сигнал	traffic_light	«скорая» проехала перекресток

Продолжение таблицы 6

Имя	Тип	Локально для диаграммы	Содержательное описание
NUM	параметр шаблона	cross_diag	порядковый номер улицы
av_color	int	cross_diag	текущий цвет светофора на проспекте
st_color	int	cross_diag	текущий цвет светофора на улице
AtoR	сигнал	cross_diag	светофор на проспекте должен стать красным
AtoG	сигнал	cross_diag	светофор на проспекте должен стать зеленым
AtoY	сигнал	cross_diag	светофор на проспекте должен стать желтым
StoR	сигнал	cross_diag	светофор на улице должен стать красным
StoG	сигнал	cross_diag	светофор на улице должен стать зеленым
StoY	сигнал	cross_diag	светофор на улице должен стать желтым
avc	таймер	av_light	время обработки сигнала переключения светофором на проспекте
stc	таймер	st_light	время обработки сигнала переключения светофором на улице
c	таймер	controller	время нахождения управляющего устройства в текущем режиме
amb	bool	controller	присутствует ли «скорая» на перекрестке?
turn	bool	controller	очередь светофора на проспекте быть зеленым?
ambc	таймер	ambul	время нахождения «скорой» в текущем режиме

Диаграммы **av_light** и **st_light** описывают обработчики сигналов смены цвета светофора и работают следующим образом. Диаграмма находится в состоянии Home, ожидая сигнала переключения цвета от управляющего устройства. Как только сигнал получен, диаграмма переходит в состояние обработки (состояние с суффиксом toRed, toYellow, toGreen, зависящем от цвета, который согласно сигналу должен появиться на светофоре). Ровно через 1 у.е.в. после этого происходит смена цвета (переменной с суффиксом color).

Работа управляющего устройства светофора (**controller**) такова. Параллельный регион counter отслеживает появление «скорой» на перекрестке, изменяя переменную amb. Второй параллельный регион работает так. Ровно 1 у.е.в. оба светофора на перекрестке красные (BothRed). Затем светофор немедленно выбирает режим работы (ChooseReg). В зависимости от того, есть ли на перекрестке «скорая» и чья очередь менять цвет, он выбирает обычный (верхний переход) или экстренный (второй сверху переход) режим работы светофора на

проспекте либо обычный (нижний переход) или экстренный (второй снизу переход) режим работы светофора на улице. В обычном режиме ровно 45 у.е.в. (или меньше, если на перекрестке появляется «скорая»; сигнал Appr) горит зеленый свет, затем ровно 5 у.е.в. горит желтый свет, затем опять достигается состояние BothRed. В экстренном режиме светофор становится зеленым, как только «скорая» подъезжает к перекрестку (сигнал Before); после ее отъезда (сигнал After) ровно 5 у.е.в. горит желтый свет, затем достигается состояние BothRed.

«Скорая помощь» (**ambul**) может находиться в четырех состояниях: отсутствует (away); вот-вот появится на перекрестке с порядковым номером ambwhere (not_arrived); появилась на этом перекрестке (arrived); подъехала к этому перекрестку (before). Отсутствовать «скорая» может от 40 до 50 у.е.в. В некоторый момент она прибывает, недетерминированно выбирает направление (ambdir) и начинает проезжать перекрестки по возрастанию порядковых номеров. В состоянии not_arrived в зависимости от направления и номера, записанного в ambwhere, «скорая» либо немедленно уезжает домой, либо немедленно появляется у перекрестка, посылая сигнал Appr. За 8-10 у.е.в. «скорая» подъезжает к перекрестку и посылает сигнал Before. Затем за 6-8 у.е.в. «скорая» проезжает перекресток и достигает состояния выбора not_arrived с увеличенным на единицу порядковым номером ambwhere.

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. Свойства корректности и безопасности поведения этой системы реального времени могут быть сформулированы следующим образом.

- 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.
- Когда автомобиль скорой помощи приближается к перекрестку, у обоих светофоров всегда включен красный свет.
- Во время пересечения перекрестка машиной скорой помощи на светофоре соответствующей магистрали всегда включен зеленый свет.
- На каждой магистрали всегда соблюдается правильный порядок смены цвета сигнала соответствующего светофора.
- Автомобиль скорой помощи всегда находится в движении.

Нами были проверены свойства системы, принадлежащие всем перечисленным типам. Результаты проверки приведены в разделе 9.6.

6.3 Модель поведения бортового компьютера автомобилей

В данном разделе приведена модель РВС РВ, описывающая поведение автомобилей на нерегулируемом перекрестке. За основу взята модель нерегулируемых перекрестков, предложенная в [149]. Компоненты модели, предложенной в [149], существенно используют арифметические вычисления над действительными числами. В связи с этим для возможности описания системы в виде диаграмм, подаваемых на вход разработанному нами средству, в модели был произведен ряд модификаций и абстракций. В следующих подразделах приведены содержательное и формальное описания модели, полученные в результате адаптации применительно к разработанной нами системе.

6.3.1 Содержательное описание модели

Два основных объекта модели – это перекресток и автомобиль.

Перекресток представляет собой пересечение двух однополосных дорог с прилегающей к пересечению достаточно длинной проезжей частью. Дорожное полотно разбито на отдельные секции, как обозначено на рисунке 79 красными прямоугольниками. Размеры дорожных секций моделируются минимальным временем, которое автомобиль должен потратить, чтобы проехать от начала до конца секции. В каждый момент времени в каждой секции пересечения дорог может находиться не более одного автомобиля. На прилегающей проезжей части при этом может находиться сколь угодно много автомобилей.

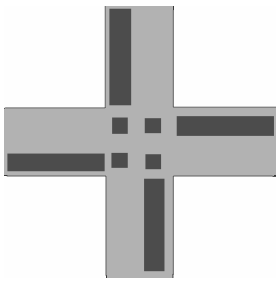

Рисунок 79 – Схема секций нерегулируемого перекрестка.

Каждый автомобиль моделируется двумя наборами параметров.

Первый набор описывает его физические характеристики: занимаемая секция, положение в секции и скорость. Положение в секции моделируется определением ближайшей секции, на которой может быть совершена полная остановка. Точное значение

скорости автомобиля заменено в модели тремя возможными скоростными характеристиками: стоит, едет и едет на максимальной скорости.

Второй набор параметров описывает управляющую часть бортового компьютера автомобиля. Для моделирования задержки при обработке поступающей информации управляющая часть разбита на два компонента: компонент принятия решений (водитель) и реагирующий компонент (обработчик). Водитель в каждый момент времени имеет доступ ко всей информации о ситуации на перекрестке и способен мгновенно принимать решения, однако не влияет напрямую на изменение физических характеристик машины. Обработчик имеет доступ к решению, принятому водителем, и способен изменять соответствующим образом скорость машины, однако делает это не мгновенно.

При появлении машины на перекрестке и до проезда перекрестка считается фиксированным направление движения машины: поворот направо, проезд прямо, поворот налево.

Одним из ограничений модели является ограничение сверху на количество машин, одновременно присутствующих на перекрестке. Это число определяется произвольным образом перед началом построения модели.

6.3.2 Учтенные правила дорожного движения

В модели учтены следующие правила, являющиеся упрощенной записью правил дорожного движения и правил движения «по договоренности».

- Водитель обязан уступить дорогу автомобилям, приближающимся справа.
- При повороте налево водитель обязан уступить дорогу автомобилям, движущимся со встречного направления и едущим прямо или направо.
- Если автомобиль, подъезжающий слева, не имеет возможности остановиться, чтобы пропустить водителя согласно первому правилу, то водитель должен пропустить этот автомобиль во избежание аварии.
- Аналогичное действие совершается для второго пункта.

Кроме того, в модели учтены и правила «здравого смысла», например, торможение, если попутная секция перекрестка занята.

6.3.3 Описание модели

Для краткости записи далее группы переменных будут объединены в массивы с использованием C-подобного синтаксиса. Для удобства чтения запись A[i][j] будет сокращаться до A[i,j]. Массив A элементов типа type с диапазоном индексов от a до b будет определяться как type[a..b]. В текущей реализации поддержка массивов отсутствует. Для

восстановления совокупности переходов из одного перехода, содержащего массив, достаточно подставить всевозможные значения переменных и, в случае если эти переменные не введены фиктивно (отличны от i, j), добавить к предусловию перехода ограничения на данные переменные.

Главная диаграмма (crossroads; рис. 80) состоит из двух паралелльно работающих компонентов – автомобиля (car) и окружения (controller). Окружение обрабатывает и предоставляет автомобилям информацию об общей ситуации на перекрестке через глобальные переменные.

В модель может быть введено любое наперед заданное число автомобилей. Для этого следует добавить нужное число параллельных регионов в диаграмму crossroads и в каждом параллельном регионе разместить такую же диаграмму, как и в регионе car, приведенном на рисунке 80.

Окружение (controller) состоит из следующих компонентов (рис. 81):

- counters – подсчитывает число автомобилей, участвующих в озвученных выше правилах дорожного движения;
- places – отслеживает занятость четырех центральных секций перекрестка и переходит в состояние *Accident* (авария), если машина пытается занять уже занятую секцию.

Автомобиль (рис. 82) задается набором физических характеристик (characteristics) и управляющим устройством (intentions). Кроме того, в связи с особенностями структуры модели автомобиль содержит компонент urgent_sender, обеспечивающий срабатывание помеченных особым образом переходов каждую условную единицу времени (у.е.в.).

Автомобиль имеет следующие физические характеристики (рис. 83): положение на перекрестке (position), скорость (speed) и ближайшая точка полной остановки (stop). Управляющее устройство автомобиля (рис. 84) разбито на два компонента: компонент принятия решений (driver) и обработчик (processor).

Остальные компоненты системы описываются последовательными диаграммами, приведенными на рисунках 85-92.

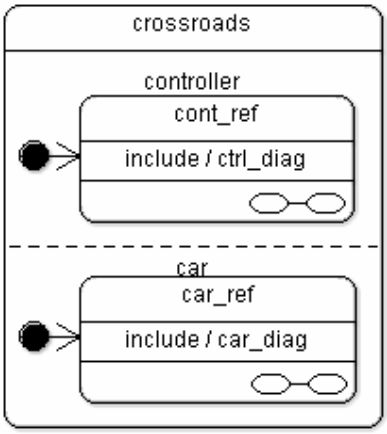

Рисунок 80. Главная диаграмма]

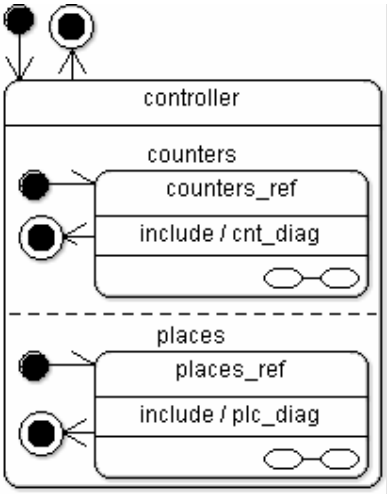

Рисунок 81. Диаграмма окружения

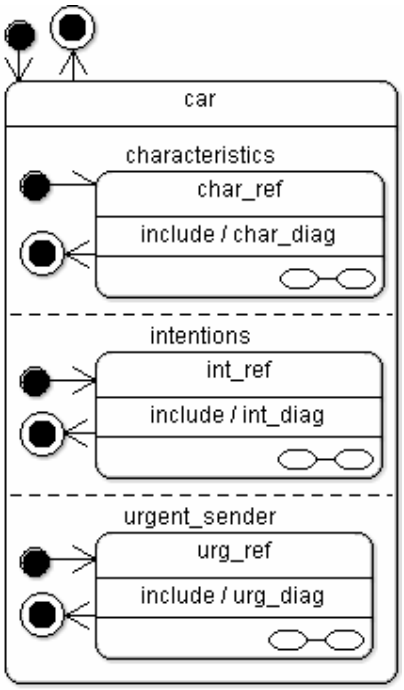

Рисунок 82. Диаграмма автомобиля

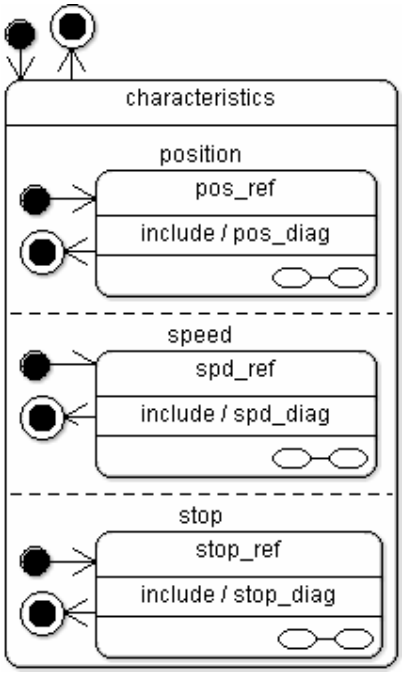

Рисунок 83. Диаграмма физических характеристик автомобиля

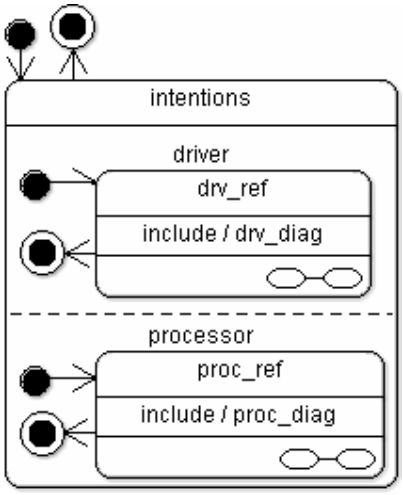

Рисунок 84. Диаграмма управляющего устройства автомобиля

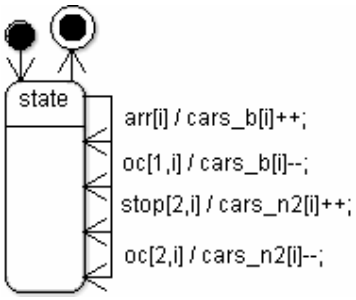

Рисунок 85. Диаграмма подсчета количества автомобилей (counters)

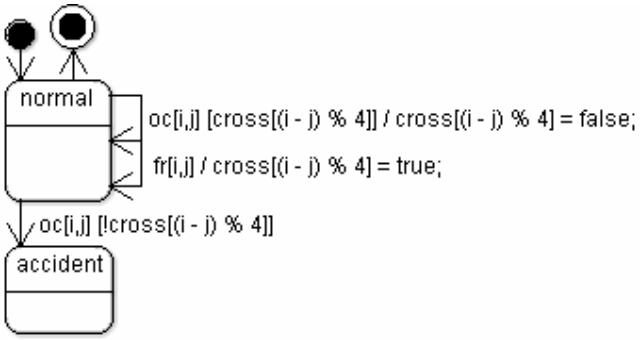

Рисунок 86. Диаграмма отслеживания занятости секций перекрестка (places)

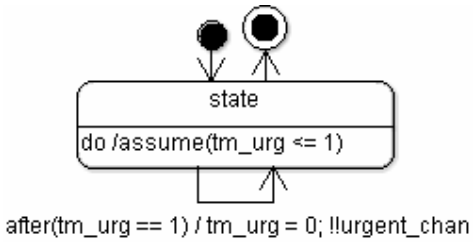

Рисунок87. Диаграмма обеспечения срочности переходов (urgent_sender)

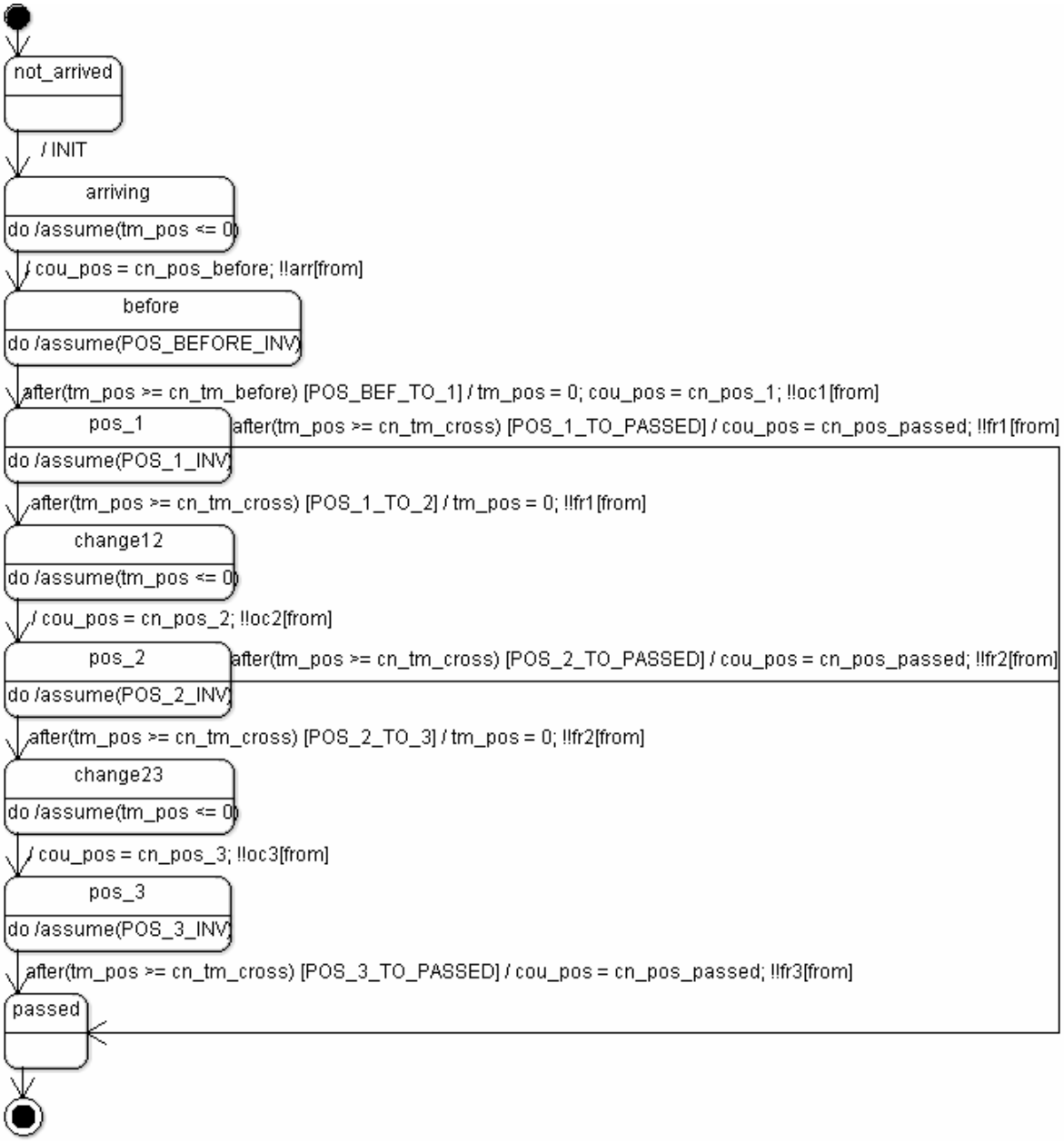

Рисунок 88. Диаграмма положения автомобиля (position)

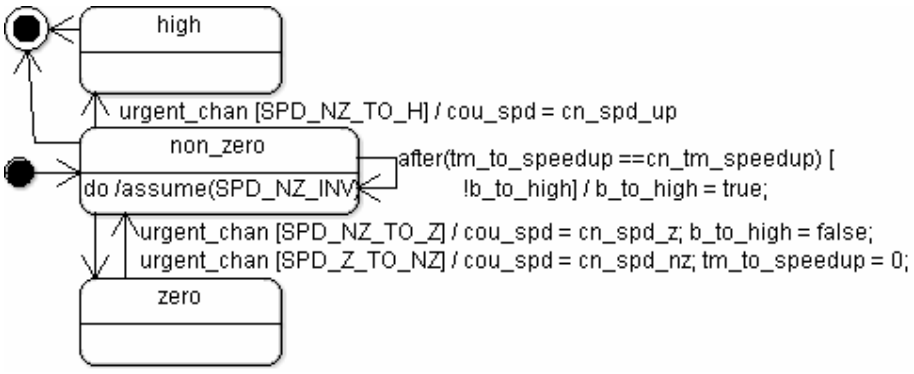

Рисунок 89. Диаграмма скорости автомобиля (speed)

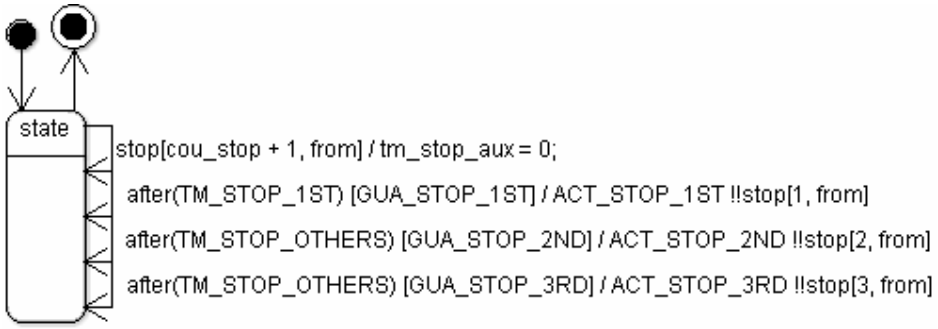

Рисунок 90. Диаграмма вычисления секции полной остановки автомобиля (stop)

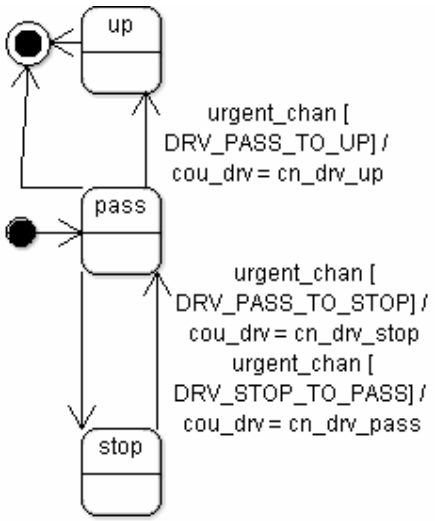

Рисунок 91. Диаграмма компонента принятия решений (driver)

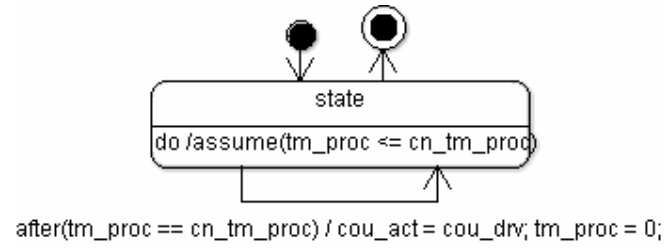

Рисунок 92. Диаграмма обработчика (processor)

Список данных модели, локальных для диаграммы crossroads, приведен в таблице 7; список данных, локальных для диаграммы car, - в таблице 8; список (глобальных) макроопределений – в таблицах 9-10. Все булевы переменные инициализируются значениями true, все переменные – нолями. Если индексы не поясняются, то в конце пояснения приводится ссылка на переменную с соответствующим значением.

Таблица 7 – Список данных модели, локальных для диаграммы crossroads

Имя	Тип	Пояснение
cross	bool[0..3]	свободна ли центральная секция перекрестка? правая нижняя, правая верхняя, левая верхняя и левая нижняя соответственно (согласно рисунку 79)
cars_b	int[0..3]	число машин, подъезжающих к перекрестку; from
cars_n2	int[0..3]	число машин, которые не могут остановиться раньше второй проезжаемой ими центральной секции; from
arr	signal[0..3]	сигнал появления машины у перекрестка; from
stop	signal[0..3,0..3]	сигнал, сообщающий о смене ближайшей секции полной остановки; первый индекс i – прилегающая часть (0) либо i-я

		проезжаемая центральная секция; второй индекс - from
fr	signal[1..3,0..3]	сигнал освобождения центральной секции перекрестка; первый индекс – номер секции по порядку проезда; второй индекс – from
oc	signal[1..3,0..3]	аналогично массиву сигналов fr с заменой «освобождения» на «захват»

Таблица 8 – список данных модели, локальных для диаграммы car

Имя	Тип	Пояснение
tm_pos	clock	время нахождения в текущей секции перекрестка
tm_proc	clock	время с момента последнего действия обработчика
tm_stop	clock	время с последней смены секции полной остановки
tm_stop_aux	clock	вычисляет время с последнего момента, в который попутная машина с тем же значением
tm_urg	clock	время с последнего выполнения срочных переходов
tm_up	clock	время, в течение которого скорость ненулевая
b_to_high	bool	можно ли сделать скорость максимальной?
from	int	откуда движется автомобиль; 0 – справа, 1 – снизу, 2 – слева, 3 – сверху
to	int	куда движется автомобиль; 0 – направо, 1 – наверх, 2 – налево
cou_drv	int	решение водителя; 0 – ехать с постоянной скоростью, 1 – разгоняться, 2 – тормозить
cou_act	int	действие обработчика; значение идентично cou_drv
cou_pos	int	положение автомобиля; 0 – не появился, 1 – на прилегающей части, 2 – на первой проезжаемой центральной части, 3 – на второй, 4 – на третьей, 5 – проехал перекресток
cou_stop	int	ближайшая секция, на которой возможна полная остановка автомобиля; значения – от 1 до 5, совпадают с cou_pos
cou_spd	int	скорость автомобиля; 0 – ненулевая, 1 – нулевая, 2 - максимальная

Таблица 9 – список макроопределений модели, заменяемых на константы

Имя	Значение	Имя	Значение	Имя	Значение
cn_tm_before	4	cn_to_l	2	cn_pos_3	4
cn_tm_cross	1	cn_act_pass	0	cn_pos_passed	5
cn_tm_proc	2	cn_act_up	1	cn_stop_before	1
cn_tm_speedup	2	cn_act_stop	2	cn_stop_1	2
cn_tm_dist	1	cn_drv_pass	0	cn_stop_2	3
cn_from_r	0	cn_drv_up	1	cn_stop_3	4
cn_from_d	1	cn_drv_stop	2	cn_stop_passed	5
cn_from_l	2	cn_pos_na	0	cn_spd_nz	0
cn_from_u	3	cn_pos_before	1	cn_spd_z	1
cn_to_r	0	cn_pos_1	2	cn_spd_up	2
cn_to_s	1	cn_pos_2	3		

Таблица 10 – Список макроопределений-сокращений

Имя	Значение
INIT	tm_pos = 0; tm_stop = 0; tm_stop_aux = 0; tm_proc = 0; tm_urg = 0; b_to_high = false; cou_act = cn_act_pass; cou_drv = cn_drv_pass; cou_stop = cn_stop_before; cou_spd = cn_spd_nz; cou_pos = cn_pos_before; from=random(); to=random();
POS_BEFORE_INV	(tm_pos <= cn_tm_before) || !(cou_spd == cn_spd_up) && (cou_stop == cn_stop_before)
POS_BEF_TO_1	(cou_spd != cn_spd_z) && cross[(1 – from) % 4] && (cou_stop > cn_stop_before)
POS_1_TO_PASSED	(to == cn_to_r) && (cou_spd != cn_spd_z)
POS_1_INV	(tm_pos <= cn_tm_cross) || !(cou_spd == cn_spd_up) && (cou_stop == cn_stop_1)
POS_1_TO_2	(to != cn_to_r) && (cou_spd != cn_spd_z) && cross[(2 – from) % 4] && (cou_stop > cn_stop_1)
POS_2_TO_PASSED	(to == cn_to_s) && (cou_spd != cn_spd_z)
POS_2_INV	(tm_pos <= cn_tm_cross) || !(cou_spd == cn_spd_up) && (cou_stop == cn_stop_2)
POS_2_TO_3	(to == cn_to_l) && (cou_spd != cn_spd_z) && cross[(3 – from) % 4] && (cou_stop > cn_stop_2)
POS_3_INV	(tm_pos <= cn_tm_cross) || !(cou_spd == cn_spd_up) && (cou_stop == cn_stop_3)
POS_3_TO_PASSED	(cou_spd != cn_spd_nz)
SPD_NZ_TO_H	b_to_high && (cou_act == cn_act_up)
SPD_NZ_INV	b_to_high || tm_to_speedup <= cn_tm_speedup
SPD_NZ_TO_Z	(cou_act == cn_act_stop) && ((cou_pos == cn_pos_before) && (cou_stop < cn_stop_1) || (cou_pos == cn_pos_1) && (cou_stop < cn_stop_2) || (cou_pos == cn_pos_2) && (cou_stop < cn_stop_3))
SPD_Z_TO_NZ	(cou_act != cn_act_stop)
TM_STOP_1ST	tm_stop >= cn_tm_before && tm_stop_aux >= cn_tm_dist
GUA_STOP_1ST	(cou_pos != cn_pos_na) && (cou_stop == cn_stop_before) && (cou_spd != cn_spd_z) && cross[(1 – from) % 4] && (cou_act != cn_act_stop)
ACT_STO_1ST	tm_stop = 0; tm_stop_aux = 0; cou_stop = cn_stop_1;
TM_STOP_OTHERS	tm_stop >= cn_tm_cross && tm_stop_aux >= cn_tm_dist
GUA_STOP_2ND	(to != cn_to_r) && (cou_stop == cn_stop_1) && (cou_spd != cn_spd_z) && cross[(2 – from) % 4] && (cou_act != cn_act_stop)
ACT_STOP_2ND	tm_stop = 0; tm_stop_aux = 0; cou_stop = cn_stop_3;
GUA_STOP_3RD	(to == cn_to_l) && (cou_stop == cn_stop_2) && (cou_spd != cn_spd_z) && cross[(3 – from) % 4] && (cou_act != cn_act_stop)
ACT_STOP_3RD	tm_stop = 0; tm_stop_aux = 0; cou_stop = cn_stop_2;
DRV_PASS_TO_UP	(cou_pos != cn_pos_na) && ((to != cn_to_r) && (cars_b[(from – 1) % 4] == 0) && (cou_stop > cn_stop_1) || (to == cn_to_l) && (cars_b[(from - 2) % 4] == 0) && (cou_stop > cn_stop_2) || (cou_pos == cn_pos_1 + var_to))
DRV_PASS_TO_STOP	(cou_pos != cn_pos_na) && ((to == cn_to_l) && (cou_stop <= cn_stop_2) && (cars_b[(from - 2) % 4] != 0) || (cars_n2[(from + 1) % 4] != 0) && (cou_stop <= cn_stop_bef) || (to != r) && (cou_stop <= cn_stop_1) && (cars_b[(from - 1) % 4] != 0))
DRV_STOP_TO_PASS	((cou_pos >= cn_pos_bef) || (cars_n2[(from + 1) % 4] == 0) && cross[(1 - from) % 4]) && ((cou_pos >= cn_pos_1) || (to != cn_to_r) || (cars_b[(from – 1) % 4] == 0)) && ((cou_pos >= cn_pos_2) || (to != cn_to_l) || (cars_b[(from - 2) % 4] == 0))

Диаграмма **counters**, принимая сигналы от машин, считает число машин в секциях, прилегающих к перекрестку, и число машин, которые не могут остановиться ранее второй проезжаемой секции и еще ее не проехали.

Диаграмма **places**, принимая сигналы от машин, меняет значения элементов массива cross. Если при этом поступает сигнал захвата уже занятой секции, диаграмма переходит в ошибочное состояние (accident), констатируя аварию.

Диаграмма **urgent_sender** каждую у.е.в. посылает сигнал по каналу urgent_chan, принуждая к выполнению все переходы, помеченные триггером приема по этому каналу.

Диаграмма **position** работает следующим образом. Прежде всего, автомобиль появляется в системе, но не на перекрестке (not arrived). Через произвольное время он подъезжает к перекрестку (before), при этом инициализируются все переменные и таймеры, недетерминированно выбираются значения переменных from и to и посылается сигнал появления (arr). Затем с некоторыми задержками автомобиль начинает последовательно проезжать центральные секции перекрестка и проезжает его, посылая сигналы о захвате и освобождении этих секций. Если при этом автомобиль едет на максимальной скорости, то задержки смены секций становятся также и ограничениями сверху на время пребывания в них.

Диаграмма **speed** регулирует скорость автомобиля, поднимая и опуская ее согласно учтенным правилам движения и решению, принятому в данный момент обработчиком. При этом максимальной скорости нельзя достичь мгновенно.

Диаграмма **stop** увеличивает номер секции, на которой может остановиться автомобиль, с учетом задержек, связанных с ограничением максимальной скорости и соблюдением дистанции между попутными автомобилями. При этом рассылаются соответствующие сигналы.

Диаграмма **driver** регулирует намерения водителя (ехать с постоянной скоростью, разогнаться или остановиться) согласно учтенным правилам дорожного движения.

Диаграмма **processor** с определенной периодичностью считывает намерение водителя (cou_drv) и устанавливает соответствующее действие (cou_act).

Целью разработки данной системы реального времени является проверка ее темпоральных свойств, таких как свойства безопасности и свойства живости. Результаты этой проверки приведены в разделе 9.6.

6.4 Модель многопроцессорной БВС

В данном разделе описывается модель многопроцессорной БВС. В разделе 6.4.1 приведено общее описание модели. Разделы 6.4.2, 6.4.3, 6.4.4 и 6.4.5 содержат описание вычислителей C_1, C_2, C_3 и C_4, входящих в состав модели.

6.4.1 Общее описание модели

РВС РВ состоит из четырех вычислителей C_1, C_2, C_3 и C_4, подключенных к общей шине. Процессоры C_2 и C_3 имеют общую память. Каждый из вычислителей выполняет свой круг задач. Вычислитель C_1 – главный процессор, управляющий вычислениями и передачей данных во всей системе. Процессор C_2 вычисляет параметры полета и готовит данные для датчиков. Процессор C_3 определяет положение самолета по показаниям датчиков и обеспечивает перемещение по требуемому маршруту, а также управляет сенсорами. Процессор C_4 обрабатывает информацию с радара и управляет полетом самолета на малой высоте.

Глобальная диаграмма модели представлена на рисунке 93. Данная модель абстрагируется от конкретных параметров полета и специфицирует преимущественно на коммуникации между процессорами. Поведение программы при вычислениях задано в виде набора состояний, соответствующих различным операциям, но их функционирование более детально не определяется

.

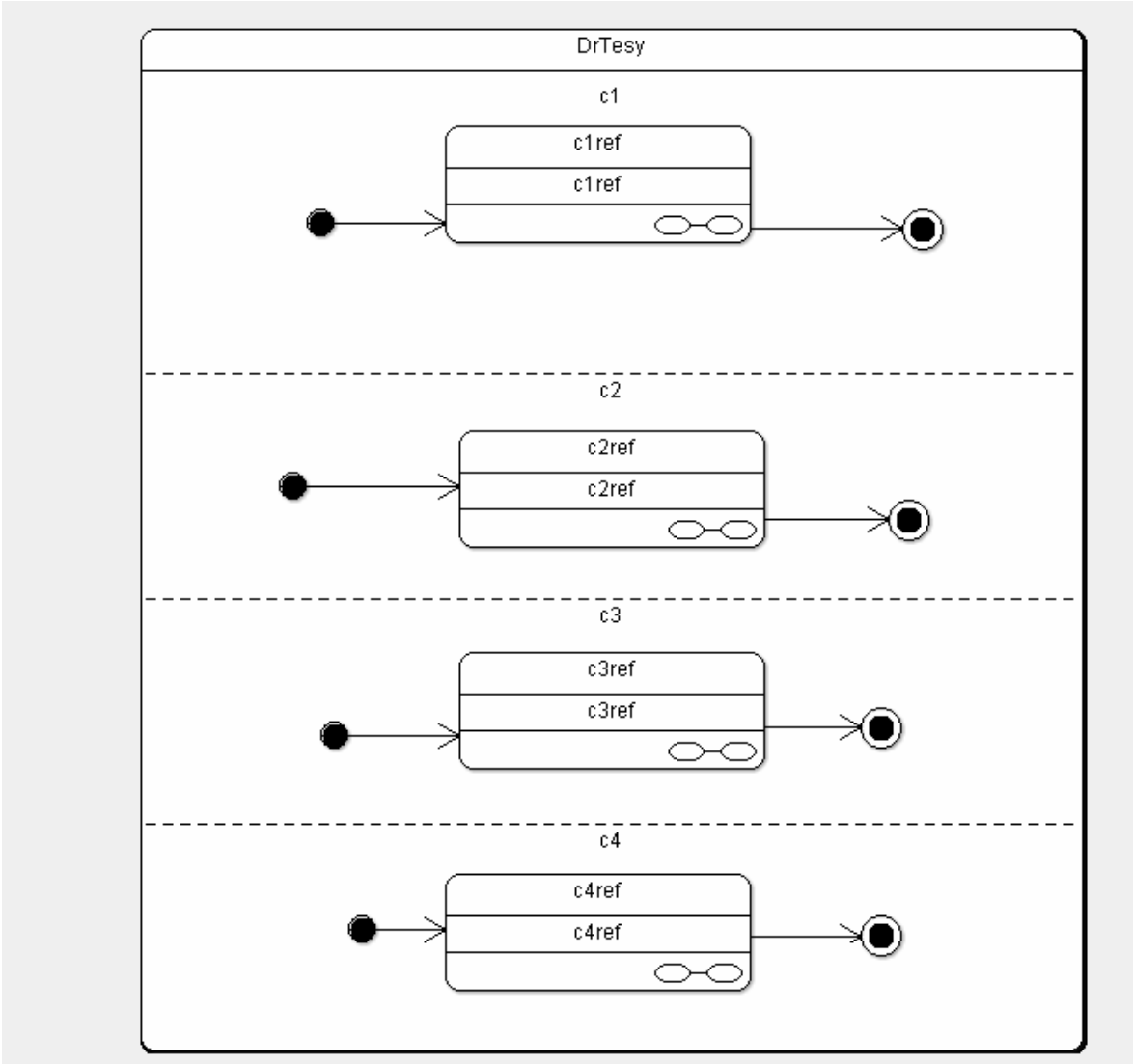

Рисунок 93. Глобальная диаграмма.

Типы данных, используемых в рассматриваемой распределенной системе таковы.

Com_Inter_C2 – прерывания для процессора C_2.

Send_status2 = 0

Send_Param1 = 1

Send_Param2 = 2

Receive_param = 3

Init_C2 = 4

Com_Inter_C3 – прерывания для процессора C_3.

Send_status3 = 0

TVS_com = 1

HVP_com = 2

FLR_com = 3

Init_C3 = 4

Com_Inter_C4 – прерывания для процессора C_4.

Send_status4 = 0

Send_data = 1

Receive_data = 2

AAR_radiate = 3

Permiss_radiate1 = 4

Permiss_radiate2 = 5

Init_C4 = 6

Com_Inter_Sensor – прерывания для сенсоров.

Send_data = 0

Send_statusSensor = 1

Mode1 – режим 1.

Cancel = 0

FLR_TVS = 1

HVP = 2

Mode2 – режим 2.

SRNS = 0

Next_WP = 1

Online_WP = 2

Manual = 3

Mode3 – режим 3.

Horizontal = 0

Aux_PRA = 1

Main_AAR = 2

None = 3

Глобальные переменные

int [0..4] COMC2_value = 0;

int [0..4] COMC3_value = 0;

int [0..6] COMC4_value = 0;

int [0..10] C4_data1 = 0;

int [0..10] C4_data2 = 0;

int [0..10] C4_data3 = 0;

int [0..10] SENSOR_value = 0;

int [0..10] Sensor_data = 0;

```
int [0..2] R1_data = 0;
int [0..3] R2_data = 0;
int [0..3] R3_data = 0;
bool sharedMemory = false;
bool REQUEST_C2 =  false;
bool REQUEST_C3 =  false;
```

6.4.2 Процессор C_1

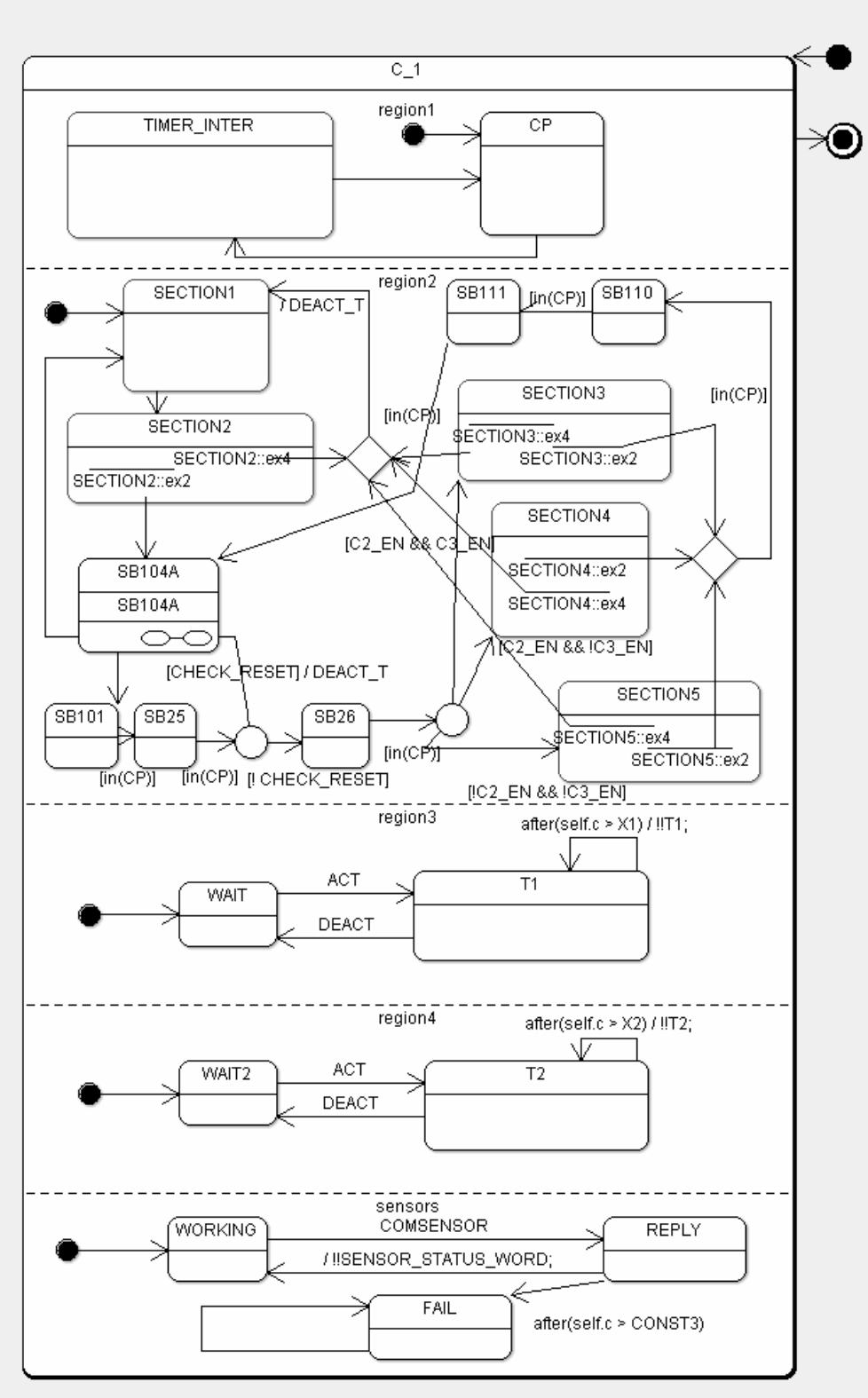

Рисунок 95. Общая диаграмма C_1.

Модель процессора C_1 представлена на рисунках 95-106. После инициализации процессор C_1 работает в бесконечном цикле. Параллельно с основным циклом выполняется

процесс, изображенный на первом из параллельных регионов. Этот процесс определяет, в каком режиме находится процессор: в рабочем (CP) или в режиме прерывания (TIMER_INTER). Благодаря этому процессу можно промоделировать прерывание по таймеру: ко всем переходам в основном цикле добавлено предусловие in(C_1.CP), гарантирующее, что процессор находится в обычном режиме. Если процессор обрабатывает прерывание, то основной цикл дожидается окончания обработчика. Процессор C_1 содержит два таймера, которые инициируют прерывания с частотой, определяющейся константами X1 и X2 соответственно. Значения констант равны 1 и 50 миллисекундам.

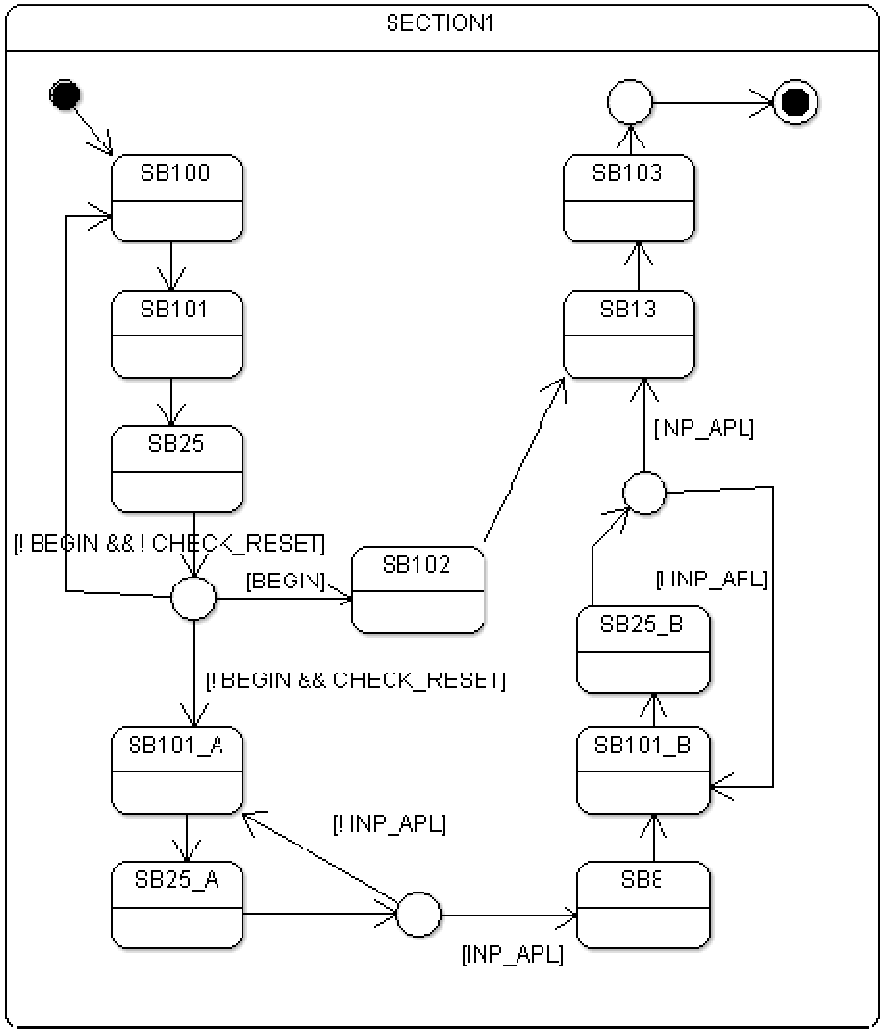

Рисунок 96. Первая секция главного цикла C_1.

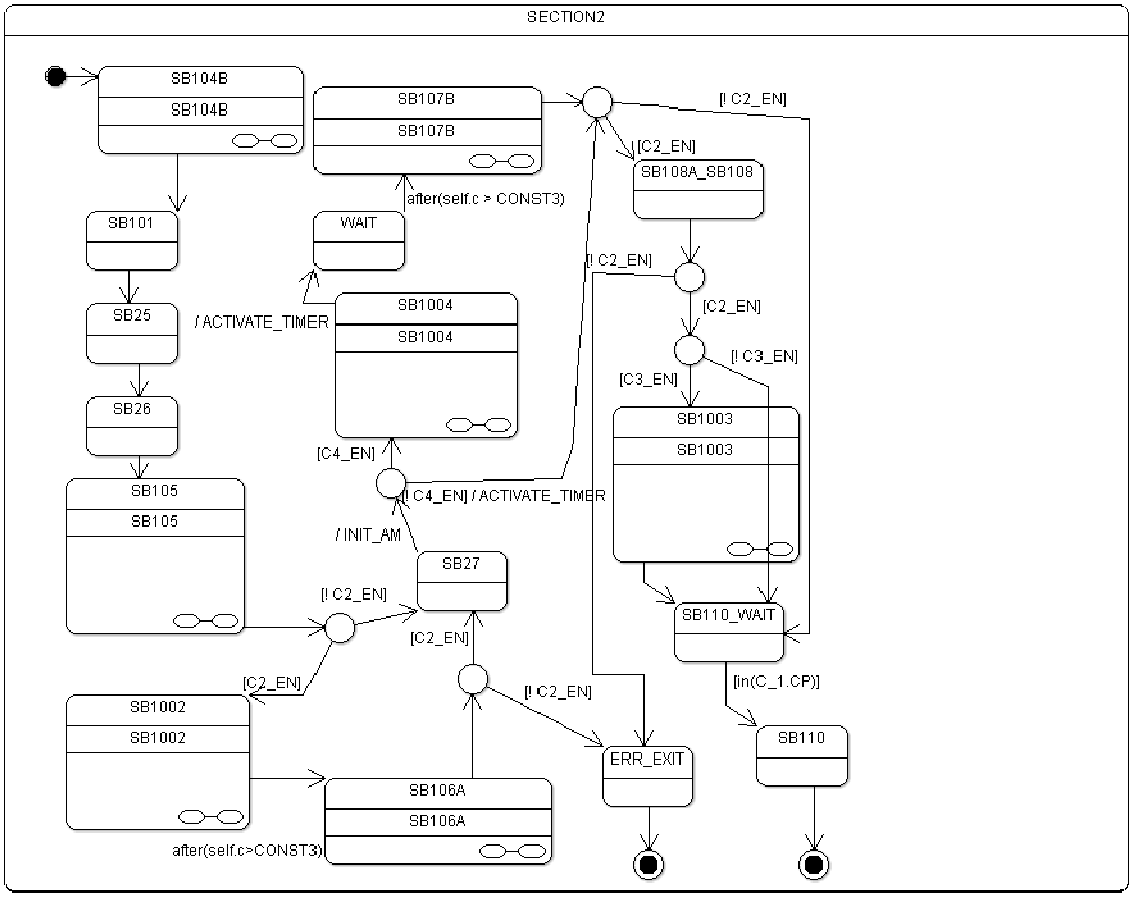

Рисунок 97. Вторая секция главного цикла C_1.

Главный цикл содержит пять основных секций, которые описывают различные сценарии поведения в зависимости от того, работают ли другие вычислители. Секции 1 и 2 (Section1, Section2) используются для загрузки данных и инициализации удаленных вычислителей. В экстренном случае, когда C_2 или C_3 теряют работоспособность, C_1 берет на себя их вычислительные задачи. Соответствующие алгоритмы содержатся в секциях 3, 4, 5. Переход в эти секции происходит при выполнении соответствующих предусловий (C2_EN, C3_EN), а флаги в них устанавливаются в блоках, отвечающих за коммуникации (SB106-109).

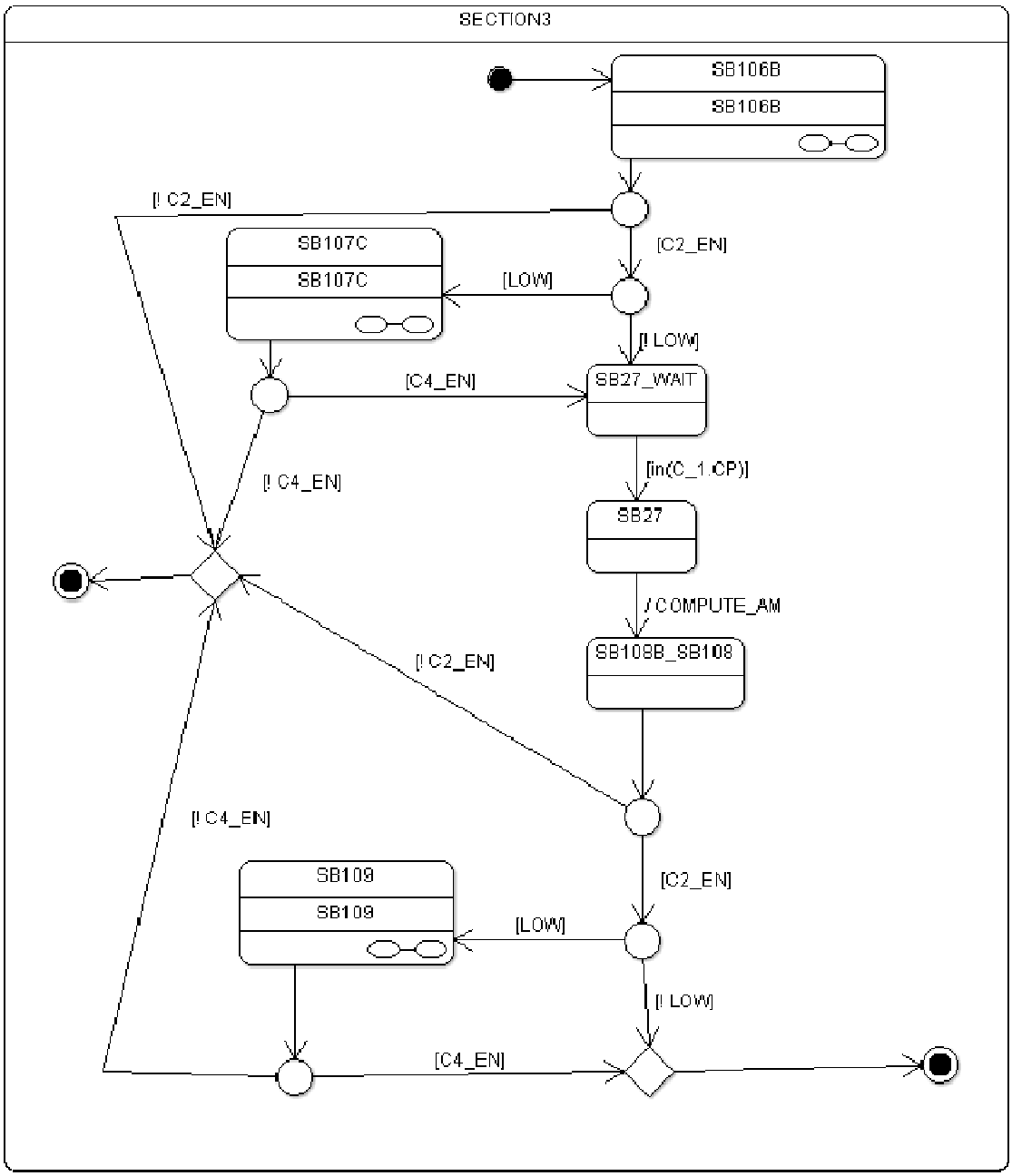

Рисунок 98. Третья секция главного цикла C_1.

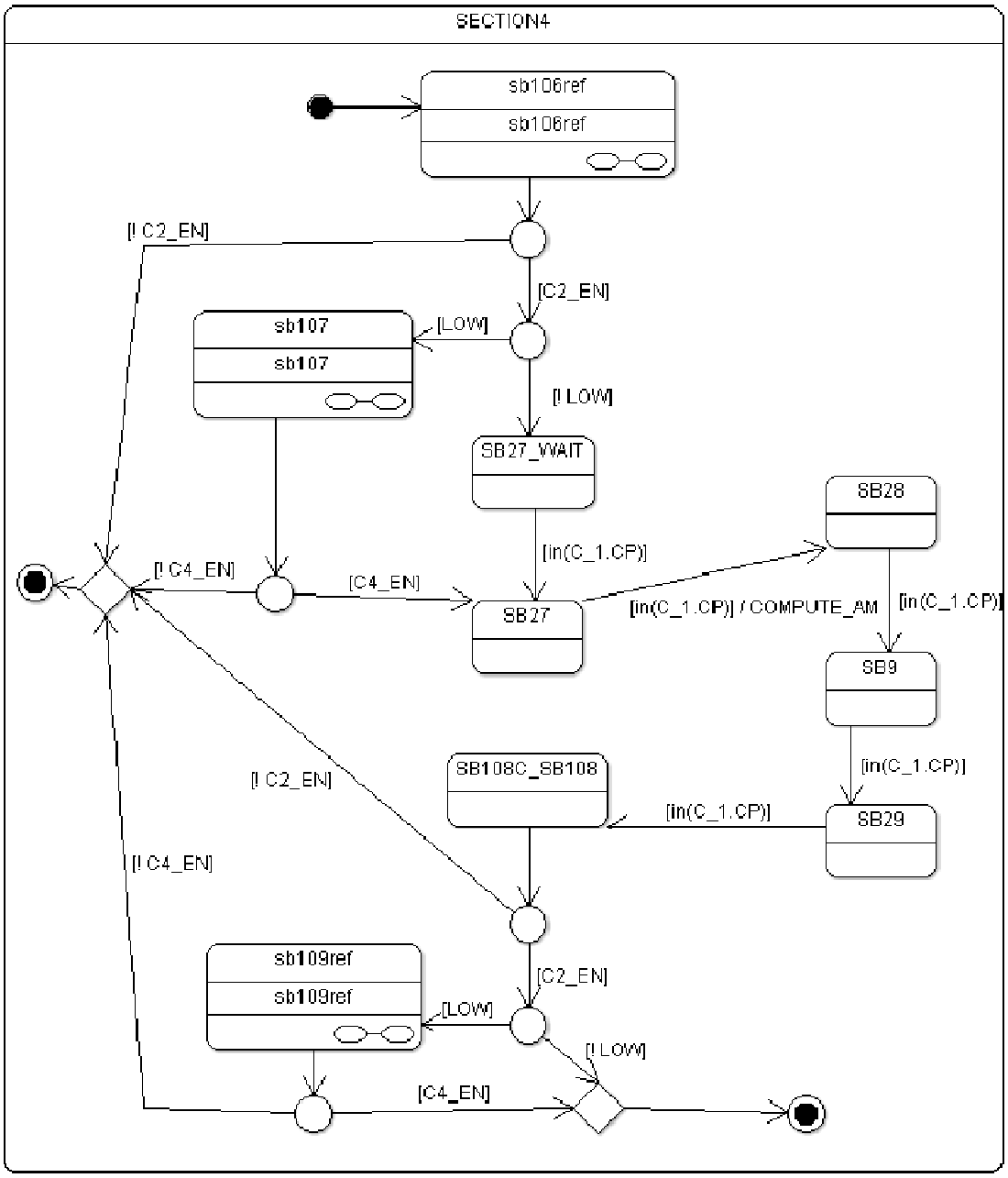

Рисунок 99. Четвертая секция главного цикла C_1.

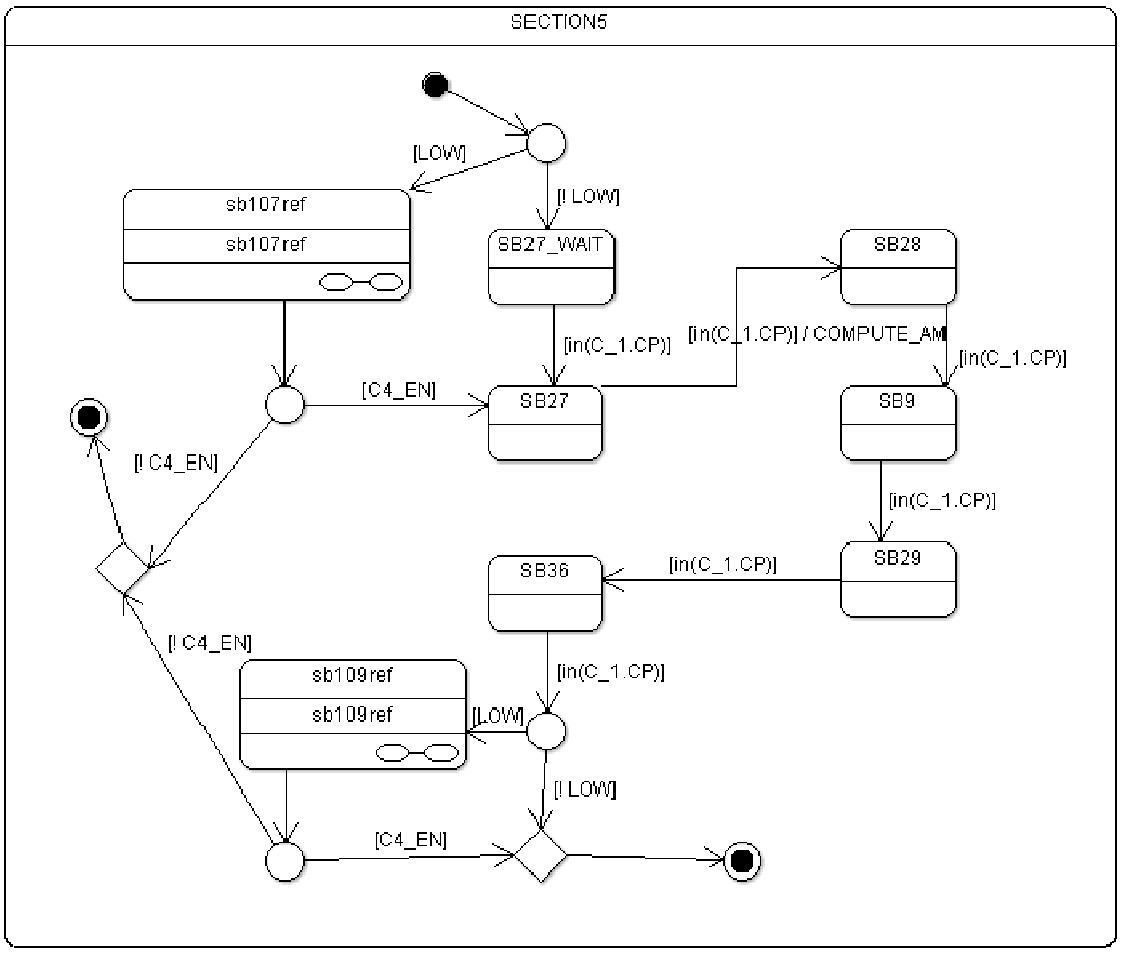

Рисунок 100. Пятая секция главного цикла C_1.

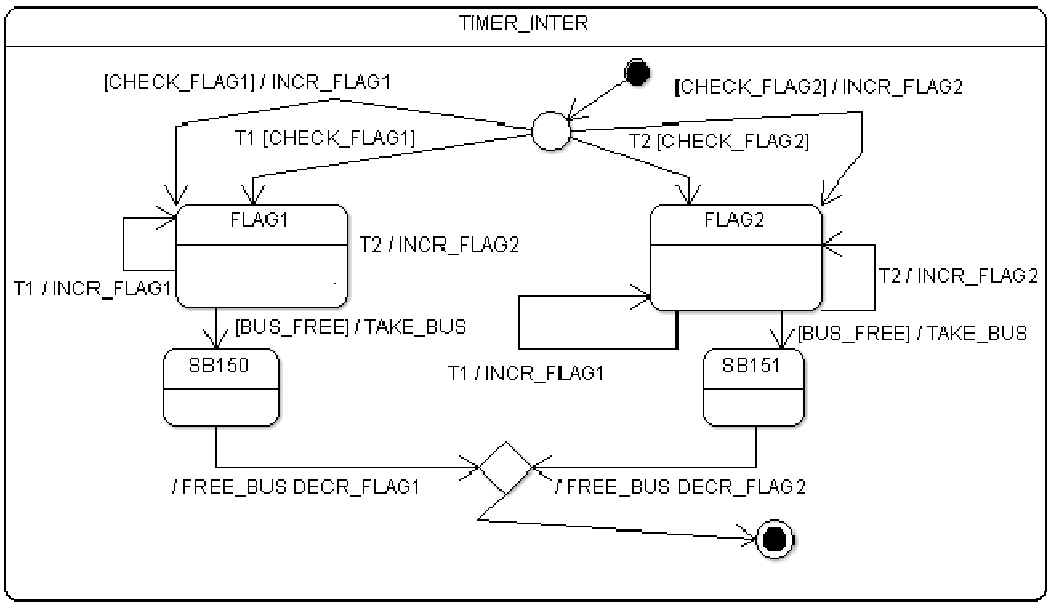

Рисунок 101. Прерывания по таймеру в C_1.

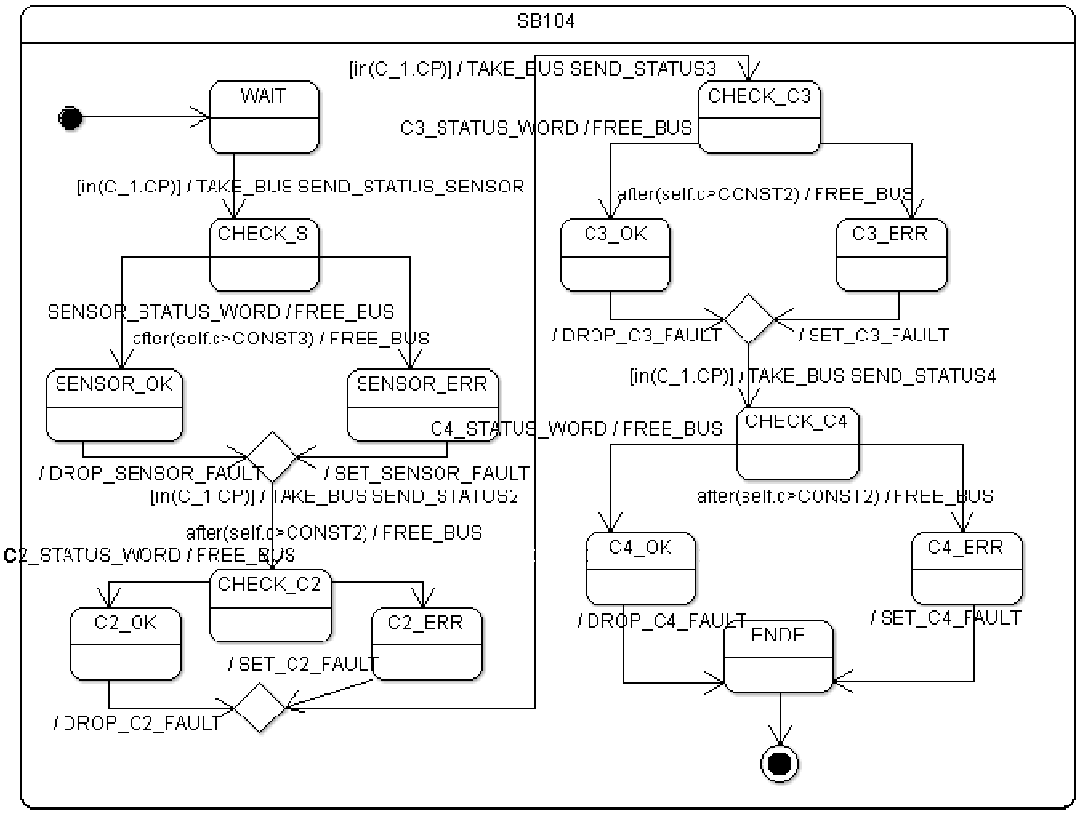

Рисунок 102. Специальный блок 104.

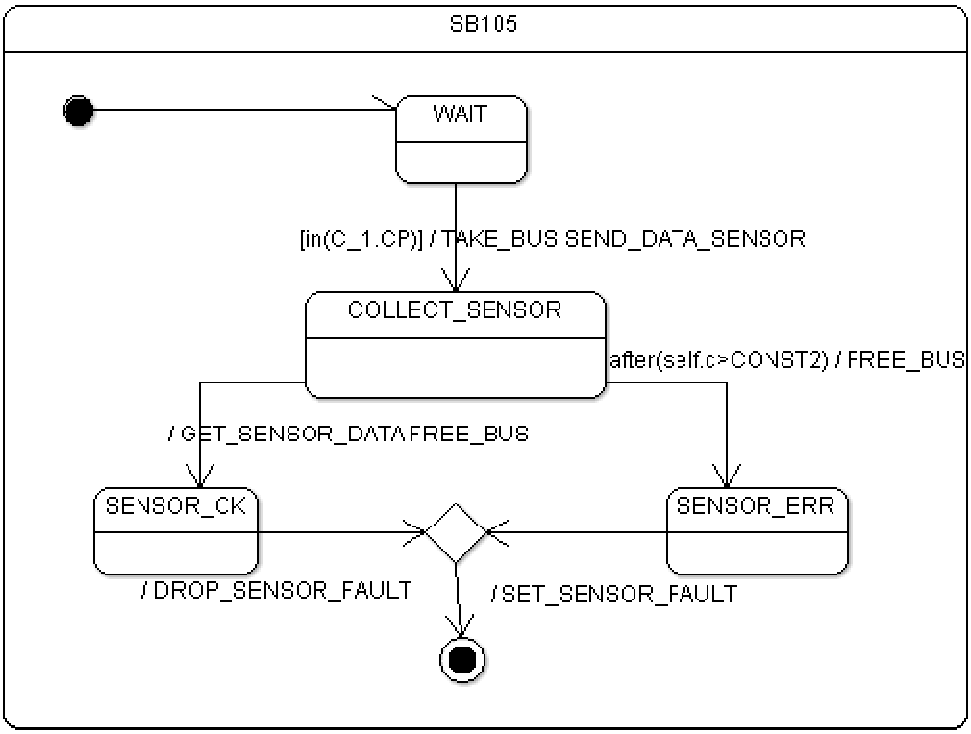

Рисунок 103. Специальный блок 105.

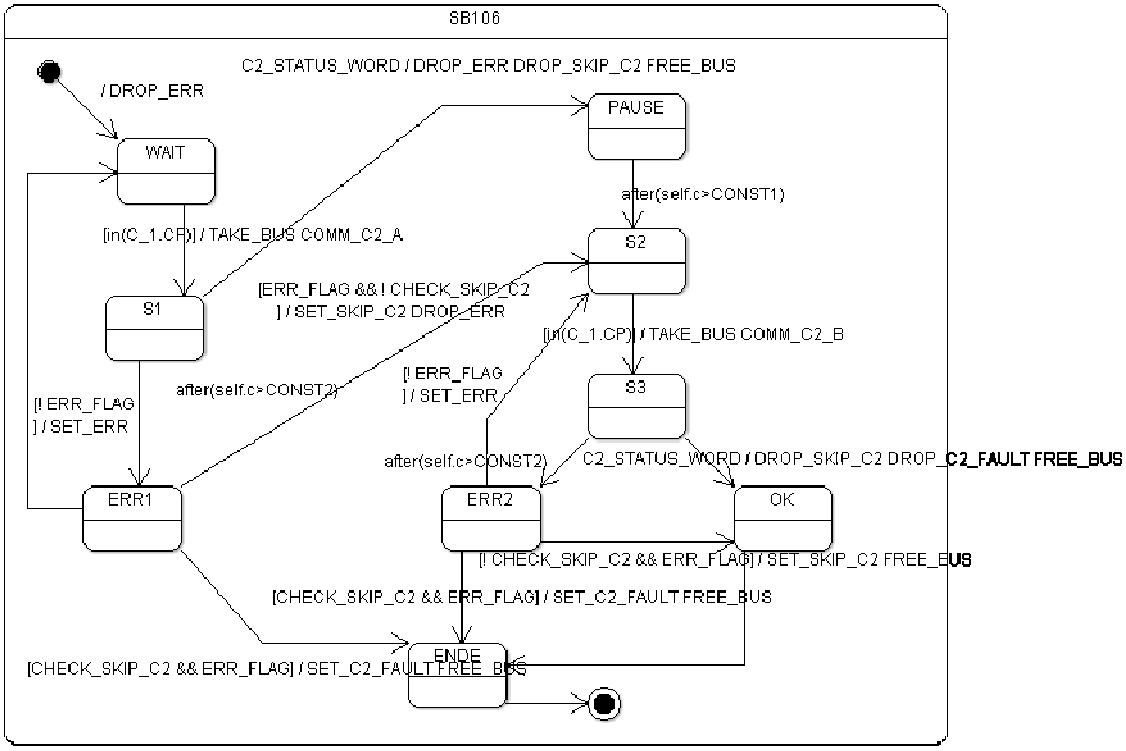

Рисунок 104. Специальный блок 106.

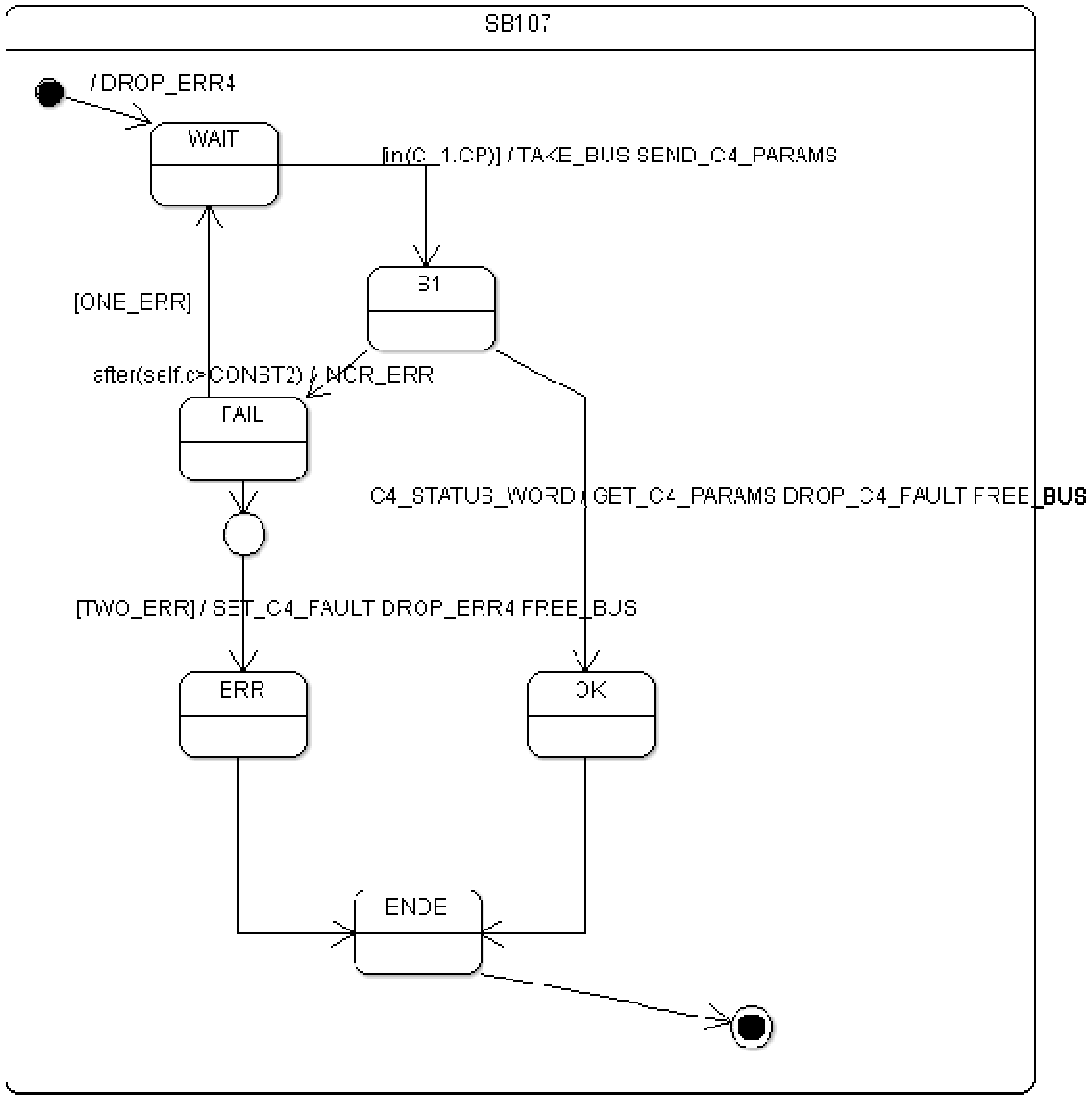

Рисунок 105. Специальный блок 107.

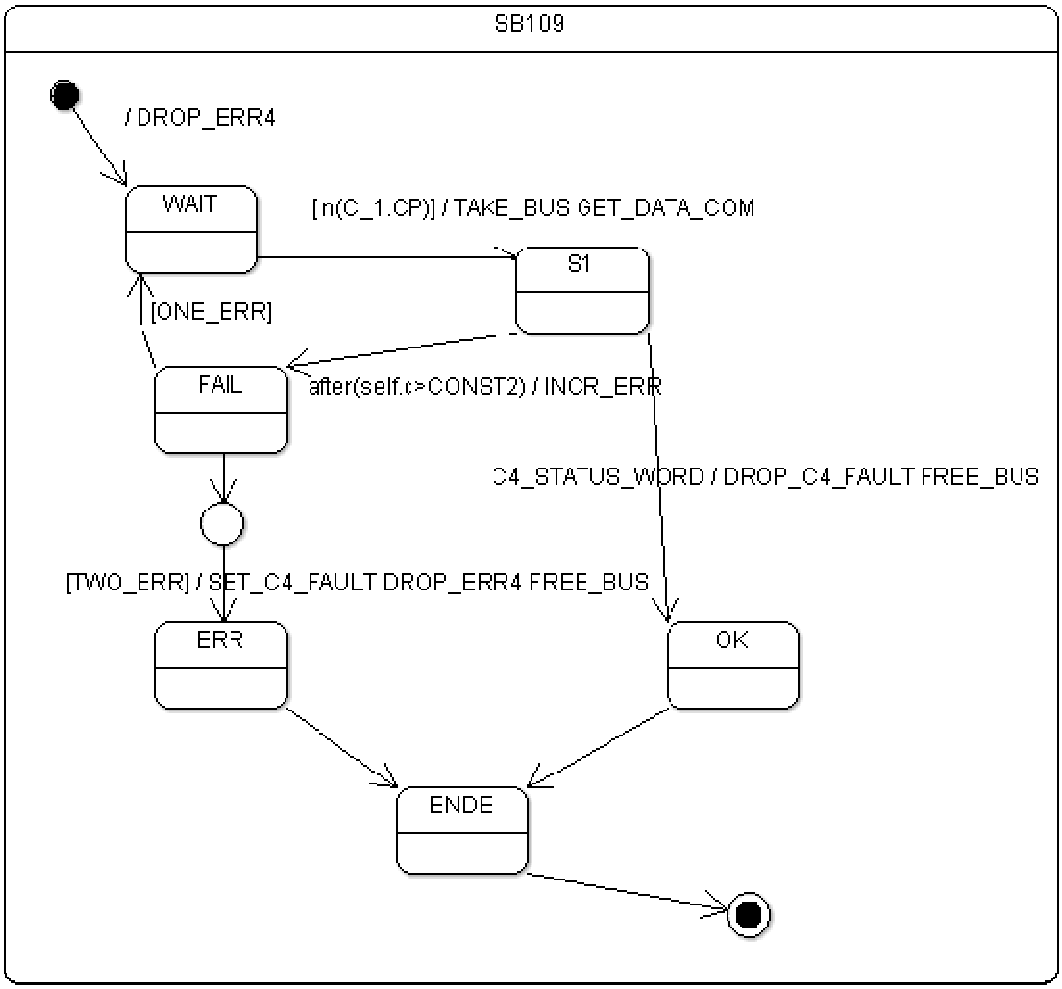

Рисунок 106. Специальный блок 109.

Возможен также перезапуск всей системы. Соответствующий флаг проверяется в главном цикле. Если система перезапускаются, таймеры также сбрасываются сигналом DEACT_T.

Большинство состояний на диаграммах названы по шаблону SB + номер, где SB – специальный блок (specific block). Такие блоки можно рассматривать как отдельные шаги алгоритма. Конкретные вычисления на диаграммах не указаны, если они не влияют на коммуникации между процессорами. Специальные блоки, отвечающие за передачу данных между процессорами, описаны отдельными диаграммами (SB104-SB109), ссылки на которые есть в диаграммах главного цикла.

Прерывание в процессоре C_1 происходит по сигналам T1 и T2 от таймеров или при установке флагов FLAG1 и FLAG2. Эти флаги означают, что после завершения текущей операции на шине необходимо передать сообщения на другие процессоры. Первый флаг указывает на необходимость запустить специальный блок 150, передающий управляющие

сигналы с C_3 на приборы для наблюдения. Второй флаг означает необходимость запустить специальный блок 151, блокирующий некоторые сенсоры в ожидании сигнала AAR radiation.

Для инициализации передачи в режиме прерывания необходимо, чтобы шина была доступна, поэтому перед обработкой прерываний проверяется флаг bus_free, и в случае успешного захвата он сбрасывается, предотвращая попытки получения шины другими процессорами. Так как во время ожидания освобождения шины могут возникнуть новые прерывания, специальные счетчики Flag1 и Flag2 хранят количество поступивших прерываний каждого типа. После завершения обработки прерывания (сигнал LEAVE) шина освобождается, а соответствующий счетчик уменьшается.

Секция 1 представляет собой простую последовательность блоков, на порядок выполнения влияют только значения флага перезагрузки и кнопки ввода.

Секция 2 содержит инициализацию компьютеров (SB1002-1004), проверку логических характеристик (SB104, SB105), и передачу сообщений на C_2, C_3 и C_4 (SB106-108). Также активируются внутренние таймеры (ACTIVATE_TIMER). Операция деактивации таймера (DEACT_T) вызывается, если коммуникации оканчиваются неудачно и необходимо перезапустить главный цикл с SB100. Паузы между событиями в этой секции обеспечиваются таймаутами.

Специальный блок 104 проверяет состояние компьютеров C_2, C_3 и C_4 и сенсоров (сенсоры считаются единым внешним устройством). Коммуникация с каждым из удаленных устройств происходит следующим образом. Сначала проверяется, что компьютера C_1 работает в обычном режиме, затем происходит захват шины и отправка сигнала-запроса о состоянии. Если в течение заданного количества миллисекунд не получен ответ, устанавливается флаг ошибки, иначе этот флаг сбрасывается. После этого шина освобождается.

Специальный блок 105 собирает данные с доступных сенсоров. Сначала проверяется, что компьютер C_1 работает в обычном режиме, затем происходит захват шины и отправка сигнала-запроса о состоянии. Если в течение 12 миллисекунд не получен ответ, устанавливается флаг ошибки SENSOR_FAULT, иначе этот флаг сбрасывается. После этого шина освобождается.

Специальный блок 106 задает коммуникации с процессором C_2 и отвечает за получение основных параметров полета. Допускается одна повторная попытка установления соединения, если на первый запрос на соединение нет ответа в течение 12 миллисекунд. Флаг skip_data используется, чтобы разрешить работать с данными, полученными во время предыдущего обмена в случае отсутствия ответа. Если ответа нет два раза подряд, то считается, что процессор C_2 отказал, и устанавливается соответствующий флаг (C2_EN).

Специальный блок 107 отвечает за получение данных с процессора C_4. Допустимы две ошибки, связанные с отсутствием соединения. Счетчик Err_C4 содержит число ошибок. Когда счетчик достигает 2, считается, что C_4 отказал.

Специальный блок 109 отвечает за передачу данных на процессор C_4. Обмен сообщениями аналогичен специальному блоку 107.

Секция 3 выполняется, если процессоры C_2 и C_3 работают. Обмен данными с процессором C_4 происходит только если самолет летит на низкой высоте (предусловие LOW). В специальном блоке 27 вычисляется текущее состояние, используемое в алгоритмах (схема COMPUTE_AM). Если передача данных на процессоры C_2 и C_4 заканчивается ошибкой, секция завершается и происходит выход в главный цикл.

Секция 4 выполняется, если процессор C_2 работает, а C_3 отказал. В ней происходят вычисления блоков 28, 9 и 29. В них не происходит коммуникаций, поэтому они не конкретизируются.

Секция 4 выполняется, если оба процессора C_2 и C_3 отказали. Передача сообщений на процессор C_2 (SB106, SB108) опускается, и выполняется дополнительный блок 36.

6.4.3 Процессор C_2

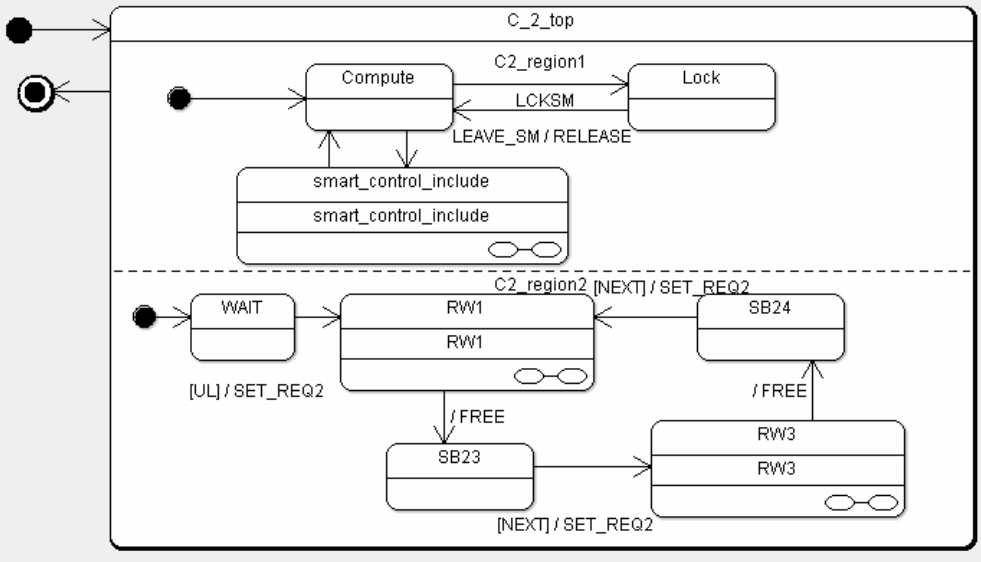

Рисунок 107. Общая диаграмма C_2.

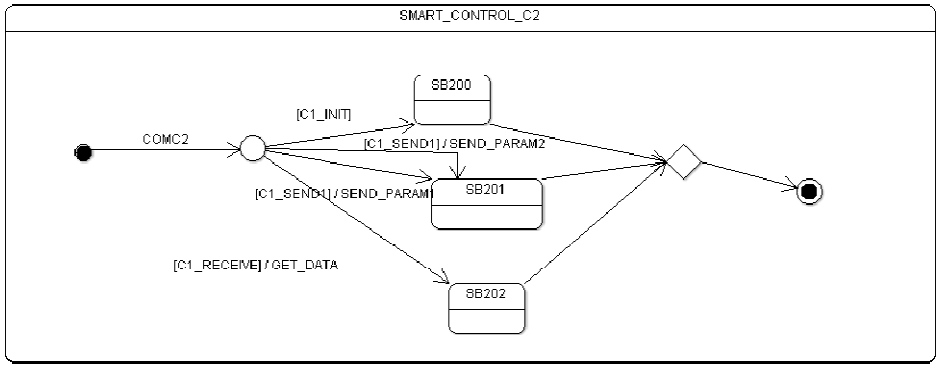

Рисунок 108. Обработка сообщений в C_2.

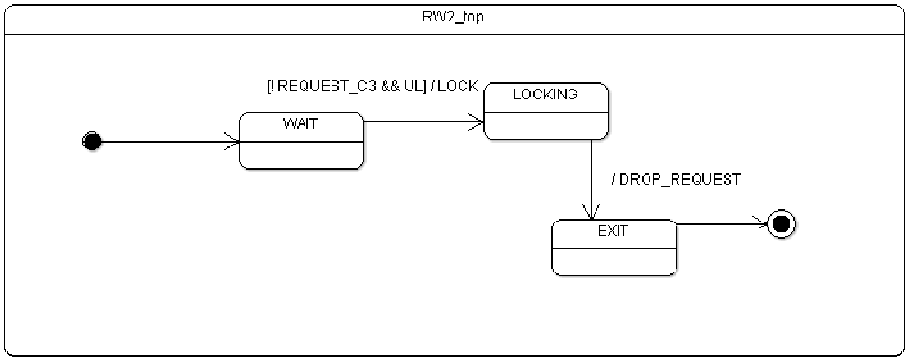

Рисунок 109. Доступ к разделяемой памяти в C_2.

Модель процессора C_2 представлена на рисунках 107-109. На процессоре C_2 в бесконечном цикле выполняются специальные блоки 23 и 24. Выполнение прерывается, когда C_1 запрашивает прием или передачу данных. Процессоры C_2 и C_3 имеют доступ к общей памяти, и цикл может быть также приостановлен, если память используется другим процессором.

На основной диаграмме два параллельных региона. Один из них описывает текущее состояние процессора, другой – текущий выполняемый блок. Процессор имеет три режима: нормальный (COMPUTE), ожидание освобождения общей памяти (LOCK) и прерывание для взаимодействия по шине с процессором C_1 (SMART_CONTROL_C2).

Процессор C_2 выполняет по очереди две задачи: расчет основных параметров полета (блок 23) и вывод данных на индикаторы (блок 24). Между этими блоками выполняется операция чтения /записи в общую память. Переход к очередному блоку возможет, если процессор работает в нормальном режиме и общая память не занята (предусловие NEXT). Перед операцией с общей памятью выставляется флаг запроса (SET_REQ), а после

окончания отправляется сигнал FREE_SM, переводящий процессор из режима ожидания в нормальный режим.

В режиме прерывания процессор C_2 обменивается данными с C_1. Возможны три случая: инициализация процессора C_2 (специальный блок 200), отправка основных параметров полета (SB201) и получение данных от C_1 (SB202).

Работа с общей памятью необходима по завершении каждого специального блока. На диаграммах состояний она моделируется специальным состоянием, помещенным между блоками. Эти состояния, имена которых начинаются с RW, реализуют механизм семафоров, предотвращающий одновременный доступ двух процессоров к памяти.

6.4.4 Процессор C_3

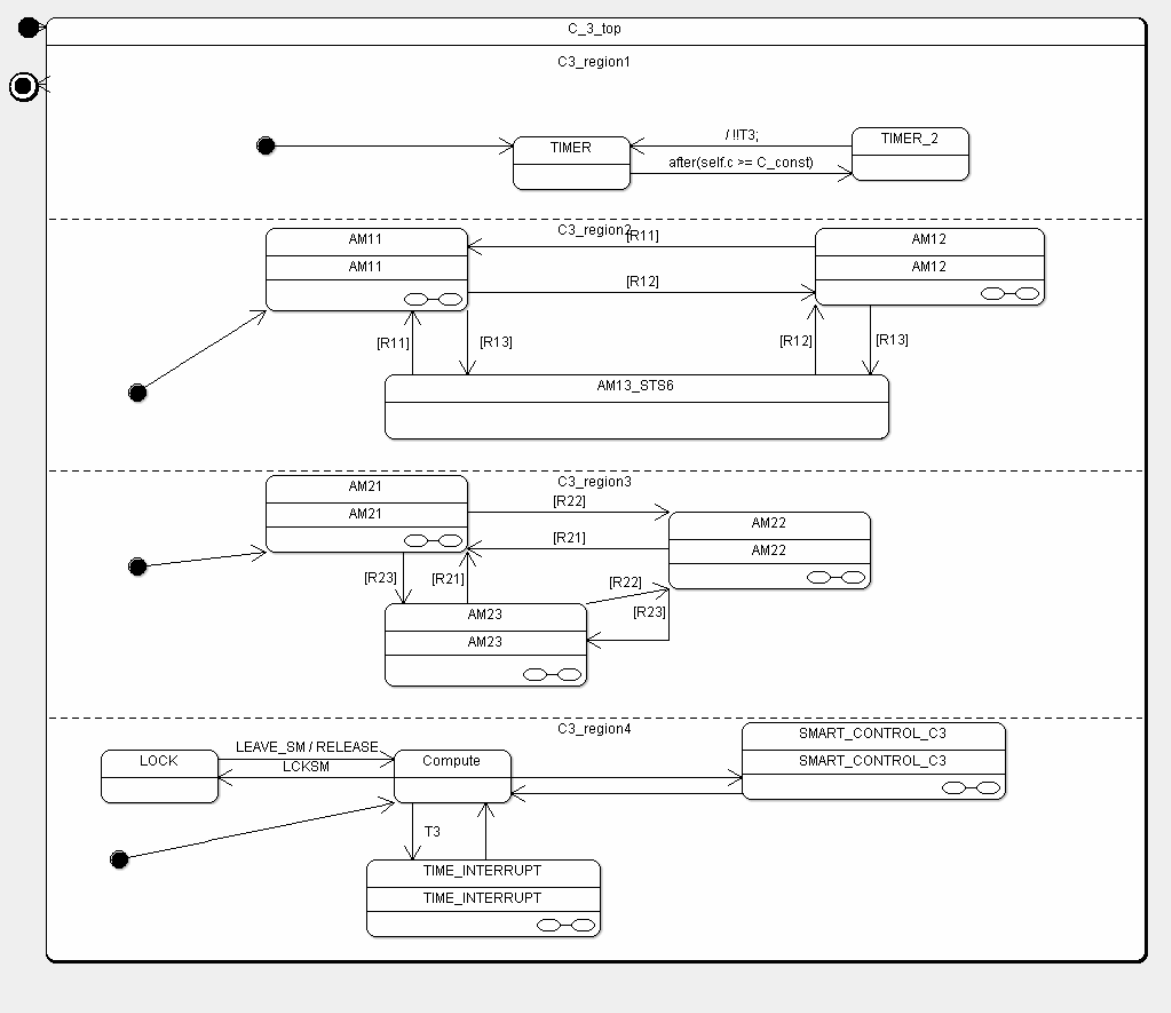

Рисунок 110. Общая диаграмма C_3.

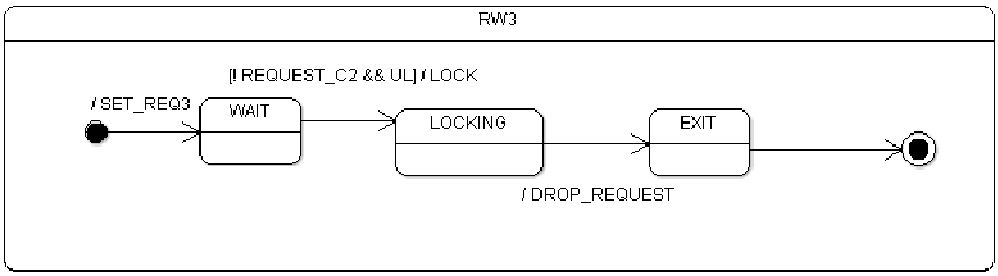

Рисунок 111. Доступ к разделяемой памяти в C_3.

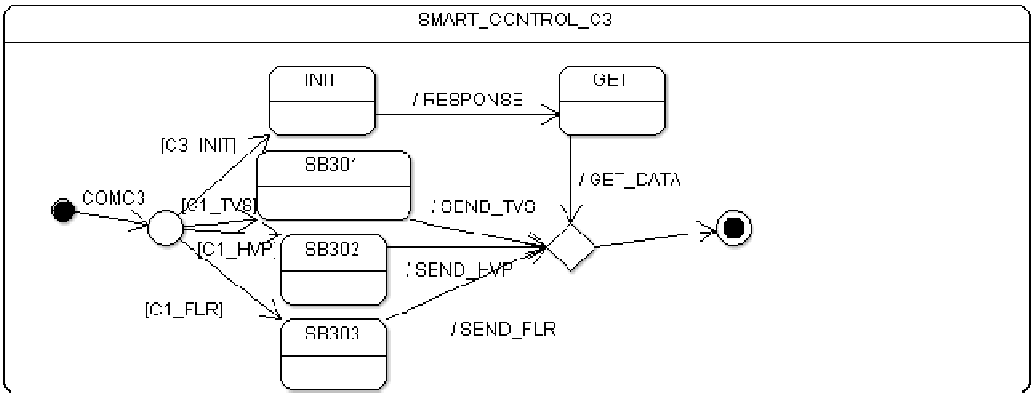

Рисунок 112. Обработка сообщений в C_3.

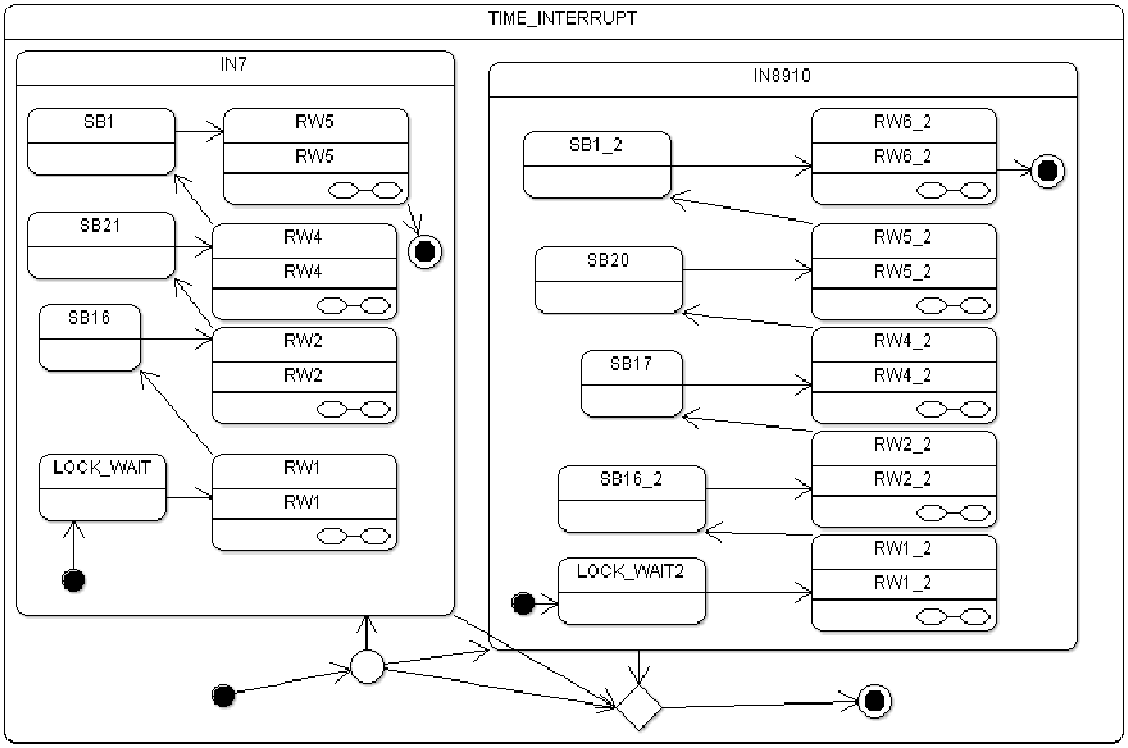

Рисунок 113. Обработка прерываний по таймеру в C_3.

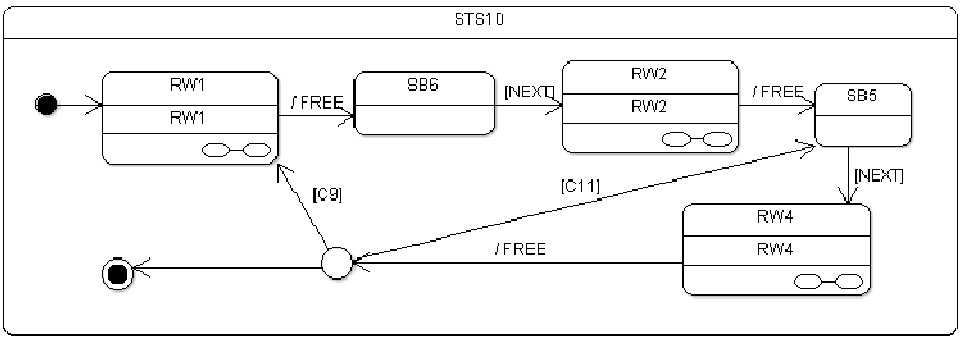

Рисунок ``4. Блок STS10.

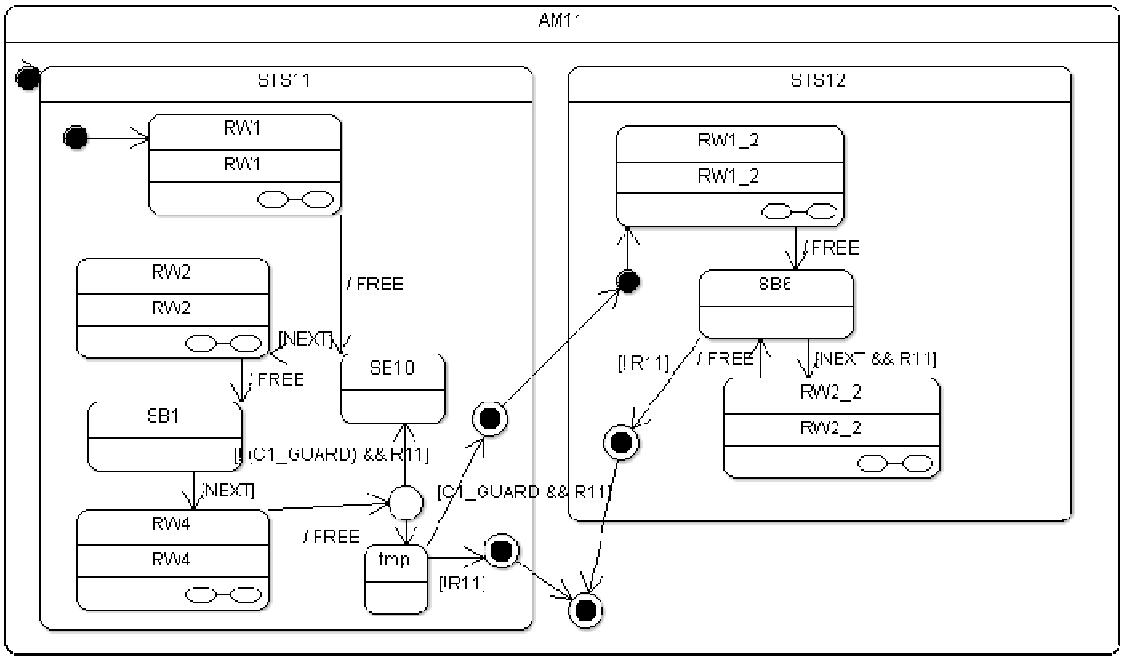

Рисунок 115. Вычислительный блок AM11.

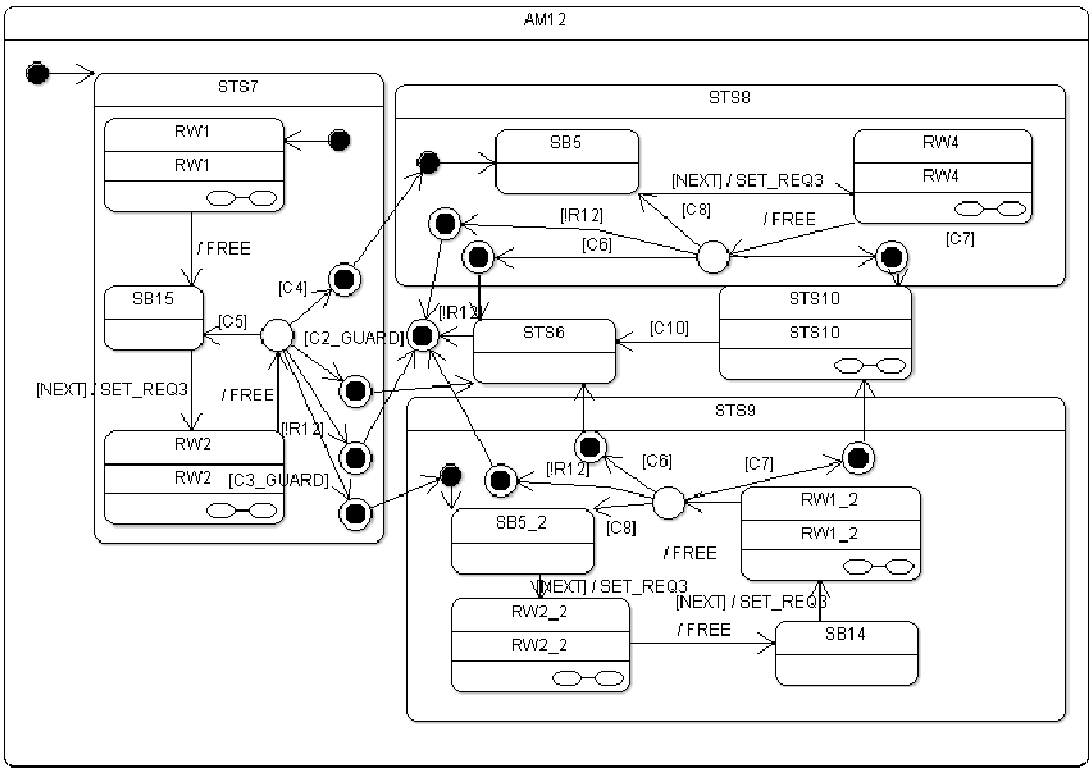

Рисунок 116. Вычислительный блок AM12.

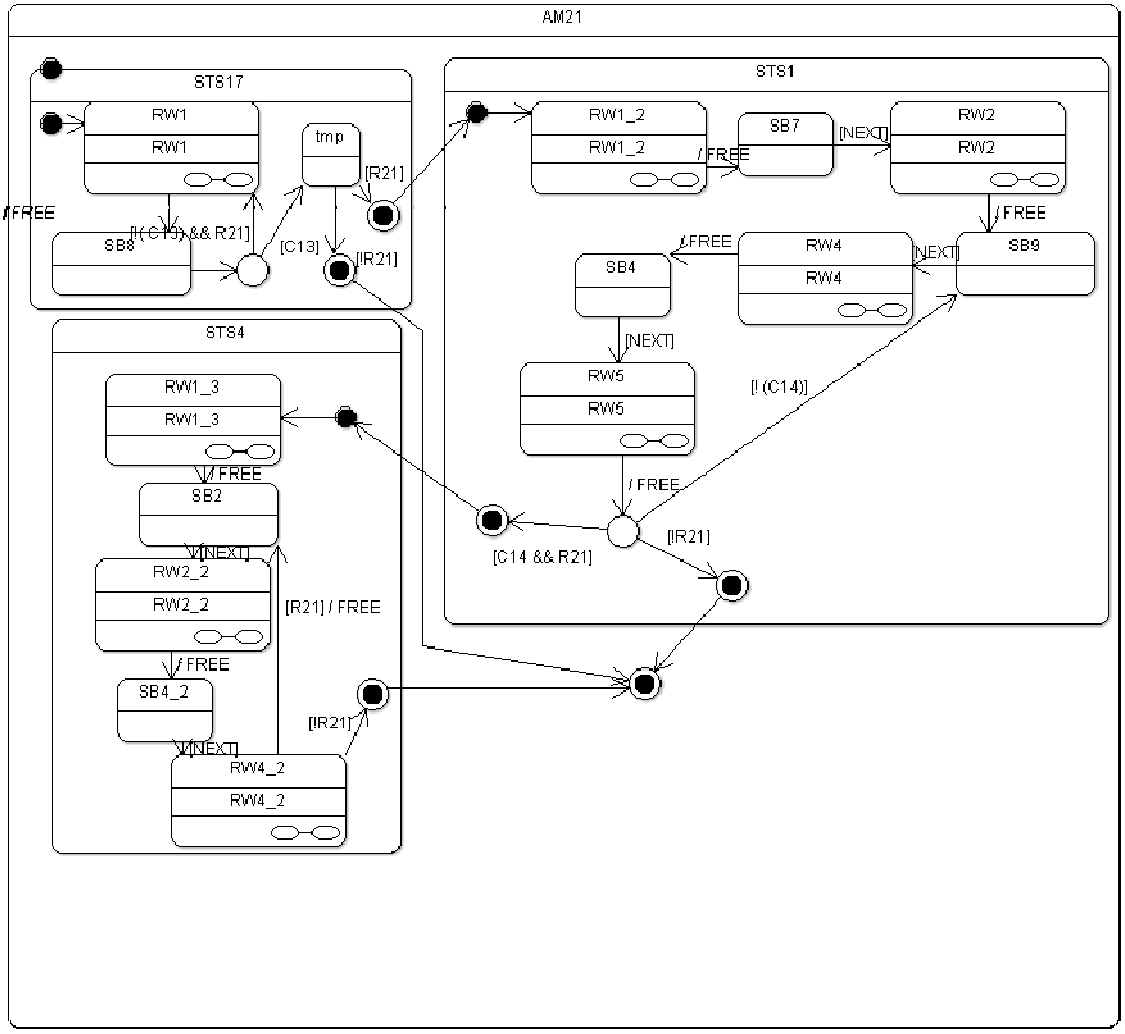

Рисунок 117. Вычислительный блок АМ21.

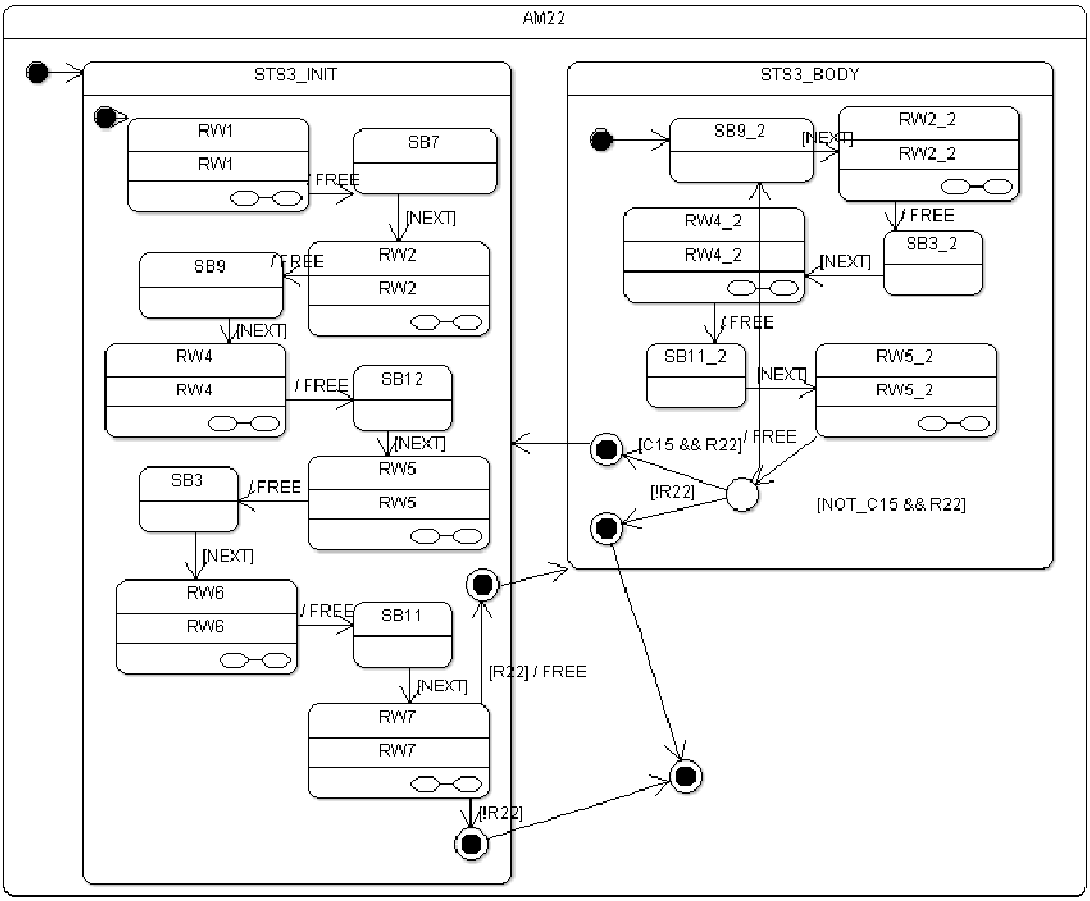

Рисунок 118. Вычислительный блок AM22.

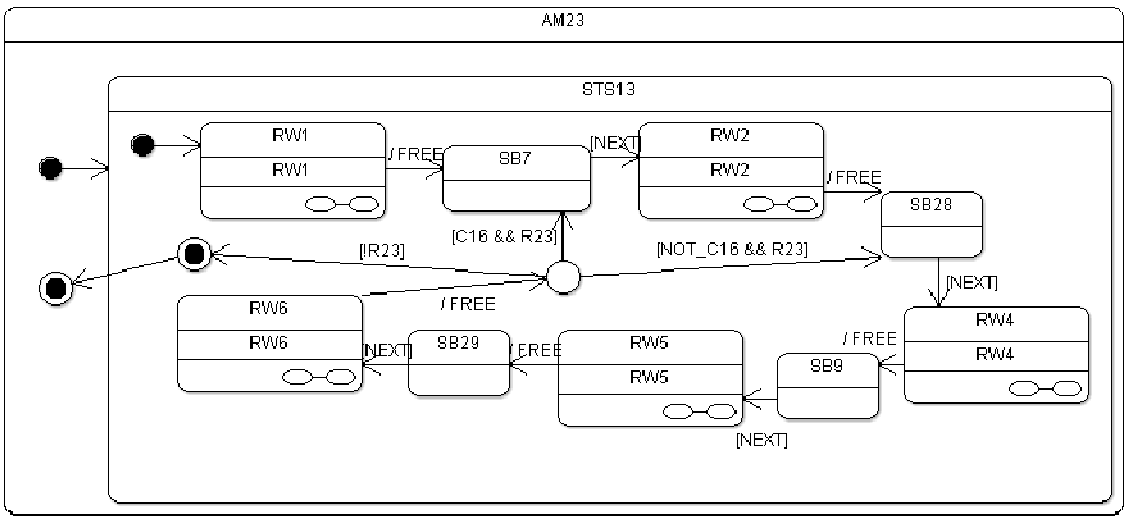

Рисунок 119. Вычислительный блок AM23.

Модель процессора C_3 представлена на рисунках 110-119. Процессор C_3 выполняет в цикле вычисления, определяемые блоками 1-22. Состав и порядок выполняемых блоков

зависят от текущего режима работы системы, который передается на процессор C_3 с процессора C_1.

Выполнение главного цикла прерывается по таймеру для пересчета значений, связанных с некоторыми сенсорами (FLR, TVS, HVP), а также при необходимости провести обмен данными по шине с C_1. Прерывания обрабатываются в специальных блоках 301-303. C_3 подключен к общей памяти, поэтому выполнение также может быть приостановлено, если память занята другим процессором.

У процессора C_3 есть четыре режима: нормальный (COMPUTE), ожидание освобождения памяти (LOCK), прерывание по таймеру (TIME_INTERRUPT) и прерывание для коммуникаций по шине (SMART_CONTROL_C3).

Выделяется четыре возможных причины прерывания для коммуникации: отправка трех параметров (FLR, TVS, HVP) и запрос о статусе процессора (C3_STATUS_WORD).

На процессоре C_3 выполняются параллельно два вида вычислений. Какой именно блок выполняется, определяется значениями переменных R1 и R2.

6.4.5 Процессор C_4

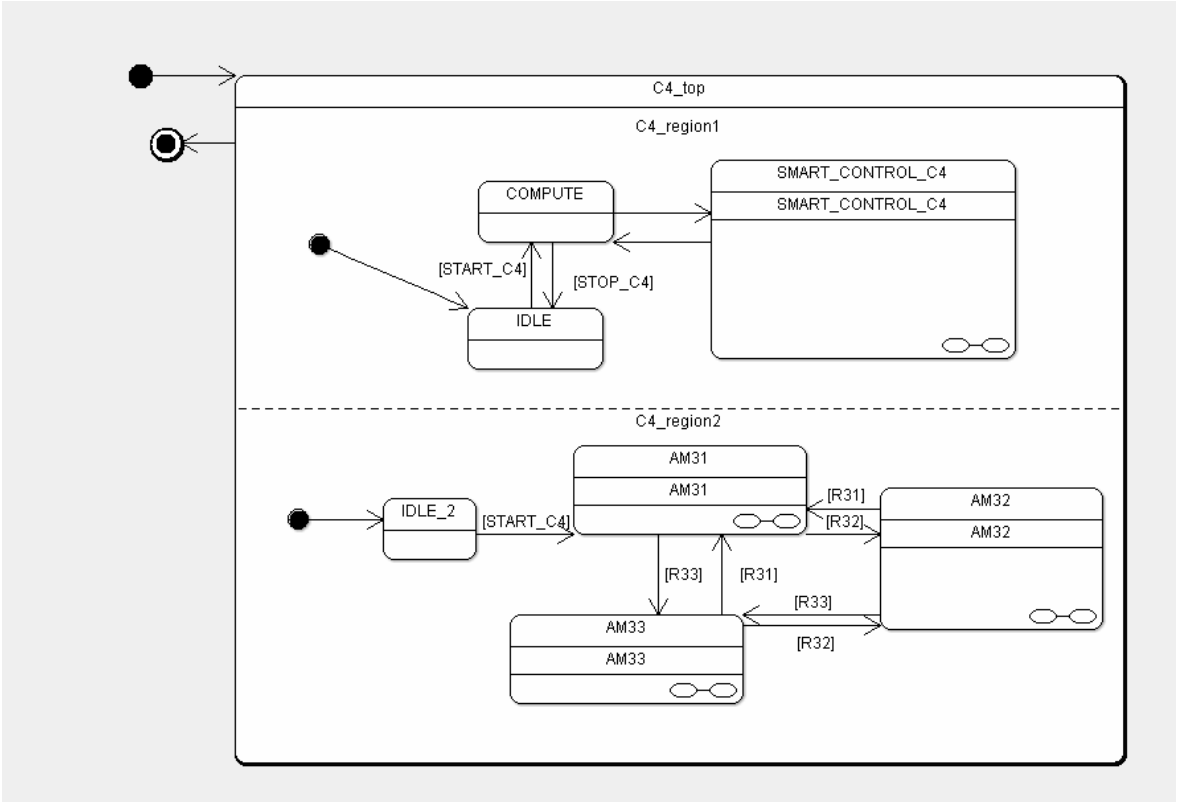

Рисунок 120. Общая диаграмма C_4.

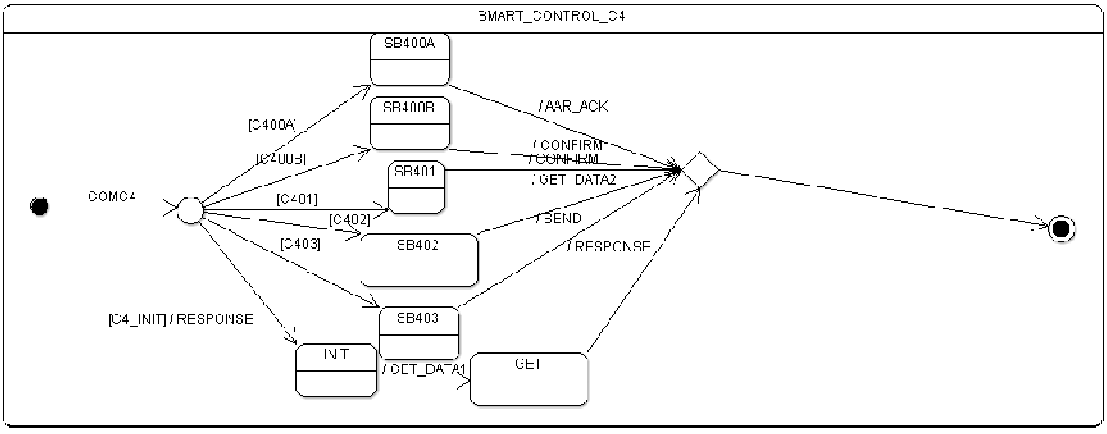

Рисунок 121. Обработка сообщений в C_4.

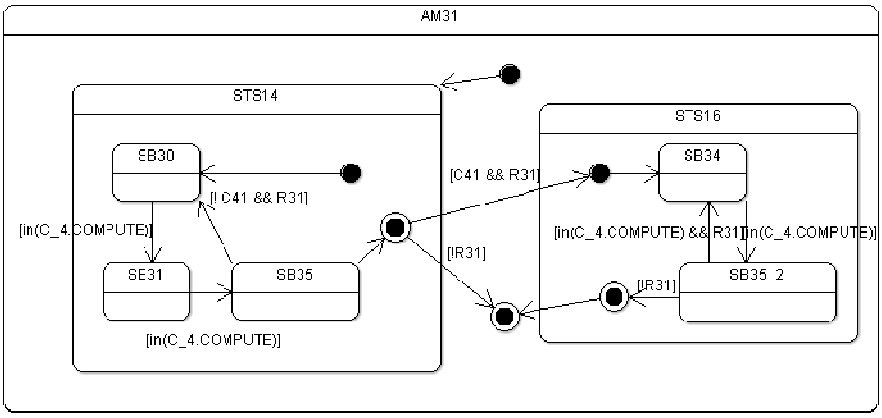

Рисунок 122. Вычислительный блок AM31.

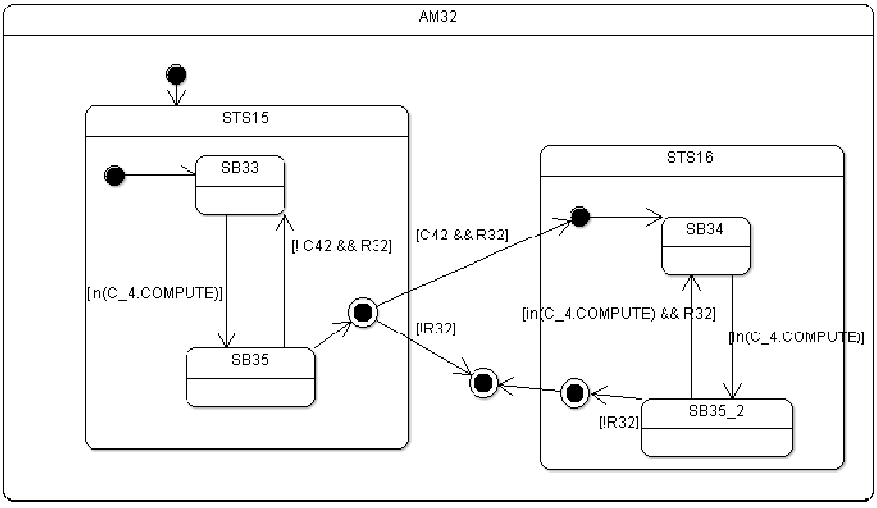

Рисунок 123. Вычислительный блок AM32.

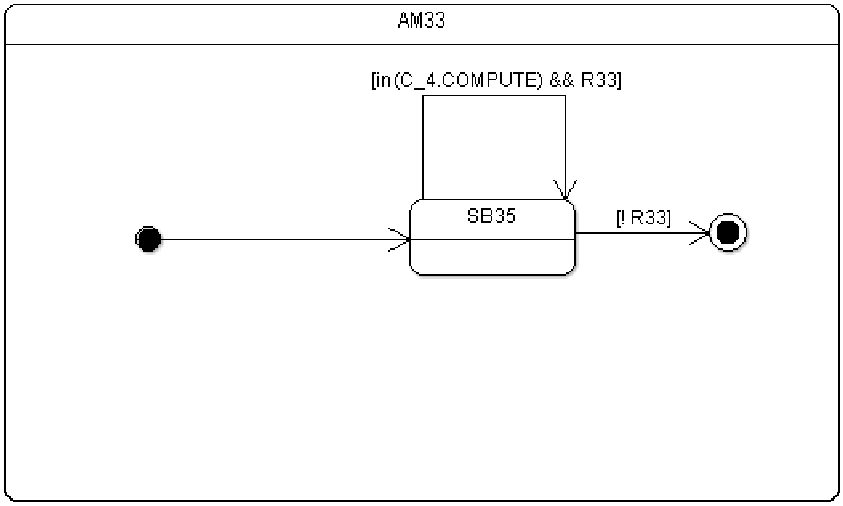

Рисунок 124. Вычислительный блок AM33.

Модель процессора C_4 представлена на рисунках 122-124. Процессор C_4 работает, когда необходим расчет параметров полета на низкой высоте или когда включается радар, предотвращающий столкновения. В зависимости от состояния процессора, которое присылается с C_1, компьютер C_4 выполняет алгоритмы, заданные специальными блоками 31-35. Выполнение прерывается в случае, если нужно получить или передать данные с процессора C_1.

Так же, как и для других процессоров, поведение процессора C_4 определяется диаграммой с двумя параллельными регионами, один из которых определяет режим работы процессора, а второй задает непосредственно вычисления. У процессора C_4 три режима: ожидание (IDLE), выполнение (COMPUTE) и прерывание (SMART_CONTROL_C4). Процессор C_4 переходит из режима ожидания в режим выполнения, когда включается радар (AAR_on == true).

В режиме прерывания процессора C_4 обрабатывает запросы в соответствие с кодом текущей операции (COMC4).

Вычисления представлены последовательностью простых состояний. Так как процессор C_4 не использует общую память, при переходах между состояниями дополнительных операций не требуется.

7 Создание экспериментального образца стенда полунатурного моделирования и интеграции РВС РВ

В данном разделе описывается экспериментальный образец стенда полунатурного моделирования и интеграции РВС РВ. В разделе 7.1 приводится описание организации полунатурного моделирования в рамках разработанной среды моделирования. Раздел 7.2 содержит описание стенда моделирования.

7.1 Схема организации полунатурного моделирования

7.1.1 Аппаратный комплекс для моделирования РВС РВ

Разработка необходимого оборудования РВС РВ обычно ведётся сразу несколькими организациями независимо друг от друга [150]. Перед ними ставятся задачи различной сложности, на выполнение которых отводятся разные сроки. В результате прототипы устройств постоянно находятся в разной степени готовности, что не позволяет провести их совместные испытания. В то же время значительная часть ошибок, допущенных разработчиками устройств, выявляется именно на стадии интеграции устройств между собой, когда прототипы близки к завершению, а стоимость исправления ошибок максимальна.

Распространённым методом раннего обнаружения интеграционных ошибок, применяемым в цикле разработки РВС РВ, является проведение экспериментов на их имитационных моделях. На стадии начального планирования разработчики РВС РВ создают грубую модель системы, позволяющую проверять её простейшие свойства и выявлять связанные с ними ошибки проектирования. По мере разработки отдельных устройств РВС РВ модель постепенно уточняется, и эксперименты с её участием позволяют выявить всё более сложные ошибки. Наконец после появления прототипов устройств их программные модели заменяются аппаратным оборудованием. При этом на всех стадиях разработки РВС РВ существуют возможности для выявления и исправления интеграционных ошибок до завершения разработки всех компонентов РВС РВ.

Смешивание аппаратных устройств РВС РВ и их программных моделей в рамках одного имитационного эксперимента требует использования специальной программно-аппаратной системы полунатурного моделирования. Использование подобных систем позволяет исследовать свойства РВС РВ на всём протяжении их разработки, постепенно увеличивая точность с грубой программной модели вплоть до аппаратной системы, целиком состоящей из прототипов устройств. Данный результат достигается с помощью итеративной замены

подключённых к среде выполнения имитационных моделей на реализующие их аппаратные устройства. Таким образом, среда выполнения системы полунатурного моделирования должна быть готова к прямому соединению с аппаратурой, использующей собственные физические каналы и протоколы передачи данных.

В результате, стенд полунатурного моделирования РВС РВ неизбежно должен представлять собой программно-аппаратный комплекс. Прежде всего, аппаратная платформа должна предоставлять физические интерфейсы для подключения прототипов устройств и каналов передачи данных. Работа части аппаратных устройств РВС РВ жёстко привязана к реальному времени, и остальные компоненты-участники эксперимента должны выполняться в том же масштабе времени. Отсюда соответствующее требование к используемой программно-аппаратной среде выполнения: программная часть среды выполнения и подключённые к ней имитационные модели должны быть способны функционировать в реальном времени. Более того, производительность данной программно-аппаратной системы должна позволять выполнять подключённые к ней имитационные модели устройств со скоростью прототипов аппаратного оборудования.

Класс современных РВС РВ включает в себя широкий диапазон программно-аппаратных комплексов. Простейшие РВС РВ встроены в бытовую технику и содержат лишь несколько датчиков и вычислителей. Сложными РВС РВ являются бортовые системы самолётов автомобилей и кораблей, которые часто состоят из сотен устройств, соединённых между собой десятками каналов передачи данных различных типов. Размеры и сложность подобных моделей не позволяют проводить эксперименты, используя персональный компьютер. Поэтому для их обработки уже долгое время используются разнообразные параллельные и распределённые системы [151].

В рамках настоящей работы была разработана распределённая среда выполнения имитационных моделей, аппаратная платформа которой может включать в свой состав произвольный набор инструментальных машин, объединённых общей компьютерной сетью. В частности, такая конфигурация делает возможным совместное моделирование РВС РВ, при котором компоненты системы находятся на значительном географическом расстоянии друг от друга.

Каждая из инструментальных машин комплекса может обладать произвольным количеством разнообразных физических интерфейсов для подключения прототипов устройств через используемые этими устройствами каналы передачи данных. Важно понимать, что каждое новое физическое устройство, подключённое к инструментальной машине, требует дополнительных вычислительных ресурсов. Чрезмерная загруженность задачами по поддержке внешних аппаратных устройств может не оставлять достаточного

количества ресурсов для работы программной части среды выполнения. Поэтому устройства должны подключаться к машинам стенда по возможности *равномерно*.

Задача распределения участников моделирования по вычислительным узлам должна решаться автоматически. Возможны две схемы, когда участники моделирования распределяются статически при запуске модели и динамически в зависимости от загрузки вычислительных узлов в процессе моделирования. На данный момент подобных инструментальных средств балансировки нагрузки вычислительных узлов авторам неизвестно, поэтому их разработка может стать одним из перспективных направлений для исследований в области систем полунатурного моделирования. Например, в стенде ПНМ устройства, участвующие в эксперименте распределяются вручную между вычислительными узлами инженером-экспериментатором [14].

7.1.2 Схемы синхронизации участников моделирования

До сих пор в данном разделе описывалась схема лишь подключения натурных компонентов к аппаратной платформе среды выполнения. Между тем, основная роль среды выполнения – синхронизация подключённых участников моделирования во время проведения имитационных экспериментов на программном уровне. Более точно, среда выполнения должна согласовать потоки событий от независимых компонентов модели, которые могут иметь, вообще говоря, программную или физическую природу, в едином модельном времени [152].

Существует несколько принципиально разных подходов к решению данной задачи. Первый из них предполагает использование единого модельного времени: производится высокоточная синхронизация аппаратных часов всех инструментальных машин. При этом любые сообщения внутри системы упорядочиваются по времени автоматически. Однако простота такого решения зачастую не позволяет эффективно использовать доступные вычислительные мощности аппаратной платформы из-за низкой степени параллелизма.

Второй подход предлагает использовать независимое логическое время для каждого участника моделирования, и изменять его по распределённому алгоритму синхронизации. Эта идея позволяет достичь большей степени параллелизма по сравнению с идеей единого времени, но требует применения сложных алгоритмов синхронизации, эффективность которых существенно зависит от конкретной имитационной задачи. Такие алгоритмы обычно разделяются на консервативные и оптимистические [153]. Консервативная схема предполагает приостановку участников моделирования, если существуют факторы, способные повлиять на результат его работы. Например, участник не может продвинуть своё

логическое время, пока другой участник может прислать ему сообщение в текущий момент этого времени, что может приводить к общему замедлению системы моделирования.

Оптимистические алгоритмы, напротив, никогда не приостанавливают участников. Их задача – разрешение возникших в результате такого подхода проблем синхронизации с помощью отката модельного времени назад. Оптимистические алгоритмы сложнее в реализации, требуют возможности сохранения и восстановления состояний участников и, как следствие, много памяти. Кроме того, возможность отката времени нарушает свойство предсказуемости системы, затрудняя её использование для решения задач с привязкой к реальному времени.

Помимо рассмотренных алгоритмов продвижения времени исследователи иногда выделяют смешанные алгоритмы, которые позволяют объединять участников модели, работающих по различным схемам [152]. При умелом применении, смешанные алгоритмы позволяют совместить преимущества этих схем, и построить быструю систему моделирования, сохранив при этом возможность подключения реальных устройств. В тоже время неправильное сопряжение разных алгоритмов продвижения времени может привести к падению производительности. Поэтому использование смешанных алгоритмов требует тщательного анализа имитационной модели и разработки соответствующих инструментальных средств.

Система полунатурного моделирования может быть построена на базе схемы с единым модельным временем, консервативной схемы или же смешанной схемы продвижения логического времени. Однако использование единого модельного времени при условии географической удалённости компонентов системы невозможно, что исключает возможность проведения совместных экспериментов, которые используются при разработке РВС РВ. Кроме того, схема с единым модельным временем использует полную синхронизацию системы и полностью исключает возможность «забегать вперёд» для отдельных компонентов модели. Поэтому в ряде случаем она может быть менее эффективной, чем любая из схем, использующих распределённое логическое время.

Реализация смешанной схемы на базе системы моделирования CERTI также может быть затруднительной, так как данная система не поддерживает даже оптимистическую составляющую синхронизации [95]. Таким образом, наиболее перспективной для целей проекта схемой синхронизации времени является консервативная схема.

7.1.3 Синхронизация полунатурной модели

Задача полунатурного моделирования предполагает синхронизацию программных моделей с аппаратными устройствами, внутреннее время которых движется независимо от

модельного времени системы. По условию имитационной задачи обычно известны лишь некоторые свойства, индивидуальные для каждого отдельного устройства РВС РВ. Например, датчик должен ответить на запрос вычислителя лишь в течение заданного директивного интервала. Необходимость учёта подобных свойств делает невозможным подключение устройств напрямую к среде выполнения – для этого нужна дополнительная программная прослойка, которая учитывала бы расписание обменов данного устройства и жёстко привязывала бы эти обмены к реальному времени.

Привязка планирующей прослойки каждого из натурных компонентов исследуемой РВС РВ к реальному времени требует их дополнительной синхронизации между собой. Реализация подобной идеи может использовать, например, выделенный сервер. Однако такое техническое решение требует поддержания дополнительной инфраструктуры и работы в обход существующей среды выполнения, что чрезмерно усложняет систему. Альтернативным решением, которое было реализовано на практике, является внедрение служебного *федерата-пульсатора*, который отправляет остальным федератам сообщения синхронизации.

Рассмотрим изложенную идею на примере. Пульсатор посылает первое *сообщение-импульс*. При его получении программная прослойка аппаратного датчика переводится в режим ожидания, и ждёт сообщений от данного устройства. Пульсатор засыпает на время, заданное соответствующим директивным интервалом и посылает следующий импульс. Если сообщение от датчика не было получено до момента получения нового импульса, то фиксируется ошибка – датчик не уложился в свой директивный интервал. Аналогично можно проверять и программные компоненты модели. Таким образом, федерат-пульсатор играет роль искусственного счётчика, который продвигает модельное время системы в соответствии с предопределённой логикой обменов данными и заданным набором директивных интервалов.

Недостатком данного решения является необходимость вычисления интервалов между отправкой импульсов синхронизации индивидуально для каждой модели исходя из набора её требований. Однако существуют методы их автоматического вычисления на основе формального описания модели [154]. В результате федерат пульсатор может быть построен подсистемой генерации исходных кодов модели в автоматическом режиме.

7.2 Общая схема стенда моделирования

В данном разделе приводится описание принципиальной схемы аппаратной и программной составляющих разработанного стенда моделирования. Также здесь содержится

описание принципов работы и реализации оркестратора – вспомогательной программы, управляющей проведением имитационных экспериментов.

7.2.1 Техническая организация

Разработанная система полунатурного моделирования может быть развернута на аппаратной платформе из одного или нескольких серверов, каждый из которых находится под управлением операционной системы из семейства Linux: Ubuntu или Debian. Здесь стоит отметить, что большая часть разработанных в рамках настоящего проекта средств кроссплатформенные и способны работать на других операционных системах, таких как системы семейства Windows. Однако тестирование работы средств на других системах в достаточной мере не производилось.

Логические роли серверов системы моделирования разделяются на две категории [14]. Это *автоматизированные рабочие места* (АРМ), которые используются для подготовки имитационной модели, и *инструментальные машины среды выполнения* (ИМ), которые необходимы для её прогона. Эти роли могут пересекаться: единственный физический сервер может одновременно являться и сервером АРМ, и сервером ИМ.

На сервера АРМ установлено программное обеспечение для управления всеми стадиями цикла разработки имитационной модели. Это интегрированная среда разработки моделей UML, включающая редактор диаграмм состояний UML, средства автоматической трансляции моделей UML в модели UPPAAL и модели HLA, средства верификации моделей, средство оценки наихудшего времени выполнения программ и средства управления моделированием и многие другие. Подробнее об их составе и назначении можно прочитать в разделе 4. Никаких специфических требований к серверам АРМ при этом не предъявляется.

В отличие от серверов АРМ, к аппаратуре которых не предъявляется никаких специфических требований, оборудование серверов ИМ требует особой конфигурации. Например, сервера ИМ обычно должны обладать высокой вычислительной мощностью, чтобы программная модель могла работать со скоростью приемлемой для взаимодействия с аппаратными участниками моделирования. С другой стороны, аппаратура сервера АРМ должны включать внешние устройства для взаимодействия с пользователем (монитор, клавиатура, мышь), что совсем не обязательно для сервера ИМ.

При таких разных требованиях к машинам АРМ и ИМ, совмещение этих ролей в рамках одного физического сервера может быть нецелесообразным, поэтому обычно они разделяются по разным машинам. Сервера ИМ образуют распределённый вычислительный комплекс, мощности которого хватает для удовлетворения требований выполняющихся на

нём моделей. Сервера АРМ, подобно терминальным машинам суперкомпьютера, лишь предоставляют интерфейс для управления более мощными серверами ИМ.

7.2.2 Схема выполнения имитационной модели

Разработанная система моделирования позволяет проводить как эксперименты, включающие только программные модели устройств, так и эксперименты с натурными компонентами. В первом случае выполнение модели производится в режиме быстрого моделирования – модельное время продвигается с максимальной скоростью, которую способна дать среда выполнения. В случае участия в имитационном эксперименте натурных компонентов модельное время жёстко привязано к астрономическому времени, и поэтому не может продвигаться быстрее. В этом случае для синхронизации элементов модели используется служебный федерат *пульсатор*.

В общем случае для проведения имитационного эксперимента на развёрнутом аппаратном комплексе должны быть запущены следующие программные компоненты:

1. *Оркестратор* эксперимента, описанный в разделе 7.2.3, позволяет автоматизировать запуск и остановку остальных программных компонентов системы;
2. Центральный компонент *CRC* реализует внутреннюю логику инфраструктуры RTI, которая согласовывает между собой действия участников эксперимента;
3. Служебный федерат «*Сборщик трасс*» подключается к инфраструктуре RTI и записывает последовательность изменения состояний модели;
4. Служебный федерат «*Пульсатор*» позволяет синхронизировать участников моделирования при проведении полунатурных экспериментов;
5. Множество федератов, выполнение которых не привязано к реальному времени;
6. Множество федератов, привязанных к астрономическому времени, таких как программные прослойки для подключения натурных компонентов.

Запуск оркестратора осуществляется на сервере АРМ, запуск остальных компонентов – на серверах ИМ, причём отображение этих компонентов на сервера ИМ может быть неоднозначным. Каждый из программных компонентов может быть запущен на собственном сервере ИМ, или же сразу несколько этих компонентов могут быть запущено на одном и том же сервере. Задача распределения программных компонентов модели по серверам в рамках настоящей работы подробно не изучалась, и данный вопрос требует дополнительных исследований. Частично вопросы, связанные с привязкой различных программных компонентов среды выполнения к конкретным серверам, а так же свойства

масштабируемости модели при наращивании количества серверов, затрагиваются в разделе 9.2.

Общая схема стенда моделирования изображена на рисунке 125. Эксперимент проводится с участием серверов 1-6, находящихся в нескольких различных локальных сетях. Оркестратор запускается пользователем через терминал на сервере АРМ 1. На старте эксперимента он подключается к серверам-участникам и запускает на них оставшиеся программные компоненты. На рисунке центральный компонент среды выполнения CRC запускается на сервере 3, сборщик трасс – на сервере 4, а пульсатор – на сервере 6. Каждый сервер может выполнять произвольный набор компонентов системы. Например, через сервера 2 и 5 подключаются лишь натурные участники моделирования, в то время как сервер 6 выполняет сразу несколько программных и несколько натурных компонентов модели (натурные компоненты модели изображены в виде пиктограмм процессора, а программные – этой же пиктограммы на облаке).

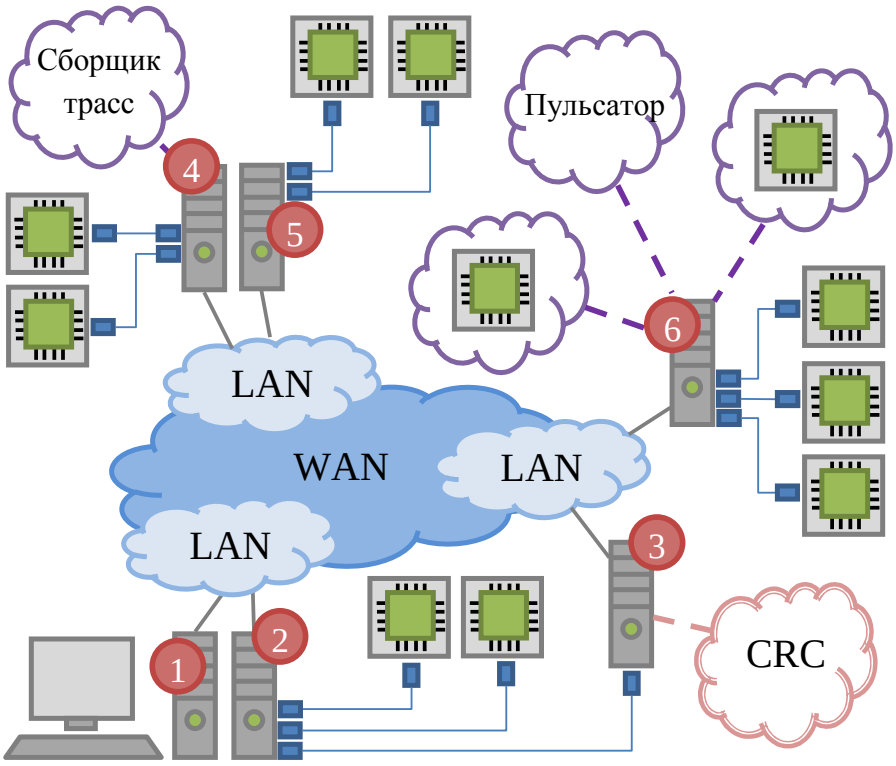

Рисунок 125. Общая схема стенда моделирования.

7.2.3 Устройство оркестратора

Большинство из перечисленных в предыдущем разделе программных компонентов реализованы в виде обычных федератов, подключаемых к среде выполнения RTI, и их

описание, как и описание центрального компонента среды выполнения CRC можно найти в разделе 4.4 настоящей работы. Описание же оркестратора ранее не приводилось.

Оркестратор реализован в виде программы на языке Python и его исходный код создаётся встроенным в систему моделирования генератором в автоматическом режиме. На вход оркестратор принимает конфигурацию серверов стенда моделирования, набор программ, которые нужно на этих серверах запустить, а так же спецификации поведение этих программ. Код оркестратора активно использует библиотеку тестирования DTest [155], предназначенную для проверки соответствия поведения программы её формальным спецификациям.

Запуск каждого программного компонента осуществляется оркестратором через подключение к соответствующему серверу по протоколу SSH. Созданная сессия SSH присоединяется к автомату DTest, описывающему поведение компонента в терминах его входного и выходного потоков: состояние программы отслеживается через её вывод, а изменение этого состояния осуществляется через посылку команд ей на вход. Например, любой федерат, построенный с помощью разработанного генератора исходных кодов, выводит при своём завершении соответствующее сообщение. Оркестратор отслеживает выходной поток запущенного федерата и при получении такого сообщения детектирует его завершение. Кроме того, библиотека DTest включает в себя базовые средства синхронизации запущенных программ между собой. Например, оркестратор может отложить запуск федератов до тех пор, пока компонента CRC среды выполнения не будет готов к их подключению.

На старте имитационного эксперимента оркестратор первым делом инициирует запуск компонента CRC, реализующего внутреннюю логику среды выполнения. В случае системы моделирования CERTI, которая используется в рамках настоящего проекта, компонент CRC реализован процессом RTI Gate (RTIG). Затем в произвольном порядке запускаются остальные служебные и пользовательские компоненты системы, каждый из которых реализован в виде независимого участника моделирования.

Выполнение модели осуществляется оркестратором до тех пор, пока последний из федератов не отсоединится от процесса RTIG и не завершит своё выполнение, или же пока не завершит наперед заданный таймаут эксперимента. Затем оркестратор завершает все незавершённые программные компоненты и сохраняет историю сообщений входного и выходного потоков каждого из компонентов.

В этом разделе описана общая схема стенда моделирования. Методика использования программных средств, входящих в состав стенда приведена в главе 8, а эксперименты с этими средствами в главе 9.

8 Разработка методики совместного применения созданных методов и инструментальных средств для поддержки разработки и интеграции РВС РВ

В данном разделе описывается Методика совместного применения созданных методов и инструментальных средств для поддержки разработки и интеграции РВС РВ. В разделе 8.1 приводится общее описание методики разработки моделей РВС РВ. Раздел 8.2 содержит описание методики использования редактора UML-диаграмм. В разделе 8.3 приводится методика использования средства визуализации трассы. Раздел 8.4 содержит методику использования средства трансляции диаграмм состояний UML во временные автоматы. В разделе 8.5 приведено описание методики использования средства верификации модели. Раздел 8.6 содержит описание методики использования средств моделирования совместно со средствами синтеза архитектур и построения расписаний.

8.1 Общее описание методики разработки моделей РВС РВ

При разработке моделей РВС РВ с помощью созданных методов и инструментальных средств, рекомендуется придерживаться типичной последовательности действий, приведенной в данном разделе.

Основным средством, реализующим взаимодействие всех созданных инструментов с пользователем, является интегрированная среда разработки моделей, описанная в разделе 4.11. Описание экспериментального исследования интегрированной среды разработки приведено в разделе 9.9. Предполагается, что все остальные средства запускаются через ее интерфейс, если явно не указано обратное.

Первым этапом разработки модели обычно является построение ее описания в виде диаграмм UML. Для этого необходимо воспользоваться редактором ArgoUML (см. разделы 4.2 и 8.2). Для моделируемой РВС РВ строятся диаграммы, описывающие ее поведение. На данном этапе необходимо определить цель моделирования и строить UML-диаграммы так, чтобы конструируемая модель была корректна (то есть исследуемые характеристики объекта в модели должны быть эквивалентны своим прообразам в исходном объекте) и адекватна (то есть в ней должны присутствовать характеристики, существенные для цели моделирования). В диаграмму могут быть включены фрагменты кода на C++. После завершения редактирования модели необходимо средствами ArgoUML экспортировать полученные диаграммы в формат XMI.

Следующим этапом является проверка того, что построенная модель удовлетворяет некоторым заданным свойствам. Этот этап необходим при исследовании сложных моделей РВС РВ, корректность которых должна быть строго доказана. Для проведения верификации необходимо преобразовать UML-диаграммы во временные автоматы UPPAAL с помощью разработанного нами транслятора (см. разделы 4.8 и 8.4). Входными данными транслятора является XMI-файл, выходными файлы формата UPPAAL. Транслятор может обнаруживать несколько видов ошибок в диаграммах, генерировать файлы, позволяющие использовать средства оценки наихудшего времени выполнения кода (WCET) (разделы 5.5, 5.6), оптимизировать (уменьшать) количество получаемых временных автоматов (раздел 5.3). В случае использования средств оценки наихудшего времени выполнения кода (WCET) необходимо также задать характеристики целевой архитектуры.

Следующий шаг можно пропустить, но если ошибок не обнаружено, можно переходить к самой верификации с помощью средства UPPAAL (см. раздел 4.9, 8.5). Для этого необходимо в отдельных файлах задать проверяемые свойства модели и запустить процедуру верификации. Верификатор определяет, выполнено ли определенное свойство. Если свойство не выполнено, верификатор строит контрпример. Также возможна ситуация, когда процедура верификации требует слишком много ресурсов и длительное время не завершается. В таком случае необходимо построить менее детальную модель. Контрпример в формате UPPAAL можно конвертировать в трассу переходов UML с конкретными значениями параметров и таймеров. Существует возможность работы с временными автоматами непосредственно в графическом интерфейсе средства UPPAAL.

Затем можно приступить к трансляции модели из UML-представления в код на C++ (см. раздел 4.3). Процесс трансляции проходит в несколько этапов. Сначала необходимо по XMI-файлу сгенерировать диаграмму состояний в формате SCXML. Данный формат гораздо проще XMI и не содержит лишней информации о расположении объектов на диаграмме состояний. Простые модели можно строить сразу в формате SCXML, что особенно удобно в случае автоматического построения моделей. Для построения/изменения SCXML-файлов существует редактор, интегрированный в нашу среду разработки моделей. Кроме того, SCXML-диаграмму можно преобразовать во временные автоматы UPPAAL и провести её верификацию. На следующем этапе SCXML-диаграмма преобразуется в исходный код федератов на C++, генерируются служебные файлы, необходимые для запуска процесса моделирования. Содержимое всех этих файлов можно просмотреть.

После того, как код федератов получен, можно запустить имитационный эксперимент в среде CERTI (см. раздел 4.4).

Результатом моделирования является файл, содержащий трассу эксперимента в формате OTF, а также несколько служебных файлов, просмотр которых может быть полезен для отладки моделей. Для просмотра и исследования трассы эксперимента используется разработанное нами средство анализа и визуализации трасс Vis4 (см. разделы 4.7, 8.3). Данное средство позволяет получать информацию о событиях модели и информационных обменах между компонентами модели, осуществлять поиск события, навигацию по трассе, масштабирование трассы. После анализа трассы пользователь может либо убедиться в том, что цель моделирования достигнута, либо вернуться на один из предыдущих этапов, изменить модель и провести эксперимент снова.

Еще одной полезной функциональностью разработанных средств является возможность интеграции со средствами планирования расписаний и синтеза архитектур, для оценки времени выполнения работ расписания (см. раздел 5.8). Разработчику средств планирования (синтеза) представляется скрипт, который по расписанию заданного формата строит имитационную модель, проводит эксперимент, анализирует результаты и генерирует файл, содержащий время выполнения всех работ расписания. Таким образом, средство планирования должно лишь в определенные моменты генерировать файл с расписанием и вызывать скрипт. В качестве примера приведено средство решения задачи выбора механизмов обеспечения отказоустойчивости РВС РВ (раздел 5.7), позволяющее строить РВС РВ высокой надежности при ограничениях на стоимость и время выполнения программ. Далее в разделах 8.2-8.6 приведено подробное описание методики использования каждого средства.

8.2 Методика использования редактора UML-диаграмм

В разделе 4.2 был описан редактор UML-диаграмм ArgoUML, используемый в разработанной в рамках данной НИР системе моделирования. В этом разделе описывается методика использования этого редактора.

Данное средство позволяет создавать и редактировать диаграммы состояний, учитывающие ограничения, описанные в разделе 3.4. В данном разделе приведено руководство по созданию таких диаграмм. В конце раздела приведен пример диаграммы, учитывающий большую часть используемых особенностей.

Для начала работы следует запустить новый проект (*File — New project*) и создать новую диаграмму состояний () (*Create — New statechart diagram*). В меню в левой части экрана будут отображаться все объекты текущей модели, в том числе все созданные диаграммы состояний. Для перехода к редактированию конкретной диаграммы следует

развернуть подменю, соответствующее этой диаграмме, и активировать щелчком мыши описание диаграммы (по умолчанию — *UntitledModel*).

Состояние добавляется на диаграмму щелчком на изображение состояния и затем в нужном месте на поле. На диаграммах разрешается использовать простые состояния () (*Simple state*), композитные состояния () (*Composite state*), ссылки () (*Submachine state*), входы () (*Initial state*) и выходы () (*Final state*). Для добавления переходов следует активировать щелчком значок перехода () и перетаскиванием (*drag-n-drop*) соединить два состояния. Переходы далее для краткости будем называть дугами.

Созданные состояния и дуги отображаются в меню в левой части экрана. Состояния, вложенные в s, и инцидентные ему дуги появляются в подменю, соответствующему s.

Композитное состояние состоит из одного или нескольких параллельных регионов () (*Concurrent region*), в каждый из которых вложена диаграмма, учитывающая описанные далее ограничения. Вложенность диаграмм соответствует геометрической вложенности их изображений в средстве ArgoUML. Диаграммы, вложенные в композитное состояние, представляют собой параллельно работающие компоненты этого состояния. Каждая диаграмма при этом подразумевает последовательное выполнение. Добавить параллельный регион можно правым щелчком на композитном состоянии и выбором соответствующего пункта меню.

Простое состояние представляет собой элементарное состояние компонента системы, выраженного объемлющим композитным состоянием.

Вход и **выход** – это служебные состояния, обозначающие, соответственно, начало и завершение работы компонента системы, выраженного объемлющим параллельным регионом (или композитным состоянием без параллельных регионов). В каждый параллельный регион должны быть вложены ровно один вход и не более одного выхода.

Дуги могут соединять между собой состояния произвольным образом. При этом дуге, соединяющей состояния различных диаграмм, равносильна следующая конструкция. Вычисляется ближайшее общее объемлющее состояние этих диаграмм. На каждом уровне вложенности между этим предком и двумя диаграммами добавляются выходы и входы, и в них направляются дуги так, чтобы поочередно завершить и затем начать работу диаграмм всех уровней. Для корректной работы в этой равносильной конструкции не должно быть дуг, соединяющих диаграммы разных параллельных регионов. Пример описанной равносильной конструкции приведен на рисунке 126. На входы и выходы налагается еще ряд ограничений,

а именно: из входа должна исходить ровно одна дуга, из выхода – ни одной, во вход не должно входить ни одной дуги.

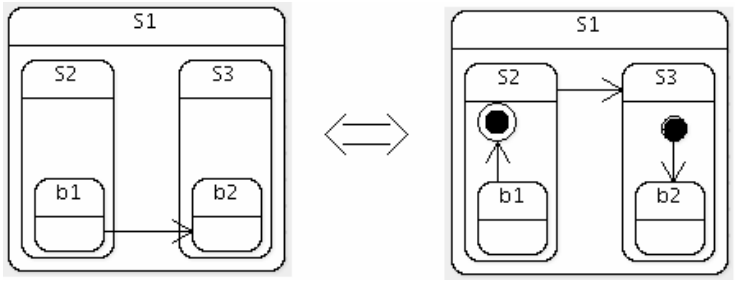

Рисунок 126. Значение входов и выходов

Ссылки используются для сокращения записи диаграммы. Ссылка используется в диаграмме так же, как и простое состояние. При трансляции ссылки устраняются путем подстановки вместо них диаграмм, на которые они ссылаются. Для оформления ссылки в модели заводится новая диаграмма состояний. В поле *Подавтомат* (*Submachine*) свойств ссылки указывается диаграмма, которая будет подставлена при трансляции.

Кроме того, ссылка может иметь параметры. Для добавления параметров нужно в выпадающем меню свойства *Действие при входе* (*Entry action*) выбрать значок 

и затем в многострочном поле *script* определить список фактических значений параметров. Запись *@param = val;* в этом поле присваивает параметру *param* значение *val*. При подстановке диаграммы вместо ссылки все ее параметры подставляются тем же способом, что и макроопределения (см. блок деклараций).

Блок деклараций создается следующим образом. Выделяется композитное состояние. По щелчку на значке комментария () (*Comment*) создается комментарий, привязанный к выделенному состоянию. Все имена, описываемые в комментарии, являются локальными для выделенного состояния. Многострочное поле *Body* созданного комментария может содержать следующие записи:

- *int [a..b] x = c;*, где *a, b, c* – целые числа, – задание новой переменной *x*, принимающей целые значения в отрезке от *a* до *b* с начальным значением *c*;
- *bool b = val;*, где *val* – либо *true*, либо *false*, – описание булевой переменной *b* с начальным значением *val*;
- *clock c;* – задание таймера *c*, принимающего действительные значения;
- *#define Name Text* – задание макроопределения в стиле языка С: при дальнейшем разборе выражений на место каждого идентификатора *Name* будет подставлено выражение *Text*;
- *signal s;* – задание канала *s* широковещательной синхронизации.

Так как все переменные, таймеры и каналы, описанные в комментарии, являются локальными для состояния, при наличии нескольких ссылок на одну диаграмму они считаются различными; при совпадении имен различных локальных переменных транслятор переименовывает последнюю встретившуюся. Все переменные, таймеры и каналы, используемые в выражениях, должны быть заданы в комментариях с учетом локальности. Отдельно следует отметить, что для каждого состояния считается определенным таймер *self.c*, обозначающий время, прошедшее с начала последней активации данного состояния.

Метки дуг. Дуга может быть помечена предусловием, действиями, триггером приема сигнала и временным триггером. Предусловие создается щелчком на значке (поле *Сторожевое условие*, *Guard*). Действие создается щелчком на значке  в выпадающем меню поля *Результат* (*Action*). Триггеры создаются щелчком на нужный пункт в выпадающем меню поля *Переключающее событие (триггер)* (*Trigger*): триггер приема сигнала обозначен значком  , временной триггер –  .

Если требуется не создавать новую метку, а редактировать уже существующую, то следует активировать соответствующую метку двойным щелчком на поле справа от кнопки создания (для предусловий и действий) либо щелчком на значке  справа от кнопки создания (для триггеров).

Один и тот же триггер (как приема сигнала, так и временной) может быть назначен различным дугам. Изменение содержания триггера влечет его изменение на всех дугах, которым он назначен. Назначение общего триггера приема сигнала означает прием широковещательного сообщения по общему широковещательному каналу связи, назначение общего временного триггера – общие временные условия перехода.

Для описания меток дуг условимся символом *E* обозначать арифметические выражения над переменными, символом *x* – переменные, символом *t* – таймеры.

Предусловие – это необходимое условие выполнения перехода. Оно описывается в многострочном поле *expression* после создания и выражается булевой формулой (&&, ||, !) над выражениями вида $(E_1\ op\ E_2)$ и $(t\ op\ E)$, где *op* – один из знаков сравнения <, ≤, =, ≥, >. Синтаксис и значение этих и всех дальнейших выражений совпадает с таковыми для выражений языка C.

Действия – это список присваиваний вида *x = E;* и *t = 0;*, после которого может идти посылка *!!c* широковещательного сигнала по каналу *c*. Кроме того возможно использование записи *x = random();* для обозначения присваивания произвольного значения переменной *x* целочисленного типа. Действия описываются в многострочном поле *script* после создания.

Триггер приема сигнала с именем *s* означает прием широковещательного сигнала по каналу *s*. Имя триггера устанавливается в поле *Имя* (*Name*) его свойств. Имя созданного триггера отображается в меню в левой части экрана.

Временной триггер при создании также отображается в меню слева. В поле *Имя* (*Name*) можно задать имя триггера, в многострочном поле *when* записывается сравнение таймера с арифметическим выражением, которое при трансляции будет добавлено к предусловию дуги. Изменение поля *when* триггера приводит к одновременному изменению выражений на всех дугах, которым он присвоен.

Перейдем теперь к **свойствам состояний**. Допустимыми метками состояний являются имя, параметр шаблона и инвариант.

Имя состояния задается в поле *Имя* (*Name*). Имя состояния может использоваться при описании свойств модели.

Инвариант задается в поле *Деятельность выполнения* (*Do activity*) – пункт  в выпадающем меню. Инвариант – это выражение вида *assume(E)*, где сравнениями в выражениях, содержащих таймеры, могут быть только < и ≤.

В целях компактности записи диаграммы и группы диаграмм могут быть «упакованы» в **классы**. Для этого в меню в левой части экрана с помощью меню, выпадающего по щелчку правой кнопкой мыши, нужно создать класс () (*New class*) и затем в диаграмме указать этот класс как контекст в поле *Контекст* (*Context*).

На рисунках 127–130 приведен пример диаграммы, использующей большую часть описанных особенностей.

После завершения написания диаграммы необходимо экспортировать ее в формат XMI (*Файл – Экспортировать как XMI*; *File – Export as XMI*). Вся дальнейшая работа производится с полученным файлом в формате XMI.

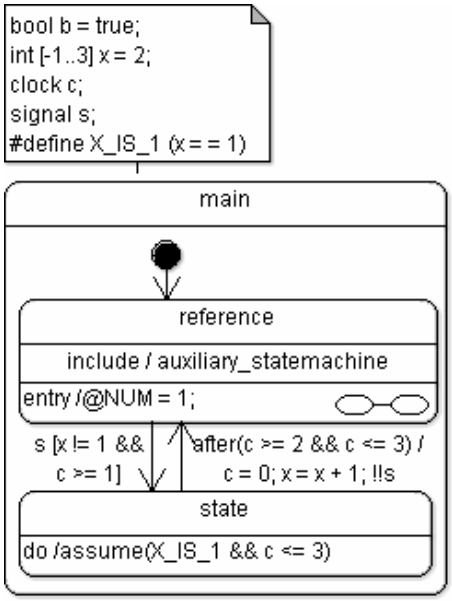

Рисунок 127 – Основная диаграмма

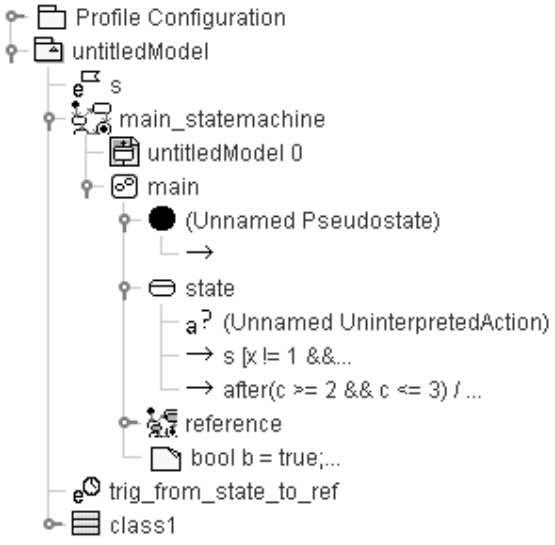

Рисунок 128 – Меню основной диаграммы

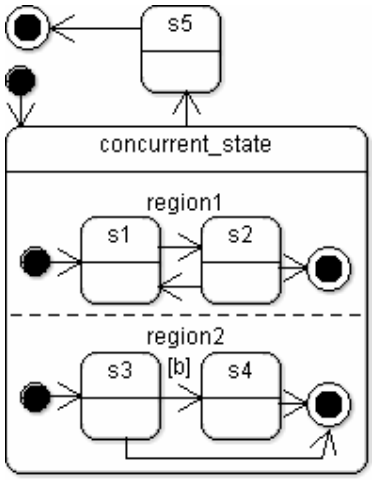

Рисунок 129 – Диаграмма ссылки

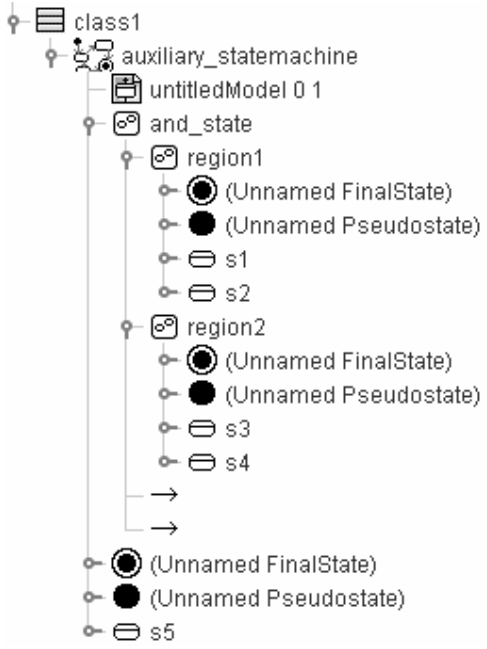

Рисунок 130 – Меню ссылки

8.3 Методика использования средства визуализации трассы

При работе со средством визуализации Vis4 (раздел 4.7) можно выделить три панели: панель инструментов; информационная панель, отображающая информацию о текущем выделенном элементе (трассе, компоненте, состоянии, обмене) и непосредственно панель визуализации трассы. Для работы с Vis4 предварительно необходимо установить библиотеку libotf.

В рамках методики использования средства визуализации и анализа трассы Vis4 проверяются следующие основные возможности средства:

- Открытие и закрытие трассы;
- Получения информации о событии, обмене и состоянии компонента РВС РВ;
- Масштабирование трассы;
- Навигация по трассе;
- Навигация по иерархии компонентов.

8.3.1 Запуск Vis4, открытие и закрытие трассы

В Vis4 используется формат OTF, выбранный на основе обзора в разделе 3.7. Для открытия трассы необходимо запустить терминал (рисунок 131) и выполнить команду

./vis «название трассы.otf»

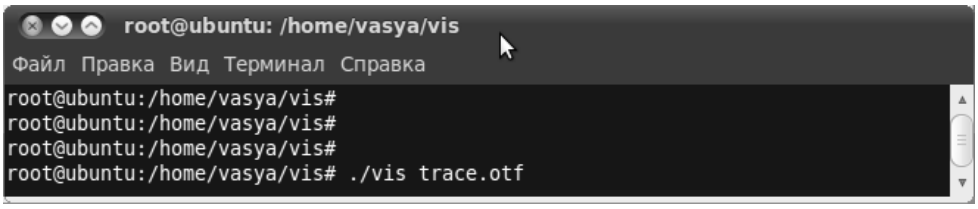

Рисунок 131. Запуск Vis4 в терминале.

Для закрытия трассы необходимо закрыть Vis4 с помощью кнопки  .

8.3.2 Панель управления

Панель управления Vis4 представлена на рисунке 132. Панель управления позволяет осуществлять перемещение по трассе, масштабирование и печать трассы. Панель управления предоставляет пользователи возможности навигации по трассе, масштабирования трассы, печати трассы.

Рисунок 132. Запуск Vis4 в терминале.

Масштабирование трассы может осуществляться следующими способами:

1. Нажать кнопку  . Масштаб будет установлен таким, что в рабочей области отображения временной диаграммы будет виден весь временной интервал эксперимента (с учётом замечания ниже). Временная диаграмма будет перерисована в соответствии с новым масштабом (время перерисовки может составлять десятки секунд).

2. Нажать кнопку  или  . Масштаб соответственно будет увеличен или уменьшен в два раза относительно исходного. Цена большого деления временной оси будет соответственно уменьшена или увеличена в два раза. Временная диаграмма будет перерисована в соответствии с новым масштабом. Значения левой и правой границ будут изменены соответственно новому масштабу относительно неизменного центра числовой оси.

3. Нажать на клавиатуре клавишу *Shift* и левую кнопку мыши или *Shift* и правую кнопку мыши. Масштаб будет изменен аналогично предыдущему способу. Значение левой и правой границы числовой оси будет изменено соответственно новому масштабу относительно неизменной точки под курсором мыши.

4. Нажать клавиши «+» или «-» в правой части клавиатуры. Масштаб соответственно будет увеличен или уменьшен вдвое. При этом значения левой и правой границ числовой оси будут изменены соответственно новому масштабу относительно положения визирной линии.

Навигация по трассе осуществляется следующим способами:

– Кнопки  предназначены для «плавного» перемещения по трассе.

– Кнопки  предназначены для быстрого перемещения в начало или конец трассы.

Для печати выбранного фрагмента трассы необходимо нажать кнопку  панели инструментов.

8.3.3 Информационная панель

На информационной панели (рисунок 133), в процессе работы постоянно отображается информация о текущем выделенном объекте трассы (трассе, компоненте, состоянии, событии), а также отображаются подсказки раздела «помощь». Например, для получения информации о состоянии компонента (тип, время начала, продолжительность) необходимо навести указатель мыши на состояние на линии жизни выбранного компонента.

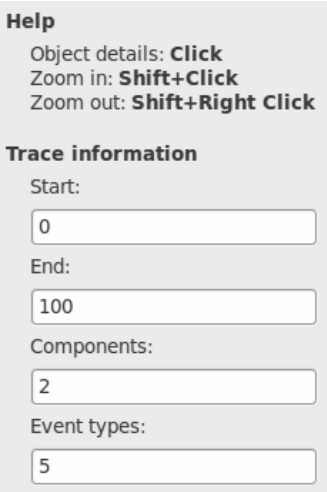

Рисунок 133. Информационная панель.

8.3.4 Панель визуализации трассы

На панели визуализации (рисунок 134) трассы поддерживаются следующие возможности: навигация по иерархии объектов, навигация вдоль линий жизни элементов, возможности изменения единиц и формата времени, масштаба трассы.

Рисунок 134. Панель визуализации трассы

В РВС РВ компоненты могут быть простыми и составными. Составные компоненты имеют в своем составе подкомпоненты, по которым в свою очередь может быть осуществлена навигация. Простые компоненты отображаются в рабочей области отображения ВД в виде прямоугольников 

, а составные в виде стопки прямоугольников 

.

К составным компонентам может быть применена операция «раскрытия». Для раскрытия компонента следует щёлкнуть мышью на имени компонента в рабочей области отображения временных диаграмм. В результате в рабочей области выводится поддерево имен и линий жизни подкомпонентов в составе раскрываемого составного компонента Для возврата вверх на уровень по иерархии следует щёлкнуть мышью на имени раскрытого компонента, находящемся в корне изображенного поддерева.

Для масштабирования трассы в области ее визуализации необходимо воспользоваться колесиком мыши.

Значения модельного времени на временной оси могут отображаться как в единицах времени простого формата («**ч**», «**м**», «**с**», «**мс**» или «**мкс**»), так и формата с разделителями с указанием точности («**ч:м:с.**<единицы точности>»).

Для изменения формата значений модельного времени временной оси надо выполнить следующие действия:

а) щелкнуть мышью по полю единиц времени (рисунок 135);

б) во всплывающем меню установить курсор мыши на строке «**Формат времени**»;

в) в появившемся списке форматов щелкнуть мышью по полю одного из форматов: «**простой**» или «**с разделителями**» (рисунок 135) - единицы времени в новом формате будут установлены в поле единиц времени; большие деления временной оси будут помечены значениями модельного времени в новом формате.

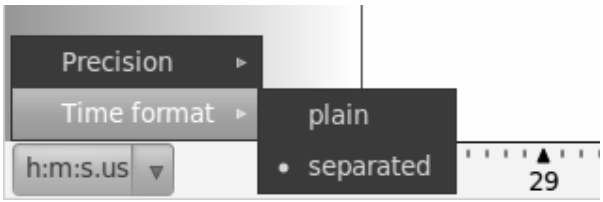

Рисунок 135. Выбор формата времени

Для изменения единиц измерения значений модельного времени временной оси надо выполнить следующие действия:

а) установить простой формат времени как описано выше (если он не был предварительно установлен);

б) во всплывающем меню (рисунок 136) установить курсор мыши на строке «**Единицы измерения**»;

в) выбрать единицы в появившемся списке единиц измерения времени: «**ч**», «**м**», «**с**», «**мс**» или «**мкс**» - выбранные единицы установятся в поле единиц времени; большие деления временной оси будут помечены значениями модельного времени в выбранных единицах измерения.

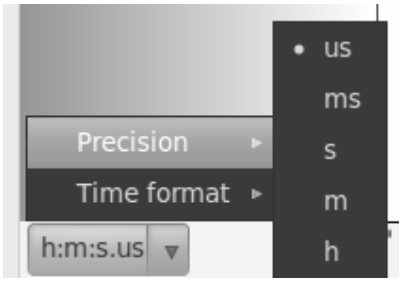

Рисунок 136. Выбор формата времени

В данном разделе была описана методика использования средства визуализации трасс Vis4.

8.4 Методика использования средства трансляции UML во временные автоматы

В данном разделе описано применение интегрированной среды для верификации моделей на UML. Для верификации свойств моделей РВС РВ, написанных на языке UML, необходима трансляция в автоматы UPPAAL, подробно описанная в разделах 4.8, 5.1, 5.2.

Первый шаг – запуск программы. Открывается главное окно (рис. 137).

Далее необходимо создать новый проект через пункт меню Файл – Новый проект (рис. 138).

Создался пустой проект с четырьмя папками и без файлов (рис. 139).

Далее необходимо создать модель на UML. О создании диаграмм UML подробно описано в разделе 8.2. В примере на рисунке 140 для иллюстрации нарисована простейшая диаграмма с двумя сменяющими друг друга состояниями. В одно из них добавлен исходный код, подаваемый средству оценки наихудшего времени выполнения.

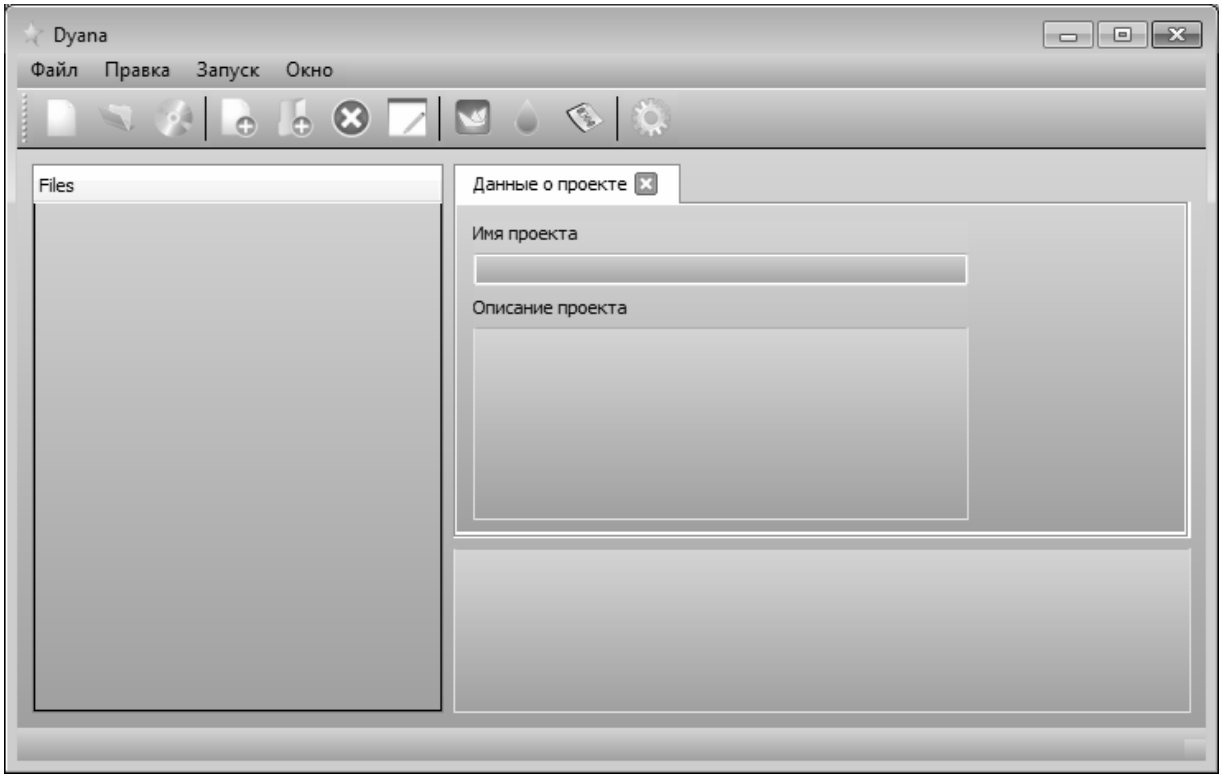

Рисунок 137. Главное окно

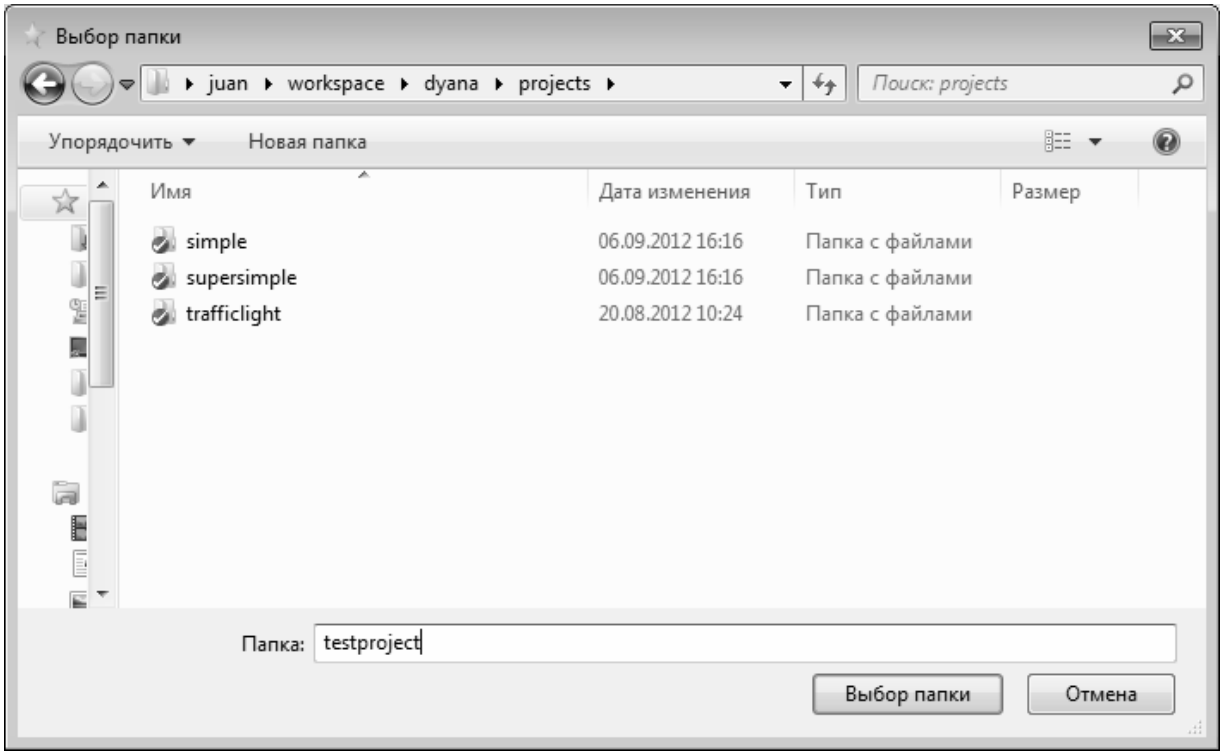

Рисунок 138. Создание нового проекта.

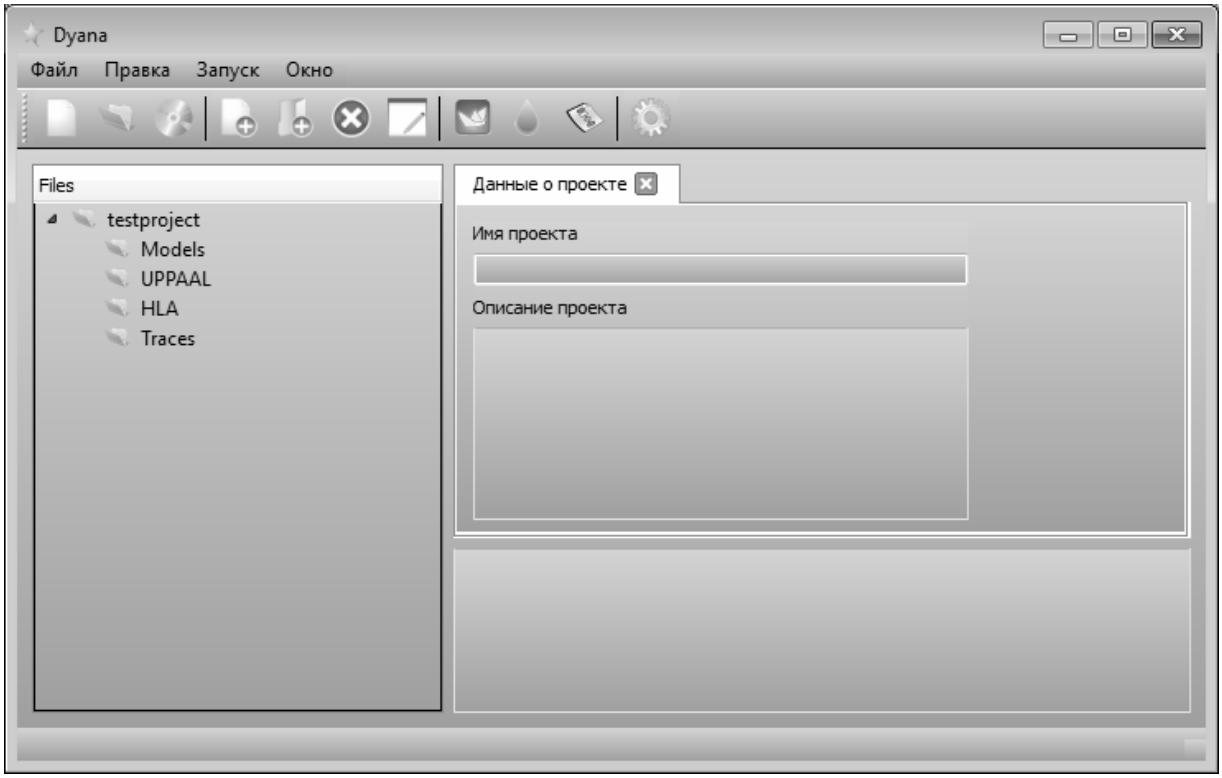

Рисунок 139. Создан Новый проект.

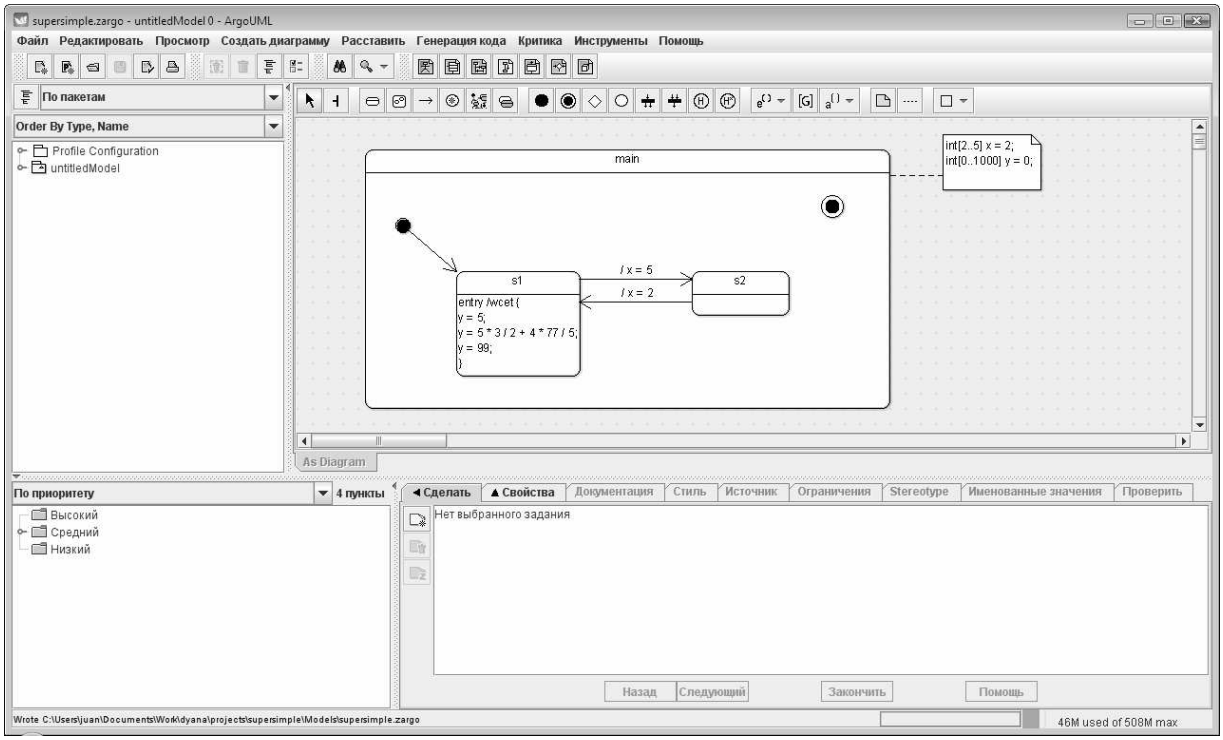

Рисунок 140. Диаграмма UML

Используя пункт контекстного меню «Добавить файл», можно добавить в проект созданный файл с расширением zargo (в данном примере supersimple.zargo) и

соответствующий ему файл с расширением xmi в папку Models (рис. 141). Также можно добавить файлы со свойствами UPPAAL, о которых речь пойдет в следующем разделе.

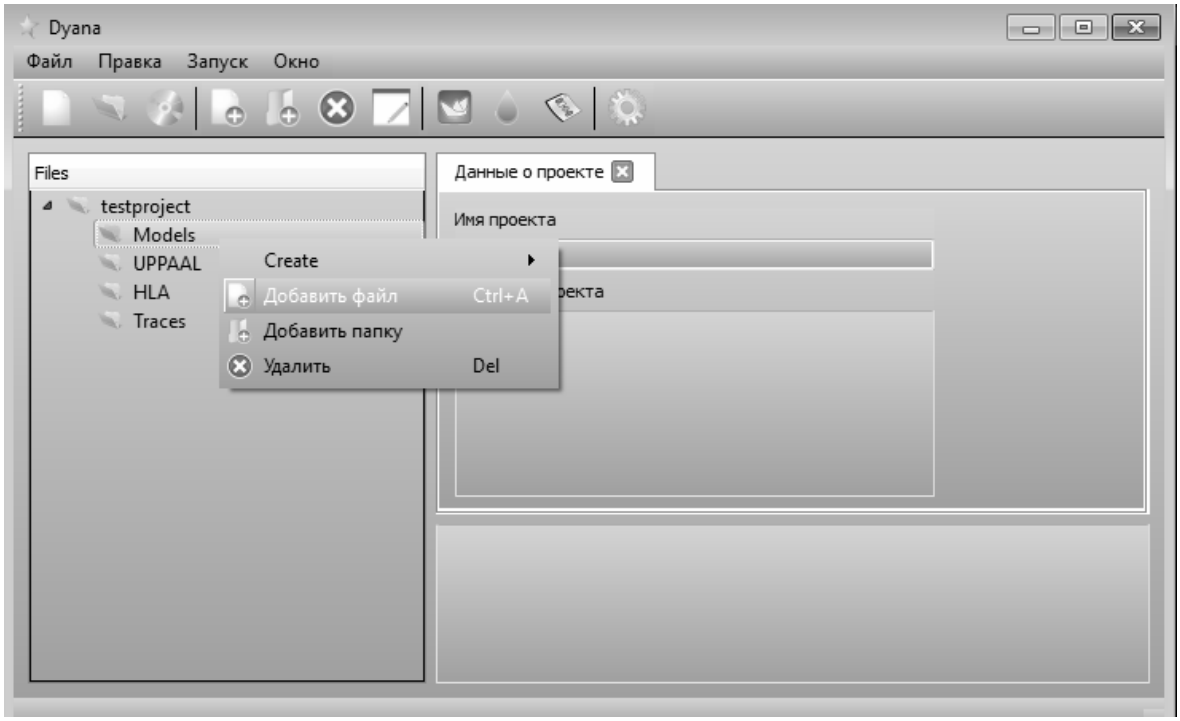

Рисунок 141. Добавление файлов.

В результате получается проект с четырьмя файлами (рис. 142).

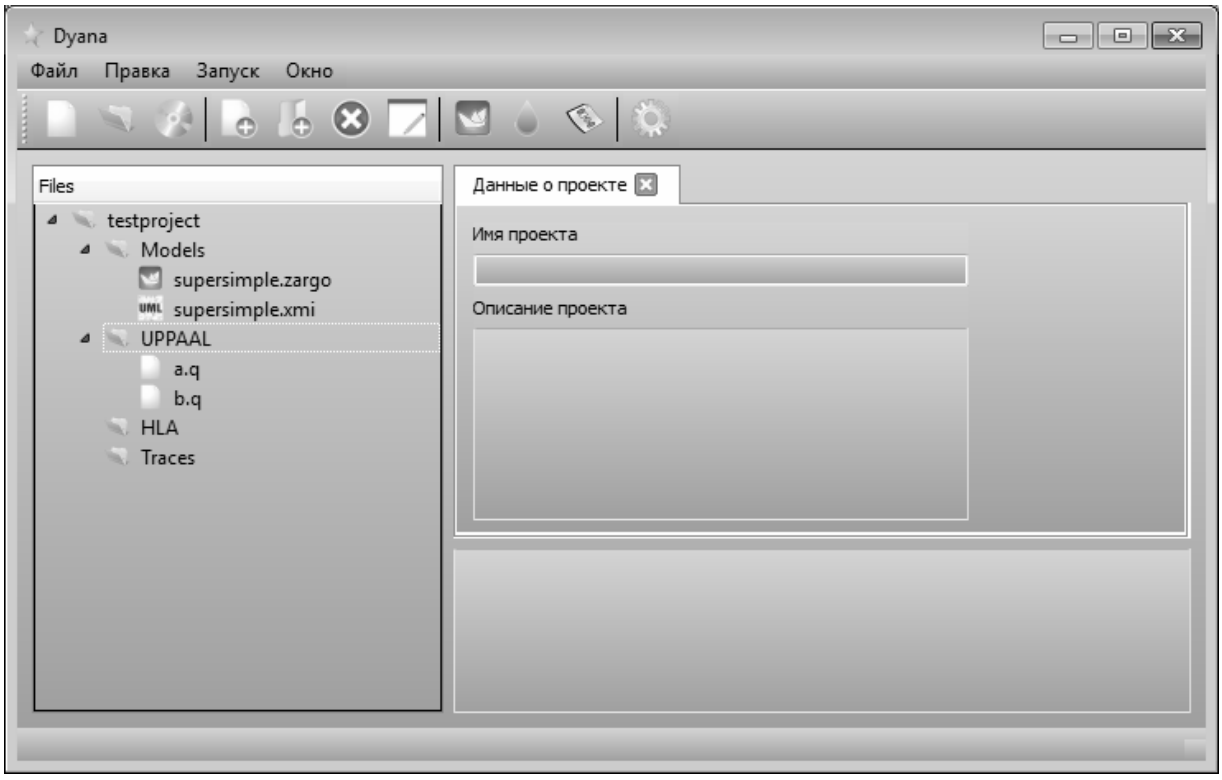

Рисунок 142. Проект после добавления файлов

Файл supersimple.xmi открывается двойным кликом. Появляется вкладка с возможными действиями (рис. 143).

Селектор «Оптимизация» позволяет уменьшить число автоматов в получаемой системе UPPAAL, селектор «Оценка времени» включает использование средства оценки времени выполнения (WCET). После установки обоих этих селекторов, нажатие на кнопку «Преобразовать» запускает конвертацию в автомат UPPAAL.

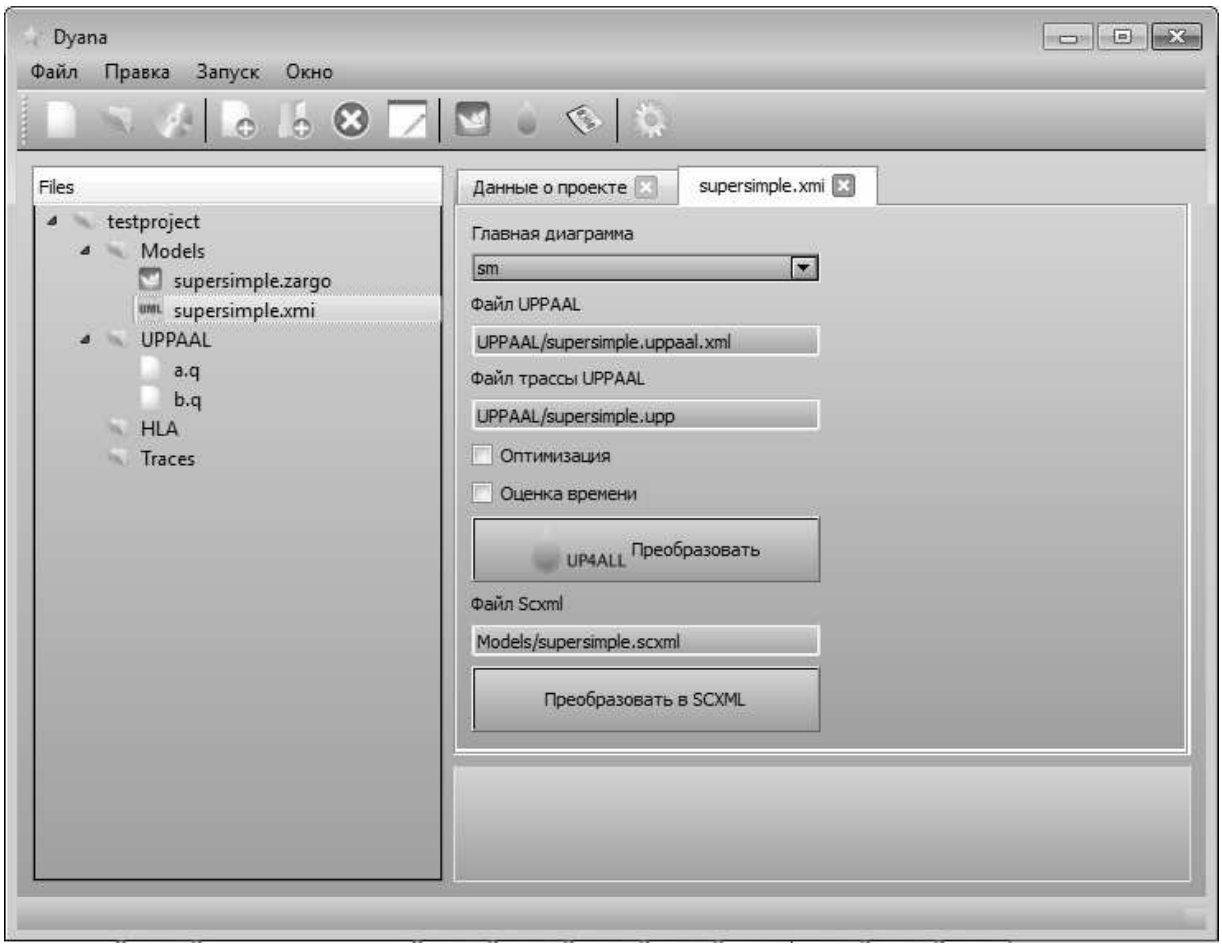

Рисунок 143. Вкладка для файла .xmi

В результате создается файл supersimple.uppaal.xml, готовый для запуска в UPPAAL, и служебный файл supersimple.upp (рис. 144).

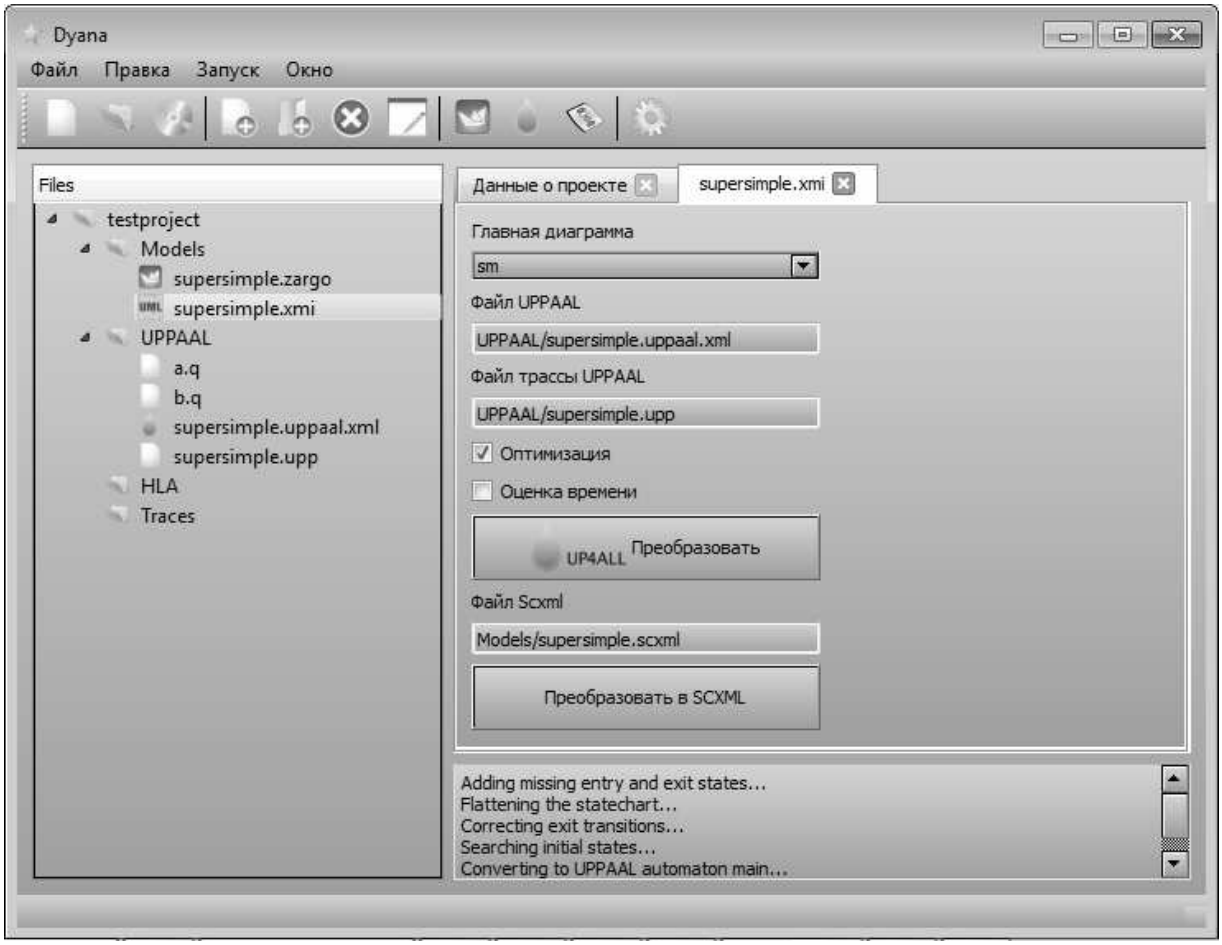

Рисунок 144. Файлы UPPAAL, сгенерированные программой

Методика использования самого средства верификации UPPAAL. будет приведена в разделе 8.5.

8.5 Методика использования средства верификации модели

В предыдущем разделе было описано, как создать систему автоматов UPPAAL по диаграмме UML. В данном разделе будет показано, как верифицировать свойства полученной системы в UPPAAL.

В папке UPPAAL лежат два файла со свойствами для верификатора. Файл a.q содержит свойство A[] x > 0, которое заведомо выполнимо. Файл b.q содержит свойство A[] x < 3, которое не выполняется.

Создавшийся файл .uppaal.xml можно открыть двойным кликом. Обратим внимание, что список доступных файлов со свойствами и файлов .upp подгрузился автоматически (рис. 145).

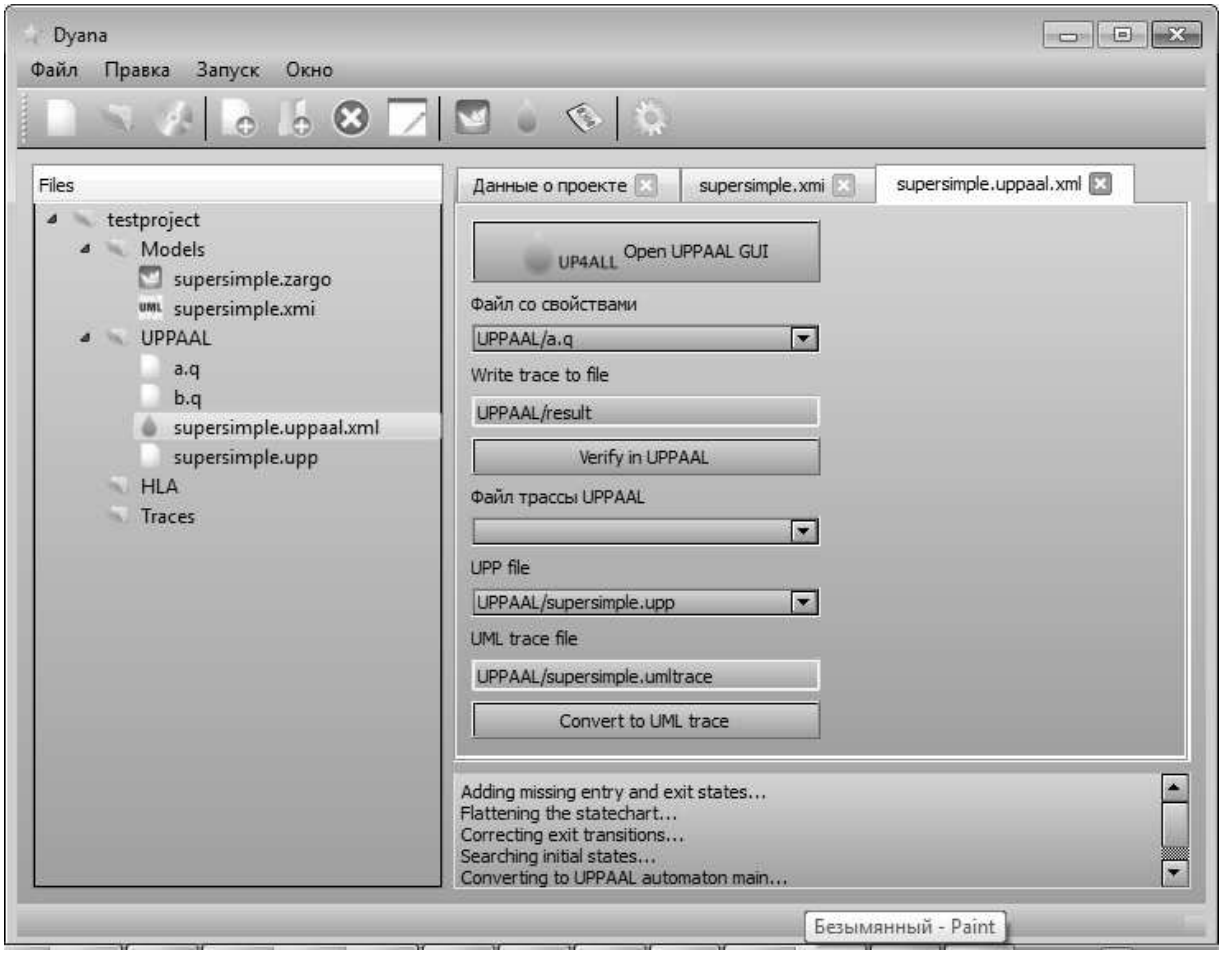

Рисунок 145. Вкладка для файла UPPAAL

Чтобы запустить верификацию, надо выбрать файл со свойствами (чтобы проиллюстрировать работу с контрпримерами, выбран файл b.q) задать путь к трассе контрпримера «UPPAAL/result», и кликнуть «Верифицировать в UPPAAL». В результате в проекте появится контрпример в файле result-1.xtr (рис. 146, рис. 147).

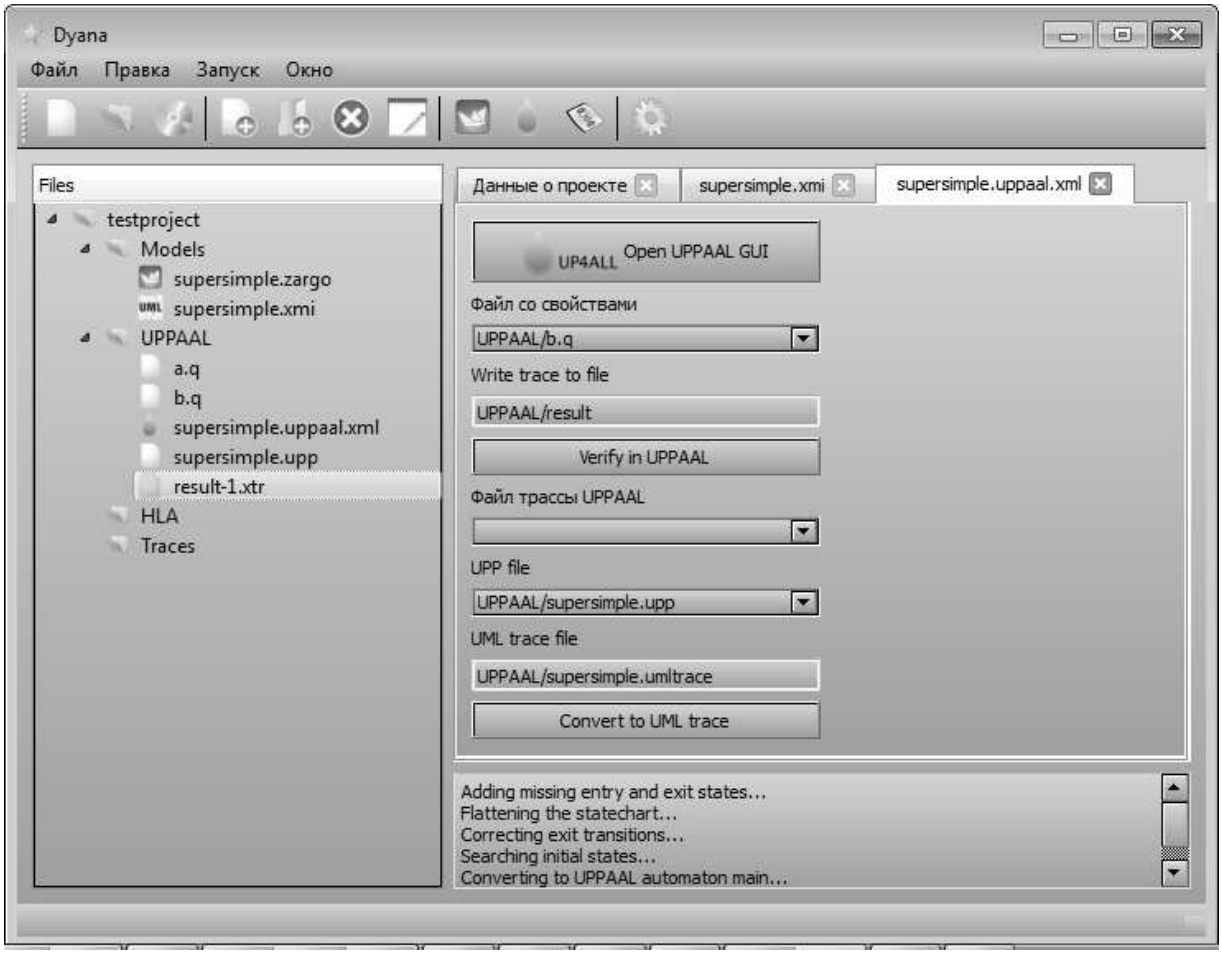

Рисунок 146. Появление контрпримера

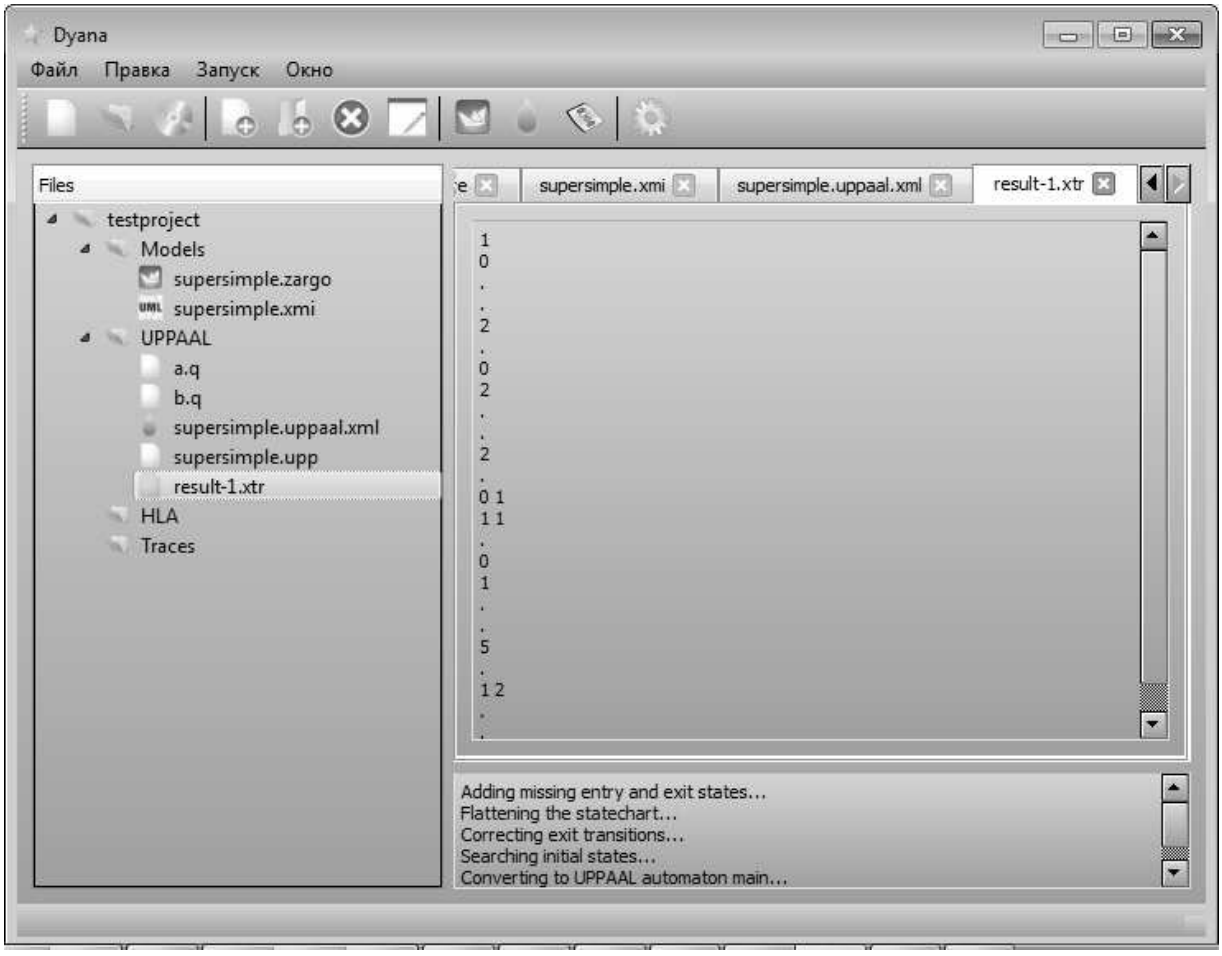

Рисунок 147. Содержимое файла контрпримера UPPAAL

Чтобы увидеть контрпример в терминах диаграммы UML, нужно вернуться на вкладку для диаграммы UPPAAL, выбрать в списках нужный контрпример, файл .upp, задать имя для UML-трассы и нажать кнопку «Конвертировать в трассу UML». В результате появится файл с трассой UML (рис. 148).

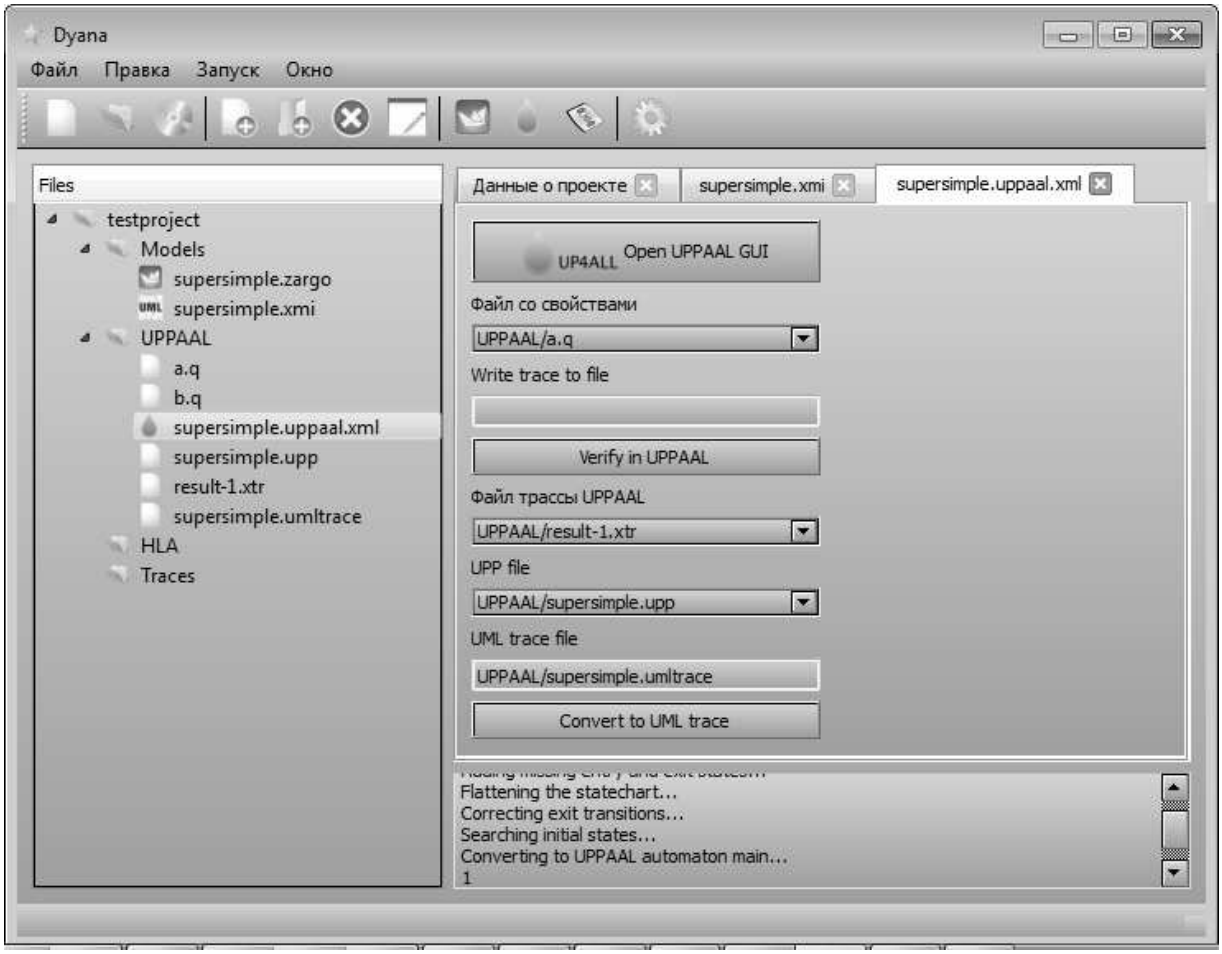

Рисунок 148. Создание файла трассы UML

Полученный файл .umltrace, отражающий найденный контрпример в терминах UML-диаграммы, можно просмотрев, открыв его двойным кликом (рис. 149).

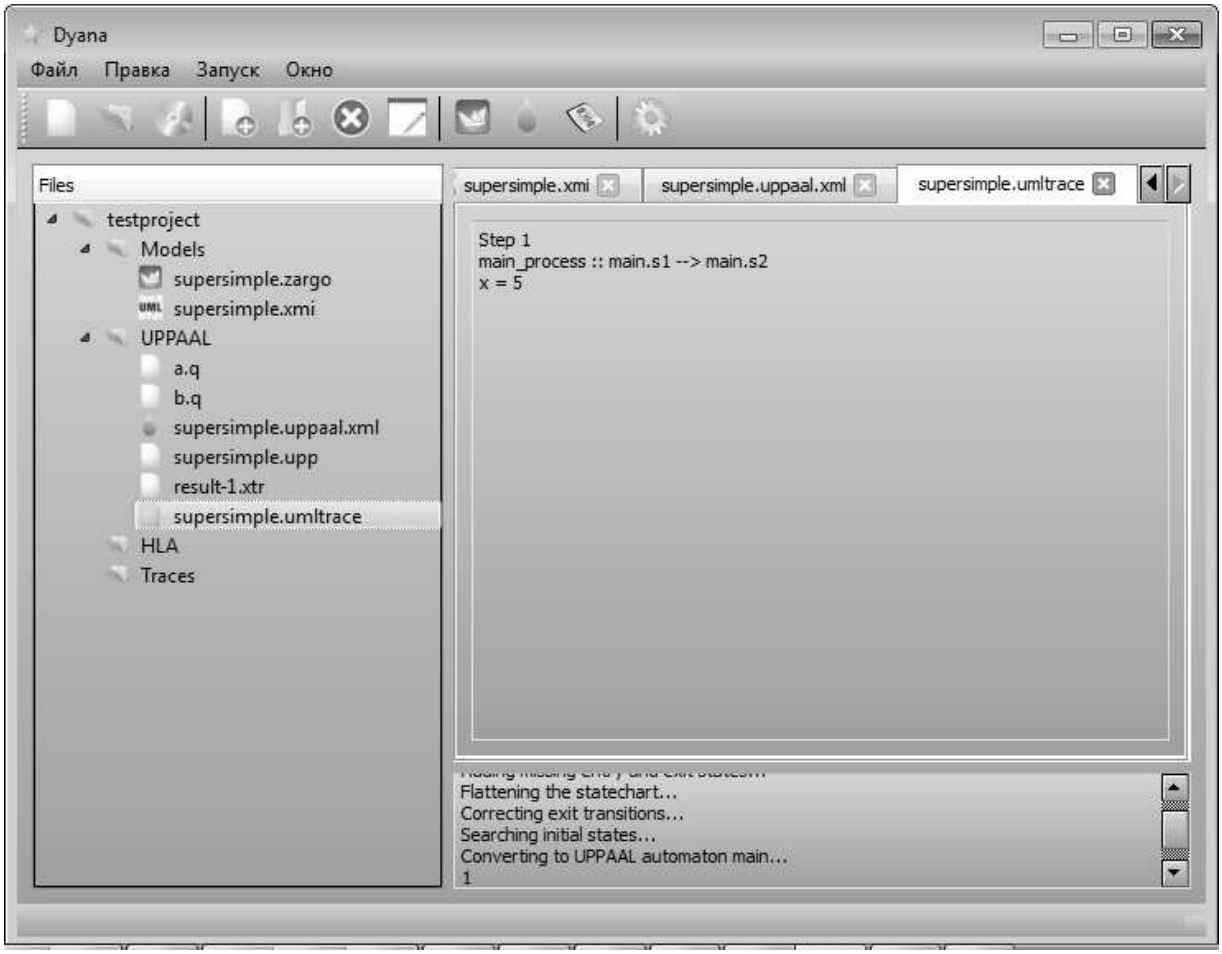

Рисунок 149. Содержимое трассы UML

Также можно с системой автоматов работать непосредственно в графическом интерфейсе UPPAAL. Запустить его можно кнопкой «Открыть UPPAAL GUI». Графический интерфейс UPPAAL визуализирует систему автоматов (рис. 150). На иллюстрации виден инвариант wcet_timer_1 < 10, добавленный средством оценки наихудшего времени выполнения.

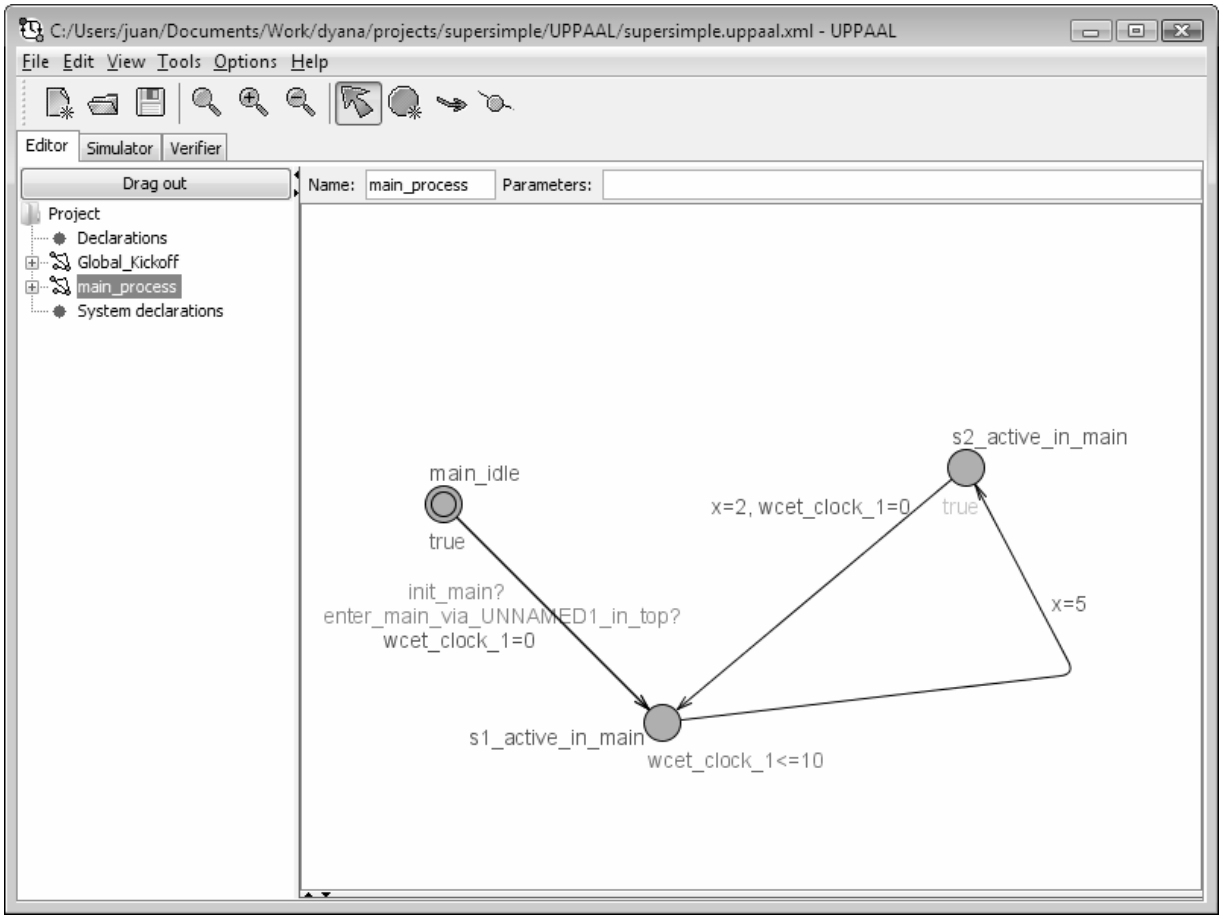

Рисунок 150. Графический интерфейс UPPAAL

Перейдя на вкладку «Verifier», можно верифицировать те же самые свойства (рис. 151).

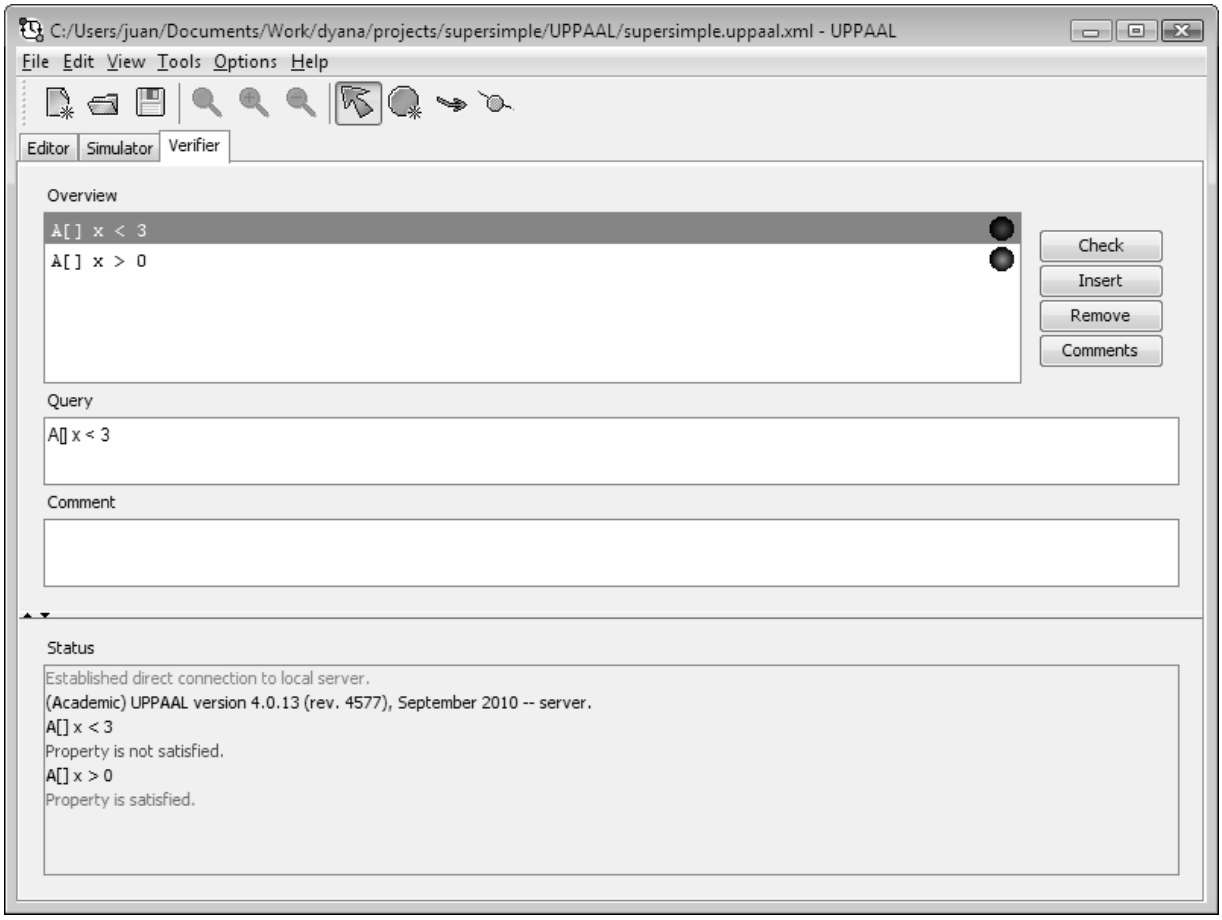

Рисунок 151. Верификация свойств в UPPAAL

На вкладке «Simulator» визуализируется трасса контрпримера (рис. 152).

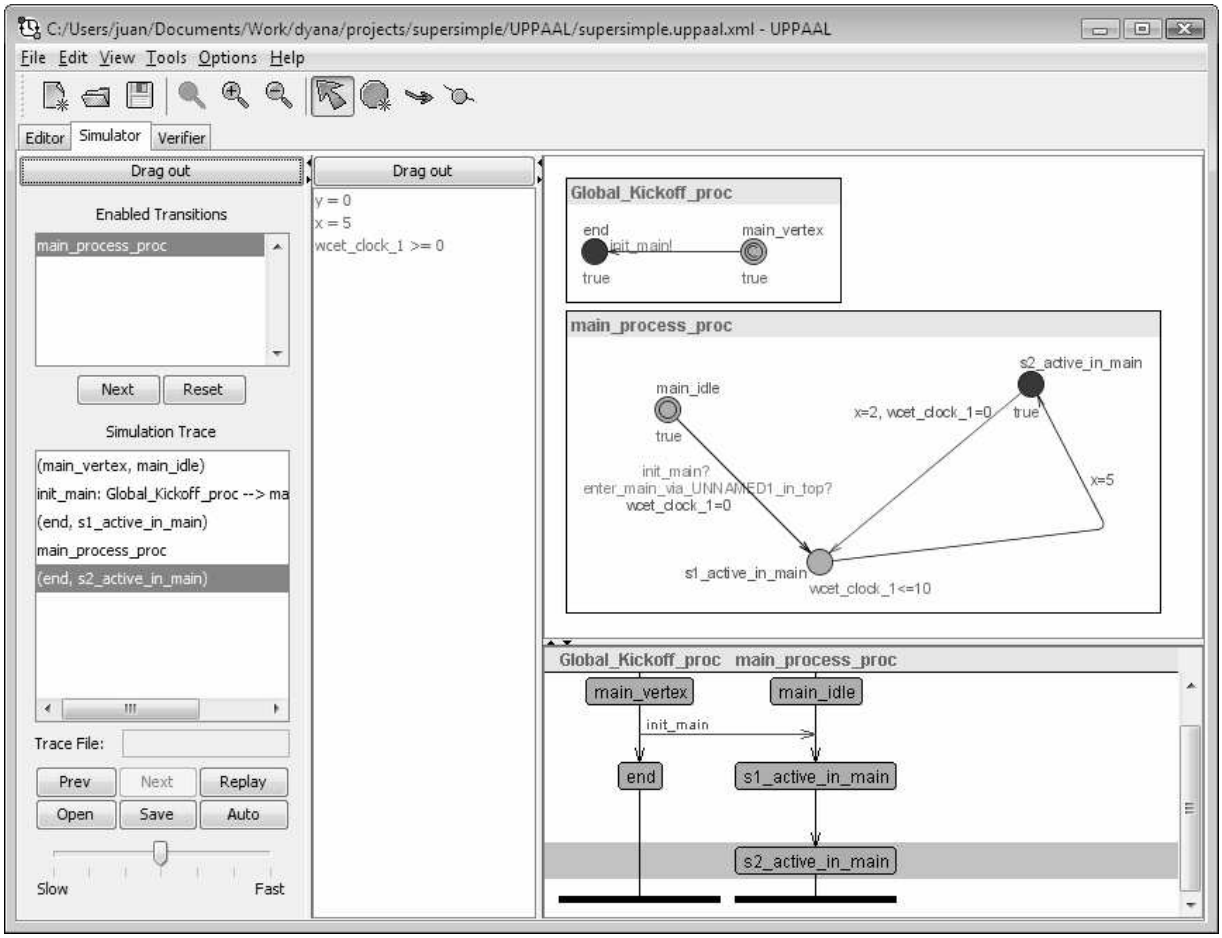

Рисунок 152. Просмотр трассы в UPPAAL

В этом разделе было показано, как верифицировать свойства полученной системы в UPPAAL. Подробное экспериментальное исследование средств верификации приведено в разделе 9.6.

8.6 Методика использования средств моделирования совместно со средствами синтеза архитектур и построения расписаний

В данном разделе приведена методика использования средства решения задачи выбора МОО РВС РВ. Задача выбора МОО РВС РВ является задачей синтеза архитектуры и при её решении необходима интеграция со средствами моделирования. Постановка задачи и реализованные алгоритмы её решения подробно описаны в разделе 5.8.

8.6.1 Общее описание программного средства

Рассматриваемое в данном разделе средство представляет собой программу на C++, которая позволяет находить конфигурацию РВС РВ, имеющую высокую надежность при ограничениях на стоимость и время выполнения программ в каждом модуле. Входными

данными для средства является описание общей структуры РВС РВ, наборы доступных вариантов аппаратных и программных компонентов каждого модуля и времена выполнения программных компонентов для каждой пары (аппаратный компонент, программный компонент). Описание общей структуры РВС РВ представляет собой описание графа зависимостей по данным между его модулями, а также описание доступных МОО, максимальной допустимой стоимости и максимальных допустимых времен завершения работы программных компонентов в каждом модуле. Для каждого доступного варианта аппаратного или программного компонента задана его стоимость и надежность.

Для решения задачи выбора МОО РВС РВ можно использовать один из четырех видов эволюционных алгоритмов, либо алгоритм имитации отжига. Так как эволюционные алгоритмы имеют много параметров, от которых сильно зависит их эффективность, то входными данными также является набор параметров алгоритма.

Выходными данными программного средства является лучшая найденная конфигурация РВС РВ, ее характеристики, а также статистика работы алгоритма.

8.6.2 Формат вызова

Реализованное средство оптимизации надежности РВС РВ имеет консольный интерфейс со следующим форматом вызова:

./rap [параметры] system.xml [time.txt]

Программное средство поддерживает следующие параметры командной строки:

‘-a ALG’ – тип алгоритма оптимизации (“EA” – эволюционный алгоритм, “SA” – алгоритм имитации отжига). Если данный параметр не задан, запускается эволюционный алгоритм. Способ задания конкретного вида ЭА (гибридный, классический, островной классический, островной гибридный) будет описан ниже.

‘-c’ – в ходе работы алгоритма не учитывать ограничения на время.

‘-m’ – интеграция со средствами моделирования (проведение имитационных экспериментов для проверки ограничений на время выполнения программ). Если данный параметр не задан, время окончания работы программных компонентов в каждом модуле оценивается аналитически.

‘-x file.xml’ – путь к файлу, содержащему настройки эволюционного алгоритма. Для алгоритма имитации отжига аналогичный файл не нужен. Формат данного файла будет приведен в разделе 8.6.3. По умолчанию берется файл **“input/ga.xml”.**

system.xml – xml-файл, содержащий описание РВС РВ, для которой решается задача. Формат данного файла будет рассмотрен в разделе 8.6.4.

time.txt – файл, содержащий информацию о «чистом» времени выполнения программ на заданной аппаратуре. Для каждой тройки (номер модуля, номер аппаратного компонента, номер программного компонента) указано время работы данного программного компонента на данном аппаратном компоненте.

Для одного набора входных данных оптимизационный алгоритм запускается несколько раз, и результаты его работы записываются в файл формата CSV. Для каждого запуска алгоритма в этом файле приводится полученное решение, значение его целевой функции, стоимость, времена работы всех программных компонентов, количество итераций алгоритма, количество проведенных имитационных экспериментов, время работы алгоритма.

8.6.3 Формат описания исследуемой РВС РВ

Исследуемая РВС РВ должна быть описана в файле формата XML, содержащем следующие теги:

<system> – корневой элемент в описании системы. Имеет атрибут limitcost – максимальное допустимое значение стоимости РВС РВ. Содержит внутри себя теги <tool> и <module>.

<tool> – механизм обеспечения отказоустойчивости. Имеет атрибуты name – имя МОО; use – флаг использования МОО (use=”1”, если данный МОО используется, и use=”0” иначе).

<module> – модуль РВС РВ. Имеет атрибуты num – номер; name – имя модуля; pall – вероятность одновременного отказа всех версий программного компонента; prv – вероятность отказа между двумя версиями программного компонента; pd – вероятность отказа схемы голосования в модуле; limittime – максимальное допустимое время завершения работы компонента; volume – объем выходных данных, передаваемых каждому из модулей, зависящих от данного модуля. Имеет внутри себя теги <software>, <hardware> и секцию <waitfor>, содержащую теги <mod> и задающую модули, от которых данный модуль зависит по данным. Тег <mod> имеет единственный атрибут num – номер модуля, от которого необходимо ожидать данные.

<software> – вариант программного компонента, который может быть использован в модуле. Имеет атрибуты name – имя, reliability – надежность (вероятность безотказной работы), cost – стоимость.

<hardware> – вариант аппаратного компонента, который может быть использован в модуле. Имеет атрибуты name – имя, reliability – надежность, cost – стоимость.

8.6.4 Формат описания эволюционного алгоритма

Используемый эволюционный алгоритм должен быть описан в файле формата XML, содержащем следующие теги:

<algorithm> – корневой элемент описания. Содержит теги population, stopcondition, selection, crossover, mutation, fuzzylogic, iga. Имеет атрибут type – тип ЭА. Поддерживаемые типы ЭА:

- cga – классический ЭА;
- hga – гибридный ЭА;
- icga – островной классический ЭА;
- ihga – островной гибридный ЭА.

<population> – популяция. Имеет атрибут size – количество решений в популяции.

<stopcondition> – условие останова. Имеет атрибут itermax – количество итераций алгоритма без улучшения целевой функции.

<selection> -- селекция. Имеет атрибут selpercent – процент лучших решений, отбираемых для скрещивания.

<crossover> -- скрещивание. Имеет атрибуты crosprob – вероятность скрещивания; crospercent – процент лучших решений из популяции, полученной после скрещивания, попадающих в следующую популяцию.

<mutation> -- мутация. Имеет атрибуты nomutpercent – процент лучших решений, которые не мутируют; mutprob – вероятность мутации.

<iga> – используется, если выбран один из островных ЭА. Имеет атрибуты iter – количество итераций между миграциями; algnum – количество алгоритмов, работающих параллельно; migrnum – количество мигрирующих особей.

<fuzzylogic> – используется, если выбран гибридный ЭА. Содержит теги <parameter>. Элементы <parameter> описывают параметры, значения которых меняются в ходе работы алгоритма. <parameter> имеет атрибуты name, min, norm, max, задающие имя регулируемого параметра и его минимальное, среднее и максимальное значения.

В данном разделе была описана методика использования средства решения задачи выбора МОО РВС РВ. Экспериментальное исследование данной задачи приведено в разделе 9.9.

9 Апробация интегрированной среды на стенде

В данном разделе описаны результаты экспериментального исследования разработанной среды моделирования. Для каждого из элементов среды, описанных в главе 4, проведены испытания. В разделе 9.1 описано функциональное тестирование эксперименты со средством трансляции UML в исполняемые модели совместимые со стандартом HLA. Раздел 9.2 содержит описание экспериментального исследования среды выполнения моделей CERTI. В разделе 9.3 описаны исследования, связанные со сравнением различных форматов трасс. Описание апробации средств трассировки и внесения неисправностей содержится в разделе 9.4. В разделах 9.5-9.7 рассмотрены эксперименты со средствами трансляции диаграмм состояний UML во временные автоматы UPPAAL, в частности, тестируется сам алгоритм трансляции, в том числе функция оценки наихудшего времени выполнения, проводится верификация свойств некоторых моделей, описанных в главе 6, и проводится тестирование функции восстановления параметров модели по контрпримеру UPPAAL. В разделе 9.8 описаны эксперименты со одним из средств синтеза архитектур, а именно со средством поиска решения задачи выбора оптимального набора механизмов отказоустойчивости (RAP). Наконец, в разделе 9.9 приведен пример использования интегрированной среды разработки для моделирования одной из систем, описанных в главе 6, заданной диаграммой состояний UML.

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

В данном разделе приведены функциональные тесты, показывающие корректную работу генератора исходного кода моделей, совместимых с HLA, на основе UML диаграмм состояний. Эксперимент будет описываться поэтапно в соответствии со всеми стадиями генерации исходного кода модели. Подробное описание исследуемого средства приведено в разделе 4.3.

9.1.1 XMI представление диаграммы состояний

После создания UML диаграммы состояний представляющую модель, необходимо сохранить данную модели в специализированом XML представлении - XMI. Данная функциональность является стандартной для использованного нами редактора UML. Пример XMI представления приведен на рисунке 153.

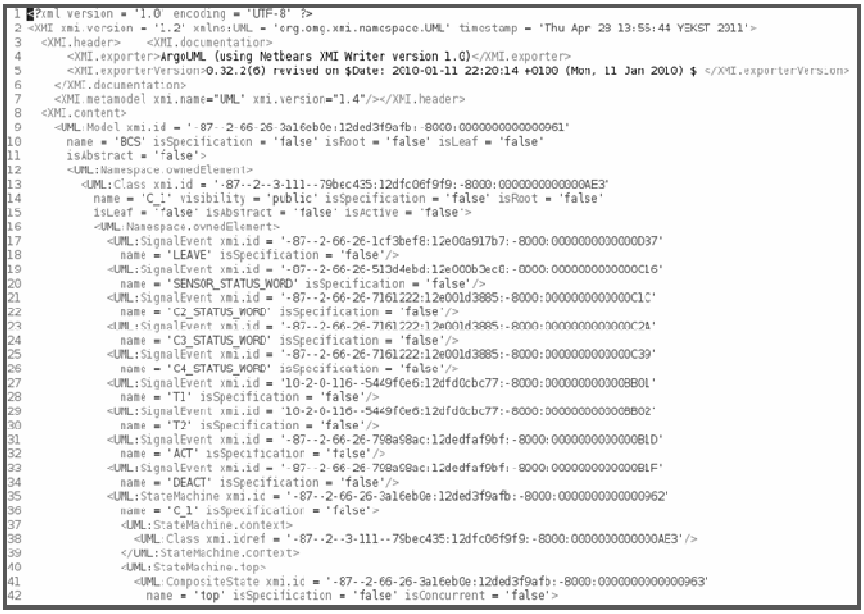

Рисунок 153. XMI представление модели Лавина.

Однако данный формат не является удобным для понимания и представления конечного автомата для описания внутренней логики. В связи с этим был выбран специализированный XML формат описания диаграмм состояний – SCXML. А также были разработаны и реализованы трансляторы их XMI в SCXML и наоборот.

9.1.2 SCXML представление диаграммы состояний

Как было сказано выше данный формат является наиболее удобным для описание диаграмм состояний и конечных автоматов, так как в своей структуре оперируется необходимыми параметрами и примитивами для описания данных структур. Пример SCXML представления экспериментальной модели представлен на рисунке 154.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- car_alarm.xml - state chart XML file -->
<scxml initialstate="Federation">
 <state id="FED_sender">
  <transition event="sender">
   <target next="receiver"/>
   <parametr name= "par1"></parametr>
  </transition>
  <parametr name= "par1" type="int" initval=".0"></parametr>
  <attribute name="regulating"></attribute>
 </state>
  <state id="FED_receiver">
  <attribute name="constrained"></attribute>
 </state>
</scxml>
```

Рисунок 154. SCXML представление модели Лавина.

9.1.3 Генерация исходного кода

На выходе генератор исходного кода по шаблонам, описанные в разделе 4.3, предоставляет исходные коды на языке C++ представленные в UML (на первой стадии генерации) распределенной модели. В данных исходниках полностью отображена функциональность внутренней логики (описанной при помощи конечного автомата), и прописаны интерфейсы для взаимодействия федератов внутри федерации через HLA RTI. Примеры исходных кодов представлен на рисунках 155-161.

```
#pragma once
#ifndef SENDER_□PP
#define SENDER_□PP

#include <base_federate_ambassador.hpp>
#include <base_federate.hpp>

#include "object_model/som.hpp"
#include "state.hpp"

class SenderAmbassador :
   public □aseFederateAmbassador<State,SenderInteractionAmbassador> {
public:
 static SenderAmbassador *self;
 static void pulse□andler(const Pulse &pulse, const rti1516::LogicalTime*, const char*);
 SenderAmbassador() {
   intamb.set_handler(pulse□andler);
   self = this;
 }
};
class Sender : public □aseFederate<SenderAmbassador> {
public:
 Sender( State *state ) : □aseFederate(state) {}
private:
 enum STATES {INIT,LOOP,FINIS□} cur;
 typedef STATES (Sender::*Func)();
 const static Func handlerTable[];

 STATES init();
 STATES loop();

 // Overloaded functions
 void configure();
 void execute() {
   cur = INIT;
   do cur = (this->*(handlerTable[cur])) ();
   while (cur != FINIS□);
 }
};
#endif // SENDER_□PP
```

Рисунок 155. Пример сгенерированного файла заголовка федерата

```
#include "sender.hpp"

#include <limits>

SenderAmbassador* SenderAmbassador::self;
void SenderAmbassador::pulse□andler(const Pulse &pulse, const LogicalTime* lt, const char*) {
  self->state->logicalTime = lt2d(*lt);
  self->state->pulse_received = true;
  *(self->state->pulse) = pulse;
}

const Sender::Func Sender::handlerTable[] = {&Sender::init,&Sender::loop};

void Sender::configure() {
  AvalancheSom::init_handles( *rtiamb );
  Pulse::subscribe( *rtiamb );
  state->pulse = new Pulse(*rtiamb);
  Knock::publish( *rtiamb );
  state->knock = new Knock(*rtiamb);
  state->knock->p_<KnockInteraction::Number>() = 0;
  state->knock->p_<KnockInteraction::Id>() = 0;
  state->lookahead = 0.5;
  constrain();
  regulate();
}

Sender::STATES Sender::init() {
  callNMR();
  if (!state->pulse_received) {
    std::cout << "Unexpected state in init" << std::endl;
    return FINIS□;
  }
  if (state->pulse_received) {
    state->pulse_received = false;
    std::cout << "Iter: " << state->pulse->p_<PulseInteraction::Iter>() << std::endl;
    if (state->pulse->p_<PulseInteraction::Iter>() == 0) {
      return FINIS□;
    }
  }
  return LOOP;
}

Sender::STATES Sender::loop() {
  int num = state->knock->p_<KnockInteraction::Number>();
  RTI1516fedTime theTime(state->logicalTime+1);
```

```
  state->knock->send(&theTime);
  state->knock->p_<KnockInteraction::Number>() = num + 1;
  callTAR(state->logicalTime+1);
  if (!state->pulse_received) {
    std::cout << "Unexpected state in loop" << std::endl;
    return FINIS□;
  }
  if (state->pulse_received) {
    state->pulse_received = false;
    std::cout << "Iter: " << state->pulse->p_<PulseInteraction::Iter>() << std::endl;
    if (state->pulse->p_<PulseInteraction::Iter>() == 0) {
      return FINIS□;
    }
  }
  return LOOP;
}

void run( State &state ) {
  Sender sender(&state);
  sender.run();
}
```

Рисунок 156. Пример сгенерированного cpp файла федерата

```
(FED
  (Federation □□VMTest)
  (FedVersion v1516)
    (federate "receiver" "Public")
  (federate "sender" "Public")
    (spaces)
    (objects)
    (interactions
    (class InteractionRoot  reliable timestamp
      (class RTIprivate  reliable timestamp)
      (class □LAinteractionRoot best_effort receive
        (class Pulse reliable receive)
        (class Knock reliable timestamp
          (parameter Id)
          (parameter Number)
        )    )   ) ))
```

Рисунок 157. Пример сгенерированного файла федерации.

```
#ifndef INTER_□LASS_TA□LE_□PP
#define INTER_□LASS_TA□LE_□PP
#include <protox/hla/i_class.hpp>
#include <protox/hla/keywords.hpp>
#include "parameter_table.hpp"
using namespace protox::hla;
//// Interaction □lass Table ////////////////////////////////////////////////
  struct inter_class_table : i_class< □LAInteractionRoot, none, child<
// +------------------------+
// | □lass                  |
// +------------------------+
  i_class< KnockInteraction, params<KnockInteraction::Id, KnockInteraction::Number> >,
// +------------------------+
  i_class< PulseInteraction, params<PulseInteraction::Iter> >
// +------------------------+
  > > {};
#endif // INTER_□LASS_TA□LE_□PP
```

Рисунок 158. Пример сгенерированной таблицы взаимодействий федерации.

```
#ifndef PARAMETER_TA□LE_□PP
#define PARAMETER_TA□LE_□PP
#include <protox/hla/param.hpp>
#include <protox/hla/keywords.hpp>
#include <protox/hla/name.hpp>
#include "simple_datatype_table.hpp"
using namespace protox::hla;
//     Parameter Table
//    +-----------------------------------------------------+-----------+------------------------+--------------------------+
//    | Interaction Name                                    | Parameter | Datatype               | String Name              |
//    +-----------------------------------------------------+-----------+------------------------+--------------------------+
 struct □LAInteractionRoot {□LA_NAME( "□LAInteractionRoot" )
     };//-----------------------------------------------------+-----------+------------------------+--------------------------+
 struct PulseInteraction   {□LA_NAME( "Pulse" )
                                    struct Iter     : param <IterType>       {□LA_NAME( "Iter" )};
     };//-----------------------------------------------------+-----------+------------------------+--------------------------+
 struct KnockInteraction   {□LA_NAME( "Knock" )
                                    struct Id       : param <IdType>         {□LA_NAME( "Id" )};
                                    struct Number   : param <NumberType>     {□LA_NAME( "Number" )};
     };//-----------------------------------------------------+-----------+------------------------+--------------------------+
#endif // PARAMETER_TA□LE_□PP
```

Рисунок 159. Пример сгенерированной таблицы параметров, которыми обмениваются федераты.

```
#ifndef SIMPLE_DATATYPE_TA□LE_□PP
#define SIMPLE_DATATYPE_TA□LE_□PP

#include <protox/dtl/simple.hpp>
#include <protox/hla_1516/basic_data_representation_table.hpp>

using namespace protox::dtl;
using namespace protox::hla_1516;

//  +--------------+------------------------+----------+
//  | Name         | Representation         | Units    |
//  +--------------+------------------------+----------+
struct IdType      : simple< □LAinteger32LE,   unitless > {PROTOX_SIMPLE( IdType )};
//  +--------------+------------------------+----------+
struct NumberType  : simple< □LAinteger32LE,   unitless > {PROTOX_SIMPLE( NumberType )};
//  +--------------+------------------------+----------+
struct IterType    : simple< □LAinteger64LE,   unitless > {PROTOX_SIMPLE( IterType )};
//  +--------------+------------------------+----------+

#endif // SIMPLE_DATATYPE_TA□LE_□PP
```

Рисунок 160. Пример сгенерированной таблицы типов параметров, которыми обмениваются федераты.

```
#ifndef SOM_□PP
#define SOM_□PP

#include <protox/hla/interaction_amb.hpp>
#include <protox/hla/i_class_type.hpp>
#include <protox/hla/som.hpp>

#include <boost/mpl/vector.hpp>

#include "inter_class_table.hpp"

typedef protox::hla::som< null_o_class, inter_class_table > AvalancheSom;

typedef i_class_type< AvalancheSom, q_name<PulseInteraction> >::type Pulse;
typedef i_class_type< AvalancheSom, q_name<KnockInteraction> >::type Knock;

typedef  protox::hla::interaction_amb<
boost::mpl::vector<Pulse>
>::type PulserInteractionAmbassador;

typedef protox::hla::interaction_amb<
boost::mpl::vector<Pulse>
>::type SenderInteractionAmbassador;

typedef protox::hla::interaction_amb<
boost::mpl::vector<Pulse,Knock>
>::type ReceiverInteractionAmbassador;

#endif // SOM_□PP
```

Рисунок 161. Пример сгенерированной SOM схемы федерации.

Ввиду громоздкости шаблонов для генерации различных элементов исходных кодов федерации, приведем пример только одного шаблона. На рисунке 162 приведен шаблон для генерации таблицы параметров, которыми обмениваются федераты.

```
#ifndef PARAMETER_TA□LE_□PP
#define PARAMETER_TA□LE_□PP

#include <protox/hla/param.hpp>
#include <protox/hla/keywords.hpp>
#include <protox/hla/name.hpp>

#include "simple_datatype_table.hpp"

using namespace protox::hla;
//     Parameter Table
//    +-----------------------------------------------------+-----------+------------------------+--------------------------+
//    | Interaction Name                                    | Parameter | Datatype               | String Name              |
//    +-----------------------------------------------------+-----------+------------------------+--------------------------+
  struct □LAInteractionRoot {□LA_NAME( "□LAInteractionRoot" )
      };//-----------------------------------------------------+-----------+------------------------+--------------------------+

#set $interactions=[]
#for $subState in $state.state
#if $subState.__dict__.has_key('transition'):
   #for $transition in $subState.transition:
      #if $transition.__dict__.has_key('event') :
         #if $transition.__dict__.has_key('parametr') and $interactions.count($transition.event)==0:
  struct ${transition.event}Interaction   {□LA_NAME( "${transition.event}" )
            #for $parametr in $transition.parametr :
${transition.event}Interaction::${parametr.name},
                                    struct Iter     : param <${parametr.name}Type>       {□LA_NAME(
"${parametr.name}" )};
            #end for
      };//-----------------------------------------------------+-----------+------------------------+--------------------------+
         $interactions.append($transition.event)
         #end if
      #end if
   #end for
#end if
#end for
#for $subState in $state.state
   #for $otherstate in $state□hartXML.state[0].parallel[0].state:
      #for $subState1 in $otherstate.state
         #if $subState1.__dict__.has_key('transition') :
```

```
            #for $transition in $subState1.transition :
              #if $transition.__dict__.has_key('target') :
                #if ($transition.target == $subState.id) :
                  #if $transition.__dict__.has_key('event') :
                    #if $transition.__dict__.has_key('parametr') and
$interactions.count($transition.event)==0:
 struct ${transition.event}Interaction   {□LA_NAME( "${transition.event}" )
                      #for $parametr in $transition.parametr :
                                struct Iter     : param <${parametr.name}Type>      {□LA_NAME(
"${parametr.name}" )};
                      #end for
    };//-----------------------------------------------------+-----------+------------------------+-------------------------+
                      $interactions.append($transition.event)
                    #end if
                  #end if
                #end if
              #end if
            #end for
          #end if
        #end for
      #end for
#end for
#endif // PARAMETER_TA□LE_□PP
```

Рисунок 162. Пример шаблона для генерации таблицы параметров, которыми обмениваются федераты.

9.1.4 Выводы

В ходе проведения были получены корректные исходные кода на языке C++ для работы в среде HLA RTI. Описание модели (создателем модели) в в виде диаграммы состояний UML существенно уменьшает времени разработку модели.

Подбор удобных форматов для представления данных на промежуточных этапах генерации кода делает удобным тестирование разработанного средства. А также способствует удобной ручной коррекции данных на различных этапах генерации исходного кода.

Наблюдаемые результаты соответствуют ожидаемым, следовательно, функциональное тестирование проведено успешно.

9.2 Экспериментальное исследование среды выполнения моделей

В данном разделе приводятся результаты исследования производительности построенной в ходе работы многопоточной среды выполнения Multi-Threaded CERTI (MT-

CERTI), а так же вспомогательной библиотеки-обвязки, призванной упростить разработку федератов, участвующих в моделировании. Полученные показатели сравниваются с аналогичными результатами, показанными оригинальной средой выполнения CERTI и федератами, написанными без использования вспомогательной обвязки вручную. Дополнительно раздел содержит аналогичные показатели специализированной системы полунатурного моделирования Стенд ПНМ, распределённая среда выполнения которой не реализует спецификации стандарта HLA [156].

9.2.1 Исследуемые системы

Подробное описание MT-CERTI, а так же библиотеки-обвязки, предоставляющей высокоуровневый интерфейс поверх инфраструктуры RTI можно найти в разделе 4.4 настоящего отчёта. В данной секции документа приводится лишь краткое напоминание их основных положений.

Многопоточная среда выполнения MT-CERTI является модификацией среды CERTI. Коренное отличие между этими двумя средами выполнения заключается в заложенной в их основу процессной архитектуре. Компонент LRC базовой версии CERTI состоит из двух полновесных процессов: (1) собственно процесса федерата, участника моделирования, и (2) процесса RTIA, предоставляющего набор сервисов стандарта HLA этому федерату. Взаимодействие между процессами осуществляется через механизм передачи сообщений, который требует дополнительной предобработки данных и, как следствие, отрицательно сказывается на общей производительности среды выполнения.

В отличие от своего прародителя, среда MT-CERTI реализует компонент LRC внутри единственного полновесного процесса: процессу RTIA базовой версии CERTI теперь сопоставляется дополнительный поток управления в контексте процесса федерата. Тем самым, взаимодействие составных частей компонента LRC переносится с уровня процессов на уровень потоков, который предоставляет более эффективные средства взаимодействия. Потоки выполняются в едином контексте и имеют общее адресное пространство. Поэтому для передачи данных достаточно предоставить лишь указатель на эти данные, не прибегая к их дополнительному преобразованию и копированию памяти.

Вспомогательная библиотека-обвязка предоставляет высокоуровневый интерфейс над сервисами и службами среды выполнения RTI, который позволяет значительно упростить процесс разработки новых федератов. Во-первых, обвязка активно использует библиотеку Proto-X, позволяющую оперировать с передаваемыми данными в терминах встроенной системы типов языка C++ и автоматически строить функции, необходимые для передачи этих данных через инфраструктуру RTI. Во-вторых, библиотека-обвязка позволяет быстро

описывать федерат в виде абстрактного автомата, состоящего из набора функций, описывающих логику его поведения внутри состояния, и системы переходов между этими функциями. Таким образом, разработанная библиотека-обвязка значительно упрощает процесс автоматической генерации исходного кода федератов.

В тоже время нельзя не заметить, что повышение уровня абстракции интерфейса между инфраструктурой RTI и подключёнными к ней федератами ведёт за собой падение их производительности. В ряде случаев ограниченная гибкость сгенерированных функций передачи сообщений и автоматного подхода к описанию логики федератов не позволяют создать столь же эффективный код, что и ручное программирование федератов поверх базового интерфейса RTI.

Таким образом, необходимо было разработать методику, позволяющую сравнивать между собой системы моделирования, использующие разные реализации инфраструктуру RTI совместно с дополнительной библиотекой-обвязкой и без неё наравне со средами выполнения не реализующими HLA вообще, и сравнить показатели производительности этих систем моделирования между собой. Описание методики, обладающей достаточной гибкостью для такой нетривиальной задачи, и полученные в результате оценки систем моделирования приводятся в последующих секциях данного раздела.

9.2.2 Методика исследования

Методика оценки сред выполнения в силу сложности этих систем должна учитывать множество разнородных факторов. В данном контексте под средой выполнения понимается вся программная прослойка, обеспечивающая взаимодействие между участниками эксперимента. В случае разработанных в рамках настоящей научно-исследовательской работы средств под такой прослойкой подразумевается совокупность инфраструктуры RTI и библиотеки-обвязки над её интерфейсом, непосредственно с которой и работают федераты. В настоящей работе предлагается методика, учитывающая следующие параметры:

1. Аспекты тестирования – набор характеристик среды выполнения, важных для её эффективного функционирования. Примером аспектов тестирования могут служить время отклика, ограничивающее минимальный размер допустимого директивного интервала и, как следствие, диапазон задач реального времени, которые система способна решать.
2. Тестовые сценарии – набор имитационных моделей, точечно нагружающих отдельные компоненты среды выполнения и позволяющие оценить выделенные показатели производительности сред выполнения. Например, набор моделей для

тестирования масштабируемости систем должен включать модели с разным количеством участников.

3. Тестовый стенд – включает в себя описание используемого во время прогона тестов программного и аппаратного окружения. Используемое окружение должно соответствовать тестовому сценарию и быть одинаковым для всех исследуемых сред выполнения.
4. Измеряемые параметры – множество показателей производительности системы, которые измеряются в процессе тестирования. Если аспекты тестирования – характеристики среды выполнения, как единой системы, то измеряемые параметры – показатели её отдельных составляющих. Примерами измеряемых параметров могут служить загрузка процессора, использование памяти на конкретной машине, интенсивность сетевой передачи данных.

9.2.3 Аспекты тестирования

Время отклика

В общем случае одной из основных характеристик среды выполнения является её среднее *время отклика* – размер временного интервала с момента передачи события среде выполнения и до завершения обработки этого события. В случае распределённой RTI, описанной стандартом HLA, временем отклика можно считать время, прошедшее с момента отправки сообщения федератом-издателем и до момента его доставки всем федератам-подписчикам. Таким образом, время отклика определяет скорость реакции системы моделирования на изменения в состоянии модели.

При выполнении задач моделирования реального времени, решение которых является одной из первостепенных целей настоящей работы, время отклика модели накладывает ограничения на минимальный размер директивных интервалов. Тем самым, время отклика существенно влияет на диапазон имитационных задач, которые могут быть решены с использованием заданной среды выполнения.

Пропускная способность

Другим важным параметром среды выполнения является её *пропускная способность*. Участники моделирования могут рассматривать среду выполнения как транспортную среду, обеспечивающую доставку передаваемых данных от отправителя к получателю. Действительно, распределённая инфраструктура RTI скрывает от федератов детали сетевого взаимодействия и представляет им интерфейс для передачи сообщений. Поэтому RTI, как и любую транспортную среду, можно охарактеризовать её пропускной способностью – объёмом данных, которые она способна передать в единицу времени.

Во время выполнения имитационной модели пропускная способность среды выполнения ограничивает интенсивность взаимодействия участников моделирования. Пусть несколько участников моделирования обмениваются данными, причём их обмены не зависят друг от друга, и поэтому не зависят от времени отклика модели. Если обмен данными будет достаточно интенсивным, то RTI с низкой пропускной способностью не будет успевать их обрабатывать и станет «узким местом» системы.

Нагрузка на логические компоненты

На логическом уровне любая распределённая среда выполнения состоит из набора локальных компонентов LRC и, возможно, центрального компонента CRC (подробнее об этом можно прочитать в разделе 4.4). На более низком уровне абстракции каждый из этих логических компонентов может быть образован несколькими процессами, пусть даже работающими на разных машинах. Например, компонент LRC системы CERTI состоит из процесса RTIA и библиотеки libRTI, выполняющейся в контексте процесса федерата.

Анализ производительности среды выполнения на уровне крупных логических блоков позволяет определить «узкие места» системы и понять, каким образом её можно оптимизировать на уровне макроэлементов. Например, при интенсивном обмене данными между участниками моделирования узким местом может стать компонент CRC, в случае CERTI образованный процессом RTIG. Поэтому целесообразным является исследование возможностей для снижения нагрузки на RTIG, например, построение среды выполнения с каскадной архитектурой.

Масштабируемость

Время отклика и пропускная способность среды выполнения могут значительно изменяться в зависимости от размера выполняемой имитационной модели. Чем больше имитационная модель, тем сложнее поддерживать её в согласованном состоянии, и тем менее производительной становится среда выполнения. Динамика подобных изменений называется *масштабируемостью* среды выполнения.

В случае распределённой среды RTI, описанной спецификациями стандарта HLA, вопрос масштабируемости можно рассматривать сразу с нескольких позиций:

1. Масштабируемость среды выполнения **на уровне федератов** предполагает исследование зависимости эффективности среды выполнения от количества участников моделирования и их характеристик. Например, нагрузка на RTI от участника-логгера, продвигающего своё логическое время крупными шагами, и получающего сообщения лишь определённого вида, будет гораздо меньше, чем от активно взаимодействующего федерата.

2. Масштабируемость **на уровне инструментальных машин** изучает вопрос производительности среды выполнения в зависимости от используемых хостов. Вместе с наращиванием мощности задействованного оборудования показатели производительности системы должны возрастать, однако рост распределённой системы приводит к соответствующему росту расходов на её синхронизацию и снижению эффективности использования ресурсов каждой отдельной машины. Принципиальным здесь является вопрос совместного размещения федератов: что выгоднее: использовать больше инструментальных машин или увеличивать количество участников моделирования на существующих машинах, какие виды взаимодействующих федератов выгодно размещать вместе, а какие раздельно.
3. Стандарт HLA предполагает возможность использования одной и той же среды выполнения для одновременного проведения сразу нескольких имитационных экспериментов. Масштабируемость среды выполнения **на уровне федераций** рассматривает целесообразность такого её использования. Важно исследовать вопрос взаимного влияния моделей друг на друга, определить динамику падения производительности системы при увеличении числа исполняемых моделей, найти максимальное количество моделей, одновременное выполнение которых может быть целесообразным.

9.2.4 Тестовые сценарии

Основными характеристиками производительности среды выполнения являются время отклика и пропускная способность модели. В самом деле, загрузка логических компонентов может быть определена одновременно с измерением времени отклика и пропускной способности, а масштабируемость изучает зависимость данных величин относительно размера имитационной модели и используемого аппаратного комплекса. Поэтому для настоящей работы достаточным было разработать только два типа моделей.

Имитационную модель можно рассматривать как одну распределённую программу, состоящую на логическом уровне из инфраструктуры RTI и множества подключённых к ней участников моделирования – федератов. Однако в рамках данного раздела нужно исследовать характеристики среды выполнения RTI, минимизировав их корреляции со свойствами используемых при этом множеством федератов. Поэтому разработанные модели были максимально упрощены.

Каждая из созданных моделей состоит из федератов двух типов – терминала и вычислителя. Каждый терминал передаёт заданное количество сообщений вычислителям. В модели «Лавина» вычислитель лишь регистрирует полученные сообщения. В модели «Пинг-

Понг» вычислитель дополнительно отвечает на каждое сообщение собственным сообщением с тем же телом, а терминал не передаёт новых сообщений, пока не получит уведомление от вычислителя. Подробнее об этих моделях можно прочитать в разделах 6.1. Несмотря на простоту описанных моделей, аналогичные им имитационные задачи часто используются в практике исследования систем [94]. Аналогичные модели входят, например, в пакет тестирования «RTINGv6-Benchmarks», использовавшимся разработчиками системы моделирования DMSO.

Для измерения пропускной способности среды выполнения в настоящей работе будет использоваться модель «Лавина». В рамках этого тестового сценария участники моделирования работают полностью асинхронно: терминал лишь генерирует поток данных, а вычислитель только отмечает доставленные ему данные. Поэтому скорость моделирования напрямую зависит от количества данных, которые система может передать в единицу времени – пропускной способности системы. Таким образом, значение пропускной способности может быть получено в виде отношения количества переданных данных ко времени выполнения модели.

Модель «Пинг-Понг» хорошо подходит для измерения времени отклика системы. Действительно, терминал не посылает новых сообщений, пока не получит уведомления о доставке предыдущего сообщения от вычислителя, а вычислитель не отправляет это уведомление, пока не получит сообщения от терминала. Однако время отклика среды выполнения в рамках распределённой системы моделирования определяется как размер временного интервала с момента передачи сообщения от участника-отправителя до момента его доставки участником-получателем.

Принимая время обработки поступившего сообщения вычислителем пренебрежимо малым, можно считать, что время между отправкой сообщения терминалом и получением ответа от вычислителя равно удвоенному времени отклика среды выполнения. Если время обработки уведомления от вычислителя также пренебрежимо мало, то время работы терминала с момента отправки им своего первого сообщения и до получения последнего уведомления от вычислителя равно произведению времени отклика среды выполнения на удвоенное количество отправленных терминалом сообщений.

В качестве передаваемого сообщения на данном этапе работы был использован единственный целочисленный параметр, значение которого устанавливалось до начала выполнения модели и уменьшалось на единицу при каждой передаче. Терминал и вычислитель завершали своё выполнение, когда передаваемый параметр достигал нулевого значения. Упаковка единственного параметра в необходимый формат требует минимальных затрат, а уменьшение значения на единицу позволяет легко контролировать корректность

передаваемых данных. В дальнейшем, однако, необходимо дополнительно исследовать зависимость показателей производительности системы моделирования от количества, структуры и размера передаваемых между участниками сообщений. Излишняя сложность формата передачи данных, предусмотренного стандартом HLA, может негативно сказываться на производительности системы моделирования.

9.2.5 Режимы работы

Система «CERTI» реализует спецификации стандарта моделирования HLA, и изначально разрабатывалась как распределённая система. При этом разработчики не уделяли особенного внимания случаю её нераспределённого использования, когда все участники моделирования запускались бы на единственной инструментальной машине. В результате, даже при работе на одной единственной машине CERTI обеспечивает взаимодействие участников моделирования с использованием тех же самые механизмов, что и при распределённом моделировании. В то же время при работе на одной машине взаимодействие и синхронизация федератов может быть организована гораздо более эффективно. Во-первых, здесь появляется возможность синхронизации с использованием общих часов. Во-вторых, процессы могут обмениваться данными, не привлекая сложные механизмы сетевой коммуникации.

Среда «Стенд ПНМ» способна работать в нескольких режимах: жёсткого РВ, мягкого РВ, вне РВ и режиме отладки. Режим отладки замедляет работу системы и не годится для тестирования производительности. В режиме «вне РВ» система «Стенд ПНМ» способна работать лишь на одной инструментальной машине. При этом для взаимодействия нескольких участников моделирования, работающих на одной машине, используются механизмы, гораздо более эффективные, чем средства сетевого взаимодействия. Таким образом, сравнение производительности системы «CERTI» и «Стенда ПНМ» в режиме «вне РВ» нецелесообразно.

В режимах жёсткого и мягкого РВ система «Стенд ПНМ» может использовать несколько инструментальных машин, однако при этом система работает с привязкой к реальному времени и производит синхронизацию логического времени моделей со временем астрономическим. На каждое действие участника эксперимента выделяется промежуток времени фиксированной длины. Если участник отрабатывает его за меньший срок, то выполнение этого участника искусственно задерживается.

Оригинальная версия системы «CERTI» предлагает лишь средства для выполнения модели без привязки ко времени AFAP (As-Fast-As-Possible), что недостаточно для решения задач моделирования в реальном времени, и, в частности, задачи полунатурного

моделирования. Данная проблема решается с помощью добавления служебного федерата пульсатора, который привязывает модельные события к астрономическому времени, задерживая при необходимости своё выполнение, и отправляет синхронизационные импульсы остальным участникам моделирования.

Дополнительные задержки, вносимые средами выполнения для синхронизации модельного и астрономического времени, вносят значительные искажения в реальные показатели их производительности. Поэтому для обеспечения максимально корректных результатов модули, обеспечивающие временные задержки, были отключены в большей части экспериментов. Для этого была собрана особая версия системы «Стенд ПНМ», а система «CERTI» запускалась без федерата-пульсатора или же этот федерат генерировал поток синхронизационных сообщений с фиксированной частотой, но проверка этих сообщений на соответствие логике участников моделирования была отключена. При этом первый вариант больше соответствует запуску систем моделирования в режиме AFAP, а второй – в режиме полунатурного моделирования.

9.2.6 Результаты исследования

На данном этапе работы было проведено сравнение времени выполнения моделей различными версиями системы «CERTI» и системой «Стенд ПНМ» с отключёнными временными задержками. Время работы каждого участника моделирования измерялось с момента начальной синхронизации модели и до завершения его выполнения. Итоговое время выполнения модели считалось как среднее арифметическое от полученных значений. Так как основной интерес для исследования на данном этапе представляют собой оригинальная система CERTI и её многопоточная версия MT-CERTI, то эти системы участвовали в большем количестве экспериментов.

Эксперимент 1: единственный сервер

Тестовый стенд первого эксперимента состоял из единственной инструментальной машины (Intel Xeon 2,4GHz, 10Gb RAM), на которой были запущены все компоненты системы моделирования. Результаты измерения времени выполнения моделей (Таблица 11) показывают, что прирост производительности MT-CERTI по сравнению с оригинальной версией CERTI достигает 30%.

С увеличением числа передаваемых сообщений время выполнения модели «Пинг-Понг» растёт линейно, в то время как модель «Лавина» демонстрирует экспоненциальный рост. Такое поведение связано с используемым CERTI распределённым механизмом временной синхронизации участников моделирования. В модели «Лавина» логическое время федерата-терминала не зависит от логического времени других федератов, поэтому он может

генерировать сообщения с произвольной скоростью. В данном случае скорость генерации превышает скорость обработки сообщений федератом-получателем. Поэтому инфраструктура RTI вынуждена буферизовать сообщения внутри себя. Вместе с ростом числа сообщений в буферах, скорость их обработки падает ещё больше, а размер буферов растёт ещё быстрее. Этот замкнутый круг приводит к экспоненциальному росту времени выполнения модели.

Описанная проблема может быть решена как на уровне системы моделирования, так и на уровне имитационной модели. С одной стороны можно ограничить размер внутреннего буфера сообщений RTI и останавливать федерат-отправитель, если его сообщения заполнили отведённую квоту. С другой стороны внутрь федерата-отправителя можно встроить дополнительные временные задержки, искусственно ограничивающие скорость генерации сообщений. Практическая эффективность предложенных способов, однако, существенно зависит от конкретной имитационной задачи, и подбор наилучшего размера внутреннего буфера сообщений инфраструктуры RTI или оптимального времени задержки между передачей сообщений требует дополнительных исследований.

Таблица 11 - Зависимость времени выполнения моделей от числа передаваемых сообщений при использовании единственной инструментальной машины, мс.

Число сообщений	**Лавина**		**Пинг-Понг**	
	CERTI	**MT-CERTI**	**CERTI**	**MT-CERTI**
10	3,8	2,4	6,8	4,6
100	35,4	19,4	73,1	45,6
1000	334	201,4	734,4	475,1
10000	3202,8	1888,9	7172,1	4728,6
100000	57335,9	50286,7	72134,8	49386,4

Эксперимент 2: кластер

Для проведения второго эксперимента использовался кластер из двух серверов (Intel Core2Duo 2,6GHz, 2Gb RAM), на каждом из которых выполнялся свой компонент имитационной модели. В случае систем «CERTI», на одной из них так же выполнялся и процесс RTIG. Результаты распределённого эксперимента (Таблица 12) показывают, что многопоточная MT-CERTI так же выигрывает до 30% у оригинальной версии системы, но в то же время проигрывает специализированной системе «Стенд ПНМ» примерно в 3 раза. Таким образом, система «Стенд ПНМ» способна выполнять значительно более сложные модели с меньшими директивными интервалами.

Таблица 12. Зависимость времени выполнения моделей от числа передаваемых сообщений при использовании комплекса из двух инструментальных машин, мс.

Число сообщений	Лавина			Пинг-Понг		
	CERTI	MT-CERTI	Стенд ПНМ	CERTI	MT-CERTI	Стенд ПНМ
10	4,1	2,8	1,6	10,2	6,3	2,3
100	38,1	26,1	7,6	94,4	65,2	22,8
1000	399,7	269	84,8	884,6	666,2	228
10000	6063	3015,2	1127,6	8770,7	6570,6	2280
100000	60601	30182,4	11722,1	87643,2	66524,8	22800

Эксперимент 3: Географическая удалённость

Аналогично предыдущему, данный эксперимент использовался комплекс из двух машин (AMD Opteron 2,4GHz, 12Gb RAM; Intel Core i7 1,6GHz, 4Gb RAM), но на этот раз они были соединены через сеть Интернет (среднее время ping-а 13,6 мс). Так как разница между скоростями работы CERTI и MT-CERTI невелика по сравнению со временем передачи сообщения через сеть, то в данном эксперименте принимала участие только система CERTI. Более того, отсутствие какие-либо механизмов контроля качества на пути передачи данных приводит к существенному дрожанию результатов. Это отражено в Таблице 13, содержащей минимальное, среднее и максимальное время выполнения моделей среди проведённых проб.

Сравнение приведённых результатов с экспериментом 2, в котором комплекс из двух машин был объединён локальной сетью, показывает, что динамика увеличения времени выполнения модели «Пинг-Понг» выше, чем аналогичный показатель модели «Лавина». Эта закономерность объясняется разным числом сетевых сообщений, которые передаются между компонентами моделей.

Таблица 13. Зависимость времени выполнения моделей системой CERTI от числа передаваемых сообщений при использовании двух удалённых друг от друга машин, мс.

Число сообщений	Лавина			Пинг-Понг		
	Минимум	Среднее	Максимум	Минимум	Среднее	Максимум
10	11,5	14,88889	18,5	146	161,6667	176
100	62,5	84,9	222	1328	1487,45	2395,5
1000	597	1752,15	5955,5	13443	14703,6	16833
10000	5951,5	9651,19	28786,5	137689,5	149603,2	165342

Эксперимент 4: одновременное выполнение

Стандарт моделирования HLA IEEE 1516 2000 предусматривает возможность одновременного проведения сразу нескольких экспериментов с использование одной и той же инфраструктуры RTI [96]. Поэтому в рамках данного эксперимента модели «Пинг-Понг» «Лавина» запускались как последовательно, так и параллельно на одной и той же инфраструктуре RTI. При этом использовался тот же аппаратный комплекс, что и во время описанного выше эксперимента 2.

Как видно из результатов тестирования (Таблица 14), падение скорости при одновременном запуске тестовых моделей значительно меньше времени их выполнения. Таким образом, одновременное проведение нескольких экспериментов с использованием единственного программно-аппаратного комплекса может быть целесообразным. Данный вопрос, однако, требует дальнейшего и более тщательного изучения.

Таблица 14. Зависимость времени выполнения моделей от числа передаваемых сообщений при их последовательном и параллельном запуске, мс.

Число сообщений	Лавина		Пинг-Понг	
	Последовательно	Параллельно	Последовательно	Параллельно
10	9	10,4	19,5	20,7
100	96,1	87,4	182,2	209,8
1000	889,3	1253,9	1794,3	2332,4

9.2.7 Выводы

Полученные результаты

Согласно результатам проведённого исследования, созданная на данном этапе работы многопоточная среда «MT-CERTI» обрабатывает события значительно быстрее, чем оригинальная версия этой системы: средний прирост скорости составляет более 30%. При этом различие в производительности двух систем заметно как при использовании единственной инструментальной машины, так и комплекса из нескольких машин, объединённых локальной сетью. В то же время многопоточная среда выполнения «MT-CERTI», так же как и оригинальная «CERTI» отстаёт от системы «Стенд ПНМ».

Совместное моделирование географически удалённых участников, объединённых сетью Интернет, приводит к существенному дрожанию результатов эксперимента, что не позволяет сравнивать между собой оригинальную и модифицированную версию CERTI.

Одну и ту же инфраструктуру RTI может быть целесообразно использовать для одновременного проведения сразу нескольких имитационных экспериментов.

Дальнейшая деятельность

Перспективной выглядит возможность измерения не только времени выполнения моделей, но и нагрузки, которую оказывает система моделирования на аппаратуру: расход оперативной памяти, затраченное процессорное время, нагрузка на сеть передачи данных. Некоторые из перечисленных характеристик не могут быть получены непосредственно из кода моделей, и потребуют модификации кода среды выполнения или использования внешних средств мониторинга программ.

В ходе проведённого исследования был выявлен экспоненциальный рост времени выполнения имитационной модели при перегрузке инфраструктуры RTI интенсивными потоками сообщений. Для борьбы с подобными перегрузками был предложено два ортогональных механизма, каждый из которых, однако, нуждается в дополнительном исследовании.

Проведённое исследование практически не задаётся вопросом масштабируемости среды выполнения. Оно лишь поверхностно рассматривает возможность использования одновременного проведения нескольких имитационных экспериментов с помощью одной единственной инфраструктуры RTI. Перспективным является более детальное изучение динамики изменения производительности среды выполнения в зависимости от количества участников моделирования и числа используемых инструментальных машин.

9.3 Экспериментальное исследование форматов трасс

В данном разделе приводится методика и результаты экспериментального исследования трасс в форматах TRC, OTF и OTFz для трассировки моделей РВС РВ.

9.3.1 Цель и задачи экспериментального исследования

В разделе 3.7 наиболее предпочтительными для хранения трасс РВС РВ являются форматы TRC, OTF и OTF с сжатием (OTFz).

Цель экспериментального исследования заключается в сравнении формата OTF с форматом TRC. Количественными параметрами для сравнения трасс в форматах OTF и TRC с одинаковой исходной информацией выбраны размер трассы и скорость обработки запросов к трассе.

Для исследования необходимо: подготовить исходные данные для трасс, разработать методику проведения экспериментов, произвести замеры размеров трасс и скорости обработки запросов, проанализировать результаты. На основе результатов исследования необходимо выбрать формат для трассировки моделей РВС РВ в рамках НИР.

9.3.2 Подготовка исходных данных

Для проведения экспериментального исследования форматов TRC, OTF и OTF с сжатием (OTFz) были выбраны 5 трасс различных размеров полученных в результате моделирования реальных РВС РВ морского назначения. Эти трассы обозначены следующим образом: Test_2010_06_01, Test_2010_10_28, Pohod88, Kingstown, VeryBigTrace. Изначально трассы даны в формате TRC.

В таблице 15 представлены общие параметры трасс: количество компонентов в моделируемой РВС РВ, время моделирования, количество состояний и событий, произошедших в системе. В таблице 16 представлены размеры файлов трасс в формате TRC и общий размер трасс.

Таблица 15. Общие параметры трасс в формате TRC.

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

Таблица 16. Размеры трасс в формате TRC.

	Test_2010_06_01	Test_2010_10_28	Pohod88	Kingstown	VeryBigTrace
stand.sta	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)
stand.trc	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)
stand.trcext	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)
stand.dbg	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)
Общий размер трассы	1,88 Мб	10,47 Мб	14,05 Мб	218,55 Мб	1,37 Гб

В таблицах 17 и 18 приведено распределение информации в процентах и байтах между файлами основной трассы, внешней трассы и трассы состояний в формате TRC.

Таблица 17. Распределение информации между файлами в трассах формата TRC.

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

Таблица 18]. Распределение информации между файлами в трассах формата TRC.

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

Из таблиц 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 19 приведено распределение событий в рассматриваемых трассах по типам. На рисунке 163 распределение событий по типам приведено в процентах.

Таблица19. Статистика событий в трассах TRC

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

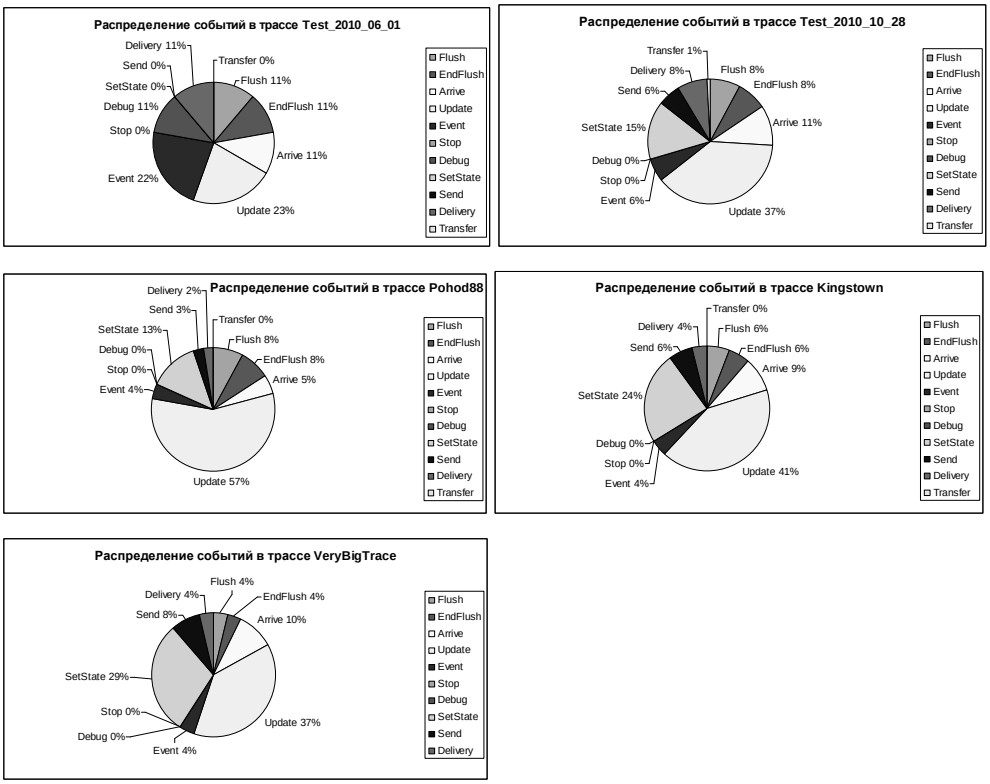

Рисунок 163. Распределение событий в трассах по типам (в процентах)

Приведённые диаграммы показывают, что большинство событий в трассе – это события типа Update обновления параметров (от 23% до 57%). На втором месте - события

типа SetState установления состояния (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 более половины происходящих в РВС РВ событий относится к изменению значений параметров внутри моделей компонентов и изменению режимов их функционирования.

9.3.3 Методика исследования форматов трасс

Сравнение трасс в форматах OTF и TRC по двум параметрам: размеры трасс и скорость последовательного чтения трассы при поиске некоторого события в ней.

Для сравнения трасс по размеру необходимо:

- Исходные трассы в формате TRC сконвертировать в трассу в формате OTF и OTFz с помощью средств trc2txt и txt2otf, разработанных на втором этапе НИР, использую промежуточное представление трассы в текстовом формате txt.
- Провести сравнение размеров трасс в форматах TRC, TXT, OTF и OTFz. Общий размер трассы определяется как сумма всех файлов, составляющих трассу.

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.

Для поддержки таких запросов к трассе обычно 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.

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.

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.

Для сравнения трасс по скорости последовательного чтения необходимо:

- Для каждой трассы с помощью модуля ResearchModul определить 4 события, расположенные на отметках 25%, 50%, 75%, 100% от размера трассы по количеству событий.
- Произвести замер времени линейного поиска этих событий по 100 итерациям в автоматическом режиме с помощью модуля TraceAnalyzerTRC и TraceAnalyzerOTF.

9.3.4 Результаты сравнения размеров трасс

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 20 и 21.

Таблица20. Сравнение размеров трасс в форматах TRC, TXT, OTF и OTFz.

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

Таблица 21. Сравнение размеров трасс в форматах TRC, TXT, OTF и OTFz.

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

Соотношение размеров трасс в форматах TRC, TXT, OTF и OTFz приведено на рисунках 164 и 165.

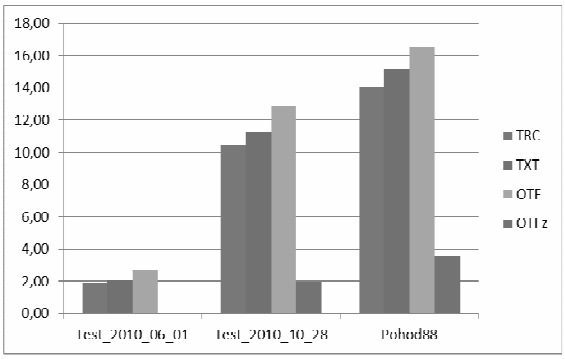

Рисунок 164. Соотношение экспериментальных трасс в разных форматах (в Мб.)

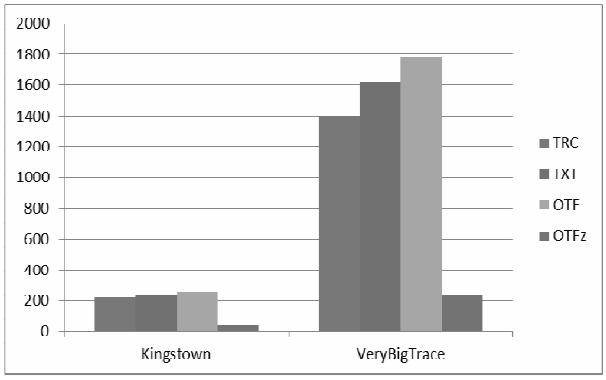

Рисунок 165. Соотношение экспериментальных трасс в разных форматах (в Мб.)

Из рисунков видно, что трасса в формате OTF практически всегда превосходит по размеру трассы в форматах TRC и TXT, однако в трассы формате OTFz с сжатием имеют наименьший размер и также могут читаться визуализаторами (Vite, Vampir) без дополнительных действий по их распаковке. В таблицах 22 и 23 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 22. 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**
TRC	1,88	1,00	10,47	1,00	14,05	1,00
TXT	2,09	0,90	11,25	0,93	15,2	0,92
OTF	2,70	0,70	12,9	0,81	16,5	0,85
OTFz	0,68	2,76	2	5,24	3,58	3,92

Таблица 23. Коэффициент сжатия трасс в форматах TRC, TXT, OTF и OTFz.

	Kingstown		**VeryBigTrace**	
	Мб	**Коэффициент сжатия**	**Мб**	**Коэффициент сжатия**
TRC	218,55	1,00	1402,88	1,00
TXT	231,2	0,95	1617,92	0,87
OTF	258,6	0,85	1781,76	0,79
OTFz	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.

9.3.5 Результаты последовательного поиска событий в трассах

В таблицах 24, 27, 30, 33, 36 приведена информация о событиях, поиск которых осуществляется в трассах Test_2010_06_01, Test_2010_10_28, Pohod88, Kingstown, VeryBigTrace соответственно. В таблицах 25, 28, 31, 34 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 26, 29, 32, 35, 38.

Последовательный поиск в трассе Test_2010_06_01

Таблица24. События для поиска в трассе Test_2010_06_01.

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

Таблица25. Время прохождения фрагментов трассы.

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

Таблица 26. 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 27. События для поиска в трассе Test_2010_10_28.

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

Таблица28. Время прохождения фрагментов трассы.

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

Таблица29. 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 30. События для поиска в трассе Pohod88.

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

Таблица 31. Время прохождения фрагментов трассы.

Pohod88 (100 итераций)					
Формат	**Показатель**	**1**	**2**	**3**	**4**
TRC	*min time*	0,017489	0,034666	0,051655	0,069239
	av time	**0,019749**	**0,039437**	**0,059419**	**0,077069**
	max time	0,025542	0,047428	0,135977	0,095499

OTF	*min time*	0,015059	0,032968	0,048951	0,066074
	av time	**0,017438**	**0,034902**	**0,055497**	**0,070133**
	max time	0,023918	0,042304	0,066322	0,083297
OTFz	*min time*	0,015293	0,031111	0,048881	0,061113
	av time	**0,017280**	**0,035967**	**0,054012**	**0,069054**
	max time	0,019786	0,040318	0,061098	0,079067

Таблица 32. 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 33. События для поиска в трассе Kingstown.

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

Таблица 34. Время прохождения фрагментов трассы.

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

Таблица 35. 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 36. События для поиска в трассе VeryBigTrace.

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

Таблица 37. Время прохождения фрагментов трассы.

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

Таблица 38. 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	91,2	87,5	90,8	88,3
OTFz	92,1	90,7	91,2	89,5

Из таблиц 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.

9.3.6 Выводы и анализ результатов

Таким образом, в результате экспериментального исследования:

Были подготовлены исходные данные, разработана методика исследования и определены количественные параметры для сравнения трасс, проведено экспериментальное исследование и анализ полученных результатов.

Были получены следующие результаты сравнение трасс в форматах TRC, OTF и OTFz:

- 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 ACSII составляет от 0,70 до 0,85.
- Формат OTFz позволяет сжимать трассу в формате TRC примерно в 3-6 раз.

На основании экспериментального исследования для использования в рамках разработанной системы моделирования выбран формат OTF с сжатием, то есть OTFz.

9.4 Экспериментальное исследование средств трассировки моделей и внесения неисправностей

В разделе 4.5 было рассмотрено средство внесения неисправностей, а в разделе 4.6 были рассмотрены возможные схемы трассировки имитационных экспериментов в разрабатываемой среде моделирования РВС РВ. Эти средства объединяет то, что в среду выполнения моделей добавляются новые федераты, решающие специализированные задачи. Для реализации схемы трассировки моделей были выбраны централизованная схема «федерат-сборщик» и распределенная схема на основе использования RTI-интерфейса для сбора трассы в каждом федерате. На основе экспериментального исследования форматов трасс (раздел 9.4) трассы записывают в формате OTFz. В распределенной схеме требуется дальнейшее объединение трасс, которое осуществляется с помощью средства otfmerge, входящего в состав библиотек otflib. Также было разработано средство внесения неисправностей и выбрана схема с одним федератом-перехватчиком сообщений, который позволяет перехватывать сообщения и портить значения переменных.

9.4.1 Цель и задачи исследования

Целью экспериментального исследования является оценка влияния процесса трассировки и внесения неисправностей на процесс моделирования. В качестве критерия сравнения выбрано время моделирования в зависимости от количества событий (переданных сообщений), то есть, на сколько увеличивается время при включении опции трассировки событий или внесения неисправностей.

Для проведения исследования необходимо подготовить модели для трассировки, разработать методику проведения экспериментов, провести эксперименты и анализ результатов.

9.4.2 Методика проведения экспериментов

Для проведения экспериментов использовались модели «Лавина» и «Пинг-Понг», описанные в разделе 6.1. Для каждой из них была добавлена опция выбора схемы трассировки и опции для автоматизации проведения экспериментов (с указанием количества запусков эксперимента и записью результатов экспериментов в виде таблицы в файл с расширением csv).

Для каждой модели эксперименты проводятся следующим образом:

1. Запускается модель без трассировки с 10, 100, 1000, 10000 сообщениями (по 100 запусков). Фиксируется среднее время выполнения модели.
2. Аналогично запускается модель со схемой трассировки «сборщик-федерат».
3. Запускается модель с распределенной схемой трассировки.

Нужно отметить, что при определении времени выполнения модели с последней схемой трассировки не учитывается время на сбор полной трассы.

Эксперименты проводились на единственной инструментальной машине, на которой были запущены все компоненты распределённой системы моделирования. Машина имела следующие характеристики: процессор Core i5 560M 2660 МГц, 4 Гб DDR3 оперативной памяти, SSD жесткий диск, 128 Гб, операционная система Ubuntu 10.10.

Аналогичные эксперименты для сравнения производительности среды выполнения без перехватчика и среды выполнения с перехватчиком были выполнены для средства внесения неисправностей. Для этого была взята тестовая модель «Лавина». Для этого модель в обоих случаях запускали с параметром N = 500, 1000, 1500, …, 5000, где N – число сообщений, передаваемых отправителем. Для справедливости эксперимента запуск с каждым параметром проводился несколько раз и полученные значения усреднялись. Запуски производились на компьютере Core 2 Duo 2660 МГц, 4 Гб DDR2 оперативной памяти, жесткий диск, 320 Гб, операционная система Ubuntu 12.

9.4.3 Результаты экспериментального исследования

Результаты экспериментального исследования выполнения моделей «Лавина» и «Пинг-Понг» с различными схемами трассировки приведены в таблицах таблицах 39, 40.

Таблица 39. Среднее время выполнения модели Лавина с различными схемами трассировки, мкс

Количество событий	Время выполнения модели Лавина, мкс.		
	без трассировки	с федератом-сборщиком	с трассировкой каждого федерата
10	5	6	5

100	45	48	47
1000	508	581	548
10000	4703	5192	5023

Таблица 40. Среднее время выполнения модели Пинг-Понг с различными схемами трассировки, мкс

Количество событий	Время выполнения модели Пинг-Понг, мкс.		
	без трассировки	с федератом-сборщиком	с трассировкой каждого федерата
10	8	9	8
100	84	95	87
1000	887	1010	962
10000	8706	9934	9385

Результаты экспериментального исследования выполнения модели «Лавина» с федератом-перехватчиком и без него приведены в таблице 41.

Таблица 41. Среднее время выполнения модели Лавина без федерата-перехватчика и с федератом-перехватчиком, мкс

Количество событий	Время выполнения модели Лавина, мкс.	
	без федерата-перехватчика	с федератом-перехватчиком
500	239	405
1000	421	825
1500	654	1214
2000	854	1639
2500	1074	2085
3000	1327	2479
3500	1473	2846
4000	1692	3321
4500	1971	3867
5000	2198	4458

9.4.4 Выводы

По результатам проведенных экспериментов со схемами трассировки можно сделать следующие выводы:

- Использование централизованной схемы с федератом-сборщиком увеличивает время выполнения модели от 7 до 15%.
- Использование распределенной схемы трассировки на основе использования RTI-интерфейса увеличивает время выполнения модели от 3 до 9 %.

Таким образом, результаты проведенных экспериментов показали, что рассматриваемая распределенная схема трассировки оказывает наименьшее влияние на процесс моделирования. Однако, можно сделать вывод и о том, что требуется проведение дополнительного экспериментального исследования:

- на большем количестве моделей;
- для трассировки разнообразных типов событий;
- для трассировки моделей, размещенных на различных машинах, взаимодействующих по сети.

Также есть предположение, что в зависимости от количества и расположения моделей на хостах напрямую зависит скорость трассировки и эффективность той или иной схемы.

Как видно из результатов экспериментов со средством внесения неисправностей, при увеличении числа сообщений, время выполнения модели с перехватчиком значительно больше времени выполнения модели без перехватчика. В модели с перехватчиком каждое сообщение от отправителя сначала идет перехватчику, а затем отправителю, то есть фактически число сообщений, проходящих через RTIG, удваивается. Как было сказано в [2], CERTI имеет централизованную архитектуру, поэтому процесс RTIG является узким местом системы, что объясняет полученные результаты.

9.5 Экспериментальное исследование средств трансляции UML во временные автоматы

В данном разделе приводится экспериментальное исследование средств трансляции диаграмм состояний UML во временные автоматы UPPAAL, описанного в разделе 4.8.

9.5.1 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 166) присутствуют базовые элементы диаграмм состояния UML: простые и композитные состояния (s1, s2, s3), переходы между ними. В

автомате UPPAAL, соответственно, должно получиться одно начально состояние и три состояния, соответствующих трем состояниям диаграммы UML (см. рисунки 167, 168).

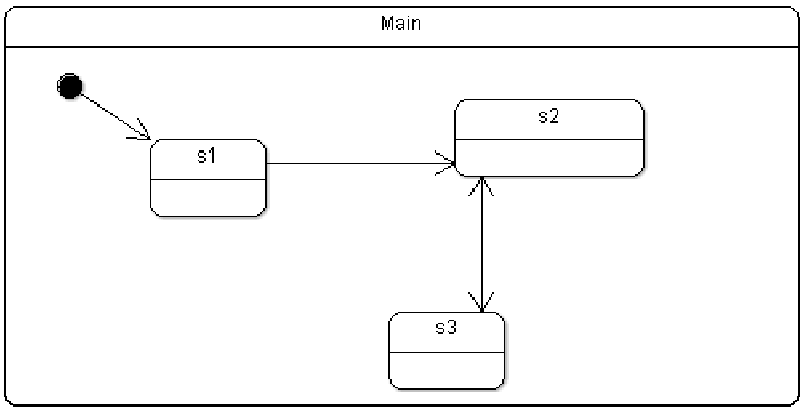

Рисунок 166. Первый тест

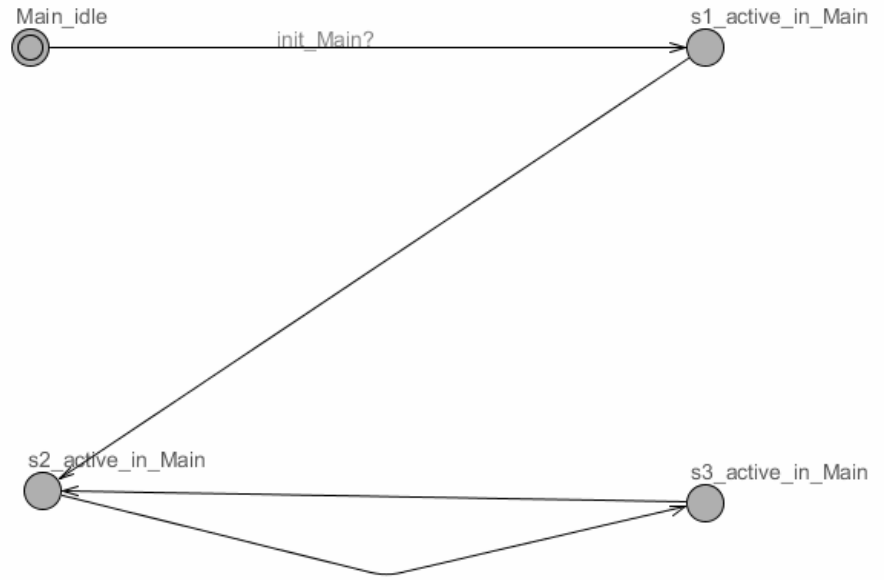

Рисунок 167. Ожидаемая диаграмма UPPAAL для первого теста

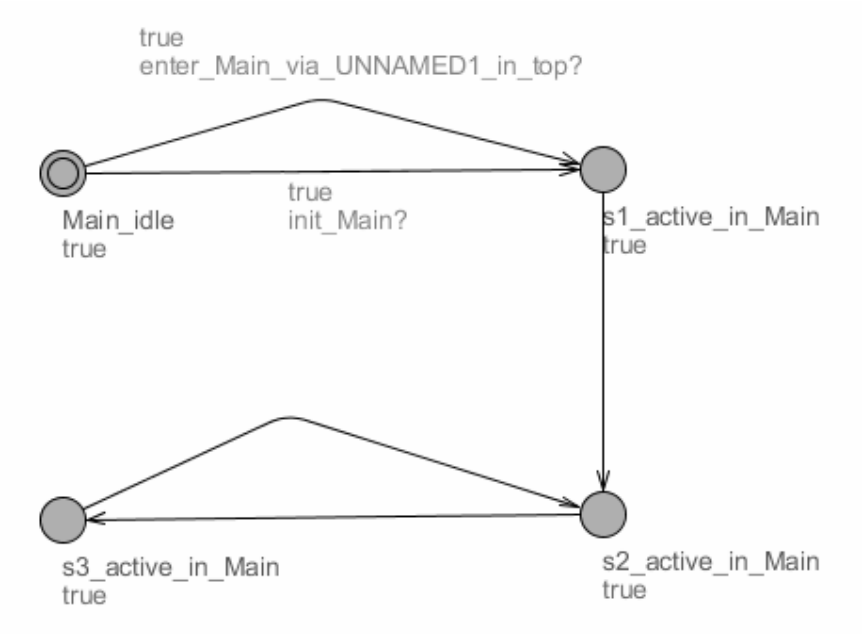

Рисунок 168. Полученная диаграмма UPPAAL для первого теста

Во втором тесте (см. рисунок 169) присутствует композитное состояние типа And. Для каждого параллельного региона должен создаться автомат UPPAAL с соответствующими простыми состояниями, а также один автомат UPPAAL для инициализации (см. рисунки 170, 171).

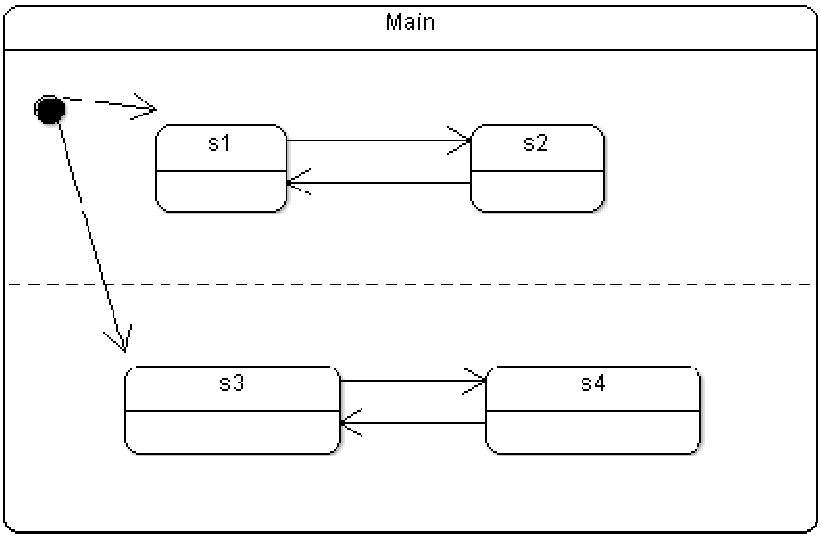

Рисунок 169. Второй тест

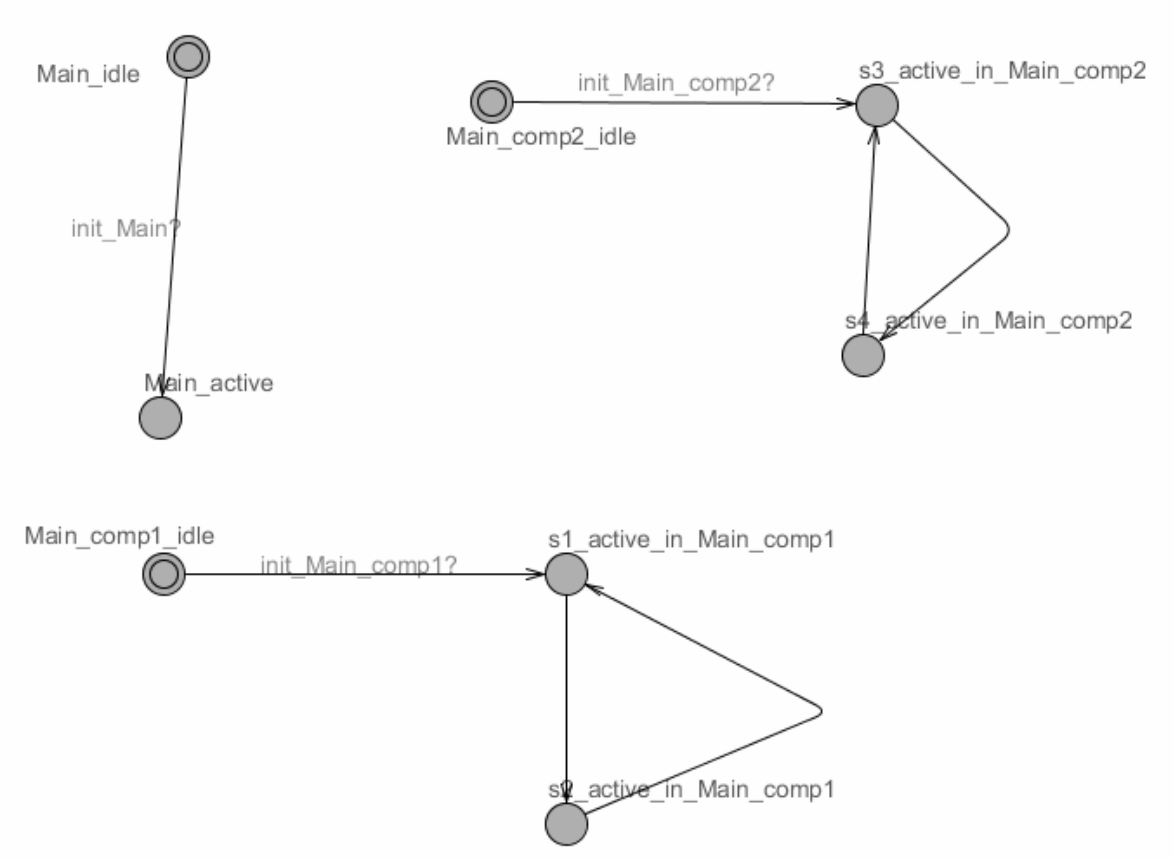

Рисунок 170. Ожидаемая диаграмма UPPAAL для второго теста.

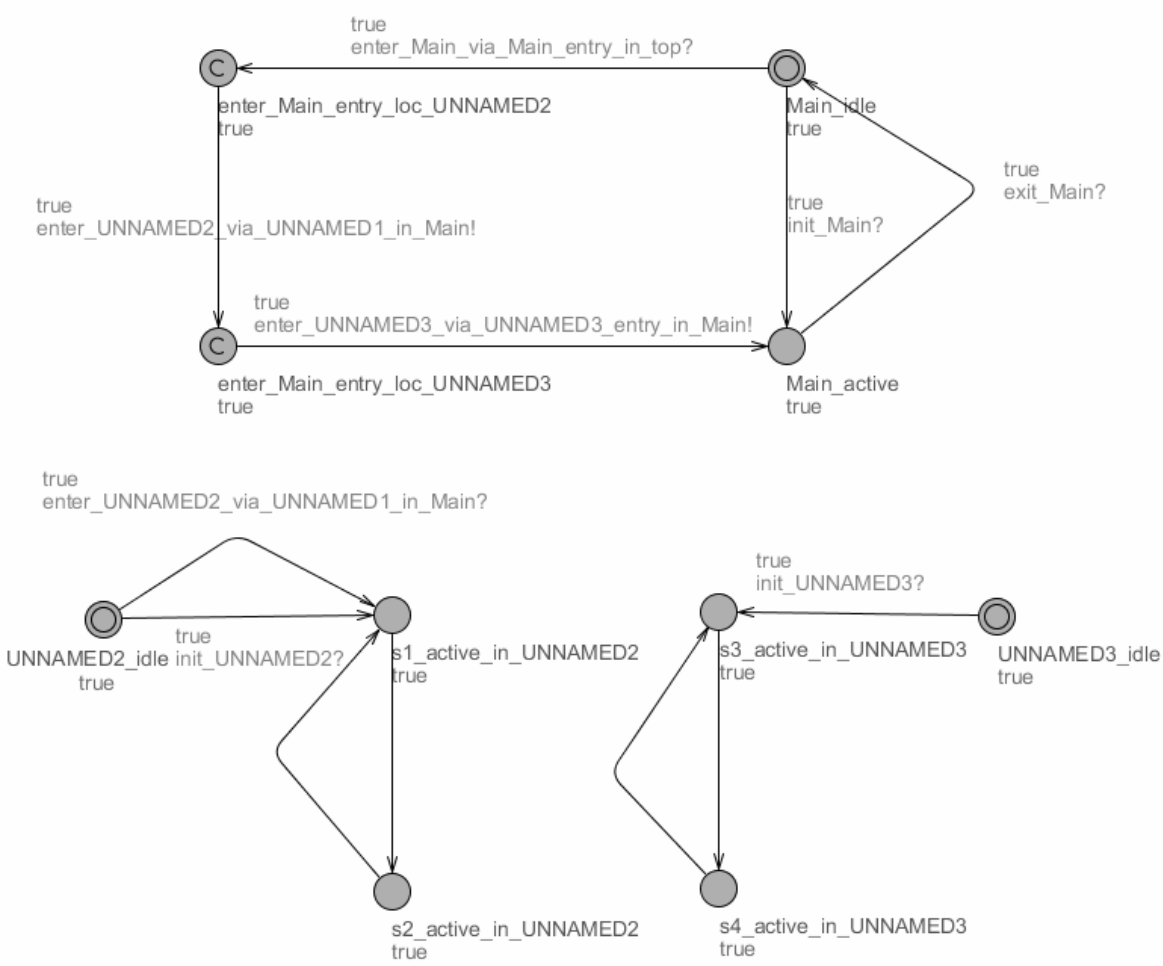

Рисунок 171. Полученная диаграмма UPPAAL для второго теста

В третьем тесте (см. рисунок 172) присутствует нетривиальное множество выходных деревьев (функция make_tree_set в алгоритме из раздела 5.1). В данной диаграмме присутствует выход из состояния типа XOR, вложенного в состояние типа AND. Соответственно, в диаграмме UPPAAL должны появиться четыре последовательности состояний, соответствующих разным порядкам выхода из вложенных состояний (см. рисунки 173,174).

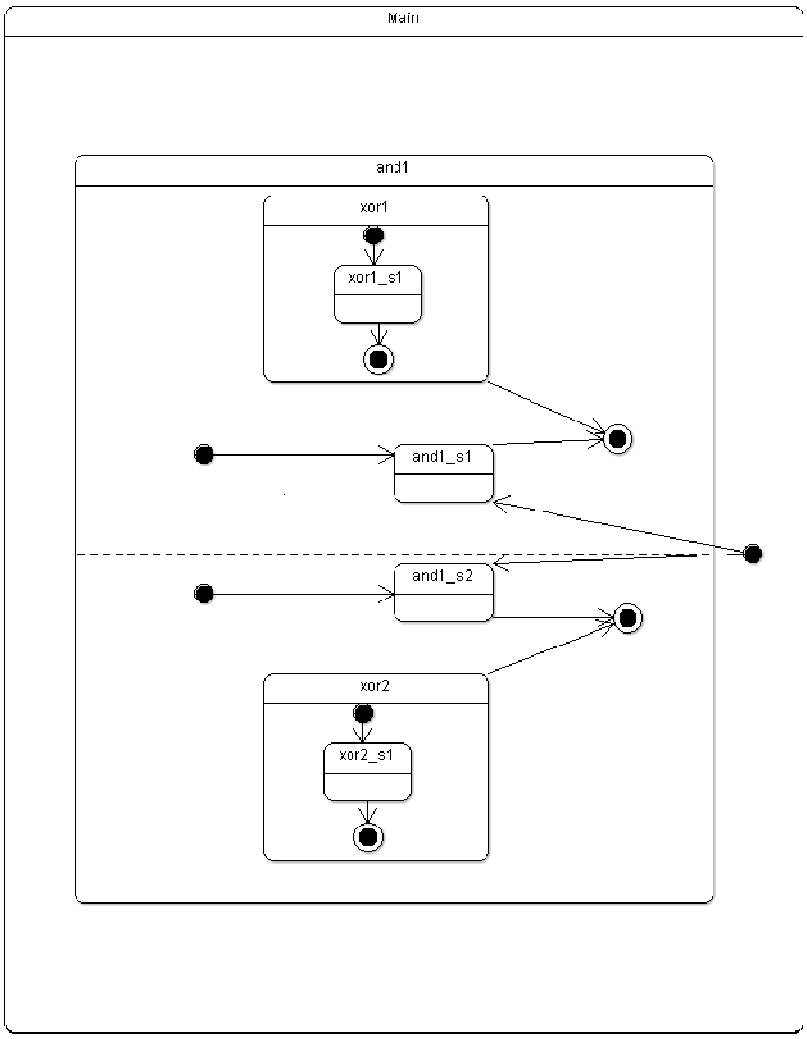

Рисунок 172. Третий тест

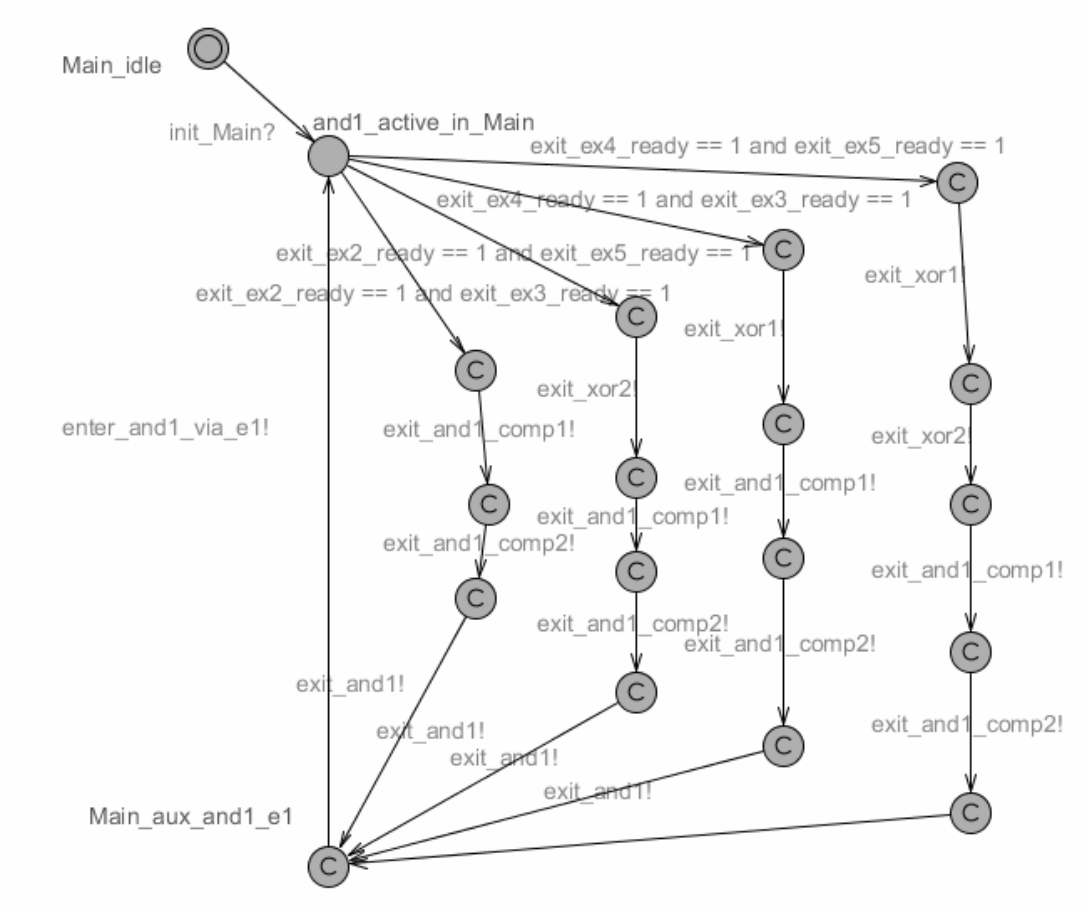

Рисунок 173. Ожидаемая диаграмма UPPAAL для третьего теста (фрагмент с деревом выходов)

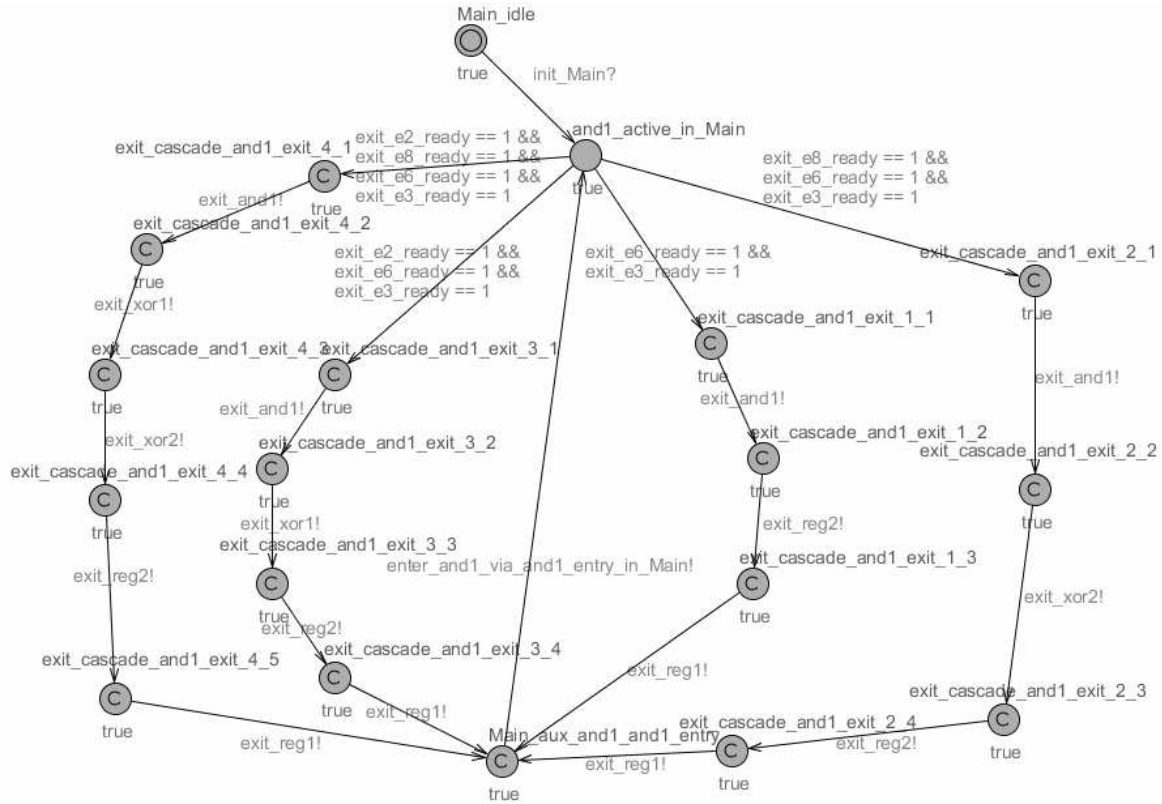

Рисунок 174. Полученная диаграмма UPPAAL для третьего теста (фрагмент с деревом выходов)

В четвертом тесте (см. рисунок 175) требуется нетривиальное назначение предусловий (функция add_exit_guards в алгоритме из раздела 5.1), так как выход в состоянии xor1 возможен сразу из двух состояний: s1 и s2. В UPPAAL должны в правильных местах появиться присваивания значений переменной exit_ex1_ready (см. рисунки 175, 176).

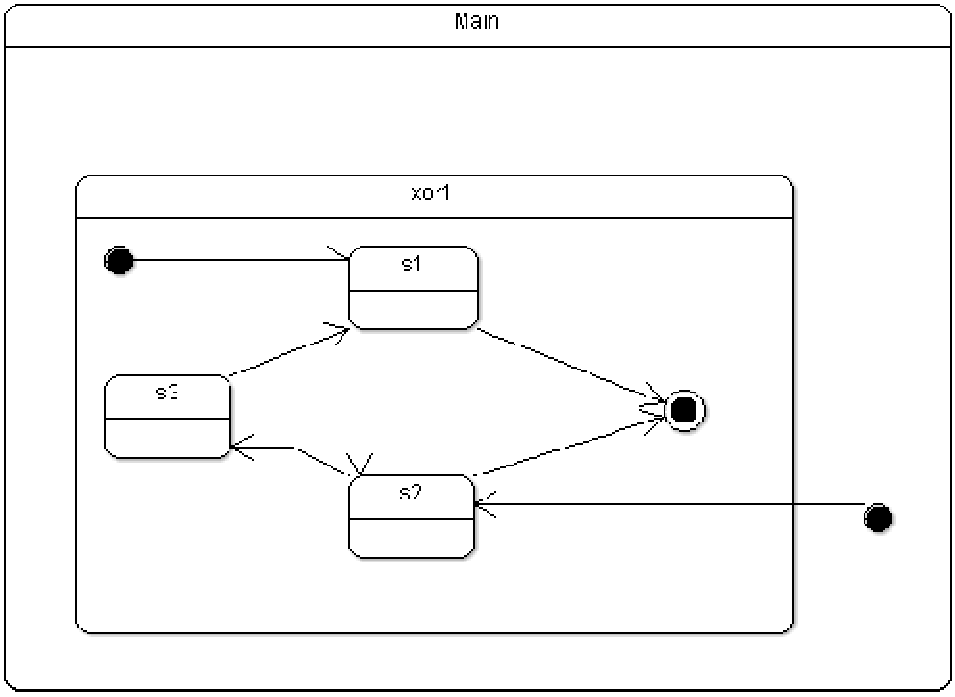

Рисунок 175. Четвертый тест

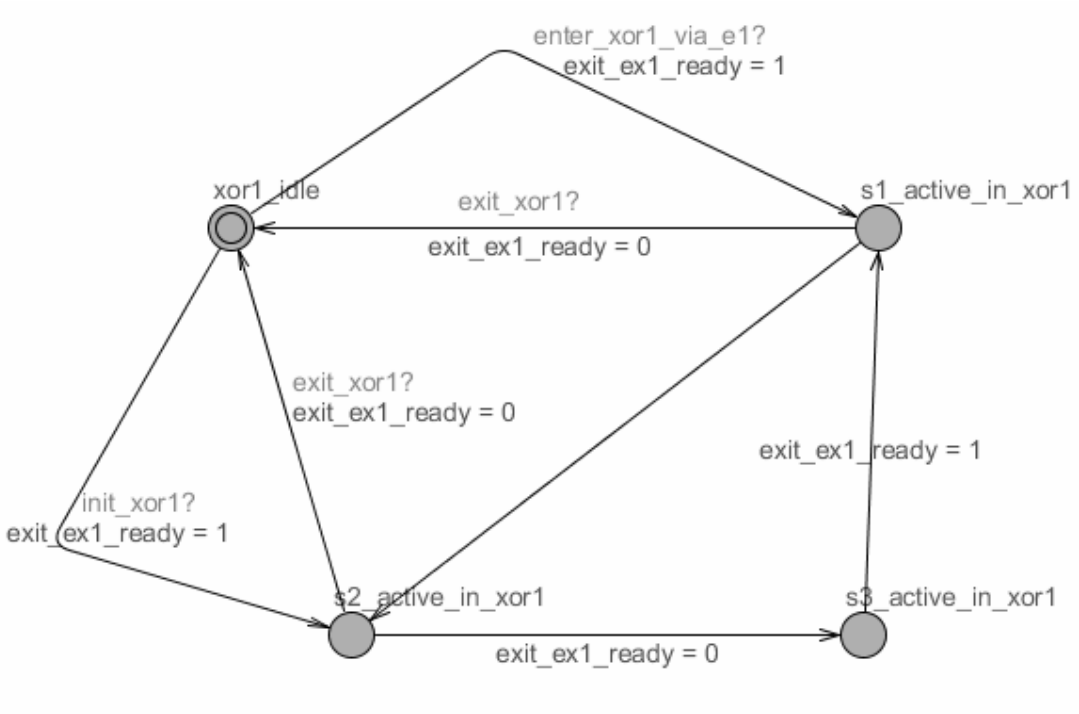

Рисунок 176. Ожидаемая диаграмма UPPAAL для четвертого теста.

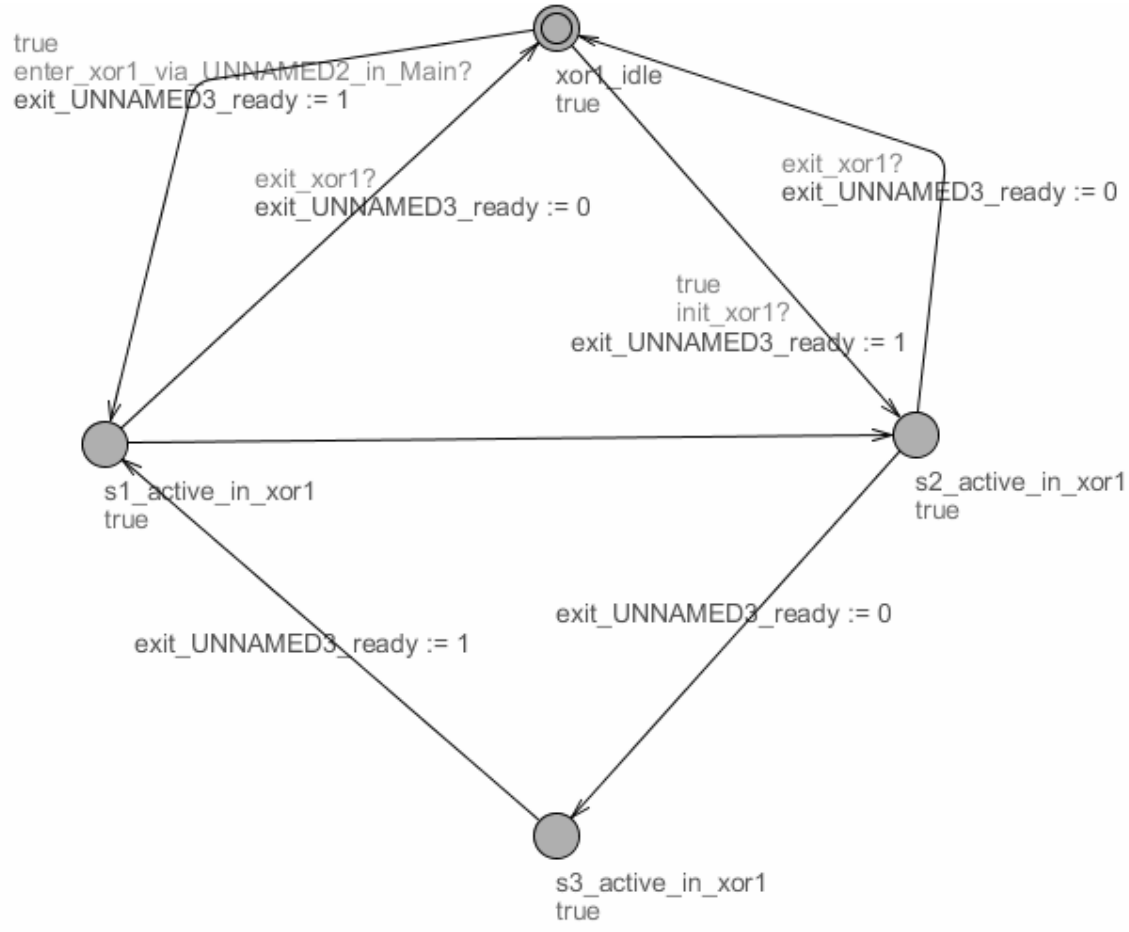

Рисунок 177. Полученная диаграмма UPPAAL для четвертого теста

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 4.8.

9.5.2 Исследование на модели «лавина»

После трансляции модели «лавина», описанной в разделе 6.1, в UPPAAL получилась система из 5 автоматов: инициализирующий, главный и три автомата, соответствующие трем параллельным регионам модели. На рисунках 178 – 182 приведены автоматы.

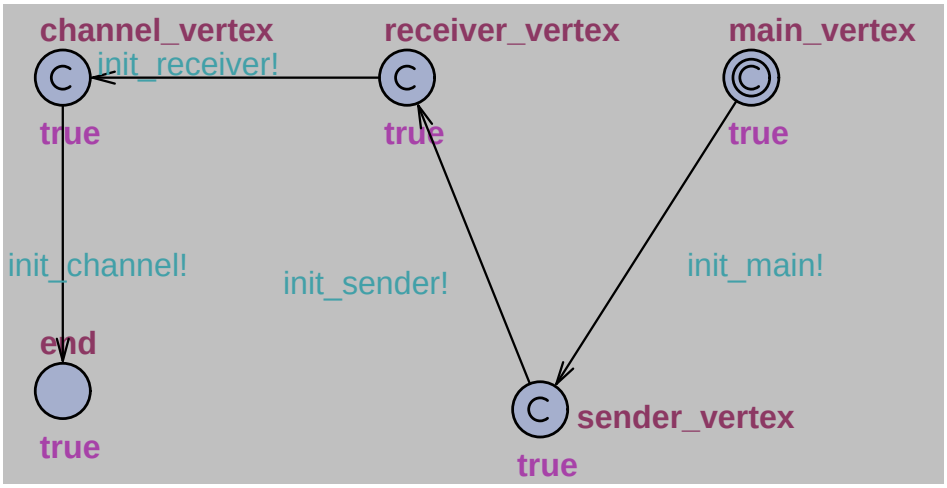

Рисунок 178. Автомат Global kickoff

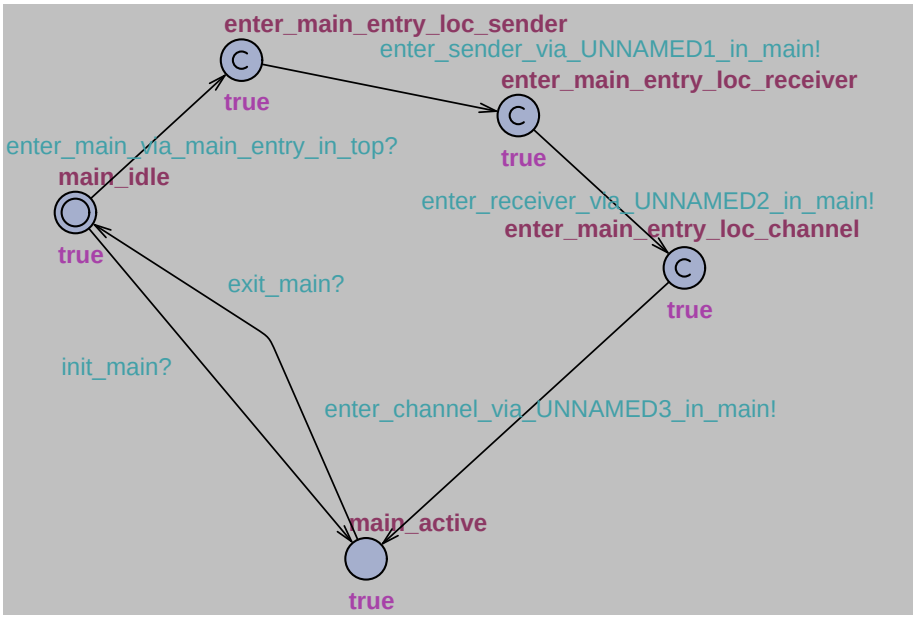

Рисунок 179. Автомат Main

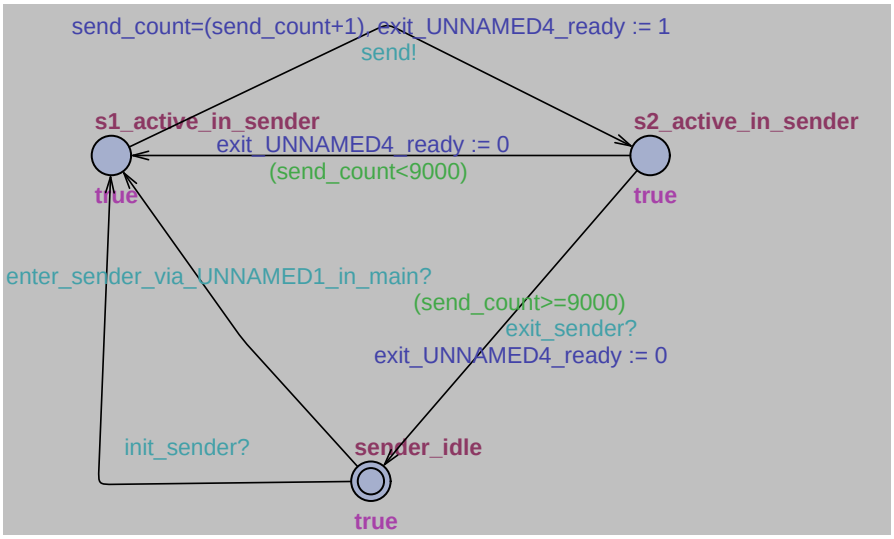

Рисунок180. Автомат sender

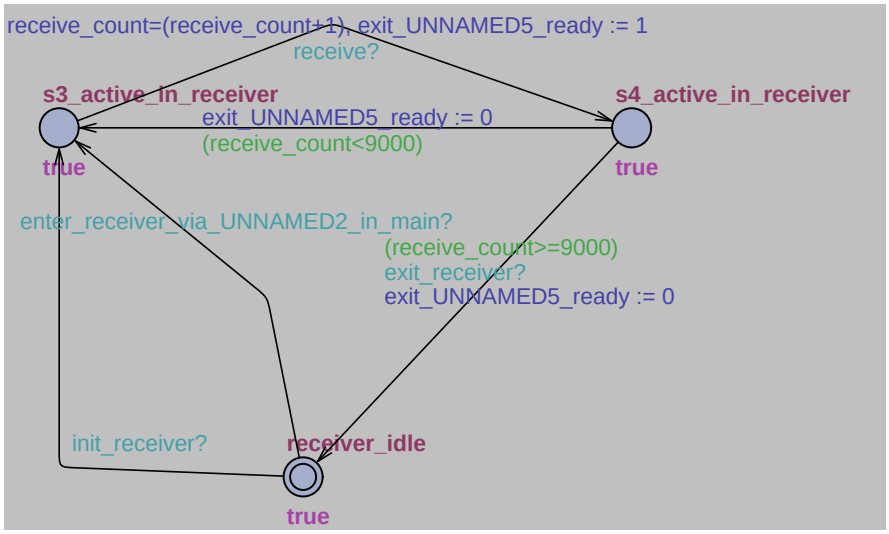

Рисунок 181. Автомат receiver

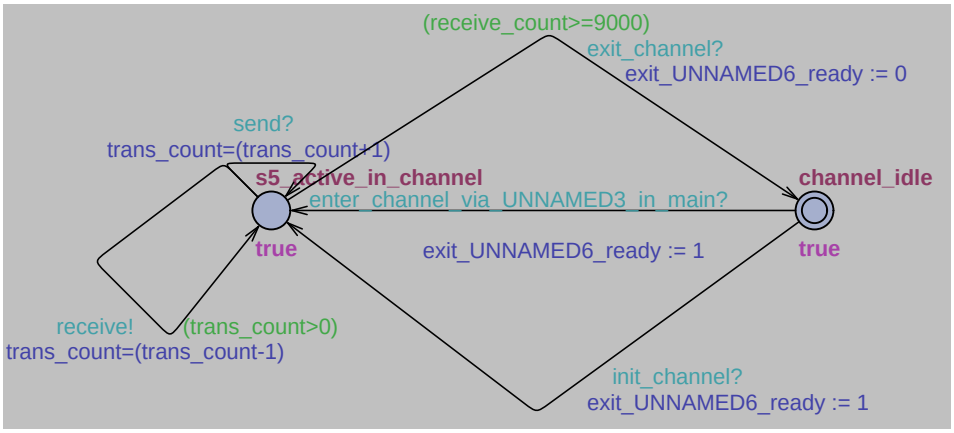

Рисунок 182. Автомат channel

9.5.3 Исследование на модели регулируемого перекрестка

После трансляции модели регулируемого перекрестка, описанной в разделе 6.2, в UPPAAL получилась система из 12 автоматов:

- Global_kickoff
- StreetCrossing
- TrafficController
- Ambulance
- AvenueLight
- StreetLight
- LightController
- AvenueTurn
- StreetTurn
- AmbulanceArriving
- AmbulanceOnAvenue
- AmbulanceOnStreet

На рисунках 183, 184, 185 приведены некоторые из автоматов.

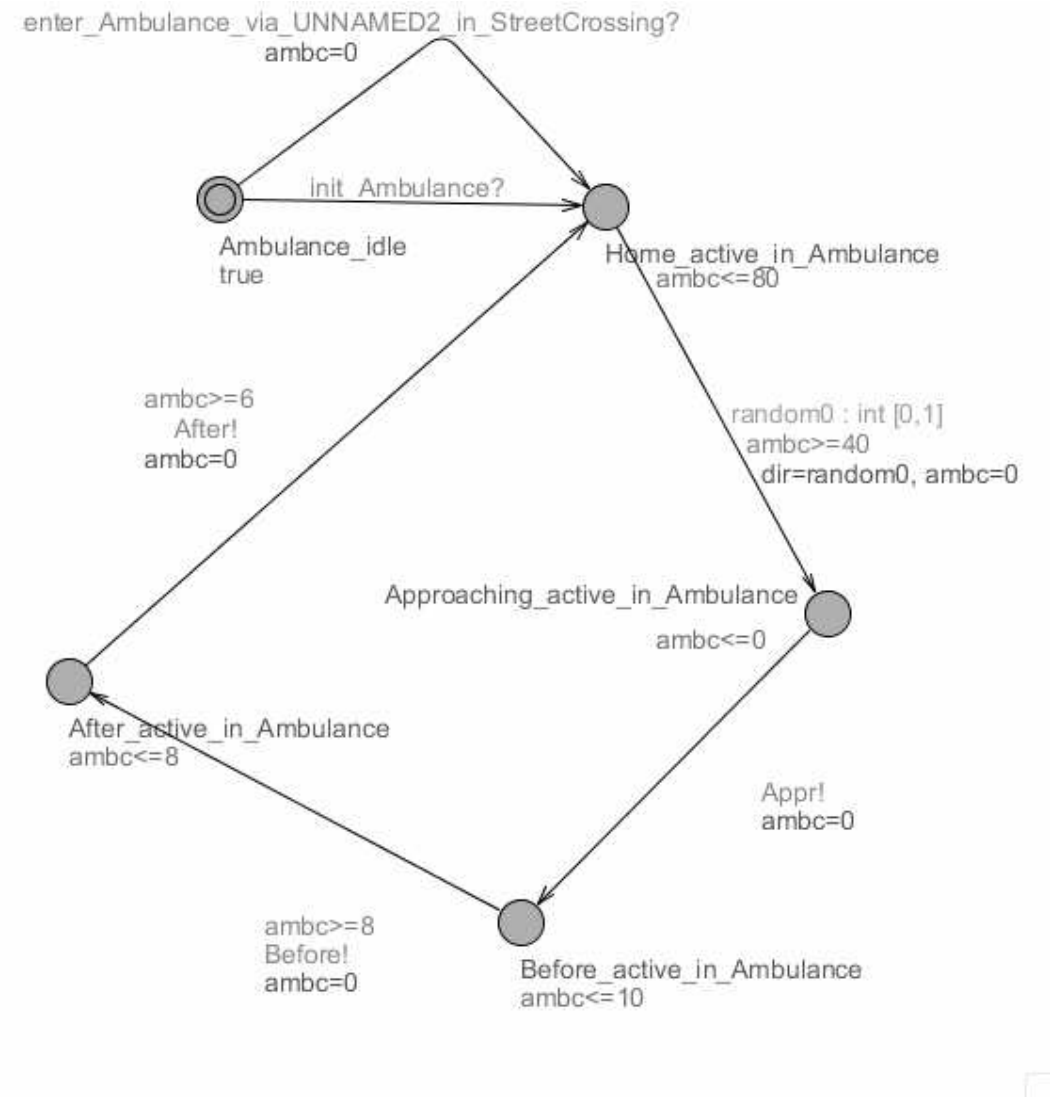

Рисунок 183. Автомат Ambulance.

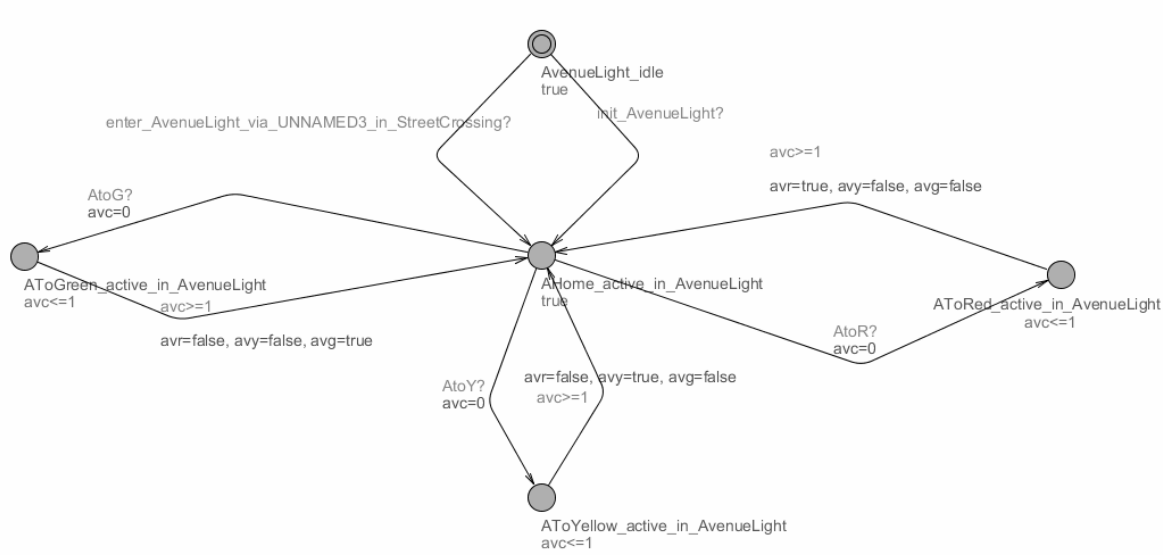

Рисунок 184. Автомат AvenueLight.

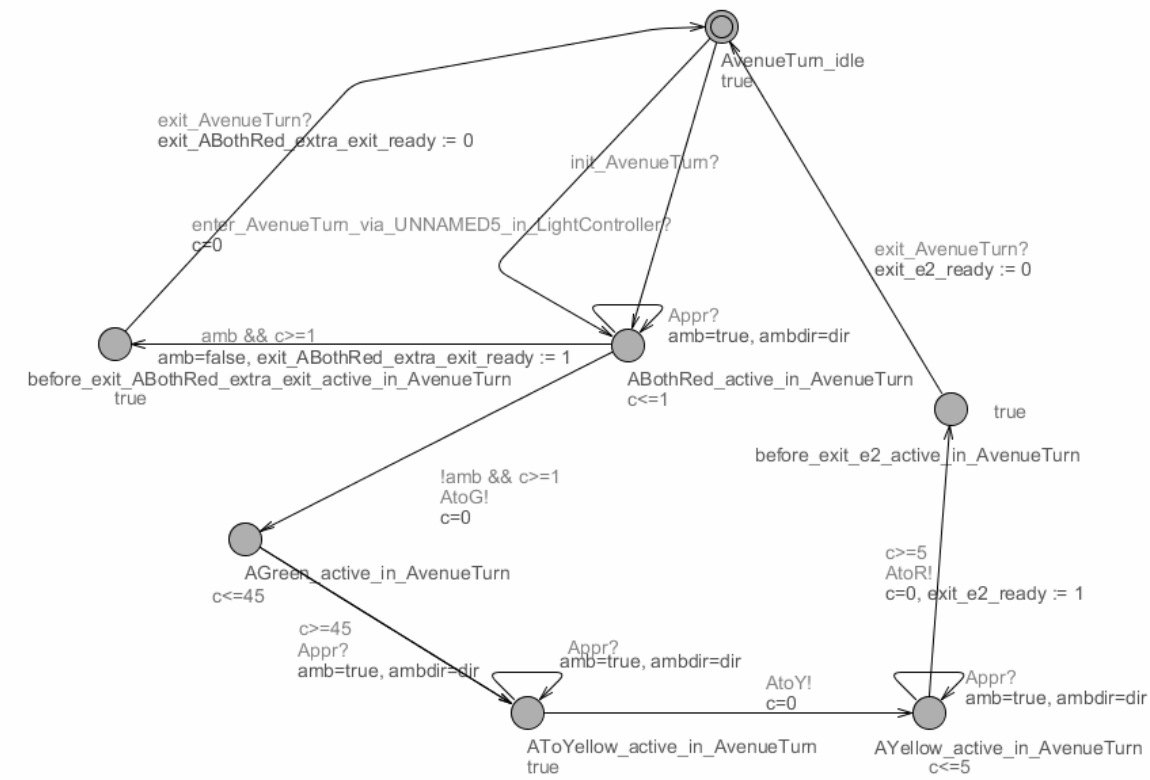

Рисунок 185. Автомат AvenueTurn.

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.

Функциональное тестирование алгоритма подтвердило, что алгоритм работает верно и позволяет получать системы автоматов UPPAAL, соответствующие диаграммам состояний UML. Эксперименты с собственно верификацией моделей приведены в следующем разделе 9.6.

9.6 Экспериментальное исследование средств верификации

Описанный в разделе 5.1 алгоритм трансляции позволяет по предложенной модели построить сеть плоских временных автоматов, используемых средством верификации UPPAAL. Целью трансляции в сеть плоских временных автоматов является проверка выполнимости спецификаций сети, выраженных на языке формул темпоральной логики TCTL, имеющих простую структуру. При этом множество формул, синтаксически допустимых в средстве UPPAAL, достаточно широко, чтобы проверять важные свойства системы, такие как отсутствие тупиков и свойства безопасности и живости системы, которые

можно охарактеризовать соответственно как «в системе не произойдет ничего плохого» и «в системе обязательно будет происходить что-то хорошее».

В следующих подразделах приведены результаты исследования структуры результата трансляции и проверки свойств моделей, описанных в разделах 6.2 – 6.4. Приведенные результаты показывают возможности и эффективность проверки темпоральных свойств с помощью разработанной нами системы.

9.6.1 Модель регулируемого перекрестка

Нами были успешно проверены следующие темпоральные свойства модели регулируемого перекрестка, описанной в разделе 6.2.

1. **A[]! deadlock**
2. **A[] av_color == red or st_color == red**
3. **E<> av_color == green && st_color == green**
4. **A[] av_color == green || st_color == green**
5. **ambul.arrived --> ambul.away**
6. **ambul.away --> ambul.arrived**

Первое свойство гарантирует отсутствие потенциальных тупиков в работе системы. Свойство выполняется.

Второе свойство гарантирует корректное переключение светофоров. Если на одном из светофоров горит зеленый или желтый свет, то на втором горит обязательно красный. Дословный перевод свойства на естественный язык: в любом сценарии поведения хотя бы на одном из светофоров горит красный свет. Свойство выполняется.

Третье свойство утверждает, что существует сценарий выполнения, в котором оба светофора горят зеленым светом. Это свойство не выполняется.

Четвертое свойство утверждает, что всегда один из светофоров горит зеленым светом. Это свойство не выполняется, так как возможна ситуация, в которой один светофор горит красным светом, а другой – желтым. Верификатор, кроме ответа, констатирующего невыполнение свойства, также предоставил ошибочную трассу, соответствующую описанной ситуации. Фрагмент этой трассы приведен на рисунке 186.

Последние два свойства гарантируют прогресс в системе. Пятое свойство утверждает, что если «скорая» появилась на перекрестке, то она проедет все перекрестки и уедет домой. Шестое свойство утверждает, что «скорая» не может отсутствовать на перекрестках вечно. Два этих свойства вместе гарантируют, что состояния arrived и away в любом сценарии выполнения достигаются бесконечное число раз. Свойство выполняется.

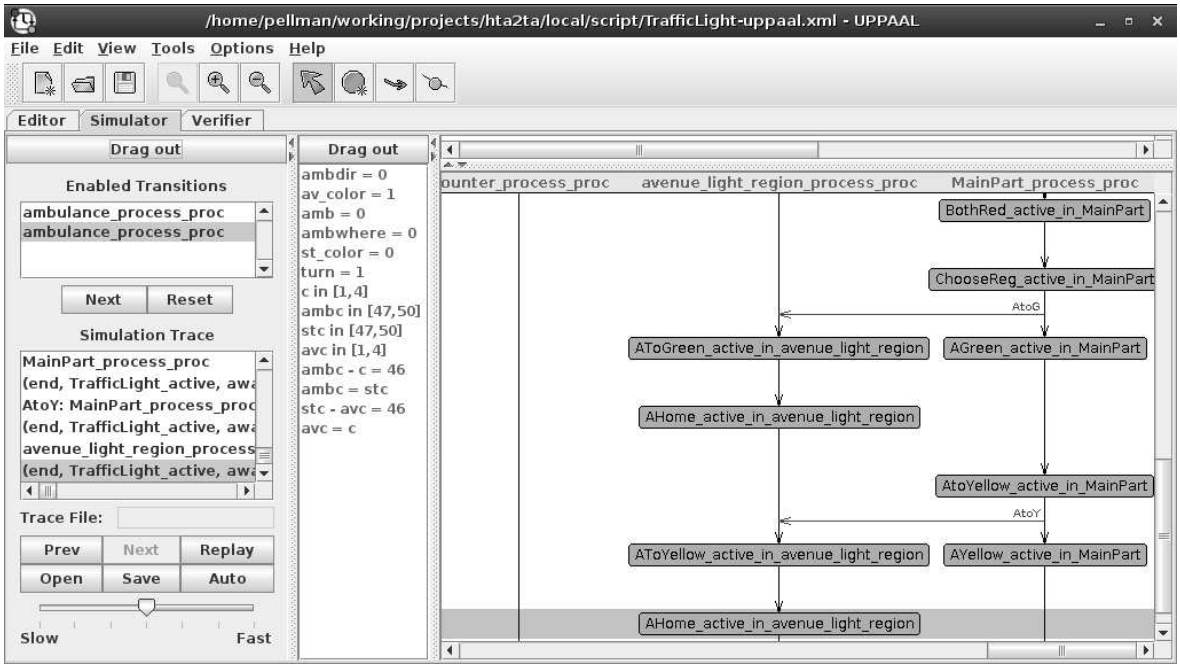

Рисунок 186 — Фрагмент ошибочной трассы UPPAAL

В таблице 42 приведено время проверки свойств на модификациях модели, содержащих от 1 до 4 перекрестков.

Таблица 42 – время проверки свойств модели регулируемого перекрестка (в секундах)

Номер свойства	Количество перекрестков			
	1	2	3	4
1	< 1	< 1	2	38
2	< 1	< 1	1	25
3	< 1	< 1	1	26
4	< 1	< 1	< 1	< 1
5	< 1	< 1	4	74
6	< 1	< 1	2	35

Заметим, что данной модели присущ экспоненциальный и сверхэкспоненциальный рост времени проверки свойств при увеличении числа перекрестков. Это связано с ростом пространства состояний и ростом размера представления одного состояния системы. Экспериментальное исследование показало, что разработанное нами средство работает достаточно эффективно, однако может не справиться с большими моделями. Актуальной в связи с этим является задача оптимизации результата трансляции диаграмм состояний.

9.6.2 Модель поведения бортового компьютера автомобилей

Для модели, описанной в разделе 6.3, нами были проверены следующие свойства.

1. **A[] not deadlock**
2. **car1_position.before imply car1_position.passed**
3. **car1_position.pos_1 imply car1_position.passed**
4. **A[](car1_from == cn_from_r && car2_from == cn_from_d imply (not car1_position.pos_1 or not car2_position.pos_2))**
5. **A[] not places.accident**

Первое свойство гарантирует отсутствие тупиков в модели: все вычисления модели являются бесконечными; иначе говоря, какой бы конфигурации (множества активных состояний, значений переменных и значений таймеров) мы ни достигли, всегда можно или выполнить какой-либо переход, или увеличить время. Данное свойство, помимо прочего, позволяет оценить размер пространства состояний построенной модели.

Второе свойсто является свойством живости: если автомобиль подъехал к перекрестку, то она обязательно его проедет. В описанной модели такое свойство живости не выполняется. Можно привести контрпример к данному свойству со следующим содержательным описанием. Автомобиль появляется на перекрестке, после чего бесконечно долго сбавляет скорость, не останавливаясь и не выезжая на пересечение дорог. В терминах диаграмм, автомат position может бесконечно долго оставаться в состоянии before.

Третье свойство также является свойством живости: если автомобиль выехал непосредственно на пересечение дорог, то он обязательно проедет перекресток. Его выполнимость показывает, что контпример, приведенный для второго свойства, является, по сути, единственным.

Последние два свойства являются свойствами безопасности. Четвертое свойство является свойством взаимного исключения (mutual exclusion) и может быть сформулировано следующим образом: если один автомобиль едет справа, тогда как второй – снизу, то они не могут одновременно занимать правую верхнюю секцию. Аналогичные свойства можно сформулировать для любых направлений и любой секции перекрестка, и все они выполняются.

Выполнимость пятого свойства подтверждает недостижимость «аварийного» состояния в модели, т.е., согласно содержательному описанию модели и условию перехода в «аварийное состояние», отсутствие аварий на перекрестке. Пятое свойство фактически совпадает со всеми аналогами четвертого свойства, взятыми в совокупности, однако формально отличается, формулируется более кратко и проверяется за меньшее время.

В таблице 43 представлено время проверки описанных темпоральных свойств для системы с двумя автомобилями. Следует отметить, что проверка осуществлялась на маломощном вычислительном устройстве класса нетбук.

Таблица 43 – Свойства модели поведения бортового компьютера автомобилей

Свойство	Время проверки	Выполнимость
A[] not deadlock	8 секунд	выполнено
car1_position.before **imply** car1_position.passed	1 секунда	не выполнено
car1_position.pos_1 **imply** car1_position.passed	6 секунд	выполнено
A[](car1_from == cn_from_r && car2_from == cn_from_d **imply** (**not** car1_position.pos_1 **or not** car2_position.pos_2))	3 секунды	выполнено
A[] not places.accident	2 секунды	выполнено

Проверка свойств для модели уже с тремя автомобилями требует больших вычислительных мощностей. Так, время проверки свойства отсутствия тупиков в модели с тремя автомобилями на том же вычислительном устройстве заняло около часа. Такое увеличение времени проверки связано как с увеличением пространства состояний сети плоских временных автоматов, так и с увеличением размера представления состояний, что выражается в падении скорости обработки состояний в десятки раз. Увеличение размера представления состояний связано с существенным увеличением числа процессов сети при добавлении в модель автомобиля.

9.6.3 Модель «БВС»

На рисунках 187-189 приведены некоторые из автоматов сети, получающейся в результате трансляции модели, приведенной в разделе 6.4. Результирующая сеть содержит 23 временных автомата, и на рисунках приведены те из них, которые имеют достаточно малый размер.

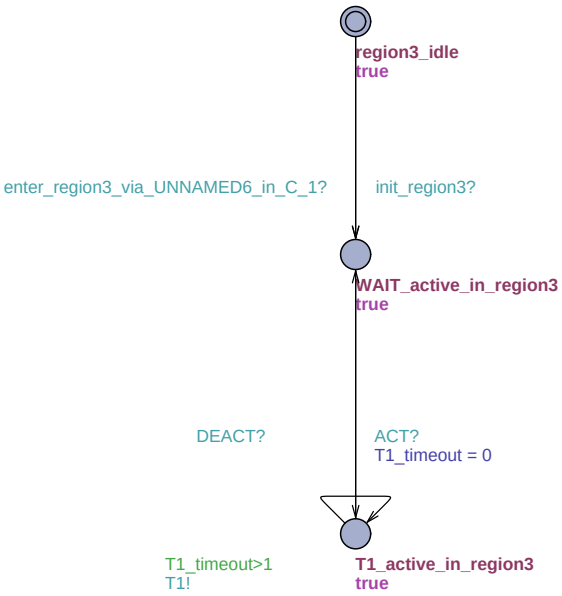

Рисунок 187 — Временной автомат UPPAAL для диаграммы C_1.region3

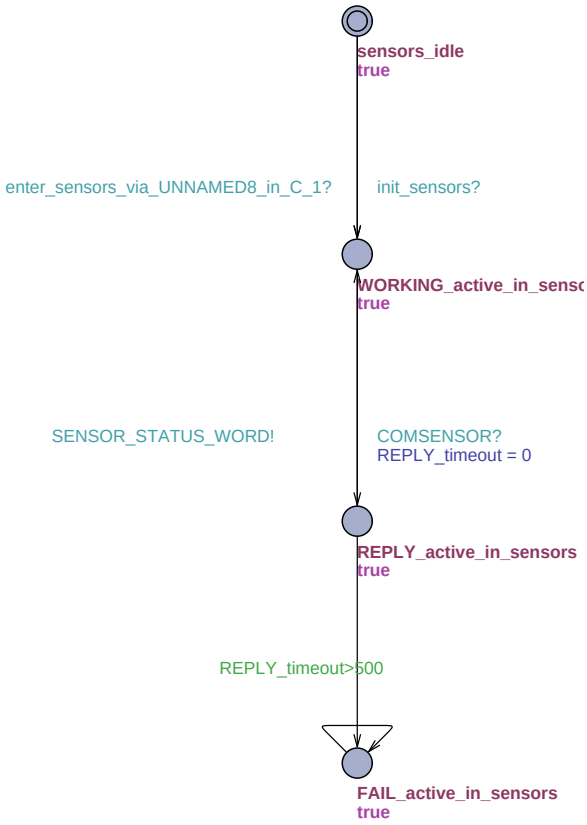

Рисунок 188 — Временной автомат UPPAAL для диаграммы C_1.sensors

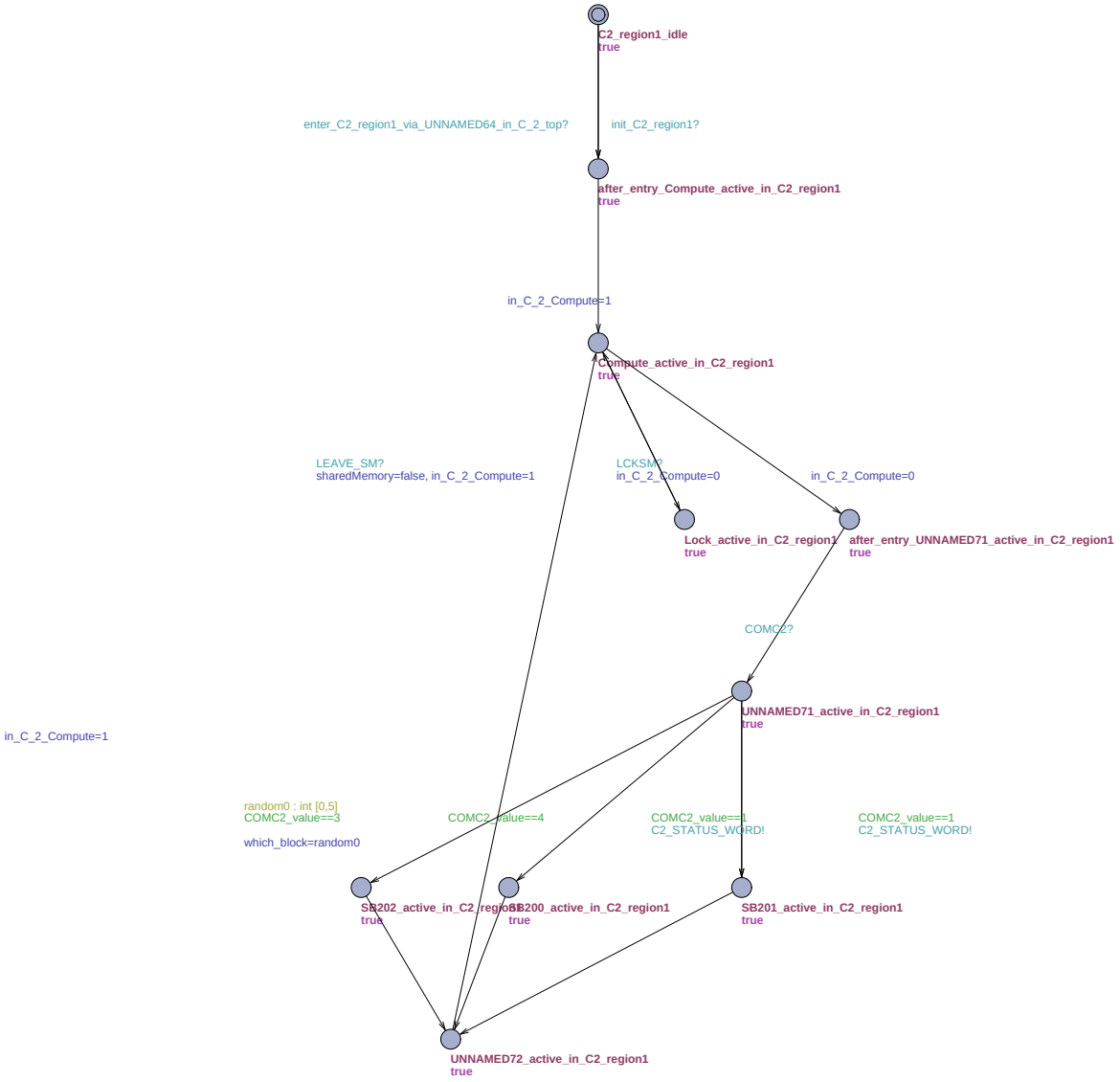

Рисунок 189 — Временной автомат UPPAAL для диаграммы C_2_region1

Также нами был проверен ряд темпоральных свойств этой модели. Примеры свойств приведены далее.

- **E<> region2_process_proc.SB13_active_in_region2**
- **E<> region2_process_proc.S2_active_in_region2**
- **E<> region2_process_proc.SB27_active_in_region2**
- **E<> region2_process_proc.ENDE_from_sb109ref_active_in_region2**
- **E<> region2_process_proc.SB26_active_in_region2**
- **E<> region2_process_proc.SB111_active_in_region2**

Достижимость различных состояний. Время верификации этих свойств варьируется от нескольких секунд до порядка 10 минут в зависимости от того, насколько близко данное состояние к начальному. Объем расходуемой памяти варьируется от 5 Мб до более 100 Мб.

- **region2_process_proc.WAIT_4_active_in_region2 --> C2_region1_process_proc.SB201_active_in_C2_region1**
- **region2_process_proc.S2_active_in_region2 --> C2_region1_process_proc.SB201_active_in_C2_region1**
- **region2_process_proc.WAIT_from_SB106A_active_in_region2 --> C4_region1_process_proc.SB403_active_in_C4_region1**
- **region2_process_proc.WAIT_2_1_active_in_region2 --> C4_region1_process_proc.SB402_active_in_C4_region1**

Свойства, гарантирующие, что на различные управляющие сообщения по шине компьютер C_2 реагирует переходом в соответствующее состояние с обработкой события.

- **(R3_data == 1) --> C4_region2_process_proc.SB30_active_in_C4_region2**
- **(R3_data == 2) --> C4_region2_process_proc.SB35_3_active_in_C4_region2**
- **(R3_data == 3) --> C4_region2_process_proc.SB33_active_in_C4_region2**

Проверка правильности переключения состояний процессора C_4. При получении сигнала с кодом R3_data процессор переходит к соответствующему состоянию.

- **C3_region2_process_proc.AM13_STS6_active_in_C3_region2 --> C3_region3_process_proc.SB7_from_AM23_active_in_C3_region3**
- **C3_region2_process_proc.SB1_active_in_C3_region2 --> C3_region3_process_proc.SB8_from_c3ref_active_in_C3_region3**
- **C3_region2_process_proc.SB15_active_in_C3_region2 --> C3_region3_process_proc.SB7_from_AM22_active_in_C3_region3**

Проверка правильности переключения режимов в двух параллельно выполняющихся циклах процессора C_3.

- **(C2_fault && C3_fault) --> region2_process_proc.WAIT_from_SECTION3_active_in_region2**
- **(C2_fault && !C3_fault) --> region2_process_proc.WAIT_from_SECTION3_active_in_region2**
- **(!C2_fault && !C3_fault) --> region2_process_proc.WAIT_from_SECTION3_active_in_region2**

Проверка правильности работы главного цикла C_1. В зависимости от того, какой или какие процессоры отказывают, осуществляется переход в ту или иную секцию.

- **A[] critical_C2 == false || critical_C3 == false**

Свойство безопасности: невозможен одновременный доступ к разделяемой памяти. Переменные critcal_C2 и critical_C3 были добавлены на диаграмму специально для

верификации и означают, что вычислители C_2 и C_3 соответственно находятся внутри критической секции.

Это свойство было проверено также следующим образом: в диаграмме RW2 было убрано изменено предусловие для входа в критическую секцию, и было проверено, что указанное выше свойство перестало выполняться.

- **A[] !deadlock**

Свойство отсутствия тупиков.

Верификация последних двух свойств требует перебора всех возможных состояний, поэтому для нее необходимы значительные ресурсы: она требует более 4 Гб памяти и длится несколько часов. При внесении изменения с явным нарушением этих свойств контрпример был найден менее чем за 5 секунд.

9.7 Эксперименты по восстановлению параметров модели по контрпримеру в UPPAAL.

В данном разделе приводятся «ошибочные» трассы, построенные в результате проверки свойств моделей, описанных в разделах 6.2, 6.4. На них можно видеть формат трассы UML, получаемой в результате построения, описанного в разделе 5.4.

Один шаг трассы обозначается строкой *Step N*, где *N* – порядковый номер шага, и следующим текстом до строки вида *Step N+1*. Текст шага состоит из указания перехода, выполненного на данном шаге, и указания множества значений переменных и таймеров после выполнения перехода и, возможно, продвижения времени. Для удобства указываются только значения тех переменных и таймеров, которые изменились по сравнению с предыдущим шагом. Указание перехода из состояния *Source_state* в состояние *Target_state* в автомате *Automaton* имеет вид *Automaton :: Source_state --> Target_state*. Множество значений переменных и таймеров описано системой равенств и неравенств.

В настоящее время средство ограничивается преобразованием трассы UPPAAL в человекочитаемую форму в терминах диаграммы UML. В перспективе планируется трансляция таких трасс непосредственно в формат OTF для визуализации в Vis4.

9.7.1 Модель регулируемого перекрестка

Первый пример, на котором рассмотрено преобразование трасс UPPAAL в трассы UML, – система управления уличным движением на перекрестке, более поздняя версия которой описана в разделе 6.2.

На рисунках 190, 191 приведены диаграммы модели, для которых строится трасса. Содержательное описание модели почти дословно (кроме возможности описания нескольких перекрестков) повторяет описание модели в разделе 6.2.

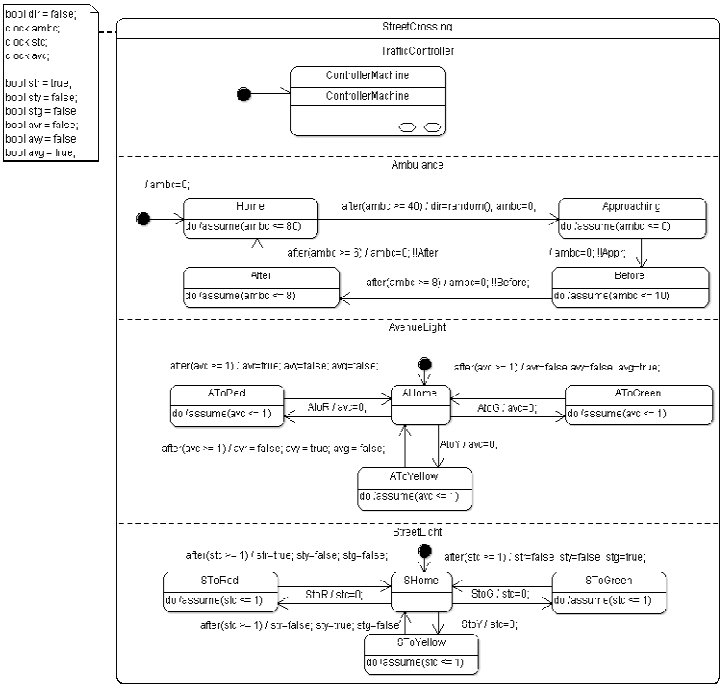

Рисунок 190. Основная диаграмма модели регулируемого перекрестка

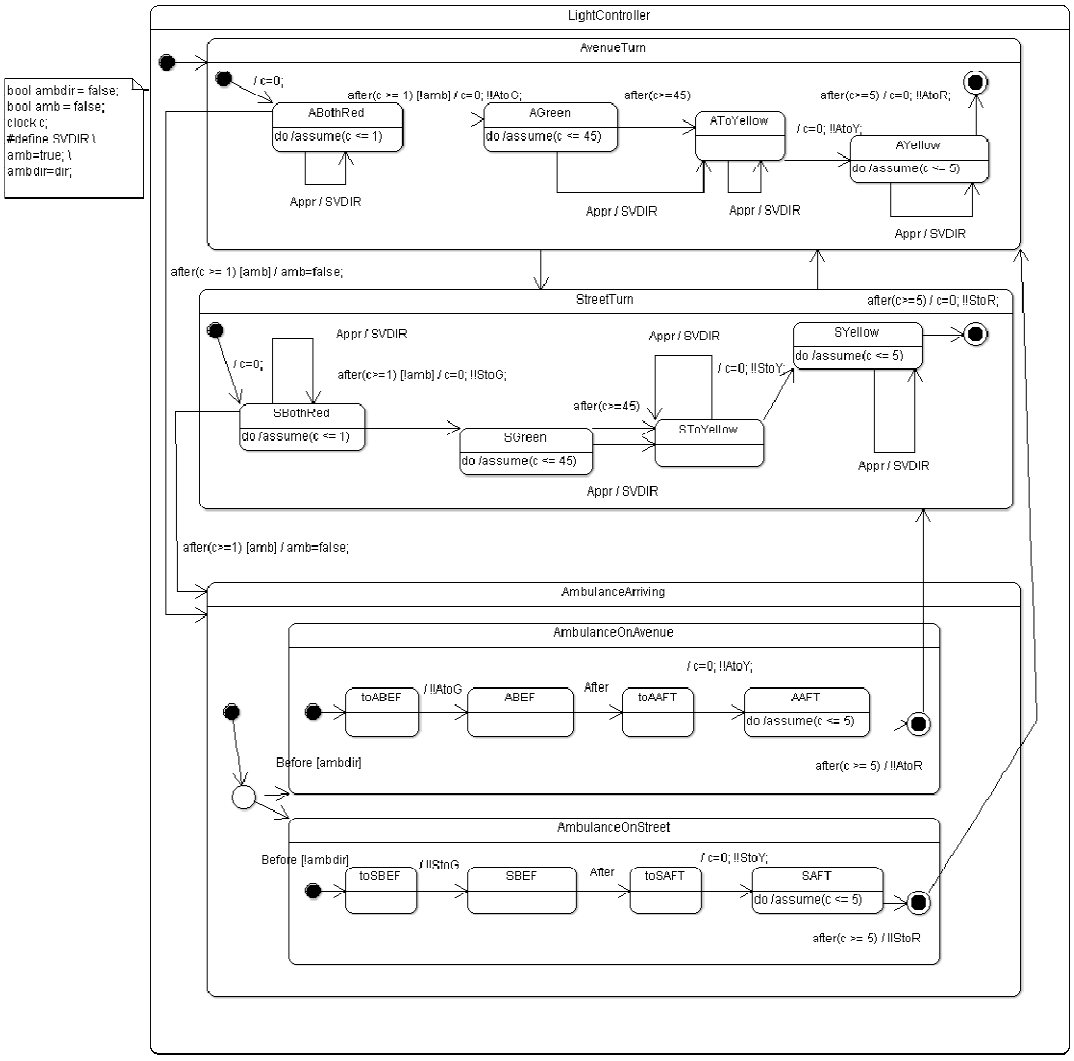

Рисунок 191. Диаграмма управляющего устройства модели регулируемого перекрестка

Приведенная ниже трасса описывает три проезда «скорой» через перекресток при разных комбинациях смены цвета светофоров.

```
Step 1
TrafficController_process :: AvenueTurn.ABothRed --> AvenueTurn.AGreen
Step 2
AvenueLight_process :: AvenueLight.AHome --> AvenueLight.AToGreen
c >= 0
c <= 1
stc >= 1
ambc >= 1
avc >= 0
avc <= 1
ambc <= 2
stc <= 2
Step 3
AvenueLight_process :: AvenueLight.AToGreen --> AvenueLight.AHome
c >= 1
c <= 45
stc >= 2
ambc >= 2
avc >= 1
avc <= 45
```

```
ambc <= 46
stc <= 46
Step 4
Ambulance_process :: Ambulance.Home --> Ambulance.Approaching
dir = 1
c >= 39
ambc >= 0
ambc <= 0
stc >= 40
avc >= 39
Step 5
TrafficController_process :: AvenueTurn.AGreen --> AvenueTurn.AToYellow
Step 6
Ambulance_process :: Ambulance.Approaching --> Ambulance.Before
ambdir = 1
amb = 1
ambc <= 10
c <= 55
avc <= 55
stc <= 56
Step 7
Ambulance_process :: Ambulance.Before --> Ambulance.After
ambc <= 8
c <= 63
c >= 47
avc <= 63
stc >= 48
stc <= 64
avc >= 47
Step 8
Ambulance_process :: Ambulance.After --> Ambulance.Home
ambc <= 80
c <= 143
c >= 53
avc <= 143
stc >= 54
stc <= 144
avc >= 53
Step 9
Ambulance_process :: Ambulance.Home --> Ambulance.Approaching
dir = 0
c >= 93
ambc <= 0
stc >= 94
avc >= 93
Step 10
Ambulance_process :: Ambulance.Approaching --> Ambulance.Before
ambdir = 0
ambc <= 10
c <= 153
avc <= 153
stc <= 154
Step 11
TrafficController_process :: AvenueTurn.AToYellow --> AvenueTurn.AYellow
Step 12
AvenueLight_process :: AvenueLight.AHome --> AvenueLight.AToYellow
c >= 0
c <= 1
avc >= 0
avc <= 1
Step 13
Ambulance_process :: Ambulance.Before --> Ambulance.After
ambc <= 1
stc >= 102
stc <= 155
Step 14
TrafficController_process :: AvenueTurn.AYellow --> StreetTurn.SBothRed
Step 15
TrafficController_process :: StreetTurn.SBothRed --> AmbulanceArriving.UNNAMED7
amb = 0
c >= 1
```

```
avc >= 1
Step 16
AvenueLight_process :: AvenueLight.AToYellow --> AvenueLight.AHome
avy = 1
avg = 0
ambc <= 8
c <= 9
stc <= 162
avc <= 9
Step 17
Ambulance_process :: Ambulance.After --> Ambulance.Home
ambc <= 80
c <= 89
stc <= 243
stc <= 242
c >= 6
stc >= 108
avc <= 89
avc >= 6
Step 18
Ambulance_process :: Ambulance.Home --> Ambulance.Approaching
c >= 46
stc >= 148
ambc <= 0
avc >= 46
Step 19
Ambulance_process :: Ambulance.Approaching --> Ambulance.Before
ambc <= 10
c <= 99
stc <= 253
stc <= 252
avc <= 99
Step 20
TrafficController_process :: AmbulanceArriving.UNNAMED7 --> AmbulanceOnStreet.toSBEF
Step 21
Ambulance_process :: Ambulance.Before --> Ambulance.After
ambc <= 8
c <= 107
stc <= 261
stc <= 260
c >= 54
stc >= 156
avc <= 107
avc >= 54
Step 22
TrafficController_process :: AmbulanceOnStreet.toSBEF --> AmbulanceOnStreet.SBEF
Step 23
StreetLight_process :: StreetLight.SHome --> StreetLight.SToGreen
stc >= 0
stc <= 1
Step 24
StreetLight_process :: StreetLight.SToGreen --> StreetLight.SHome
stg = 1
str = 0
stc >= 1
ambc >= 1
stc <= 8
c >= 55
avc >= 55
Step 25
TrafficController_process :: AmbulanceOnStreet.SBEF --> AmbulanceOnStreet.toSAFT
Step 26
Ambulance_process :: Ambulance.After --> Ambulance.Home
ambc >= 0
ambc <= 80
c <= 187
stc <= 133
stc <= 88
c >= 60
avc <= 187
avc >= 60
```

9.7.2 Модель «БВС»

В данном разделе приводится трасса для модели, описанной в разделе 6.4. Модель состоит из четырех вычислителей C_1, C_2, C_3 и C_4, подключенных к общей шине. Процессоры C_2 и C_3 имеют общую память. Каждый из вычислителей выполняет свой круг задач. Вычислитель C_1 – главный процессор, управляющий вычислениями и передачей данных во всей системе. Процессор C_2 вычисляет параметры полета и готовит данные для датчиков. Процессор C_3 определяет положение самолета по показаниям датчиков и обеспечивает перемещение по требуемому маршруту, а также управляет сенсорами. Процессор C_4 обрабатывает информацию с радара и управляет полетом самолета на малой высоте.

На приведенной ниже трассе показана инициализация системы и несколько первых шагов работы, в том числе вход в критическую секцию для работы с общей памятью.

```
Step 1
region2_process :: SECTION1_from_SECTION1.SB100 --> SECTION1_from_SECTION1.SB101
PAUSE_from_SECTION4_timeout >= 0
CHECK_C3_from_SB104A_timeout >= 0
CHECK_C4_from_SB104A_timeout >= 0
UNNAMED26_timeout >= 0
S3_timeout >= 0
S1_from_sb109ref_timeout >= 0
S1_from_SB106A_timeout >= 0
S1_from_sb109ref_from_SECTION5_timeout >= 0
S1_from_sb107_timeout >= 0
CHECK_C4_timeout >= 0
S3_from_SECTION4_timeout >= 0
COLLECT_SENSOR_timeout >= 0
TIMER_timeout >= 0
T1_timeout >= 0
CHECK_S_from_SB104A_timeout >= 0
CHECK_C2_from_SB104A_timeout >= 0
CHECK_S_timeout >= 0
S1_timeout >= 0
PAUSE_from_SECTION3_timeout >= 0
S1_from_SECTION5_timeout >= 0
S1_from_SB109_timeout >= 0
S1_from_SECTION3_timeout >= 0
S1_from_SB107C_timeout >= 0
CHECK_C2_timeout >= 0
PAUSE_timeout >= 0
REPLY_timeout >= 0
S3_from_SECTION3_timeout >= 0
CHECK_C3_timeout >= 0
WAIT_from_SECTION2_timeout >= 0
S1_from_SECTION4_timeout >= 0
T2_timeout >= 0
Step 2
C2_region1_process :: C2_region1.Compute --> C2_region1.Lock
Step 3
C3_region3_process :: RW3_from_AM21.WAIT --> RW3_from_AM21.LOCKING
Step 4
C3_region4_process :: C3_region4.Compute --> C3_region4.LOCK
critical_C3 = 1
sharedMemory = 1
Step 5
C3_region1_process :: C3_region1.TIMER --> C3_region1.TIMER_2
PAUSE_from_SECTION4_timeout >= 10
CHECK_C3_from_SB104A_timeout >= 10
```

```
CHECK_C4_from_SB104A_timeout >= 10
UNNAMED26_timeout >= 10
S3_timeout >= 10
S1_from_sb109ref_timeout >= 10
S1_from_SB106A_timeout >= 10
S1_from_sb109ref_from_SECTION5_timeout >= 10
S1_from_sb107_timeout >= 10
CHECK_C4_timeout >= 10
S3_from_SECTION4_timeout >= 10
COLLECT_SENSOR_timeout >= 10
TIMER_timeout >= 10
T1_timeout >= 10
CHECK_S_from_SB104A_timeout >= 10
CHECK_C2_from_SB104A_timeout >= 10
CHECK_S_timeout >= 10
S1_timeout >= 10
PAUSE_from_SECTION3_timeout >= 10
S1_from_SECTION5_timeout >= 10
S1_from_SB109_timeout >= 10
S1_from_SECTION3_timeout >= 10
S1_from_SB107C_timeout >= 10
CHECK_C2_timeout >= 10
PAUSE_timeout >= 10
REPLY_timeout >= 10
S3_from_SECTION3_timeout >= 10
CHECK_C3_timeout >= 10
WAIT_from_SECTION2_timeout >= 10
S1_from_SECTION4_timeout >= 10
T2_timeout >= 10
Step 6
C3_region3_process :: RW3_from_AM21.LOCKING --> RW3_from_AM21.EXIT
critical_C3 = 0
Step 7
C3_region1_process :: C3_region1.TIMER_2 --> C3_region1.TIMER
Step 8
C3_region1_process :: C3_region1.TIMER --> C3_region1.TIMER_2
Step 9
region1_process :: region1.CP --> TIMER_INTER_from_TIMER_INTER.UNNAMED19
Step 10
region2_process :: SECTION1_from_SECTION1.SB101 --> SECTION1_from_SECTION1.SB25
Step 11
C3_region1_process :: C3_region1.TIMER_2 --> C3_region1.TIMER
Step 12
region2_process :: SECTION1_from_SECTION1.SB25 --> SECTION1_from_SECTION1.UNNAMED21
Step 13
C3_region1_process :: C3_region1.TIMER --> C3_region1.TIMER_2
Step 14
C2_region1_process :: C2_region1.Lock --> C2_region1.Compute
Step 15
C3_region3_process :: RW3_from_AM21.EXIT --> STS17.SB8
Step 16
C3_region4_process :: C3_region4.LOCK --> C3_region4.Compute
in_C_3_Compute = 1
in_C_2_Compute = 1
sharedMemory = 0
Step 17
region2_process :: SECTION1_from_SECTION1.UNNAMED21 --> SECTION1_from_SECTION1.SB102
Step 18
C2_region2_process :: C2_region2.WAIT --> RW2_top.WAIT
REQUEST_C2 = 1
Step 19
C3_region3_process :: STS17.SB8 --> STS17.UNNAMED133
Step 20
C2_region1_process :: C2_region1.Compute --> C2_region1.Lock
Step 21
C2_region2_process :: RW2_top.WAIT --> RW2_top.LOCKING
Step 22
C3_region4_process :: C3_region4.Compute --> C3_region4.LOCK
in_C_3_Compute = 0
critical_C2 = 1
in_C_2_Compute = 0
```

```
sharedMemory = 1
Step 23
C3_region1_process :: C3_region1.TIMER_2 --> C3_region1.TIMER
Step 24
C2_region2_process :: RW2_top.LOCKING --> RW2_top.EXIT
critical_C2 = 0
REQUEST_C2 = 0
Step 25
C3_region1_process :: C3_region1.TIMER --> C3_region1.TIMER_2
Step 26
region2_process :: SECTION1_from_SECTION1.SB102 --> SECTION1_from_SECTION1.SB13
Step 27
C3_region3_process :: STS17.UNNAMED133 --> RW3_from_AM21.WAIT
Step 28
C2_region1_process :: C2_region1.Lock --> C2_region1.Compute
Step 29
C2_region2_process :: RW2_top.EXIT --> C2_region2.SB23
Step 30
C3_region4_process :: C3_region4.LOCK --> C3_region4.Compute
in_C_3_Compute = 1
in_C_2_Compute = 1
sharedMemory = 0
Step 31
region2_process :: SECTION1_from_SECTION1.SB13 --> SECTION1_from_SECTION1.SB103
Step 32
C2_region2_process :: C2_region2.SB23 --> RW2_top_from_RW3.WAIT
REQUEST_C2 = 1
Step 33
C3_region4_process :: C3_region4.Compute -->
SMART_CONTROL_C3_from_SMART_CONTROL_C3.UNNAMED145
in_C_3_Compute = 0
Step 34
C2_region1_process :: C2_region1.Compute --> SMART_CONTROL_C2.UNNAMED79
in_C_2_Compute = 0
Step 35
C2_region2_process :: RW2_top_from_RW3.WAIT --> RW2_top_from_RW3.LOCKING
critical_C2 = 1
sharedMemory = 1
Step 36
C2_region2_process :: RW2_top_from_RW3.LOCKING --> RW2_top_from_RW3.EXIT
critical_C2 = 0
REQUEST_C2 = 0
Step 37
C2_region2_process :: RW2_top_from_RW3.EXIT --> C2_region2.SB24
Step 38
C3_region1_process :: C3_region1.TIMER_2 --> C3_region1.TIMER
Step 39
C3_region1_process :: C3_region1.TIMER --> C3_region1.TIMER_2
Step 40
C3_region1_process :: C3_region1.TIMER_2 --> C3_region1.TIMER
Step 41
C3_region1_process :: C3_region1.TIMER --> C3_region1.TIMER_2
Step 42
C3_region1_process :: C3_region1.TIMER_2 --> C3_region1.TIMER
Step 43
region2_process :: SECTION1_from_SECTION1.SB103 --> SECTION1_from_SECTION1.UNNAMED24
Step 44
C3_region1_process :: C3_region1.TIMER --> C3_region1.TIMER_2
Step 45
region2_process :: SECTION1_from_SECTION1.UNNAMED24 --> SB104.WAIT
Step 46
C3_region1_process :: C3_region1.TIMER_2 --> C3_region1.TIMER
```

9.8 Эксперименты по совместному применению системы моделирования со средствами синтеза архитектур и построения расписаний

9.8.1 Представление конфигурации РВС РВ в виде расписания общего вида

Для применения описанной в разделе 5.8 схемы интеграции средств синтеза архитектур и построения расписаний и среды моделирования к средству сбалансированного выбора МОО РВС РВ (раздел 5.7) необходимо осуществить переход от внутреннего представления конфигурации РВС РВ к расписанию общего вида.

Покажем, что каждой конфигурации РВС РВ, определенной в разделе 5.7.2 можно однозначно поставить в соответствие расписание, определенное в разделе 5.8.1 как множество S троек *(v,m,n)*, где *v* – задание программы, *m* – процессор, на котором задание выполняется, *n* – порядковый номер задания на процессоре. Каждой конфигурации $\left\{H_i^{F_i}, S_i^{F_i}, F_i\right\}$ модуля U_i ставятся в соответствие элементы расписания *S* следующим образом, в зависимости от F_i:

1. F_i=*None* $\Rightarrow$ $H_i^{F_i}=\left\{H_i^{F_i}(1)\right\}$, $S_i^{F_i}=\left\{S_i^{F_i}(1)\right\}$. Данной конфигурации соответствует элемент расписания $s_{i1}=\left(S_i^{F_i}(1),H_i^{F_i}(1),1\right)$

2. F_i=*NVP/0/1* $\Rightarrow$ $H_i^{F_i}=\left\{H_i^{F_i}(1)\right\}$, $S_i^{F_i}=\left\{S_i^{F_i}(1),S_i^{F_i}(2),S_i^{F_i}(3)\right\}$. Данной конфигурации соответствуют элементы расписания $s_{i1}=\left(receiver_i,H_i^{F_i}(1),1\right)$, $s_{i2}=\left(S_i^{F_i}(1),H_i^{F_i}(1),2\right)$, $s_{i3}=\left(S_i^{F_i}(2),H_i^{F_i}(1),3\right)$, $s_{i4}=\left(S_i^{F_i}(3),H_i^{F_i}(1),4\right)$, $s_{i5}=\left(voter_i,H_i^{F_i}(1),5\right)$.

3. F_i=*NVP/1/1* $\Rightarrow$ $H_i^{F_i}=\left\{H_i^{F_i}(1),H_i^{F_i}(2),H_i^{F_i}(3)\right\}$, $S_i^{F_i}=\left\{S_i^{F_i}(1),S_i^{F_i}(2),S_i^{F_i}(3)\right\}$. Данной конфигурации соответствуют элементы расписания $s_{i1}=\left(receiver_i,H_i^{F_i}(1),1\right)$, $s_{i2}=\left(S_i^{F_i}(1),H_i^{F_i}(1),2\right)$, $s_{i3}=\left(S_i^{F_i}(2),H_i^{F_i}(2),1\right)$, $s_{i4}=\left(S_i^{F_i}(3),H_i^{F_i}(3),1\right)$, $s_{i5}=\left(voter_i,H_i^{F_i}(1),3\right)$.

4. F_i=RB*/1/1* $\Rightarrow$ $H_i^{F_i}=\left\{H_i^{F_i}(1),H_i^{F_i}(2)\right\}$, $S_i^{F_i}=\left\{S_i^{F_i}(1),S_i^{F_i}(2)\right\}$. Данной конфигурации соответствуют элементы расписания $s_{i1}=\left(receiver_i,H_i^{F_i}(1),1\right)$, $s_{i2}=\left(S_i^{F_i}(1),H_i^{F_i}(1),2\right)$, $s_{i3}=\left(test_i,H_i^{F_i}(1),3\right)$, $s_{i4}=\left(recovery_i,H_i^{F_i}(1),4\right)$, $s_{i5}=\left(S_i^{F_i}(2),H_i^{F_i}(1),5\right)$, $s_{i6}=\left(test_i,H_i^{F_i}(1),6\right)$, $s_{i7}=\left(sender_i,H_i^{F_i}(1),7\right)$, $s_{i8}=\left(S_i^{F_i}(1),H_i^{F_i}(2),1\right)$, $s_{i9}=\left(test_i,H_i^{F_i}(2),2\right)$, $s_{i10}=\left(recovery_i,H_i^{F_i}(2),3\right)$, $s_{i11}=\left(S_i^{F_i}(2),H_i^{F_i}(2),4\right)$, $s_{i12}=\left(test_i,H_i^{F_i}(2),7\right)$. Отметим, что в рамках поставленной задачи необходимо получить максимальное время завершения работы программ, поэтому считается,

что результат работы первой версии программного компонента всегда отвергается, происходит откат и запуск второй версии.

Никаких других элементов, кроме описанных выше, в расписании нет.

В ходе выполнения заданий $receiver_i$ происходит прием данных от других модулей; $sender_i$ – отправка данных; $voter_i$ – выбор результата, выдаваемого большинством версий программного компонента, и отправка данных; $test_i$ – проверка результата с помощью контрольного теста; $recovery_i$ – откат к начальному состоянию. Если в i-ом модуле не используется МОО, то действия по приему и отправке данных входят в задание s_{i1}.

Опишем зависимости между элементами построенного расписания:

- Если F_i=*NVP/0/1* или F_i=*NVP/1/1*, то элементы s_{i2}, s_{i3}, s_{i4} непосредственно зависят от s_{i1}; s_{i5} – от s_{i2}, s_{i3}, s_{i4}.
- Если F_i=RB*/1/1*, то s_{i2}, и s_{i8} непосредственно зависят от s_{i1}; s_{ij} – от s_{ij-1} для j={3,4,5,6,9,10,11,12}; s_{i7} – от s_{i9} и s_{i12}.
- Если модуль U_i непосредственно зависит по данным от модуля U_j, то s_{i1} непосредственно зависит либо от s_{j1} (F_j=*None*), либо от s_{j5} (F_j=*NVP/0/1* или F_j=*NVP/1/1*), либо от s_{j7} (F_j=*RB/1/1*).

Директивные сроки всех заданий, соответствующих модулю U_i, равны D_i, временем завершения работы программного компонента модуля U_i считается время завершения работы задания, осуществляющего передачу данных.

9.8.2 Результаты экспериментов

В ходе данной работы было проведено экспериментальное исследование алгоритмов, описанных в разделе 5.7. Алгоритмы сравнивались по качеству полученных с их помощью решений, то есть по значению целевой функции (надежности) найденного решения. В качестве тестовой РВС РВ была взята РВС РВ со следующими характеристиками:

n = 10 – количество модулей РВС РВ;

p_i=5, q_i=5 – количество доступных версий аппаратных и программных компонентов в каждом модуле;

R_{ij}^{hw}, R_{ij}^{sw} – надежности вариантов аппаратных и программных компонентов в каждом модуле; случайные величины из равномерного распределения на отрезке [0.85;0.99]

C_{ij}^{hw}, C_{ij}^{sw} – стоимости вариантов аппаратных и программных компонентов; принадлежат отрезку [5, 35], чем больше надежность компонента, тем больше его стоимость.

“Чистое” время выполнения программы для каждой пары (аппаратный компонент, программный компонент) – случайная величина из равномерного распределения на отрезке [100; 300]

Граф зависимостей по данным был сгенерирован случайным образом и изображен на рисунке 192:

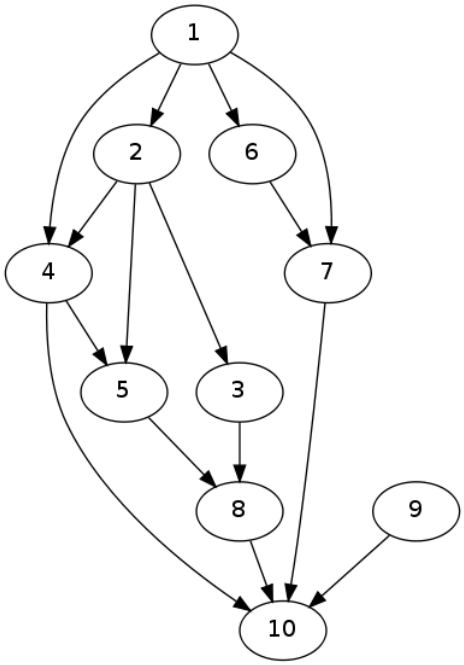

Рисунок 192. Граф зависимостей по данным исследуемой РВС РВ

В данном исследовании рассматривались три вида ограничений на стоимость и время:

Жесткие. Ограничение на стоимость: 600. Ограничения на время окончания выполнения программ в каждом модуле: 230, 460, 740, 850, 1200, 1500, 1790, 1500, 300, 2400.

Средние. Ограничение на стоимость: 800. Ограничения на время окончания выполнения программ в каждом модуле: 400, 750, 1100, 1100, 1600, 1900, 2200, 1800, 400, 2800.

Ограничения отсутствуют.

В таблице 44 приведены значения целевой функции решений, полученных с помощью классического ЭА, АГЭА, островных алгоритмов, состоящих из классических ЭА и АГЭА и алгоритма имитации отжига, усредненные по 100 запускам.

Таблица 44 - Значения целевой функции решений, полученных с помощью разных алгоритмов

Алгоритм	Ограничения на стоимость и время		
	Жесткие	Средние	Отсутствуют
Классический ЭА	0,2503	0,5560	0,8282
АГЭА	0,2434	0,7309	0,8659
Островный классический ЭА	0,2494	0,5052	0,7869
Островной АГЭА	0,2563	0,6142	0,8600
Алгоритм имитации отжига	0,3814	0,6897	0,8499

Эксперименты показали, что при отсутствии ограничений и при средних ограничениях на стоимость и время наилучшие решения могут быть получены с помощью АГЭА. При жестких ограничениях лучшим алгоритмом по качеству решений является алгоритм имитации отжига.

9.9 Экспериментальное исследование средства визуализации трасс моделей

В данном разделе проиллюстрировано совместное использование разработанных средств на примере модели «Лавина», UML-представление которой приведено в разделе 6.1 (создание UML-моделей описано в разделах 4.2, 8.2).

Создаем новый проект, загружаем в него SCXML-файл. Данный файл может быть как создан вручную в редакторе SCXML диаграмм, так и сгенерирован из XMI-представления UML (см. рисунок 193).

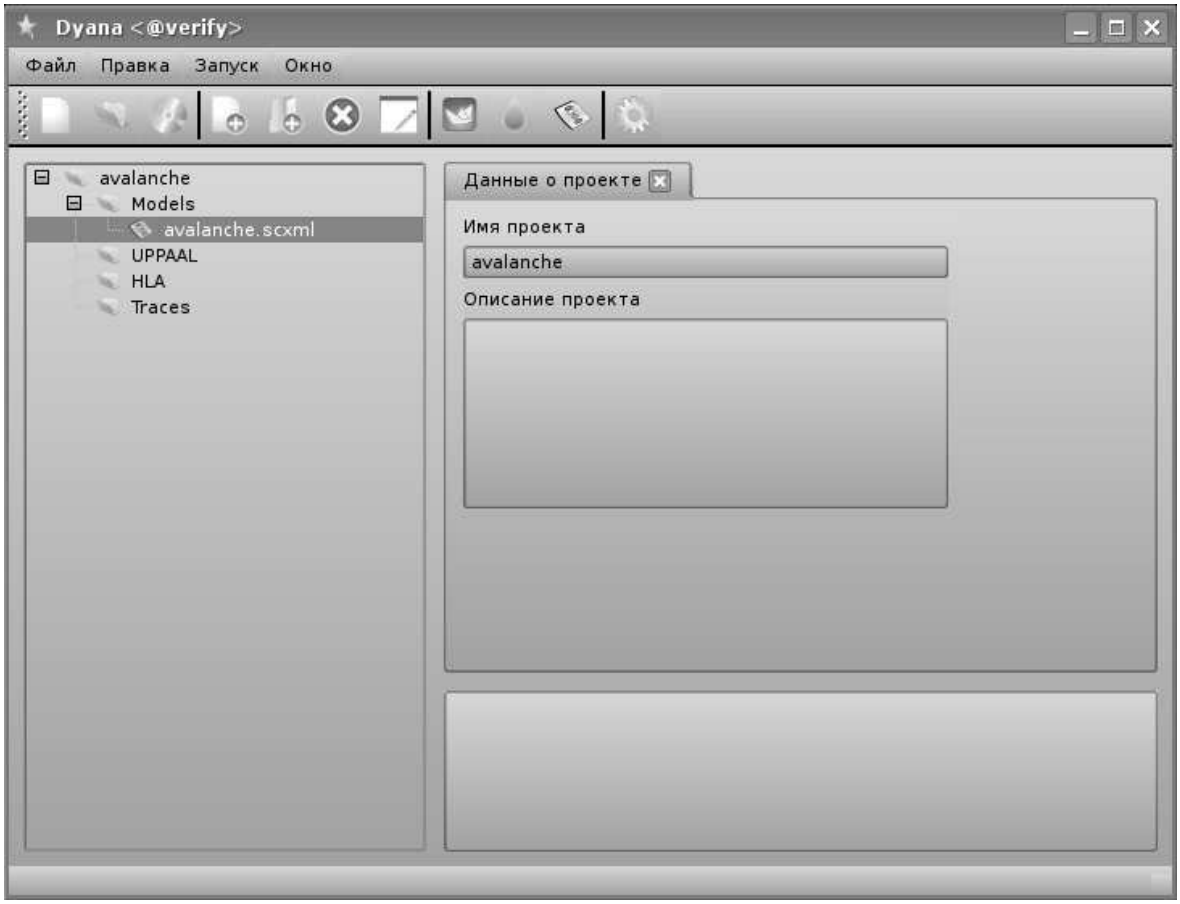

Рисунок 193. Создание нового проекта. Загрузка SCXML-файла.

Открываем вкладку для SCXML-файла (см. рисунок 194). Задаем путь для размещения сгенерированного кода федератов («HLA/federates») и нажимаем кнопку «Преобразовать в федерат HLA». В результате этого действия будет сгенерирован исходный код федератов на C++ и служебные файлы, необходимые для запуска имитационных экспериментов (процесс генерации описан в разделах 4.3, 9.1).

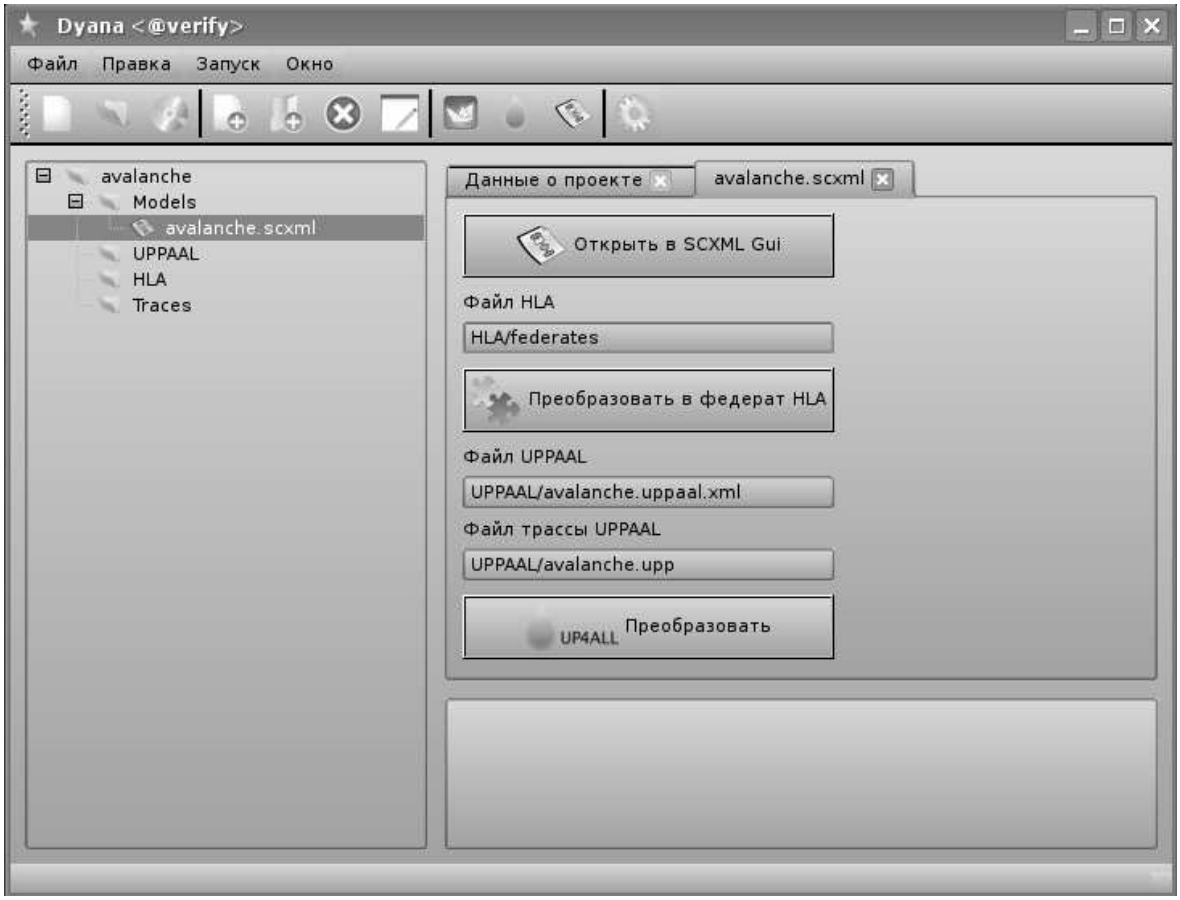

Рисунок 194. Вкладка для scxml-файла

Был сгенерирован исходный код федератов и скрипт для запуска имитационного эксперимента launcher.py. Можно просмотреть содержимое файлов с кодом федератов на C++ (см. рисунок 195).

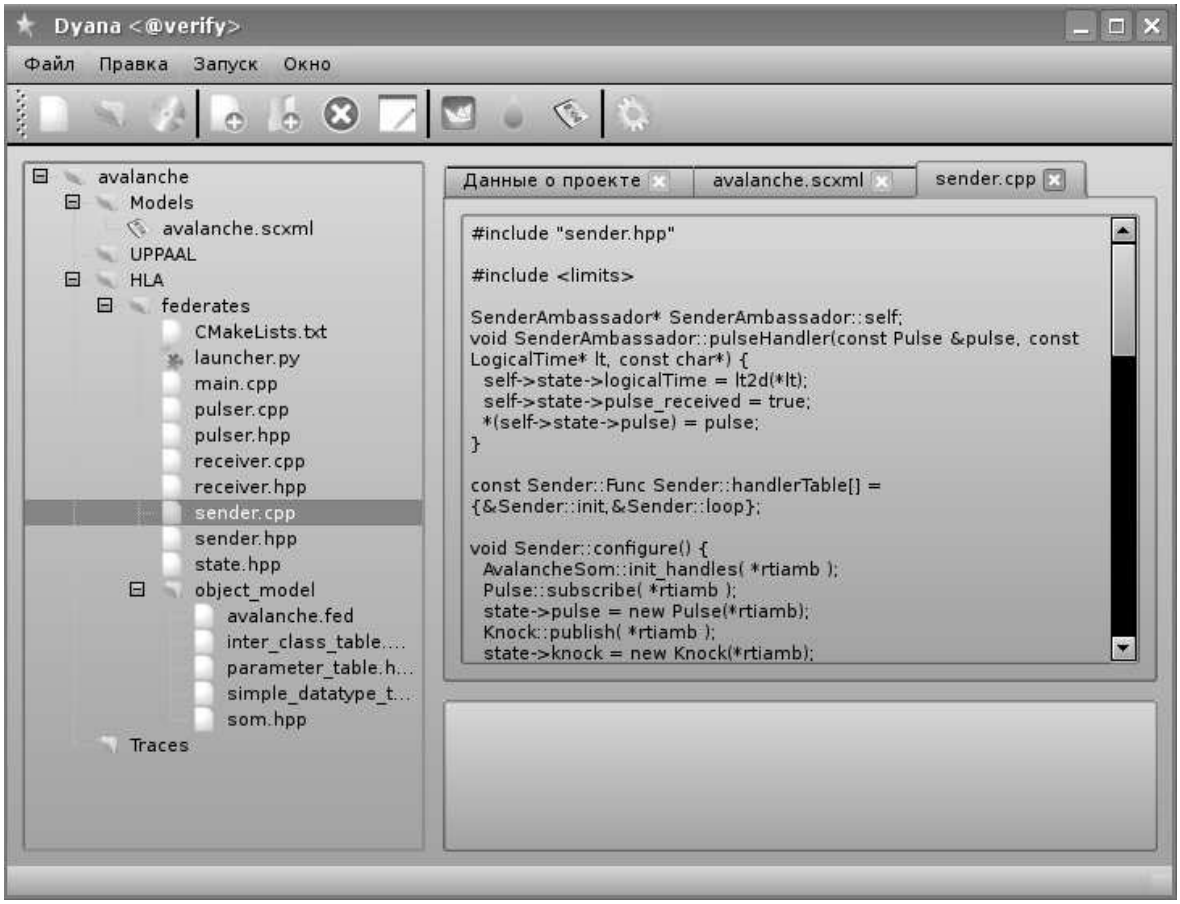

Рисунок 195. Сгенерированный код федерата на C++

Открываем вкладку для скрипта launcher.py (см. рисунок 196). Можно просмотреть его содержимое. Задаем имя файла трассы («Traces/trace»). Запускаем имитационный эксперимент, нажав кнопку «Запустить CERTI». Описание среды выполнения моделей и процесса выполнения имитационного эксперимента приведено в разделах 4.4, 9.2.

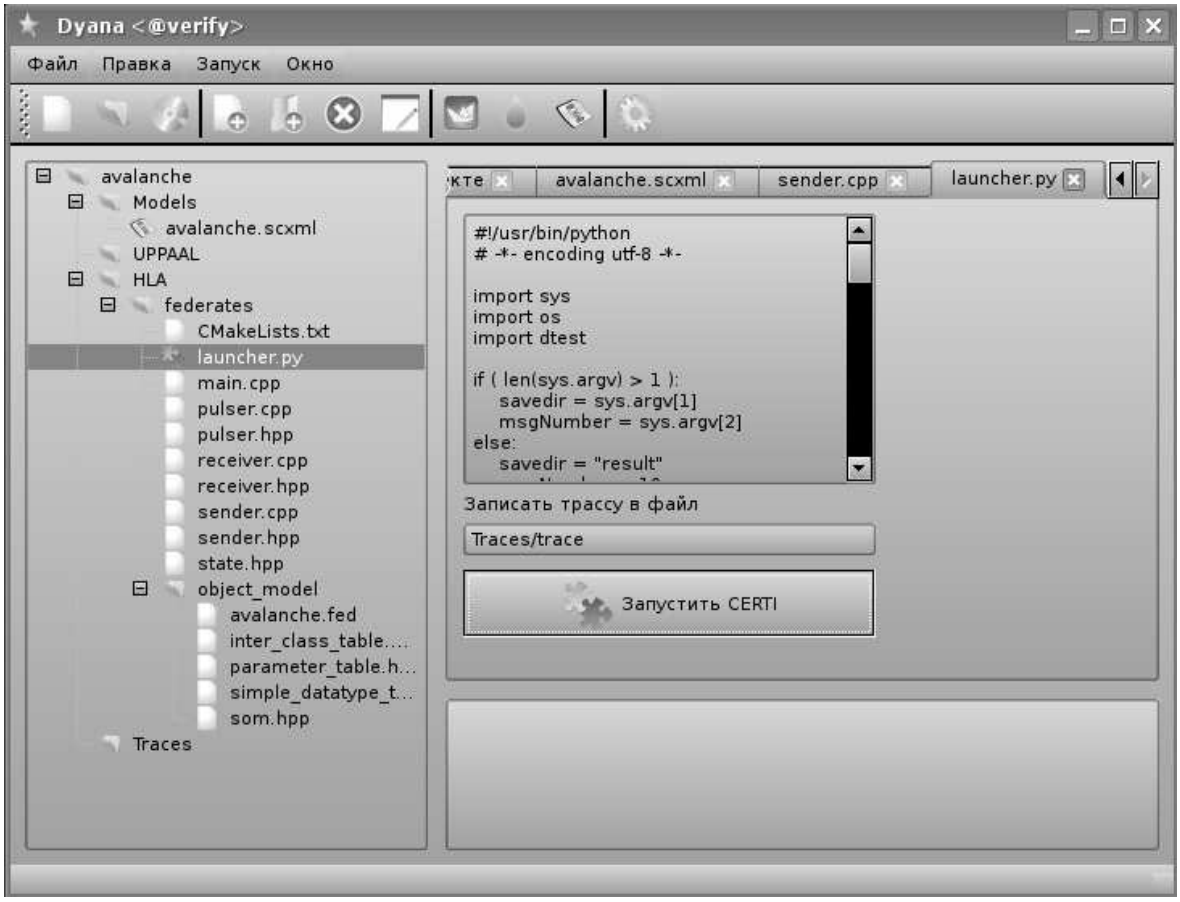

Рисунок 196. Вкладка для скрипта launcher.py

После выполнения эксперимента в каталоге «Traces» появится файл «trace», содержащий трассу эксперимента (процесс создания трассы описан в разделах 4.6, 9.4) и подкаталог «result» с файлами, в которых были записаны потоки stdin, stdout и stderr процессов, участвовавших в экспериментах. Просмотр содержимого этих файлов может быть полезен для отладки моделей (рисунок 197).

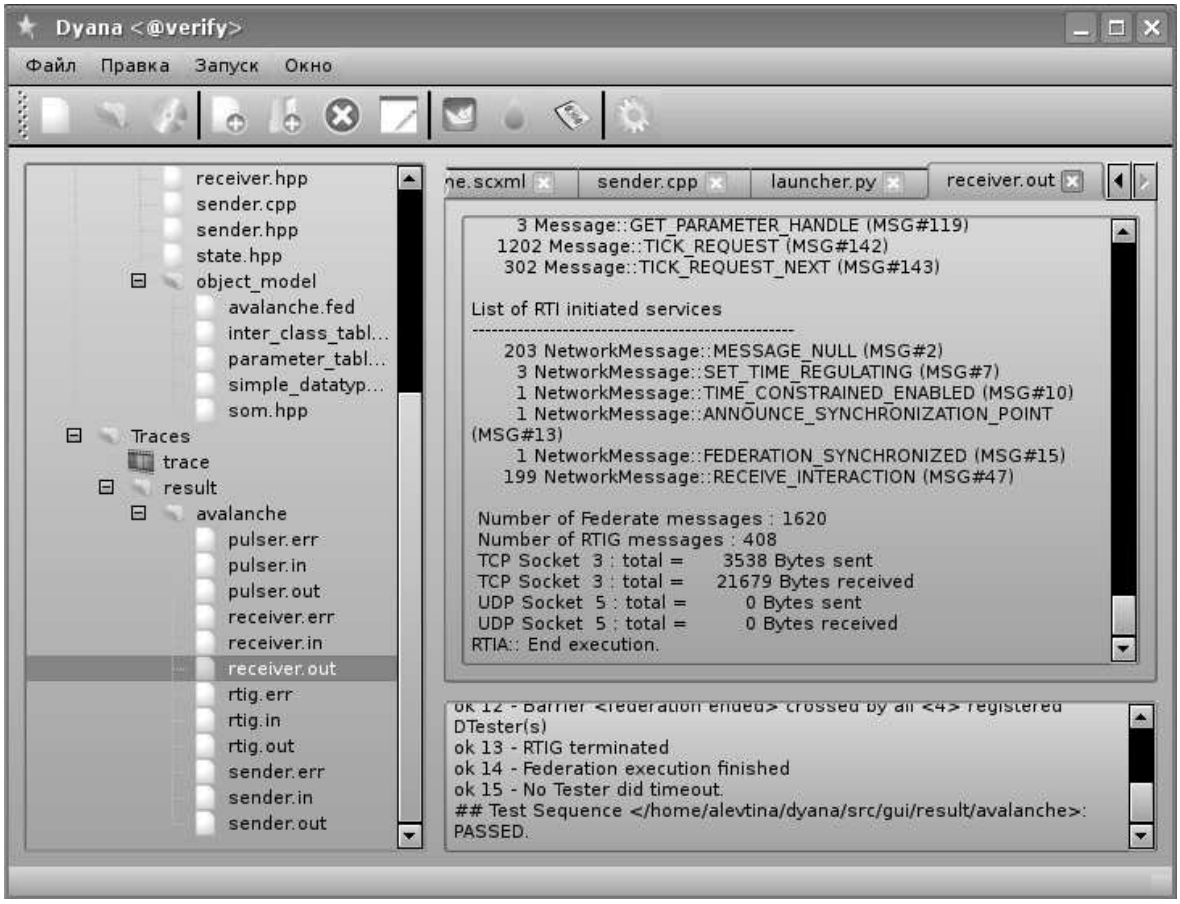

Рисунок 197. Результаты эксперимента

Далее переходим на вкладку для файла с трассой «trace» и нажимаем кнопку «Запустить vis» (рисунок 198).

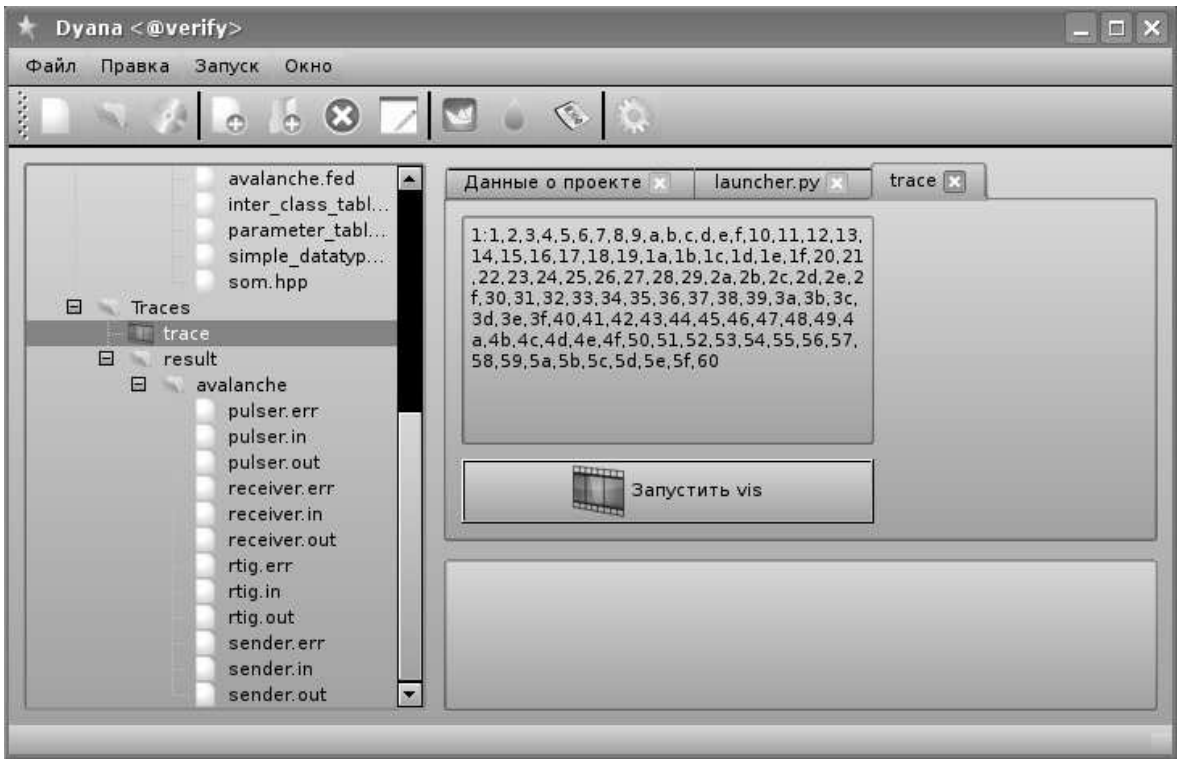

Рисунок 198. Вкладка для файла трассы

В результате откроется окно средства анализа и визуализации трасс «Vis» (описание средства приведено в разделах 4.7, 8.3). Теперь можно просмотреть информацию о событиях модели и информационных обменах между ее компонентами (рисунок 199).

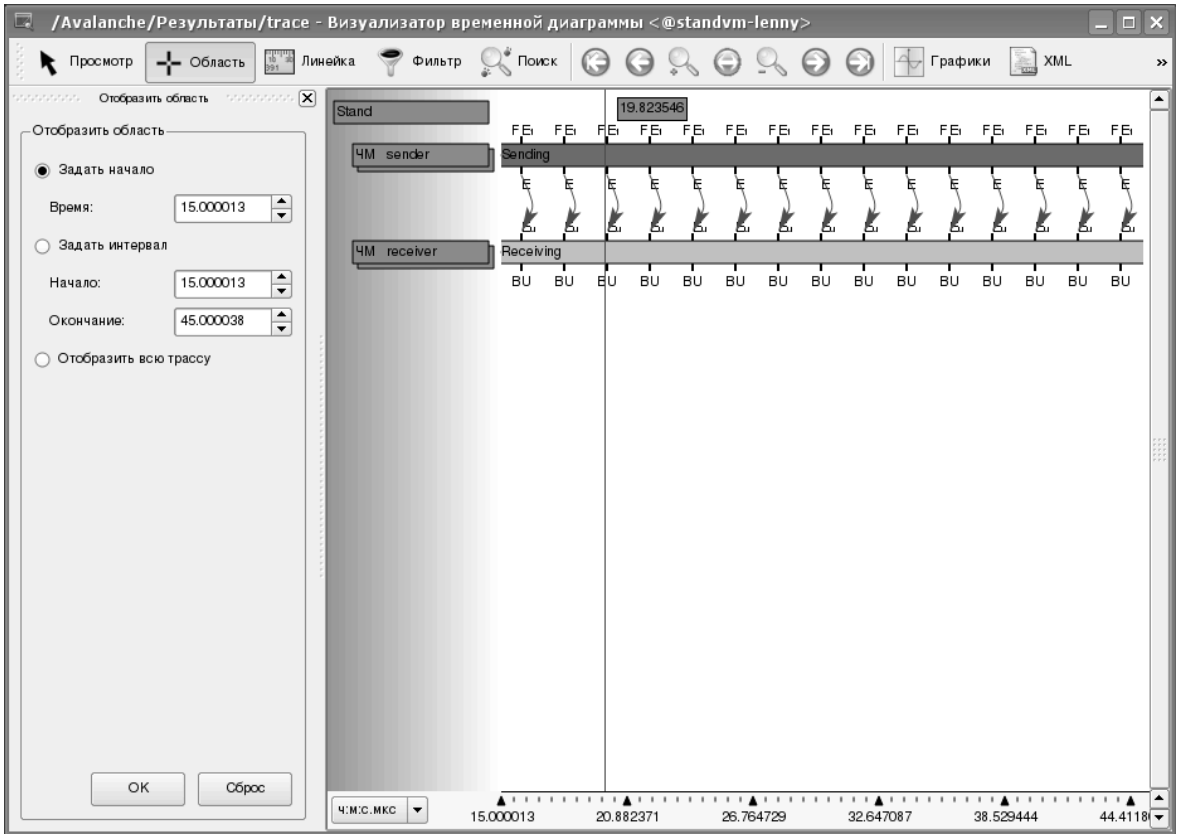

Рисунок 199. Средство анализа и визуализации трасс “Vis”

После анализа трассы нужно оценить, достигнута ли цель моделирования. Если цель не достигнута, вернуться к одному из предыдущих этапов.

В данном разделе на конкретном примере была проиллюстрирована простейшая последовательность работы со средствами, входящими в состав разработанной интегрированной среды моделирования. Подробное описание экспериментов с каждым из разработанных средств в отдельности приведено в разделах 9.1-9.8.

10 Разработка программы внедрения результатов НИР в образовательный процесс

Результаты выполненного в данной НИР исследования, как в исследовательском, так и в методологическом плане, могут быть использованы и уже используются в различных курсах, читаемых на факультете ВМК МГУ имени М.В. Ломоносова и на факультете управления и прикладной математики МФТИ. В таблице 10.1 приведён список таких курсов, количество разработанных материалов и место внедрения курса.

Таблица 45 - Курсы и место внедрения результатов НИР

	Форма внедрения	**Тип внедрения**	**Место внедрения**
1.	курс «Технологии разработки встроенных систем»	Новый курс (60 академических часов)	факультет ВМК МГУ имени М.В. Ломоносова
2.	курс «Моделирование и анализ функционирования вычислительных систем»	Новый курс (материалы для части лекций и практических занятий, 12 академических часов)	факультет ВМК МГУ имени М.В. Ломоносова
3.	курс «Методы проектирования и анализа ПО»	Новый курс (материалы для части лекций и практических занятий, 12 академических часов)	факультет ВМК МГУ имени М.В. Ломоносова
4.	курс «Математические методы спецификации и верификации ПО»	Дополнение в существующий курс в виде новых лекций и практических занятий (8 академических часов): 1. Лекция «Алгоритмы проверки отношений подобия на моделях распределенных систем» 2. Лекция «Методы верификации параметризованных моделей распределенных программ» 3. Практическое занятие: «Применение систем верификации моделей программ для проверки распределенных программ с неограниченным числом процессов»	факультет ВМК МГУ имени М.В. Ломоносова
5.	курс «Математическая логика и логическое программирование»	Изменение в существующем курсе лекций (2 академических часа): Лекция «Темпоральные логики и их применение в проектировании распределенных вычислительных систем»	факультет ВМК МГУ имени М.В. Ломоносова

Продолжение таблицы 45

	Форма внедрения	Тип внедрения	Место внедрения
6.	курс «Алгоритмы планирования вычислений в системах реального времени»	Новый курс (30 академических часов)	факультет ВМК МГУ имени М.В. Ломоносова
7.	курс «Прикладные логики»	Дополнение в существующий курс в виде новых лекций и практических занятий (2 академических часа): Лекция «Неклассические логики и их применение в информатике»	МФТИ, факультет управления и прикладной математики
8.	курс «Распределенные алгоритмы»	Дополнение в существующий курс в виде новых лекций и практических занятий (2 академических часа): Лекция «Методы верификации распределенных алгоритмов»	МФТИ, факультет управления и прикладной математики
9.	курс «Надёжность программного обеспечения»	Новый курс (20 академических часов)	факультет ВМК МГУ имени М.В. Ломоносова

Внедрение результатов НИР осуществлялось на основе разработанных на этапах 3-5 данной работы учебно-методических материалов. Авторами проекта также были изданы учебники «Компьютерные сети» и «Архитектура ЭВМ и операционные среды». Эти учебники разработаны в соответствии с федеральными государственными образовательными стандартами по направлениям подготовки "Прикладная математика и информатика", "фундаментальная наука компьютерные и информационные технологии" (квалификация "бакалавр"). Результаты НИР могут быть внедрены образовательными учреждениями в субъектах РФ на основе предложенных программ и учебно-методических материалов, размещённых на сайте Дирекции научно-технических программ http://www.sstp.ru.

11 Подготовка научно-методических материалов для учебных материалов по тематике проекта объёмом 28 академических часов

По курсам «Технологии разработки встроенных систем», «Моделирование и анализ функционирования вычислительных систем», «Методы проектирования и анализа ПО» и «Математические методы спецификации и верификации ПО» и «Алгоритмы планирования вычислений» в системах реального времени» были подготовлены учебные материалы, представленные в приложении А.

По курсу «Технологии разработки встроенных систем» были разработаны материалы в виде слайдов для презентации объёмом 2 академических часа. Подготовлены материалы для следующей лекции:

- средства разработки: средства поддержки процесса верификации и тестирования ПО. Средства внесения неисправностей.

По курсу «Моделирование и анализ функционирования вычислительных систем» были разработаны материалы в виде слайдов для презентации объёмом 12 академических часа. Подготовлены материалы для следующих лекций:

- имитационное моделирование и примеры средств имитационного моделирования;
- управление временем в средствах имитационного моделирования;
- стандарт систем распределённого моделирования High Level Architecture;
- средства анализа и визуализации трасс функционирования РВС РВ;
- методы решения задачи выбора механизмов обеспечения отказоустойчивости РВС РВ;
- эксперименты с UPPAAL.

По курсу «Методы проектирования и анализа ПО» были разработаны материалы в виде слайдов для презентации объёмом 12 академических часа. Подготовлены материалы для следующих лекций:

- язык моделирования и анализа UML;
- плоские временные автоматы;
- средство верификации UPPAAL;
- иерархические временные автоматы;
- методы обеспечения качества проектируемого ПО;
- интегрированная среда разработки моделей ПО.

По курсу «Математические методы спецификации и верификации ПО» были разработаны материалы в виде слайдов для презентации объёмом 2 академических часа. Была разработана вторая часть лекции «Алгоритмы проверки отношений подобия на моделях распределенных систем».

Заключение

В результате выполнения работ по пятому этапу НИР были выполнены следующие работы.

В соответствии с п. 5.1 календарного плана и требованиями 4.1.7 ТЗ разработана методика совместного применения созданных методов и инструментальных средств для поддержки разработки и интеграции РВС РВ. В рамках данного направления была предложена общая методика разработки моделей РВС РВ в среде моделирования, а также разработаны:

- методика использования редактора UML-диаграмм;
- методика использования средства визуализации трассы;
- методика использования средства трансляции UML во временные автоматы;
- методика использования средства верификации модели, включающая использование функции восстановления параметров модели по контрпримеру, конструируемому в результате верификации, и оценку времени выполнения фрагментов программ;
- методика использования средств моделирования совместно со средствами синтеза архитектур и построения расписаний.

В соответствии с п. 5.2 календарного плана и требованиями 4.1.5 ТЗ разработан и реализован экспериментальный образец стенда полунатурного моделирования и интеграции РВС РВ. В рамках данного направления была разработана схема организации полунатурного моделирования в рамках стенда и предложена общая схема стенда моделирования.

В соответствии с п. 5.3 календарного плана и требованиями 4.1.1, 4.1.2, 4.1.6 и 4.1.11 ТЗ проведена апробация интегрированной среды на стенде. В рамках данной апробации было проведено экспериментальное исследование средств, разработанных в рамках данной НИР. Средства исследовались как на простых моделях, таки и на моделях реальных РВС РВ. Эксперименты показали успешную работу разработанных средств.

В соответствии с п. 5.4 календарного плана и требованиями 4.1.10 ТЗ разработана программа внедрения результатов НИР в образовательный процесс. Были разработаны научно-методические материалы для целого ряда курсов, читаемых на факультете ВМК МГУ имени М.В. Ломоносова и на факультете управления и прикладной математики МФТИ.

В соответствии с п. 5.5 календарного плана и требованиями 4.1.8, 4.1.9 и 4.1.11 ТЗ разработаны научно-методические материалы для учебных материалов по курсам «Технологии разработки встроенных систем», «Моделирование и анализ функционирования вычислительных систем», «Методы проектирования и анализа ПО» и «Математические методы спецификации и верификации ПО» и «Алгоритмы планирования вычислений».

В результате выполнения исследований по данной НИР были получены следующие результаты:

1. Выделен класс РВС РВ, для проектирования которых создавалась интегрированная среда моделирования (см. главу 1).
2. Сформулированы требования к создаваемым методам и средствам (см. главу 2).
3. Выбран стандарт HLA для представления моделей на уровне среды выполнения моделей (см. разделы 3.1 – 3.2).
4. Разработано строгое описание модели, основанное на диаграммах состояний UML, удобное для проектирования РВС РВ. (см. разделы 3.3 – 3.6).
5. Выбран открытый формат хранения трасс OTF (см. раздел 3.7).
6. Разработана и реализована среда моделирования РВС РВ (см. разделы 4.1 и 4.11). Это среда включает в себя существовавшие в начале работ средства, которые используются без изменения, а именно:
   - редактор UML-диаграмм ArgoUML (см. раздел 4.2),
   - средство верификации модели UPPAAL (см. раздел 4.9) и
   - средство оценки наихудшего времени выполнения METAMOC (см. раздел 4.10).

   В среде выполнения моделей CERTI был исправлен ряд ошибок и разработана многонитевая версия данной среды выполнения (см. раздел 4.4). На базе средства визуализации трассы Vis3 было создано средство визуализации трассы Vis4, позволяющее работать с трассами в формате OTF. В рамках данной НИР были разработаны:
   - средство трансляции UML в исполняемые модели совместимые со стандартом HLA (см. раздел 4.3),
   - средство внесения неисправностей (см. раздел 4.5),
   - средство трассировки моделей (см. раздел 4.6),
   - средство трансляции UML во временные автоматы (см. раздел 4.8) и
   - интегрированная среда разработки и анализа моделей CERTI.
7. Разработан алгоритм трансляции диаграмм состояний UML в сети временных автоматов, которые используются средством верификации UPPAAL в качестве входной модели. Важной особенностью алгоритма является использование промежуточной модели (иерархических временных автоматов), которая обеспечивает математическое описание операционной семантики диаграмм состояний UML и позволяет сформулировать требования корректности трансляции (см. раздел 5.1).
8. Строго математически обоснована корректность разработанного алгоритма трансляции (см. раздел 5.2).

9. Предложен алгоритм минимизации системы переходов для сетей временных автоматов (см. раздел 5.3).
10. Разработан алгоритм восстановления параметров модели по контрпримеру, который конструируется в результате ее верификации (см. раздел 5.4).
11. Разработан и реализован метод, позволяющий применять средства верификации для оценки наихудшего времени выполнения программ (см. разделы 5.5 и 5.6).
12. Разработаны и реализованы 4 вида эволюционных алгоритмов и алгоритм имитации отжига для решения задачи механизмов обеспечения отказоустойчивости РВС РВ. Выполнена интеграция с этими методами (см. разделы 5.7 и 5.8).
13. Разработан ряд моделей, как простых, так и сложных (приближенных по размеру и структуре к реальным моделям РВС РВ), иллюстрирующих возможности и эффективность разработанной нами системы (см. главу 6).
14. Создан экспериментальный образец стенда полунатурного моделирования и интеграции РВС РВ (см. главу 7).
15. Разработана методика совместного применения созданных методов и инструментальных средств для поддержки разработки и интеграции РВС РВ (см. главу 8).
16. Выполнена апробация интегрированной среды на стенде и экспериментальное исследование, входящих в состав стенда средств, а также сравнение с существующими аналогами (см. главу 9).
17. Разработана программа внедрения результатов НИР в образовательный процесс (см. главу 10).
18. Подготовлены научно-методические материалы для учебных материалов по тематике проекта (см. главу 11).

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

1. Модификация описания модели в целях повышения удобства проектирования РВС РВ, в частности, расширение синтаксиса модели, позволяющее обрабатывать структуры данных, сокращающие запись модели.
2. Оптимизация алгоритма трансляции диаграмм UML в сети временных автоматов, призванная преодолеть эффект «комбинаторного взрыва» в пространстве состояний автоматов. Есть основания полагать, что, децентрализованная стратегия деинициализации временных автоматов, моделирующих вложенные состояния диаграмм UML, позволит получить

уменьшение (экспоненциальное по порядку) роста числа вершин в сетях временных автоматов

3. Добавление возможности обработки форматов записи модели, отличных от формата средства ArgoUML. На первых этапах выполнения проекта выбор редактора диаграмм состояний UML ArgoUML был наиболее предпочтительным; однако за прошедшее время существенно усовершенствовались функциональные возможности более удобного средства редактирования Topcased.
4. Разработка каскадной среды выполнения дискретно-событийных имитационных моделей, настраиваемой на тип моделируемых объектов.
5. Разработка и реализация гибридного консервативно-оптимистического алгоритма синхронизации времени.
6. Разработка статических и динамических анализаторов модели с целью построения оптимальной инфраструктуры среды выполнения.
7. Построение оптимального распределения внутренних компонентов среды выполнения по инструментальным машинам стенда.
8. Доработки внутренней реализации CERTI, включающие добавление 3-го потока для использования пассивного ожидания и изменение внутреннего протокола передачи сообщений между компонентами CERTI.
9. Разработка методики сравнения сред моделирования, а также пакета тестирования для оценки производительности HLA-систем и системы CERTI.
10. Разработка средства автоматического кодирования внутренних данных имитационной модели в форматы, требуемые для передачи сообщения через необходимые каналы передачи данных (натурные каналы, такие как MILSTD1553B) и среду выполнения (стандарт HLA).
11. Доработки обеспечения работы среды выполнения в реальном времени, в частности построение системы на базе стандарта DDS и исследование возможностей существующего стандарта (HLA Evolved).
12. Исследование применимости нового формата OTF2 для трассировки результатов моделирования РВС РВ, сравнение его с OTF. Возможность гибкой настройки формата и его выбора в зависимости от продолжительности и требований к результатам эксперимента.
13. Автоматизация обработки результатов моделирования.
14. Развитие новых методов визуализации данных результатов моделирования РВС РВ.

15. Развитие новых методов оперативной визуализации процесса выполнения модели РВС РВ.
16. Оценка наихудшего времени выполнения распределённой программы в РВС РВ.
17. Разработка и реализация метода трансляции из трассы UML в OTF в случае нахождения контрпримера системой верификации.
18. Доработка интегрированной среды разработки в части добавления редактора текстовых файлов, транслятора свойств из UML в UPPAAL, автообновления списков во вкладке UPPAAL, консольные версии используемых в среде моделирования трансляторов.
19. Рефакторинг парсера XMI.
20. Разработка и исследование других алгоритмов решения задачи выбора механизмов обеспечения отказоустойчивости, например, модификаций жадного алгоритма.
21. Разработка и реализация алгоритмов приближенного вычисления времени выполнения программных компонентов для задачи выбора механизмов обеспечения отказоустойчивости РВС РВ, например использование метамоделей.
22. Развитие схемы интеграции среды моделирования и средств синтеза архитектур и планирования расписаний в части интеграция со средствами оценки наихудшего времени выполнения программы и усложнения модели РВС РВ.

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. Отчёт о научно-исследовательской работе «Создание прототипа интегрированной среды и методов комплексного анализа функционирования распределённых вычислительных систем реального времени (РВС РВ)» (Этап 2) // М.:, 2011. - Стр. 189
3. Отчёт о научно-исследовательской работе «Создание прототипа интегрированной среды и методов комплексного анализа функционирования распределённых вычислительных систем реального времени (РВС РВ)» (Этап 3) // М.:, 2011. - Стр. 162
4. Отчёт о научно-исследовательской работе «Создание прототипа интегрированной среды и методов комплексного анализа функционирования распределённых вычислительных систем реального времени (РВС РВ)» (Этап 4) // М.:, 2012. - Стр. 183
5. 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.
6. Stankovic J. A. Real-time Computing. // Byte Magazine. 1992. 17. N 8. P. 155-160.
7. Павлов А.М. Принципы организации бортовых вычислительных систем перспективных летательных аппаратов // МКА, №4, 2001 г.
8. 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
9. Авиационные системы радиоуправления. В 3 т. Т.2. Радиоэлектронные системы самонаведения. Изд.2-е, перераб. и доп. / под ред. А.И. Канащенкова и В.И. Меркулова. М.: Радиотехника, 2003. 390 с.
10. Архитектура вычислительных систем для интегрированной модульной авионики перспективных летательных аппаратов / А.А. Турчак [и др.] // Радиотехника. 2001. № 8. С. 87 – 95.
11. Принципы построения бортовых информационно-управляющих систем высокоточного оружия нового поколения / Г.В. Анцев [и др.] // Радиотехника. 2001. № 8. С. 81 – 86.

12. Л.И. Пономарев, Ю.Г. Нестеров, В.М. Адодин, В.В. Мухин, Н.А. Лукин, Н.А. Дядьков Концепция построения вычислительных систем бортовых радиолокационных средств малоразмерных беспилотных летательных аппаратов // Вестник УГТУ–УПИ. Сер. радиотехн., Теория и практика радиолокации В38 земной поверхности. 2005. № 19 (71). 235 с.
13. Бобровский С. Опыт создания бортового ПО для истребителя F-22 // Корпоративные системы, №35, 2000.
14. Балашов В.В., Бахмуров А.Г., Волканов Д.Ю.,Смелянский Р.Л.,Чистолинов 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.
15. Neumann P. G. Atlantis Launch Delay. // The Risks Digest. 1989. 9. N 32. P. 2-3 [HTML] (http://catless.ncl.ac.uk/Risks/9.32.html#subj5.1).
16. Leveson N. Endeavor bug -- more details. // The Risks Digest. 1992. 13. N 57. P. 3-4 [HTML] (http://catless.ncl.ac.uk/Risks/13.57.html#subj4.1).
17. Neumann P. G. Computer Related Risks. Reading, Massachusetts, USA: Addison-Wesley, 1995. 384 p.
18. Neumann P. G. Cause of AT&T network failure. // The Risks Digest. 1990. 9. N 62. P. 2-3 [HTML] (http://catless.ncl.ac.uk/Risks/9.62.html#subj2.1).
19. Arnold D. N. Computer Arithmetic Tragedies. Patriot Missile Failure. [HTML] (http://www.ima.umn.edu/~arnold/455.f96/disasters.html).
20. Denning P. J. Computers Under Attack: Intruders, Worms, and Viruses. New York, New York, USA: ACM Press, 1990. 592 p.
21. Замятина, Е. Б.. Современные теории имитационного моделирования: специальный курс. ПГУ, Пермь, 2007.
22. Савенков К.О., Смелянский Р.Л., Масштабирование дискретно-событийных имитационных моделей // Программирование, 2006, No. 6, стр. 14-26.
23. 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).
24. Баранов А.С., Грибов Д.И., Поляков В.Б., Смелянский Р.Л., Чистолинов М.В. Комплексный стенд математического моделирования КБО ЛА // Труды

Всероссийской научной конференции "Методы и средства обработки информации" (1 октября - 3 октября 2003 г., г. Москва) -М.: Издательский отдел факультета ВМиК МГУ, 2003. - С. 282-295.

25. Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules // IEEE, 2010 c. 26.
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. Simulation Interoperability Standards Committee of the IEEE Computer Society IEEE Standard for Distributed Interactive Simulation - Application Protocols, IEEE Std 1278.1a-1998.
30. 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.
31. Modeling and Simulation (M&S) High Level Architecture (HLA) — Object Model Template (OMT) Specification // IEEE, 2000, c 260.
32. С.Г. Басиладзе, Р.Л. Смелянский, А.И. Караваев, М.В. Емельянов, В.Ф. Косов, О.З. Элоев. Экспериментальная многопроцессорная система "СТЕНД" // Модули и программное обеспечение систем автоматизации экспериментальных исследований: Учебно-методическое пособие/Под ред. С.Г. Басиладзе. – М: Изд-во Моск. ун-та. – 1990. – С. 120-129.
33. Чистолинов М.В., Бахмуров А.Г. Среда моделирования многопроцессорных вычислительных систем // "Программные системы и инструменты": Тематический сборник факультета ВМиК МГУ им. Ломоносова N1/Под ред. Л.Н.Королева - М.: МАКС Пресс, 2000, с.42-47
34. Riddle W.E. An approach to software system behavior description // Computer Languges, 1979. V.4, P.29-47.
35. Riddle W.E. An approach to software system modelling and analysis // Computer Languges, 1979. V.4, P.49-66.

36. Питерсон Дж. Теория сетей Петри и моделирование систем // Москва, Мир, 1984.
37. Молонов В.Г., Смелянский Р.Л. Комплексный подход к моделированию распределенных вычислительных систем // Программирование N.1, 1988 С.57-67.
38. Simulink Tutorial [PDF] (http://academic.csuohio.edu/dong_l/EEC510/material/Simulink%20Tutorial.pdf) (дата обращения: 13.07.2011)
39. OMG Unified Modeling Language Specification [PDF] (http://www.omg.org/spec/UML/1.4/PDF) (дата обращения: 13.07.2011)
40. Гома Х. UML. Проектирование систем реального времени, распределенных и параллельных приложений. Издательство ДМК, 2011 год, 704 стр.
41. OMG Unified Modeling Language, Infrastructure [PDF] (http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF) (дата обращения: 05.10.2012)
42. OMG Unified Modeling Language, Superstructure [PDF] (http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF) (дата обращения: 05.10.2012)
43. D. Harel. Statecharts: A Visual Formalism for Complex Systems. Sci. Comput. Programming 8 (1987), pp. 231-274.
44. Behaviour modelling with Stateflow/Simulink Tutorial [PDF] (http://www.mathworks.com/help/pdf_doc/simulink/sl_gs.pdf) (дата обращения: 05.10.2012).
45. A. David, M.O. Moller. From HUppaal to Uppaal: a translation from hierarchical timed automata to flat timed automata // Research Series RS-01-11, BRICS, Department of Computer Science, University of Aarhus, March 2001.
46. R. Alur, D.L. Dill. A theory of timed automata // Theoretical Computer Science. — 1994. — v. 126. — p. 183-235.
47. R. Alur, D.L. Dill. Automata-theoretic verification of real-time systems // Formal Methods for Real-Time Computing, Trends in Software Series, John Wiley & Sons Publishers. — 1996. — p. 55-82.
48. Э.М. Кларк, О. Грамберг, Д. Пелед. Верификация моделей программ: Model Checking. — Изд-во МЦНМО, 2002. — 416 с.
49. R. Alur, P. Madhusudan. Decision Problems for Timed Automata. A Survey // Lecture Notes in Computer Science. — 2004. — v. 3185. — p. 1-24.
50. G. Behrmann, A. David, K.G. Larsen. A Tutorial on Uppaal // Lecture Notes in Computer Science. 2004. v. 3185. p. 200-236.
51. Stephan Seidl. VTF3 – A Fast Vampir Trace File Low-Level Management Library. Dresden University of Technology. Center for High-Performance Computing, November 17, 2003.

52. Andreas Knüpfer. Advanced Memory Data Structures for Scalable Event Trace Analysis. Dissertation. March, 2009.
53. Andreas Knüpfer, Holger Brunst, Allen D.Malony, Sameer S. Shende. Open Trace Format API Specification. Version 1.1. November 13, 2006.
54. Andreas Knüpfer, Holger Brunst, Ronny Brendel. Open Trace Format Specification. April 22, 2009.
55. B. de Oliveira Stein, J. Chassin de Kergommeaux. Paje trace file format. February 24, 2010.
56. Felix Wolf, Bernd Mohr, Nikhil Bhatia, Marc-Andre Hermanns, Markus Geime. EPILOG – Binary Trace-Data Format. Version 1.4. March 7, 2006.
57. A. Chan, W. Gropp, and E. Lusk. Scalable Log Files for Parallel Program Trace Data – DRAFT. Argonne National Laboratory, Argonne, IL 60439, 2000.
58. Paraver CEP01 from knupfer
59. Проект V-Ray. // http://parallel.ksu.ru/v-ray
60. Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. M¨uller, and Wolfgang E. Nagel. The Vampir Performance Analysis Tool-Set.
61. ViTE, Technical Manual. June 16, 2009.
62. Sameer S. Shende, Allen D. Malony. THE TAU PARALLEL PERFORMANCE SYSTEM. The International Journal of High Performance Computing Applications,Volume 20, No. 2, Summer 2006, pp. 287–311.
63. Sameer Shende, Allen D. Malony, and Alan Morris. Improving the Scalability of Performance Evaluation Tools. Para 2010 – State of the Art in Scientific and Parallel Computing – extended abstract no.110.University of Iceland, Reykjavik, June 6–9, 2010.
64. Intel® Trace Analyzer User's Reference Guide.
65. Bernd Mohr and Felix Wolf. KOJAK – A Tool Set for Automatic Performance Analysis of Parallel Programs.
66. Anthony Chan, David Ashton, Rusty Lusk, William Gropp. Jumpshot-4 Users Guide. July 11, 2007.
67. Среда моделирования ДИАНА. Визуализатор временной диаграммы. Руководство пользователя. 2009.
68. Andreas Knupfer, Christian Rossel, Dieter an Mey and other. DRAFT: Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. Dresden tools workshop September, 2011.
69. Dominic Eschweiler, Michael Wagner The Open Trace Format 2 (Version 1.0 beta 1) Format and Library Specification, August, 2011.

70. OPEN TRACE FORMAT 2. USER MANUAL. Version 1.1. October 9, 2012.
71. Papyrus Homepage [HTTP] (http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?L=EN&P=55&vTicker=alleza&ITEMID=3)
72. Moskitt Homepage [HTTP] (http://www.prodevelop.es/en/tech/eclipse)
73. VioletUML Homepage [HTTP] (http://alexdp.free.fr/violetumleditor/page.php)
74. TinyUML Homepage [HTTP] (http://www.tinyuml.org/)
75. ArgoUML Homepage [HTTP] (http://argouml.tigris.org/)
76. Topcased Homepage [HTTP] (http://www.topcased.org/)
77. BOUML Homepage [HTTP] (http://bouml.free.fr/)
78. Robert C. Martin , UML Tutorial: Finite State Machines // Engineering Notebook Column C++ Report, June 1998
79. State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Working Draft 26 April 2011 [HTML] (http://www.w3.org/TR/scxml/)
80. Cheetah Users' Guide [PDF] (http://www.cheetahtemplate.org/docs/users_guide.pdf)
81. Антоненко В.А., Волканов Д.Ю., Чистолинов М.В. Средство генерации имитационной модели, совместимой со стандартом HLA. — Санкт-Петербург, 2011. — т.1, стр. 331-335.
82. Simulation Interoperability Standards Committee of the IEEE Computer Society IEEE Standard for Distributed Interactive Simulation - Application Protocols, IEEE Std 1278.1a-1998.
83. 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.
84. Chaudron, Jean-Baptiste and Saussié, David and Siron, Pierre and Adelantado, Martin Real-time aircraft simulation using HLA standard. (In Press: 2011) In: IEEE AESS Simulation in Aerospace 2011, 8 Jun 2011, Toulouse, France.
85. Adelantado, Martin and Siron, Pierre and Chaudron, Jean-Baptiste Towards an HLA Run-time Infrastructure with Hard Real-time Capabilities. (2010) In: International Simulation Multi-Conference (ISMC'10), 12-14 July 2010, Ottawa, Canada.
86. NCWare - A Real Time HLA Run Time Infrastructure // [HTML] (http://www.nexteleng.es/microsite/ncware/products/ncware.asp), 2012.
87. MAK RTI // [HTML] (http://www.mak.com/), 2012.
88. Möller B., Karlsson M. Making RTI Tuning Easy with Performance Profiles // Proceedings of 2005 Euro Simulation Interoperability Workshop, Simulation Interoperability Standards Organization, June 2005.

89. RTI NG Pro // [HTML] (http://www.raytheon.com/capabilities/products/rti/), 2012.
90. L. Bononi, M. Bracuto, D'Angelo G., and Donatiello L., "A new adaptive middleware for parallel and distributed Simulation of dynamically interacting systems," in Distributed Simulation and Real-Time Applications, 2004, pp. 178 - 187.
91. I. Birrer, B. Carnicero-Dominguez, M. Egli, B. Carnicero-Dominguez, and A. Pasetti, "EODiSP – an open and distributed simulation platform," in International Workshop on Simulation for European Space Programmes, Noordwijk, the Netherlands, 2006.
92. Portico RTI // [HTML] (http://www.porticoproject.org/), 2012.
93. B. d'Ausbourg, P. Siron, and E. Noulard, "Running real time distributed simulations under Linux and CERTI," in European Simulation Interoperability Workshop, Edimburgh, Scotland, 2008.
94. Karlsson M., Karlsson P. An In-Depth Look at RTI Process Models // In Proceedings of 2003 Spring Simulation Interoperability Workshop. — Stockholm, Sweden, 2003.
95. Noulard E., Rousselot J.-Y., CERTI, an Open Source RTI, why and how // Spring Simulation Interoperability Workshop. San Diego, USA, 2009.
96. 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.
97. Möller B., Karlsson M. Making RTI Tuning Easy with Performance Profiles // In Proceedings of 2005 Spring Simulation Interoperability Workshop, Toulouse, France, 2005.
98. P. Bieber, D. Raujol, P. Siron. Security Architecture for Federated Cooperative Information Systems. Toulouse, France, 2002.
99. Williams A. The Boost Thread Library // [HTM] (http://www.boost.org/doc/libs/1_47_1/libs/boost_thread/boost_thread.htm), 2011.
100. Patrick P. C. Lee, Tian Bu, Girish Chandranmenon A lock-free, cache-efficient shared ring buffer for multi-core architectures // In proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'09), Princeton, USA, October 19-20 2009, pp. 78-79.
101. B. Möller, M. Karlsson, B. Löfstrand Reducing Integration Time and Risk with the HLA Evolved Encoding Helpers // In Proceedings of Spring Simulation Interoperability Workshop, 2006.
102. J. Graham Creating an HLA 1516 Data Encoding Library using C++ Template Metaprogramming Techniques // In Proceedings of Spring Simulation Interoperability Workshop, 2007.

103. L. Malinga and WH. Le Roux, HLA RTI Performance Evaluation // European Simulation Interoperability Workshop, Istanbul, Turkey, 2009, pp. 1-6.

104. Волканов Д.Ю., Шаров А.А. Программное средство автоматического внесения неисправностей для оценки надежности вычислительных систем реального времени с использованием имитационного моделирования // 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.457-464.

105. Benso & P. Prinetto, Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, 2004

106. Smart, Julian; Hock, Kevin; Csomor, Stefan. Cross-Platform GUI Programming with wxWidgets, 5 AuguWst 2005, Prentice Hall, pp.744

107. Norman Wilde, Sharon Simmons, Dennis Edwards and L. Pounds. But Where Does It DO That? Locating Features in a Distributed Simulation. The 2002 Fall Simulation Interoperability Workshop, Paper number 02F-SIW-088, 2002.

108. Mr. Jerry Black. Data Collection in an HLA Federation, 1999.

109. Heng-Jie Song, Zhi-Qi Shen, Chun-Yan Miao, Ah-Hwee Tan, Guo-Peng Zhao. The Multi-Agent Data Collection in HLA-based Simulation System. 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS'07), 2007.

110. Пашков В.Н., Волканов Д.Ю. О подходах к трассировке распределенных вычислительных систем реального времени. // Материалы 17-ой международной конференции по вычислительной механике и современным прикладным программным системам (ВМСППС'2011), 25-31 мая 2011 г., Алушта. - М.: Изд-во МАИ-ПРИНТ, 2011.

111. Бахмуров А.Г., Чистолинов М.В. Среда моделирования многопроцессорных вычислительных систем. // Программные системы и инструменты №1. Москва: МАКС Пресс, 2000. С. 42-47.

112. CMake // [HTML] http://www.cmake.org/ , 2012.

113. G. Behrmann, A. David, K.G. Larsen. A Tutorial on Uppaal // Lecture Notes in Computer Science. 2004. v. 3185. p. 200-236.

114. 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.

115. 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.

116. 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.co m, 1999.

117. 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.

118. Torsten K. Iversen, K°are 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.

119. 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.

120. 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.

121. F.W. Vaandrager and A.L. de Groot. Analysis of a biphase mark protocol with UPPAAL and PVS. Technical Report NIII-R0445, Radboud University of Nijmegen, 2004.

122. 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.

123. 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.

124. 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.

125. Thomas A. Henzinger. Symbolic model checking for real-time systems. Information and Computation, 111:193–244, 1994

126. Andreas E. Dalsgaard, Mads Chr. Olesen, Martin Toft, Ren R. Hansen, Kim G. Larsen, METAMOC: modular execution time analysis using model checking, 2010

127. A. David, M.O. Moller, W. Yi. Verification of UML Statechart with Real-time Extensions // Uppsala: Department of Information Technology, Uppsala University. IT Technical Report 2003-009, 2003.

128. M.C. Browne, M.C. Clarke, O. Grumberg. Characterizing finite Kripke structures in propositional temporal logics // Theoretical Computer Science. 1988. v.59 (1-2). p.115-131.

129. E.M. Clarke, O. Grumberg, D. Peled. Model Checking // The MIT Press. 1999.

130. Reinhard Wilhelm, Jakob Engblom The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools // 2008.

131. Reinhold Heckmann, Christian Ferdinand Worst-Case Execution Time Prediction by Static Program Analysis // 2004.

132. Daniel Sandell, Andreas Ermedahl Static Timing Analysis of Real-Time Operating System Code // 2004.

133. Прус В.В. Эффективный алгоритм перебора кратчайших путей в графе //Труды Всероссийской научно-технической конференции “Методы и средства обработки информации” (МСО-2003). М.: Издательский отдел факультета ВМиК МГУ, 2003. С. 474.

134. Alexandre Davide, John Håkansson, Kim G. Larsen, and Paul Pettersson Model Checking Timed Automata with Priorities using DBM Subtraction // 2006

135. Armin Biere, Alessandro Cimatti, Edmund M. Clarke Bounded Model Checking // Advances in Computers. 2003. Vol. 58. P. 118-149.

136. Sungjun Kim, Hiren D. Patel, Stephen A. Edwards Using a Model Checker to Determine Worst-case Execution Time // 2009.

137. Shaw, A. C. Reasoning About Time in Higher-Level Language Software. IEEE Transactionson Software Engineering // 1989.

138. Andreas Engelbredt Dalsgaard, Mads Christian Olesen, Martin Toft, Ren Rydhof Hanseneand Kim, Guldstrand Larsen, WCET Analysis of ARM Processors using Real-Time Model Checking // 2009.

139. ARM9TDMI Technical reference manual // ARM DDI 0180A, ARM Ltd., March 2000. [PDF] (http://infocenter.arm.com/help/topic/com.arm.doc.ddi0180a/DDI0180.pdf) (дата обращения: 30.10.2012)

140. ARM7TDMI Technical Reference manual // ARM DDI 0210C, ARM Ltd., 2001. [PDF] (http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/DDI0210B.pdf) (дата обращения: 30.10.2012)

141. Atmel, AVR 8-bit Instruction Set // Atmel Corporation, 2010. [PDF] (http://www.atmel.com/images/doc0856.pdf) (дата обращения: 30.10.2012)

142. Wattanapongsakorn N. and Levitan S.P. Reliability Optimization Models of Fault-tolerant distributed systems // Reliability and Maintainability Symp. (RAMS), Philadelphia, PA, Jan. 22-25, 2001, pp. 193-199.

143. A.G. Bakhmurov, V.V. Balashov, A.B Glonina, V.N. Pashkov, R.L. Smeliansky, D.Yu. Volkanov. Simulation Modeling Based Method For Choosing An Effective Set Of Fault Tolerance Techniques For Real-Time Avionics Systems // Proc. 4th EUCASS European Conference for Aerospace Sciences, St. Petersburg, Russia, 2011. - Накопитель (Flash).

144. К.Тимофеев, Исследование алгоритма имитации отжига для задачи выбора модулей РВС РВ с учётом требований надёжности // Курсовая работа, Москва, 2012

145. Калашников А.В., Костенко В.А. Параллельный алгоритм имитации отжига для построения многопроцессорных расписаний // Известия РАН. Теория и системы управления. 2008. № 3. С. 101-110

146. Зорин Д.А. Способ представления и преобразования расписаний в итерационных алгоритмах структурного синтеза вычислительных систем реального времени // Программные системы и инструменты. Тематический сборник № 12, М.: Изд-во факультета ВМК МГУ, 2011., С. 1-1

147. State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Working Draft 26 April 2011 [HTML] (http://www.w3.org/TR/SCXML/)

148. Чемерицкий Е.В., Волканов Д.Ю., Смелянский Р.Л. Среда полунатурного моделирования на основе стандарта HLA / Сборник трудов конференции "Моделирование - 2012". Киев, Украина, 2012. С. 454 - 457.

149. Г.С. Осипов. Методы искусственного интеллекта // М.: ФИЗМАТЛИТ. – 2011.

150. Смелянский Р.Л. Проблемы разработки и анализа функционирования встроенных систем реального времени // Труды Всероссийской научной конференции "Методы и средства обработки информации" (1 - 3 октября 2003 г., г. Москва) -М.: Издательский отдел факультета ВМиК МГУ, 2003. - С. 57-73.

151. Fujimoto R. M., Perumalla K., Park A., Wu H. Ammar M. H., Riley G.F., Large-scale network simulation How big? How fast? // In Proceedings of the 11th IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS’03), Orlando, USA. 2003.

152. Fujimoto R.D. Parallel and Distributed Simulation Systems. Wiley Interscience. 2000.

153. Казаков Ю.П., Смелянский Р.Л. Об организации распределенного имитационного моделирования // Программирование, 1994, No. 2, стр. 45-64.

154. Chaudron, J.-B., Adelantado M., Noulard E., Siron P. HLA high performance and real-time simulation studies with CERTI. (2011) // In Proceedings of the 25th European Simulation and Modelling Conference (ESM'2011), 24-26 Oct 2011, Guimaraes, Portugal.

155. Stagiaire Onera DTIM What is DTest? // [PDF] (http://nongnu.askapache.com/tsp/dtest/what_is_dtest.pdf), 2008

156. 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.