Министерство образования и науки
Харьковский национальный университет радиоэлектроники

На правах рукописи

ГРИДЕЛЬ РОСТИСЛАВ НИКОЛАЕВИЧ

УДК 004.942: 004.272.26

МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АНАЛІЗУ РОЗПОДІЛЕНИХ ПРОГРАМНИХ МОДЕЛЕЙ GRID-СИСТЕМ

05.13.06 – информационные технологии

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель
Волк Максим Александрович,
кандидат технических наук, доцент

Харьков – 2015

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ, СОКРАЩЕНИЙ И ТЕРМИНОВ

ЛВС	локальная вычислительная сеть
ИМ	имитационная модель
ОС	операционная система
САПР	система автоматизации проектирования
СУБД	система управления базами данных
УПМ	управляющая программа моделирования
ЭВМ	электронная вычислительная машина
CAD	(Computer Aided Design) – система автоматизированного проектирования
CASE	(Computer Aided Software Engineering) – автоматизированная разработка программного обеспечения
CCS	(Communication concurrent processes) – взаимодействующие параллельные процессы
CSP	(Communicating Sequential Processes) – взаимодействующие последовательные процессы
DAI	(Distributed Artificial Intelligence) – распределенный искусственный интеллект
GRASS	(GRID Advanced Simulation System) – расширенная система моделирования GRID
GVT	(Global Virtual Time) – глобальное моделбное время
FIFO	(First In First Out) – дисциплина «первый пришел, первый обслужен»
HLA	(High Level Architecture) – архитектура высокого уровня
RTI	(Runtime Infrastructure) – инфраструктура реального времени
TSO	(Time Stamp Order) – дисциплина «неубывание временных меток»

ВВЕДЕНИЕ

В настоящее время наблюдается развитие информационных технологий, связанных с распределенной обработкой информации. К наиболее масштабным из них относятся GRID и технологии использования облачных ресурсов. Данная тенденция вызывает повышенный интерес к смежным информационным технологиям, которые используют GRID и облачные ресурсы в качестве инструмента. К таким направлениям относится распределенное имитационное моделирование. Без этапа моделирования практически не обходится сегодня ни одна научная, научно-исследовательская или инженерская работа. А при значительных объемах проектов внимание все больше обращается в сторону именно распределенного моделирования.

Применение распределенного имитационного моделирования в последнее время позволило значительно расширить класс реализуемых имитационных моделей за счет использования большого количества мощных вычислительных ресурсов, к которым можно отнести локальные и глобальные компьютерные сети, суперкомпьютеры, кластера.

За время своего существования, системы имитационного моделирования прошли путь от однопользовательских программ и сред, позволяющих создавать простейшие последовательные модели к крупным распределенным имитационным системам моделирования, объединяющим разнородные удаленные модели, созданные разными разработчиками, в качестве которых могут выступать как отдельные инженеры и ученые, так и целые коллективы и фирмы. Следствием этого стало создание международных стандартов на создание распределенных имитационных моделей, таких как HLA (High Level Architecture).

Значительную роль в формировании и развитии теории имитационного моделирования играют работы отечественных и зарубежных авторов Р. Шеннона [1], И.В. Максимея [2], Н.Н. Моисеева [3], Н.П. Бусленко [4], В.М. Глушкова [5,6], А. Прицкера [7], Д. Гордона [8] и др.

Стремительное развитие распределенных вычислительных систем, обуслов-

ленное появлением мощных вычислительных ресурсов и быстродействующих коммуникационных сетей, приводит к недостаточно эффективному их использованию со стороны имитационных систем моделирования.

Актуальность проблемы. Распределенные имитационные модели чаще всего строятся на основе существующих последовательных методов моделирования, а задача выбора совокупности ресурсов для проведения имитационного эксперимента чаще решается декларативно на основе информации о вычислительной сложности распределенной модели и производительности существующих вычислительных ресурсов. При этом такие важные характеристики как трафик, оперативная память распределенных моделей, потоки локальных задач выделенных ресурсов если и учитываются, то только как ограничивающие параметры при выборе ресурсов, не влияющие на эффективность распределения заданий в распределенной вычислительной среде.

Возникшая ситуация приводит к необходимости научных исследований по широкому кругу вопросов, связанных с теорией организации самого процесса распределенного имитационного моделирования, обусловленных изменением вычислительной среды и переходом от последовательных моделей (программ) к параллельным (или распределенным).

Высокая стоимость эксплуатации распределенных вычислительных систем и объединяющих их коммуникационных сетей требует эффективного прогнозирования поведения распределенной имитационной модели. К факторам, влияющим на стоимость проведения имитации, относятся как параметры вычислительной среды, так и характеристики самой имитационной модели. Последнее обуславливает актуальность задачи анализа распределенных имитационных моделей.

Анализ эффективности процесса распределенного имитационного, проводимый до имитационного эксперимента (а иногда и до создания модели), требует наличия формального представления распределенной имитационной модели и возможности проведения с ним определенных операций, показывающих изменение основных характеристик моделей во времени.

Среди современных работ, которые направлены на создание такого фор-

мального аппарата, можно отнести работы Окольнишникова В.В. [9,10], Вознесенской Т.В. [11,12,13,14], Волка М.А. [15,16,17]. Специфика представления распределенных имитационных моделей в виде параллельно исполняемых процессов определяет частое использование в вопросах анализа процессной алгебры, разработанной Хоаром [18,19,20] и Миллером [21].

Диссертационная работа направлена на создание математических моделей, описывающих процесс распределенного имитационного моделирования, и на разработку, на основе этих моделей, методов анализа процесса распределенного имитационного моделирования. Наличие задач, которые не были решены вышеуказанными авторами, обусловлено отсутствием в их моделях таких важных характеристик, как динамическое изменение объемов памяти распределенных моделей, объемы данных, которыми обмениваются модели в процессе имитации и др. Решение этих задач позволит более эффективно реализовать анализ распределенных моделей, исследовать их поведение под влиянием различных факторов вычислительной среды.

Таким образом, разработка моделей и методов для анализа процессов в распределенных имитационных системах моделирования является актуальным направлением диссертационного исследования.

Связь работы с научными программами, планами, темами. Диссертационная работа выполнялась в соответствии с планом научно-технических работ Харьковского национального университета радиоэлектроники в рамках госбюджетных тем:

- «Разработка структуры харьковского ресурсно-операционного GRID-центра и его ресурсов», договор № 9 между ХНУРЭ и «ИПСА» НТУУ «КПИ», выполняемая на основании Договора «ИПСА» НТУУ «КПИ» с Министерством образования и науки Украины №IT/506-2007, Государственной программы «Информационные и телекоммуникационные технологии в образовании и науке» на 2006-2010 года (№ ГР 0107U010616);

- «Разработка и исследование применения GRID-портала харьковского ресурсно-операционного GRID-центра», договор № 08-22/9 между ХНУРЭ и «ИП-

СА» НТУУ «КПІ», выполняемая на основании Договора «ИПСА» НТУУ «КПИ» с Министерством образования и науки Украины №IT/506-2013, Государственной программы «Информационные и телекоммуникационные технологии в образовании и науке» на 2006-2013 года (№ ГР 0108U008261).

В рамках указанных тем соискателем проведено распределенное имитационное моделирование различных конфигураций и системных настроек харьковского ресурсно-операционного GRID-центра и его ресурсов, а также анализ конфигураций при использовании GRID-портала на различных приложениях.

Результаты научных исследований использованы в научно-технических отчетах о НДР № 9, НДР № 08-22/9.

Цель и задачи исследования. Целью диссертационной работы является развитие технологий имитационного моделирования в распределенных вычислительных системах, разработка моделей, методов и информационной технологии анализа распределенных программных моделей, что позволит уменьшить время моделирования и необходимое для моделирования количество вычислительных ресурсов.

В соответствие с сформулированной целью необходимо решить следующие задачи:

- разработка модели предметной области, отражающей динамическое изменение основных параметров программных элементов системы моделирования;

- разработка модификаций модели для случаев синхронных, консервативных, оптимистических методов синхронизации распределенных имитационных систем моделирования;

- разработка методов анализа процессов распределенного имитационного моделирования на основе созданных моделей для методов распределения вычислительных ресурсов;

- разработка информационной технологии, алгоритмов и программного обеспечения анализа процессов распределенного имитационного моделирования на основе созданных моделей.

Объектом исследования является процесс распределенного имитационного

программного моделирования задач большой размерности.

Предметом исследования являются модели, методы и информационная технология анализа процессов распределенного программного моделирования с разными методами синхронизации частных программных моделей.

Методы исследования. Для проведения исследований предметных областей использована теория имитационного моделирования, теория множеств, общая теория систем, теория имитационного моделирования. Для отражения динамики поведения частных моделей использован формальный аппарат процессной алгебры. В основу представления распределенных программных моделей положен стандарт HLA (High Level Architecture).

Научная новизна полученных результатов. Основные результаты, которые определяют научную новизну диссертационной работы, состоят в следующем:

- впервые предложено имитационную модель процесса распределенного дискретно-событийного моделирования, которая, в отличие от существующих, учитывает динамическое изменение объемов памяти, потоков данных в локальных моделях, время простоя, методы синхронизации модельного времени и архитектуру локальных моделей стандарта HLA, что позволяет выполнить установленные ограничения на объем оперативной памяти и минимизировать время простоя вычислительных ресурсов при оптимистических и консервативных методах синхронизации программных моделей;

- впервые предложены методы оценки схемы распределения локальных программных моделей по вычислительным ресурсам, которые на основе характеристик динамического изменения объемов их сегментов данных, дампов памяти состояний моделей и оценок простоев ресурсов, выбирают приемлемые схемы распределения ресурсов, что позволяет уменьшить время выполнения и количество вычислительных ресурсов при реализации оптимистичных и консервативных методах синхронизации программных моделей;

- усовершенствована модель оценки времени выполнения программных моделей, которая, в отличие от существующей, учитывает увеличение объемов па-

мяти программных моделей и объемы передаваемых данных между программными моделями, время простоя ресурсов, что позволяет оценить время выполнения программных моделей для различных схем распределения ресурсов с учетом ограничений на объем памяти и суммарное время простоя вычислительных ресурсов;

- получили дальнейшее развитие методы распределения программных моделей по вычислительным ресурсам в распределенном имитационном моделировании GRID-систем, которые, в отличие от существующих, выбирают схемы распределения ресурсов с максимальным ходом модельного времени при ограничениях на объем оперативной памяти, времени простоя и выбирают способ синхронизации программных моделей, что позволяет повысить эффективность процесса распределенного имитационного моделирования путем уменьшения времени моделирования и количества вычислительных ресурсов.

Практические значения полученных результатов. Практическая значимость полученных теоретических результатов диссертационной работы подтверждена улучшением качества анализа распределенных имитационных моделей, работающих под управлением разных методов синхронизации. В частности, практические решения теоретических исследований заключаются в следующем;

- увеличено количество параметров, получаемых в процессе анализа распределенных имитационных моделей, что дает возможность улучшения методов распределения ресурсов, алгоритмов управления заданиями на моделирования в распределенных информационных системах;

- разработаны и реализованы алгоритмы и процедуры анализа распределенных имитационных моделей;

- разработана процедура сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени;

- разработана процедура оценки возможности распределения частных моделей на основе динамического изменения объемов виртуальной памяти;

- разработана процедура получения оценок простоя ресурсов для алгоритмов синхронизации распределенных имитационных моделей.

Практическое значение результатов диссертационной работы подтверждаются положительным эффектом от применения разработанных средств в:

- обществе с ограниченной ответственностью «Вента», г. Киев;

- обществе с ограниченной ответственностью «Альвис», г. Киев;

- при выполнении госбюджетных научно-исследовательских работ согласно тематическому плану Харьковского национального университета радиоэлектроники;

- в учебном процессе Харьковского национального университета радиоэлектроники при проведении лекционных, практических и лабораторных занятий по дисциплинам, направленным на изучение методов организации и поддержки баз данных, в курсовых и дипломных проектах, а также в спецкурсах для магистров, соискателей и аспирантов.

Личный вклад соискателя. Все научные результаты диссертационной работы, выносимые на защиту, получены автором самостоятельно. В работах, опубликованных совместно, автору принадлежат следующие результаты: в роботе [22] на основе аналитических моделей предложено обобщенную модель процессов дискретно-собфтийной распределенной имитации; в роботе [23] автор выявил основные недостатки и замечания при использовании современных методов распределения ресурсов, поставил задачу создания новых методов выбора схем распределения ресурсов; в роботе [24] автор предложил новые функциональные модули системы имитационного моделирования GRID и механизм их включения в существующее программное обеспечение, которое реализует разработанные методы распеделения программных моделей по вычислительным ресурсам; в роботе [25] рассмотрены потоки данных в системах имитационного моделирования, которые используются в процессе анализа программных моделей и методах оценивания и выбора схем их распределения по вычислительным ресурсам; в роботе [26] предложена аналитическая модель, которая в отличии от существующих, учитывает время простоя при использовании распределённых имитационных моделей с консервативными алгоритмами синхронизации; в роботе [27] предложена аналитическая модель, которая в отличии от существующих, учитывает динамическое изменение объемов памяти и потоков данных в локальных моделях при использо-

вании распределенных имитационных моделей с оптимистическими алгоритмами синхронизации; в роботе [28] формализовано множество объектов, которые обеспечивают процессы распределенной имитации; в роботе [29] предложена технология синхронизации распределенных моделей с использованием технологии предварительного анализа моделей; в роботе [30] предложены новые архитектурные и функциональные решения для включения в систему распределенного имитационного моделирования GRID-модулей анализа программных моделей; в роботе [31] изучено влияние методов и технологий синхронизации распределенных моделей на функционирование системы моделирования и методов анализа моделей; в роботе [32] разработан алгоритм интеграции общих модулей в систему моделирования; в роботе [33] предложены новые архитектурные и функциональные решения по включению в систему распределенного имитационного моделирования GRASS модулей для анализа программных моделей; в роботе [34] предложен новый метод анализа распределенных имитационных моделей с расширенным множеством параметров; в роботе [35] разработан формальный аппарат для отображения процессов распределенной имитации с использованием процессной алгебры; в роботе [36] проведен анализ влияния потоков заданий и данных в системах распределенного моделирования на эффективность их функционирования; в роботе [37] проведен анализ влияния сетевого трафика в системах распределенного моделирования на эффективность их функционирования; в роботе [38] предложены усовершенствованные аналитические модели для анализа распределенных имитационных моделей с разными методами синхронизации; в работе [39] введено понятие менеджера памяти и приведены его функции; в работе [40] проделен анализ влияния технологий синхронизации моделей на технологии их анализа в режиме реального времени и с использованием априорной информации; в роботе [41] предложена информационная технология анализа распределенных программных моделей; в роботе [42] приведены этапы технологии анализа распределенных программных моделей.

Апробация результатов диссертации. Основные положения и результаты диссертационной работы были представлены, докладывались и обсуждались на

международных научных конференциях, в частности на:

- 11-й, 12-й Международной научно-технической конференции «Системный анализ и информационные технологии» (Киев, 2009, 2010 гг.);

- 13-ом, 14-ом Международном молодежном форуме «Радиоелектроника и молодежь в XXI ст.» (Харьков, 2009, 2010 гг.); 9-й Международной научно-технической конференции «Проблемы информатики и моделирования» (Харьков, 2009);

- 1-й, 2-й Международной научно-технической конференции «Информационные технологии в навигации и управлении: состояние и перспективы развития» (Киев, 2010, 2011 гг.);

- 1-й, 2-й Международной научно-технической конференции «Современные направления розвития информационно-коммуникационных технологий и способов управления» (Киев – Харьков 2010, 2011 гг.).

На основании проведенных исследований и практической реализации представленных методов и алгоритмов разработано методическое и программное обеспечение, используемое в учебном процессе Харьковского национального университета при изучении дисциплин профилирующего цикла «Интерфейсы параллельного программирования», «Имитационное моделирование на инновационных HPC системах», спецкурсы руководителя магистерскими работами.

Публикации. По результатам диссертационных исследований опубликовано 21 научную работу. Из них 7 статей в сборниках и журналах, рекомендованных как специальные издания в технических науках (1 издание входит в международные наукометрические базы РИНЦ и WorldCat), 1 статья в иностранном научно-техническом журнале, 13 публикаций материалов (тезисов докладов) международных конференций.

Структура диссертации. Диссертационная работа состоит из введения, четырех разделов основной части, выводов, перечня использованных источников, приложений.

В первом разделе проводиться анализ предметной области диссертационного исследования. В частности, рассматриваются вопросы развития распределен-

ного имитационного моделирования, международный стандарт HLA распределенного имитационного моделирования, процессная алгебра как формальный аппарат исследования процессов, методы анализа распределенных имитационных моделей, GRID-системы как технология проведения распределенной имитации.

Во втором разделе приводиться формальные модели распределенной имитационной среды моделирования, изменения состояния имитационной модели, формально представлены консервативные и оптимистические методы синхронизации распределенного моделирования, показана совместимость моделей со стандартом HLA.

Третий раздел посвящен выбору критериев оценивания, разработке моделей, методов и алгоритмов анализа процесса распределенной имитации.

Четвертый раздел содержит описание информационных технологий исследования распределенных программных моделей, разработанных программных средств, реализованных на основе предложенных методов и алгоритмов, интеграцию их в систему имитационного моделирования GRID-систем, результаты практических экспериментов с разработанным программным обеспечением.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЦЕЛИ И ЗАДАЧ ИССЛЕДОВАНИЯ

1.1 Введение

Диссертационная работа выполнена на стыке трех научно-технических направлений. Во-первых, основной объект изучения – это теория имитационного моделирования, в которой выбран конкретный предмет исследований – методы анализа имитационных моделей и алгоритмов управления ими. Во-вторых, распределенная природа выбранного класса имитационных моделей требует учета методов и средств параллельного и распределенного программирования. Третье направление определяется выбранной средой исполнения распределенных имитационных моделей большой размерности – GRID-системами.

Все перечисленные направления являются сравнительно молодыми. Самое «старое» из них – имитационное моделирование, которое начало активно использоваться с 60-х годов XX-го века.

Анализу развития имитационного моделирования в Украине и в мире посвящен подраздел 1.2. В подразделе 1.3 приводиться анализ современного состояния распределенного имитационного моделирования. В подразделе 1.4 приводиться обзор стандарта распределенного имитационного моделирования HLA. Для изложения положений диссертационной работы, связанных с описанием процессов, протекающих в имитационной среде моделирования, выбран формальный аппарат представления динамических процессов на основе процессной алгебры, обзор которой приведен в подразделе 1.5. Подраздел 1.6 посвящен методам анализа распределенных имитационных моделей. В подразделе 1.7 рассматриваются особенности GRID-систем и их место в распределенном имитационном моделировании. В подразделе 1.8 формулируется цель и задачи диссертационного исследования.

1.2 Анализ технологий имитационного моделирования

Имитационное моделирование, как новое научное направление начало интенсивно развиваться в 60-е года XX-го века, что обусловлено широким внедрением и использованием сложных технических систем в разнообразных отраслях человеческой деятельности (космос, метеорология, биология, медицина, энергетика, транспорт, экономика, новые технологии на производстве и др.). Отличительными особенностями таких систем являлись большая размерность, наличие разнородных элементов и подсистем, отсутствие математического описания законов их поведения, а в некоторых случаях – отсутствие достоверной статистики их параметров. Более того, натурные эксперименты с такими системами как правило невозможны, либо затруднены или ограничены. Разрабатываемые методы проведения имитационных экспериментов требовали конкретной реализации, а следовательно, технологий, объединяющих существующие и специально разрабатываемые средства для построения моделей и их исполнение в процессе моделирования.

Само понятие «имитационное моделирование» имеет несколько определений. Р. Шеннон так определяет понятие имитационного моделирования: «Имитационное моделирование – процесс конструирования модели реальной системы и постановки экспериментов на этой модели с целью оценить (в рамках ограничений, накладываемых некоторым критерием или совокупностью критериев) различные стратегии, обеспечивающие функционирование системы» [1].

Термин «имитационное моделирование» вобрал в себя два понятия, которые первоначально являлись обозначением одного и того же. Все расчетные методы на ЭВМ во всех сферах науки и техники являются моделями реальных процессов. Для аутентификации математических моделей исследователи стали давать им дополнительные названия. Термин «имитационное моделирование» означает, что мы имеем дело с такими математическими моделями, с помощью которых результат нельзя заранее вычислить или предсказать, поэтому для предсказания поведения реальной системы необходимо проведение эксперимента (имитации) на

модели при заданных исходных данных. В 80-е года XX-го века имитационное моделирование стали напрямую ассоциировать с моделированием на компьютерах. «Имитация представляет собой численный метод проведения на ЭВМ экспериментов с математическими моделями, описывающими поведение сложной системы в течение заданного или формируемого периода времени» [2].

Поведение системы, взаимодействие её элементов, в имитационной модели чаще всего описываются набором алгоритмов и данных программы, реализуемых на некотором языке моделирования (или программирования). Все эти описания представляют собой программную имитационную модель, которую необходимо спроектировать, написать, отладить, испытать, а затем использовать для постановки имитационного эксперимента на ЭВМ. Поэтому под процессом имитации на ЭВМ понимаются и конструирование модели, и ее испытание, и применение модели для изучения некоторого явления или проблемы.

В Украине становление имитационного моделирования, как научной и прикладной дисциплины, связано с именем Н.П. Бусленко. Методологической основой для развития имитационного моделирования явились работы Н.П. Бусленко, В.М. Глушкова , Н.Н. Моисеева [3,4,5].

Школы под их руководством сформировали традиции имитационного моделирования, сосредоточившись на следующих направлениях:

- развитие методологии, методов и технологий моделирования;
- разработка средств и систем моделирования на базе универсальных алгоритмических языков моделирования;
- разработка пакетов моделирования широкого назначения;
- разработка проблемно-ориентированных пакетов моделирования.

Среди известных зарубежных ученых, вложивших солидный вклад в развитие имитационного моделирования следует выделить Р. Шеннона [1], А. Прицкера [7], Д. Гордона [8]. Школы имитационного моделирования в нашей стране и за рубежом развивались параллельно, что привело к созданию своих оригинальных языков и средств моделирования.

Языки непрерывного моделирования (еще его называют совмещенным [43]

или аналоговым моделированием [44,45]), в отличие от сложившегося в 50-е годы прошлого века представления об аналоговом моделировании как моделировании с использованием аналоговой и гибридной вычислительной техники [46]) имеют блочную либо операторную ориентацию. Ранее под блочным заданием системы понималось соответствие элементов цепей в аналоговых компьютерах [46]. Представителем блочной ориентации является стандарт PSpice, в котором схема задается путем указания типа элемента, его параметров и узлов подключения [7,47]. В операторных языках уравнения состояния (дифференциальные, разностные и др.) кодируются в явном виде. К ним можно отнести непрерывные расширения языков НЕДИС, ГАСП, ПМНМ, 360/System CSMP.

При событийном подходе [2,48] модель продвигается во времени от события к событию, которые изменяют состояние модели. Логика наступления событий определяет последовательность смены состояний модели, причем время продвигается от события к событию. Данный подход развивался в 60-70 годы XX века и привел к созданию таких систем моделирования как SIMSCRIPT, SMPL, ГАСП, VHDL [49,50] и т.д.

При транзактном подходе модель представляет собой совокупность двух типов объектов – динамических (транзактов) и статических (приборов обслуживания). Поток транзактов продвигают от одного статического объекта к другому, состояние модели изменяется в дискретные моменты времени после того, как закончилось перемещение транзактов в текущий момент модельного времени. Примером транзактных систем моделирования являются GPSS [51,52,53,19,20], ASPOL [43]. Система моделирования GPSS находит широкое применение сегодня у российских исследователей [54,55].

Процессный подход [2,9] заключается в представлении объекта в виде совокупности нескольких компонентов, каждому из которых ставиться в соответствие своя последовательность событий. Каждая последовательность событий реализует собой процесс – изменение состояния системы внутри одного компонента. Модель представляется как совокупность взаимодействующих квазипараллельных процессов, что более адекватным образом отражает структуру и поведение реаль-

ной системы. Квазипараллельность заключается в том, что программа, соответствующая процессу, составляется из последовательности программ событий независимо от других процессов, а исполняется с прерываниями, во время которых исполняются другие процессы [9]. Реализация таких систем, как правило, включает два типа подпрограмм: соответствующие событию и процессу. Первую из них система моделирования вызывает в моменты времени, определяемые событиями. Вторая подпрограмма активируется в той же точке, в которой процесс был приостановлен. Так как логика работы процессов сосредоточена в одной программе, облегчается описание динамики системы.

При использовании подхода сканирования активностей разработчик описывает действия, в которых принимают участие элементы системы, и задает условия, определяющие начало и окончание этих действии. События, которые начинают или завершают действие, не планируются разработчиком модели, а инициируются по условиям, определенным для данного действия. Условия начала или окончания действия проверяются после очередного продвижения имитационного времени. Если заданные условия удовлетворяются, происходит соответствующее действие в модели, сканирование условий производится для всего множества действий при каждом продвижении имитационного времени [2]. Примером языка, ориентированного на сканирование активностей является SMPL.

В объектно-ориентированных системах имитационного моделирования модель представляет собой совокупность объектов. Объекты включают данные и операции над ними. Объекты обладают специфическим интерфейсом (набором методов или функций), благодаря которому они взаимодействуют с системой моделирования и другими объектами. Этот подход широко используется при написании программных моделей на универсальных языках программирования высокого уровня (C++, Pascal). Методологические подходы к использованию объектно-ориентированных технологий систематизированы в работе Г. Буча [56]. Примером реализации объектно-ориентированного подхода является SIMULA [57,58]. Объектно-ориентированный подход получил развитие в работах Пегдена [59] с новым названием – агентный подход. Согласно этому подходу поведение всей

системы можно рассматривать как результат поведения большого числа объектов, называемых агентами, которые являются автономными, могут взаимодействовать друг с другом и преследуют свои цели [60,61,62,63].

В докторской диссертации В.В. Окольнишникова [9] приводиться определение логического подхода к имитационному моделированию. «Логический подход – это использование в имитационном моделировании концепций из области искусственного интеллекта – концепций неалгоритмического программирования, поиска по образцу и бэктрекинга. Такой подход дает возможность строить ориентированные на поиск цели или генерирующие цель самоорганизующиеся модели с переменной структурой». Другим примером может служить теория агрегатов [64,65].

При общем рассмотрении методологических подходов находятся области их пересечения. Например, «событие – состояния» аналогично понятию «сканирование активностей», в которых события не планируются, а инициируются определенным состоянием системы; процессно-ориентированный подход сочетает в себе черты событийного подхода и подхода сканирования активностей [2].

Проведенный анализ приводит к мысли о создании единого подхода к представлению имитационных моделей с целью унификации разрабатываемых алгоритмов и программного обеспечения систем моделирования. Подобные попытки проводятся постоянно. Однако общее направление исследований по данному вопросу сводится к созданию своего методологического подхода или концептуальной схемы построения имитационных моделей (например, теория агрегатов [64], сети Петри [66] и т.д.). В данной работе предлагается использовать подход, основанный на работах [2,9,15,16,17]: рассмотреть имитационные модели, построенные с точки зрения различных концептуальных схем, в рамках одного подхода, основанного на выделении в структуре имитационных моделей однородных по функциональному назначению и взаимосвязям элементов и того очевидного факта, что все перечисленные подходы реализуются программно, то есть, они содержат в себе одинаковые структурные элементы: данные, отражающие состояние модели и код, отражающий ее поведение.

1.3 Анализ технологий распределенного имитационного моделирования

Одним из наиболее актуальных направлений в области имитационного моделирования систем является распределенное (distributed) или параллельное (parallel) имитационное моделирование (simulation), проводимое с целью повышения быстродействия. Традиционно, распределенное имитационное моделирование относилось к моделированию на компьютерах с MIMD (Multipli Instruction Stream / Multipli Data Stream) архитектурой, в то время как использование архитектуры SIMD (Single Instruction Stream / Multipli Data Stream) приводило к «параллельному моделированию». Поскольку при параллельном и распределенном моделировании используется одна и та же техника, с появлением кластерных вычислительных систем и GRID-вычислений грань между этими двумя видами имитационного моделирования начала стираться. Многими авторами используется термин «распределенное моделирование» для обозначения как параллельного, так и распределенного моделирования [9,17]. Тенденция усиливается растущими возможностями к интеграции и построению многопроцессорных и многомашинных вычислительных систем моделирования, появлением GRID-систем как мощной среды проведения имитационных экспериментов [67,68].

Распределенное имитационное моделирование основывается на следующих источниках: вычисление и моделирование, требующее для своего выполнения большого количества вычислительных ресурсов, военные приложения, и компьютерные программы, обменивающиеся информацией при помощи Интернет в реальном времени. Под распределенным имитационным моделированием понимается «распределенное выполнение единой программы имитационной модели на мультипроцессорной или мультикомпьютерной системе» [10,69]. При распределенном моделировании, в отличие от последовательного моделирования, первичной единицей является не объект, а так называемый логический процесс [9] или частная модель [15]. Логический процесс – это последовательная подмодель, которая имеет собственный набор объектов и собственную управляющую программу. Частная модель содержит в себе подпрограмму, определяющую поведение

моделируемого времени и подпрограмму передвижения модельного времени.

В работах В.В. Окольнишникова [9,10] приведено следующее формальное представление распределенной имитационной модели:

$$\begin{aligned} &DM = U_{k=1}^{N} lp^k, \\ &lp^k = \{sm^k, T^k, S^k\}, \\ &S^k = \{(Q_1^k, t_1^k), (Q_2^k, t_2^k), \ldots, (Q_m^k, t_m^k) \mid t_1^k < t_2^k < t_m^k\}, \\ &T_k = t_1^k, \end{aligned} \tag{1.1}$$

где *DM* – распределенная модель;

lp – логический процесс;

sm – подмодель логического процесса;

T – значение локальных часов модельного времени;

S – локальный список событий;

Q – событие в списке событий;

t – модельное время наступления события Q (временная метка события Q).

В работах В.А. Горбачева и М.А. Волка [15] приводится другое формальное представление среды имитационного моделирования, которое более полно учитывает обобщающие параметры как программной распределенной имитационной модели, так и характеристики распределенной среды моделирования:

$$\begin{cases} \{ИМ, ЭВМ, ПП_м\}, \\ ИМ = \{АП, Н, M_T, ДМ, \varphi, \mu, \rho\}, \\ ЭВМ = \{ПП,\ CC,\ RR, W\ W\}, \\ ПП_i = \{<\Pi_i, U_i,\ J_i,\ C_i\ R_i\}, \\ \begin{cases} \varphi: \{\varphi_t : ДМ_t \times АЛ \rightarrow ДМ_t' \ \&\ ДМ_t \cap ДМ_t' = \varnothing \ \&\ t \in T \ \&\ t = const\} \\ \mu: \{\mu_{tt'} : ДМ_t' \times M_T \rightarrow ДМ_{t'} \ \&\ t, t' \in T \ \&\ t' \geq t\} \\ \rho: \{\rho : ДМ_{t'} \times Н \rightarrow ДМ_t' \ \&\ ДМ_t \cap ДМ_t' = \varnothing\} \end{cases} \end{cases} \tag{1.2}$$

где ИМ – имитационная модель;

ЭВМ – множество ресурсов;

$ПП_м$ – множество программных элементов;

АП – подпрограмма, реализующая поведение модели;

Н – подпрограмма связи;

МТ – процедура управления временем модели;

ДМ – область данных модели;

φ – функция логического преобразования данных;

μ – временная функция;

ρ – функция преобразования данных;

ПП= {ППi}, i=1, ..., n – множество независимых иерархических систем процессов;

CC= {CCj}, j=1,...,β – множество системных ресурсов (оперативная память, процессоры, устройства ввода-вывода и т. д.);

β – количество системных ресурсов;

RR={RR1, ..., RRa} – семейство множеств ресурсов пользователей;

α – число пользователей сети ЭВМ;

WW – внешняя среда, определяемая системой управления данными Wд, пользователями Wa и программными подсистемами сети ЭВМ;

U_i, J_i, C_i, R_i – множество соответственно управляющих, информационных, ресурсных по системе и ресурсных по пользователям связей между процессами.

Элементы множеств *АП*, *Н*, M_T являются исполняемыми процедурами, а функционалы φ, μ, ρ – отражают динамику поведения.

Сравнивая представление имитационных моделей в выражениях (1.1) и (1.2) можно найти много общего. Так логический процесс lp эквивалентен понятию имитационной модели ИМ, события *S*,*Q*, а также состояния модели реализуется в представлении (1.2) в виде данных модели ДМ, упорядоченность событий согласно модельному времени обеспечивается подпрограммной управления модельным временем M_T.

Основной недостаток большинства моделей – высокий уровень абстракции,

который затрудняет их применение при решении конкретных задач, а также отсутствие единого формального аппарата их представления. В связи с этим, перспективным видится использование единого формального аппарата описания распределенных имитационных моделей и функционирования имитационных систем. Такой подход предлагается в работах [15,17]. В работе [17] предлагается формальный аппарат представления имитационных моделей на основе процессной алгебры [16], который позволяет представить вычислительные процессы, протекающие в имитационной системе моделирования с естественной (программной) точки зрения. Описанных подход позволяет описывать не только элементы имитационной среды, но и учитывать специфику операционной системы, аппаратных ресурсов и других компонентов, участвующих в процессе эксперимента. Основной идеей предлагаемого аппарата является единое представление таких важных понятий имитационных моделей как состояние, модельное время, входные и выходные переменные в качестве данных программы. Упрощая систему выражений (1.2) представим любую программную имитационную модель в виде двух естественных составляющих – кода и данных (выражение 1.3):

$$\begin{cases} ИМ = \left\{ ИМ_i, i = \overline{1, I} \right\} \\ ИМ_i = \left\{ dm_i, A_i, i = \overline{1, I} \right\} \\ A_i = \left\{ A_i^j, j = \overline{1, J} \right\} \end{cases} \quad (1.3)$$

где $ИМ_i$ – частные имитационные модели;

I – количество частных имитационных моделей в системе;

dmi – данные модели;

A_i – множество из J активностей (подпрограмм), обслуживающих *i*-ю имитационную модель.

Следующий важный вопрос – синхронизация модельного времени частных моделей или вопрос управления модельным временем. Синхронизация должна обеспечивать выполнение частных моделей в пространстве единого модельного времени или в правильном хронологическом порядке, обеспечивая тем самым

ограничение причинной связи (local causality constraint) [10]. Методы управления временем в имитационных системах являются сложной, наукоемкой проблемой. В работе [10] приводится обзор методов решения этой проблемы для последовательного и распределенного имитационного моделирования.

Известны несколько классов алгоритмов синхронизации модельного времени: синхронные, с синхронизацией взаимодействия, консервативные и оптимистические. Классификация современных алгоритмов синхронизации модельного времени приведена в работах [70-74].

Первые два класса алгоритмов синхронизации продвигают модельное время только при условии, что все частные модели согласны с этим, что приводит к неравномерной загрузки вычислительных ресурсов. Их применение возможно при соблюдении нескольких условий: все модели имеют одинаковую вычислительную стоимость, вычислительные ресурсы и каналы связи между ними имеют одинаковые параметры.

В целях распределенного имитационного моделирования наиболее часто используют консервативные и оптимистические алгоритмы.

Особенностью консервативных алгоритмов является предотвращение парадоксов времени. В процессе имитации предполагается, что адресат получает сообщения в том же порядке, в котором их посылает источник, и что источник выбирает из локального списка событий события с неубывающими временными метками (time stamped). При этом возможна приостановка выполнения частных моделей, что снижает производительность процесса имитации и может привести к тупиковым ситуациям. Для устранения этих недостатков консервативные алгоритмы используют дополнительную информацию. Например, информацию о вероятности определенных связей между частными моделями.

Первые консервативные алгоритмы использовали посылку «пустых» сообщений, которые не отражают поведение частных моделей, а передают дополнительную информацию для определения момента изменения модельного времени и величины этого изменения [75-77]. Последняя величина получила название «нижней границы временных меток» (LBTS – lower bound on the time stamp) [10]

или «временного горизонта» (logical virtual time horizon) [72]. Особенности работы с временным горизонтом исследуются в [78]. Способы минимизации количества пустых сообщений обсуждаются в [79]. В работе [77] указывается, что корректная работа с временным горизонтом не гарантирует отсутствие тупиковых ситуаций, распознавание которых необходимо при возникновении ситуации зацикливания алгоритма синхронизации

Некоторые консервативные алгоритмы используют предварительный просмотр (look a head), который вызван заложенным в модель подходом к построению имитационной модели или физическими особенностями вычислительной среды.

Существуют консервативные алгоритмы, которые игнорируют возникающие парадоксы времени (с определенной точностью). Например, подход «расслабления времени» (relaxing time), допускающий отход от учета событий в соответствии с временной упорядоченности [80], что возможно только на ограниченном классе моделей и определяется природой моделируемых объектов либо предполагается, что дальнейшая статистическая обработка результатов моделирования даст допустимую точность [81,82]. Аналогичные (приближенные) результаты дают системы моделирования, синхронизация в которых достигается путем синхронизации модельного и реального. В последнем случае возникает задача синхронизации часов самих вычислительных ресурсов [83,84]. В системах с предсказанием известны кванты модельного времени, с которыми частные модели изменяют свои локальные часы [85,86]. Еще один подход основан на понятии приближенного времени (approximate time), он дает возможность использовать нечеткость времени возникновения событий: вместо момента времени используется интервал времени, в течении которого возможно возникновение события [87,88,89].

Многие консервативные алгоритмы используют понятие «расстояния» (distance), которое представляет собой временной отрезок, необходимый для прямого или косвенного воздействия одного логического процесса на другой [90,91,92]. Согласно определению, данному в [9], расстояние – это минимальное количество модельного времени, которое должно пройти, прежде чем собы-

тие в одном логическом процессе вызовет прямо или косвенно выполнение события в другом логическом процессе. Расстояние может быть использовано логическим процессом для определения временной отметки события, которое ему будет послано.

Более сложные алгоритмы консервативной синхронизации предполагают передачу между частными моделями сообщения помеченного не временной, а векторной меткой времени (Vector Time), позволяющей частной модели определить место своего события среди других [87,93,94,95].

В отличие от консервативных алгоритмов, которые не нарушают порядок выполнения событий в неубывающем модельном времени, оптимистические методы не следят за этим ограничением. В связи с этим, при таком методе происходят «парадоксы времени»: некоторые частные модели могут «уйти вперед», то есть их локальное время превышает (иногда – значительно) локальное время других частных моделей.

Наиболее известный оптимистический алгоритм – алгоритм, разработанный Д. Джефферсоном (Jefferson), называется механизмом «деформации времени» (Time Warp Mechanism) [96]. Когда логический процесс получает событие, имеющее временную отметку меньшую, чем уже обработанные события, он выполняет «откат» (rollback) и обрабатывает эти события повторно в хронологическом порядке. Откатываясь назад, процесс восстанавливает состояние, которое было до обработки событий и отказывается от сообщений, возвращаемыми в прошлое состояние. Для отказа от этих сообщений разработан механизм антисообщений. Антисообщение – это копия ранее отосланного сообщения. Если антисообщение и соответствующее ему сообщение (позитивное) хранятся в одной и той же очереди, то они взаимно уничтожаются. Чтобы изъять сообщение, процесс должен отправить соответствующее антисообщение. Если соответствующее позитивное сообщение уже обработано, то процесс-получатель откатывается назад. При неблагоприятных условиях моделирования откат частной модели может произойти до любого момента модельного времени, начиная с нулевого значения. Поэтому надо сохранять состояния частных моделей для каждого такого момента модельного

времени, что приводит к большому расходу памяти. Для устранения этого недостатка существует несколько групп оптимистических алгоритмов.

Представители первой группы ориентированы на нахождение минимальной временной метки откатов – GVT (global virtual time) [97], которая является аналогом глобального модельного времени в последовательных моделях. После вычисления GVT, память, используемая для хранения состояний всех частных моделей с временными метками, меньшими, чем GVT освобождается. Некоторые авторы [98] предлагают использовать последовательные отрезки вычислений и счётчики сообщений, эффективно решающие эти проблемы.

Одной из проблем, возникающих при реализации оптимистических алгоритмов, заключается в значительных объемах оперативной (виртуальной) памяти для хранения информации, необходимой для осуществления откатов. Подходы второй группы направлены на сокращение объемов памяти. В некоторых случаях, при переполнении памяти, хранимые состояния могут удаляться, даже если их временная метка больше GVT [99], сохранение состояний может происходить выборочно из всего множества состояний между текущим модельным временем частной модели и GVT [100,101], используются математические методы пересчета параметров моделей вместо операций с памятью (отката состояний) [102,103].

Подходы третьей группы пытаются избавиться от антисообщений или ограничивают их количество. Это подходы, использующие «просмотр назад» (lookback) [104,105], «непосредственной отмены» (Direct Cancellation) антисообщений [106,107,108], «откладывание» (lazy cancellation) антисообщений при откате [109,110].

К четвертой группе можно отнести подходы, которые к распределенной имитационной модели добавляют адаптивные программные элементы, которые следят за параметрами эксперимента и изменяют параметры среды имитации или модели с целью оптимизации того или иного ресурсного вектора процесса имитации [111,112].

Как отмечается в [9] заранее нельзя сказать, какой из множества консервативных и оптимистических подходов будет работать быстрее. Все зависит от реа-

лизации модели, параметров среды исполнения, доступных вычислительных ресурсов и каналов связи и т.п. Оптимистические алгоритмы требуют избыточных вычислений и дополнительной памяти для хранения истории изменения состояний, для определенного класса моделей, в которых откаты происходят редко, оптимистические алгоритмы дают выигрыш во времени исполнения модели за счет более эффективного использования параллелизма. В [9] перспективным видится создание библиотеки различных консервативных и оптимистических алгоритмов. После чего на основе экспертной оценки разработчика модели, тестового исполнения модели на ограниченном отрезке модельного времени, можно было бы определить, какой из классов алгоритмов, консервативный или оптимистический, более подходит для данной модели.

1.4 Анализ технологии распределенного имитационного моделирования на базе стандарта HLA

На основании анализа, проведенного в подпунктах 1.2-1.3 можно сделать вывод, что распределенное моделирование представляет собой постоянно развивающуюся область научных исследований. Появление новых объектов моделирования, новых вычислительных ресурсов приводит к появлению новых и модификации уже известных методов распределенного имитационного моделирования.

В середине 90-х годов XX-го века появился стандарт High Level Architecture (HLA – архитектура высокого уровня), который попытался объединить уже созданные модели и системы моделирования для возможности их дальнейшего совместного использования. HLA представляет собой совокупность методик, решений и стандартов для построения систем распределенного моделирования и интерактивные системы имитации. Под интерактивностью системы понимают способность ее подсистем, модулей и элементов выдавать и принимать сообщения от других подсистем, модулей и элементов. Обмен сообщениями позволяет организовать совместное функционирование различных участников моделирования. Данная архитектура определяется международными стандартами IEEE 1516, 1516.1 и 1516.2. HLA – это новая технология, предназначенная для обеспечения

взаимодействия территориально распределенных участников моделирования различных типов. Эта технология объединяет вместе системы, построенные для различных целей, технологии разных периодов времени, продукты и платформы различных фирм и позволяет им взаимодействовать в единой среде моделирования.

Основными предпосылками появления стандарта HLA считаются работы, проводимые в военных целях (США), по объединению в одном имитационном пространстве военных тренажеров и программных моделей. Первыми стандартами в этом направлении считаются стандарт взаимосвязи распределенных тренажеров DIS (Distributed Interactive Simulation) [113], вышедший из проекта SIMNET [114], и разработанный протокол ALSP (Aggregate Level Simulation Protocol) взаимодействия тренажеров [115]. Одновременно с работами, проводимыми в этой области, была создана группа DMSO (Defence Modeling&Simulation Office), которая с 1996 года координирует исследования по созданию специальной технологии HLA, определяющей общую архитектуру всех разрабатываемых в США систем моделирования. С этого момента все разработчики средств и систем моделирования должны обеспечивать внешнюю взаимосвязь с учетом стандарта HLA. В 1998 году HLA была принята стандартом в NATO. Организация SISO (Simulation Interoperability Standarts Organization) в настоящее время координирует с IEEE и OMG (Object Management Group) работы по HLA стандартам.

Одной из концепций стандарта является объединение моделей, созданных разными разработчиками в разное время. Причем это объединение достигается двумя средствами. Во-первых, это стандартизация интерфейсов модели, то есть функций (методов) при помощи которых модель взаимодействует с внешним миром. В качестве последнего для модели выступает специальная инфраструктура Runtime Infrastructure (RTI). Данная инфраструктура гарантирует обмен информацией в реальном времени между имитационными моделями и другими участниками виртуального процесса (устройствами, тренажерами, пользователями). Все они называются федератами (federates). В качестве федератов выступают имитационные модели и системы моделирования (constructive), тренажеры (virtual), реальные участники или пользователи (live), специализируемое программное обес-

печение, которое сохраняет, статистически обрабатывает и визуализирует текущую информацию о процессе. Во-вторых, стандартизация самой структуры RTI. В результате анализа стандартов [116,117,118], создана структурная схема, отображающая элементы RTI (рисунок 1.1).

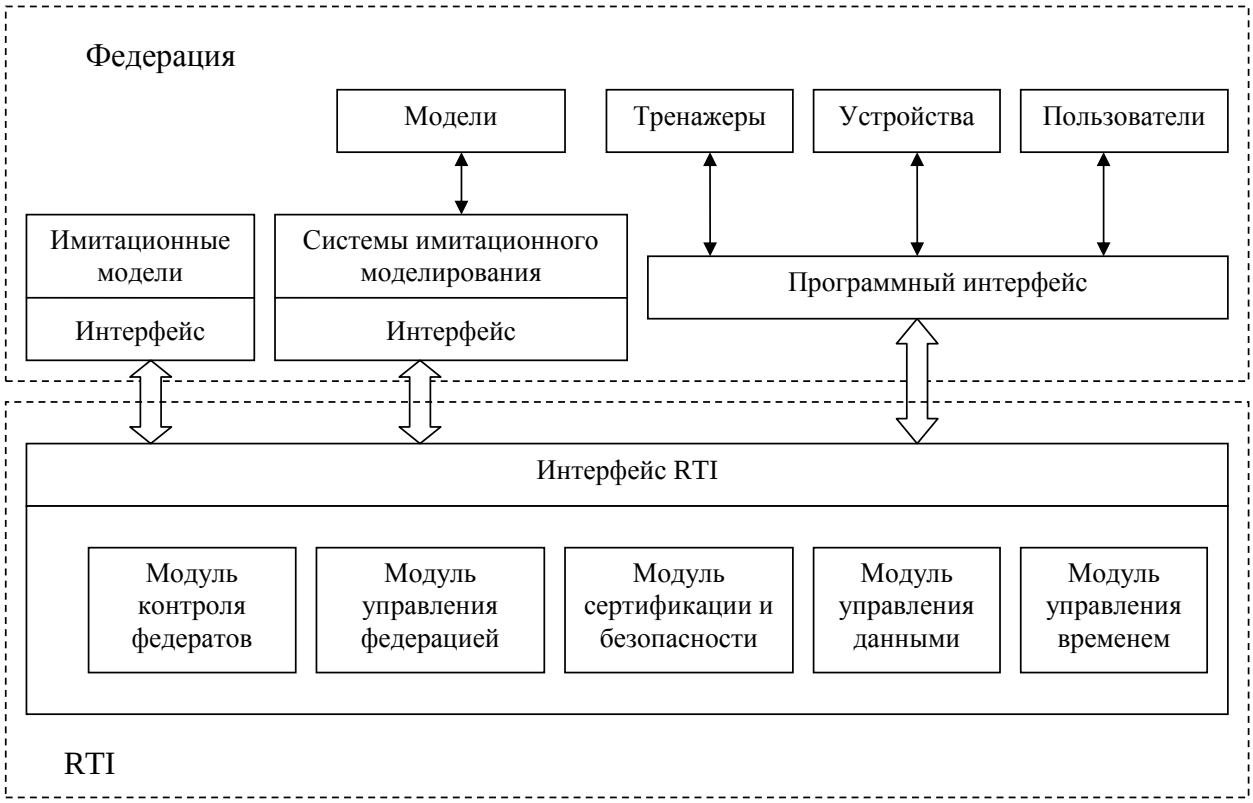


Рисунок 1.1 – Элементы архитектуры HLA

Основными составляющими стандарта является: набор общих правил HLA [119,120], правила построения федератов и организации федерации [121,122], спецификация интерфейса федератов [123]. Сервисы RTI обеспечивают управление федерацией, управление федератами, сертификацию и безопасность, управление распределением данных, управление модельным временем и синхронизацию. Примерами реализованных согласно стандарта HLA программных систем является программные продукты VR-Link, Plan View Display, DIS/HLA для Vega Prime.

Несмотря на достаточно широкое распространение данного стандарта, многие разработчики имитационных систем моделирования отмечают сложность реа-

лизации этого стандарта, а также невозможность его использования для языков декларативного типа из-за объектно-ориентированной направленности HLA.

1.5 Анализ процессных алгебр как формального аппарата для представления параллельных процессов

В первой половине 20-го века, были предложены различные способы формализации описания понятия вычислимой функции: μ-рекурсивная функция, машина Тьюринга и лямбда-исчисления, являющейся наиболее известными примерами. Однако в связи с развитием информационных технологий потребовались нетрадиционные формулировки понятия вычислений, в частности явные представления параллелизма и связей между параллельно функционирующими компонентами. Появились такие модели параллельных исчислений как Сети Петри в 1962 г. и модель «Деятеля» в 1973 г. Исследованиями процессных алгебр начали всерьез заниматься Робин Милнер, предложив процессную алгебру «Взаимодействующих параллельных систем» (CCS – Calculus of Communicating Systems), в период с 1973 по 1980 год [21,124,125,126]. Параллельно с ним работал также известный учёный Чарльз Хоар, предложивший иную концепцию, известную под названием «Взаимодействующие последовательные процессы» (CSP – Communicating Sequential Processes), которая впервые появились в 1978 году, и впоследствии превратилась в полноценную процессную алгебру в начале 1980-х [18,19,20]. По мере развития эти обе базовые концепции развивали и дополняли друг друга. В 1982 году Ян Бегстра и Ян Виллем Клоп начал работу над тем, что стало впоследствии известно, как «Алгебра взаимодействующих процессов» (ACP), собственно и предложившие термин «процессная алгебра» для описания сути их работы [127]. CCS, CSP и ACP представляют собой три основные ветви в семье процессных алгебр: все остальные теории берут свои начало у этих основополагающих концепций.

В настоящее время исследования в области процессных алгебр сосредоточены в основном на следующих проблемах:

- разработка новых процессных алгебр для более точного имитационного

моделирования;

- нахождение подходящих подмножеств алгебр для данной проблемной области. Многие из алгебр сами по себе являются достаточно обширными и общими для того, чтобы описывать конкретные специфичные процессы, а также, многие приложения не используют всех средств, предоставляемых алгебрами;

- расширение поведенческих теорий понятиями эквивалентности процессов (поиск классов эквивалентных процессов). Как правило, процессы могут считаться идентичными вне контекста параллельности, однако с учётом таковой может обнаружиться разница. К сожалению, принятие этих различий является тонкой деталью и зачастую непросто установить истинную эквивалентность.

Определение процессной алгебры обычно начинается с начального набора каналов, которые используются в для обеспечения взаимодействия процессов. В дополнение к каналам, нужны средства для создания новых процессов.

Основные операторы, представленные в той или иной форме, позволят представить: композицию параллельных процессов; спецификацию: какие каналы использовать для приема передачи данных; описание последовательности взаимодействий; сокрытие точек взаимодействия; рекурсию (репликацию) процессов.

Краткий обзор процессных алгебр приведен в [128,129], исторический анализ в [130], различные подходы, связанные с теорией эквивалентностей – в работах [131,132], процессные алгебры с рекурсией введены в [133,134], математические методы верификации представлены в [135], расширения π-calculus появились в работах [136,137,138]. Также известны вероятностные процессные алгебры, где процессы имеют дополнительные вероятностные характеристики [139-143].

Выбор той или иной концепции определяется самой задачей, для которой та или иная процессная алгебра больше подходит. Большинство процессных алгебр предлагают широкий набор аксиом и теорем для доказательства эквивалентности процессов, а также для нахождения дедлоков и тупиковых ситуаций.

Учитывая нынешнюю тенденцию к использованию многопроцессорных высокопроизводительных вычислений, большинство ранее описанных алгоритмов подвергаются переработке для обеспечения возможности распараллеливания про-

цесса вычисления. Однако при таком переходе эквивалентность процессов может быть нарушена, плюс к тому в алгоритме могут содержаться тупиковые ситуации взаимодействия параллельно работающих частей.

На сегодняшнее время наибольшее распространение получило семейство процессных алгебр CCS, включая такие расширения как π-calculus, описывающее динамически изменяемые системы, и PEPA (Performance Evaluation Process Algebra), добавляющее операторы стохастического недетерминизма.

Процессная алгебра как средство описания процессов имитационной среды моделирования используется и украинскими учеными [144,145,146]. Процессная алгебра была выбрана в качестве формального аппарата представления процессов, протекающих в распределенной имитационной среде моделирования в работах [15,16,17] и будет использована для этих целей в данной работе.

1.6 Анализ моделей процесса распределенного имитационного моделирования

Как отмечают многие из авторов, одной из основных проблем выбора того или иного методологического подхода, способа, алгоритма синхронизации, среды распределенного имитационного моделирования при решении конкретной задачи является трудность предсказания: какой из них даст более эффективное решение. Под эффективностью, в данном контексте, будет пониматься будущее быстродействие имитационного эксперимента, то есть при выборе какой из совокупностей средств моделирования для одного и того же имитационного эксперимента на одних и тех же совокупностях, и состояниях ресурсов будет получено меньшее время выполнения модели.

Проблема анализа распределенных имитационных моделей обострилась сравнительно недавно и была следствием появления (расширения) задач большой размерности и общего повышения качества и количества доступных вычислительных ресурсов. Существует несколько работ в этой области, но их количество, неуклонно увеличивается.

Выделим два основных направления в указанном анализе. Первое из них используется, когда в определенной проблемной области, для которой строиться модель, существуют традиционные средства моделирования. Например, в теории массового обслуживания, в решении транспортных задач широкое распространение получили программные системы моделирования на базе языка GPSS [51,52,53,147]. Для исследования дискретных динамических систем более широкого класса – аппарат сетей Петри [66,148]. В этом случае существует множество методических рекомендаций, научных аналитических публикаций, которые помогут разработчику выбрать нужные средства и правильно спроектировать модели [149,150,151].

Второе направление исходит из самой природы модели как представителя того или иного методологического подхода или идеи о том, что любая имитационная модель представляет собой программный продукт. К таким работам следует отнести [9,15,74,152,153].

Рассмотрим подход к анализу, приведенный в работах Вознесенской Т.В. [11-14]. Работы указанного автора обобщают модель взаимодействия двух имитационных моделей под управлением оптимистических алгоритмов синхронизации (подраздел 1.3), разработанной D. Mitra и I. Mitrani [74].

В рассматриваемой модели выделим следующие обозначения: Pr – количество процессоров. Каждая частная модель с номером n, $n \in [1,N]$ имеет свое локальное модельное время t^n. Модельное время всей распределенной модели характеризуется вектором $T = \{t^1, t^2, ..., t^N\}$. Модельное время всей модели при произвольном обращении к системе определяется минимальным значением элементов вектора $T_m = \min\limits_{n=1,N} x^n$. Процесс продвижения модельного времени зависит от выбранного алгоритма синхронизации, набора ресурсов, исходных данных эксперимента. Так как заранее все эти параметры неизвестны, то полагается, что процесс изменения модельного времени носит вероятностный характер. Для анализа вводится понятие шага наблюдения за системой моделирования i, $i \in \overline{1,\infty}$. Соответственно модельное время распределенной системой на i-ом шаге характе-

ризуется соответственно значениями T_i и T_{Mi}. Частные модели могут обмениваться сообщениями с вероятностью $a^{kl}\left(k,l \in \overline{1,N}, a^{kl} \in [0,1]\right)$. Затраты времени на внутреннюю работу процессов между двумя соседними шагами характеризуются случайными величинами $\left\{\xi_i^k\right\}, k = \overline{1,N}, \xi_i^k > 0$. Величины a_i^{kl} и ξ_i^k характеризуют распределенную имитационную модель. Выбирая количество процессоров P и закон продвижения модельного времени, можно построить модели функционирования распределенной имитационной модели во времени. Целью построения таких моделей является анализ и сравнение различных алгоритмов синхронизации. Критерий сравнения: тот алгоритм синхронизации лучше, который за одно и то же количество шагов позволит развить имитационную модель до большего значения модельного времени T.

На основании данных характеристик строятся системы уравнений, описывающие развитие моделей во времени. Так для моделей, функционирующих на одном процессоре или синхронных моделей, функционирующих на разных процессорах, система будет иметь вид:

$$\begin{cases} t_{i+1}^n = \begin{cases} t_i^n + \xi_i^n \ , если\ t_i^n = T_i \\ t_i^n , если\ t_i^n \neq T_i \end{cases} \\ t_0^n = 0 \\ n = \overline{1,N} \\ T_{i+1} = \min\limits_{n=\overline{1,N}} (t_{i+1}^n) \end{cases} \tag{1.4}$$

Существуют модификации этих моделей для консервативных и оптимистических методов синхронизации. На основании этих моделей в работах [11,12,13] рассмотрены частные случаи для известных алгоритмов синхронизации, таких как алгоритм Chandy-Misra с передачей нулевых сообщений, классический алгоритм Time Warp, алгоритмы с ограничением оптимизма, временным окном, алгоритмы с периодическим сохранением состояний и др. [154].

Остановимся на недостатках описанных методов и моделей, устранение которых рассматривается в данной работе:

- не учитывается реальное время выполнения частных моделей;

- не учитывается время передачи сообщений между частными моделями (в случае, когда частные модели находятся на удаленных ресурсах, это время может значительно превышать время выполнения определенных частных моделей или служебных);

- рассматривается идеальный случай, когда количество частных моделей равняется количеству процессоров: $N=Pr$;

- не учитываются алгоритмы статического и динамического распределения ресурсов;

- не учитываются количественные параметры частных моделей и вычислительных ресурсов: объемы оперативной, виртуальной внешней памяти, фоновая загрузка процессора, время простоя и т.п.;

- нет достаточно информации о том, каким путем можно получить некоторые параметры моделей: вероятность посылки сообщений, значения флагов состояний на определенном шаге моделирования и т.п.;

- упущен вариант возможной комбинации алгоритмов синхронизации распределенных имитационных моделей.

Указанные недостатки говорят о том, что описанные модели требуют дальнейшего усовершенствования, к которым, в частности, можно отнести работы и украинских ученых [155,156].

1.7 Анализ современных распределенных систем моделирования и GRID-технологий

Последние два десятилетия наблюдается широкий интерес к GRID-технологиям, предоставляющим научному сообществу доступ к большому количеству распределенных ресурсов. GRID дает возможность увеличения размерности имитационных моделей, особенно тех, которые имеют повышенные требования к вычислительной стоимости и большому объему обрабатываемых данных. GRID объединяет различные гетерогенные ресурсы для решения задач большой размерности в науке и инженерии [157].

Многие пакеты программ, рассчитанные на параллельные вычисления адаптируются для работы в GRID. Существует много реализаций распределенных систем имитационного моделирования, например, SPEEDES и PARASOL [158]. Большинство из них поддерживают стандарт (подраздел 1.3). Адаптация существующих систем приводит к ограничению их возможностей, так как программирование симуляторов выполняется в виде переделки существующего программного обеспечения (ПО). Данный подход не дает естественной интеграции системы моделирования в GRID пространство и ведет к созданию новых систем моделирования [144,145,159].

В работе [160] предлагается пересмотреть саму структуру моделирующих программ с целью оптимального включения в инфраструктуру GRID, создав своего рода middleware систем имитационного моделирования и сформулированы требования к системе имитационного моделирования GRID.

В результате проведенного анализа предложено несколько структур систем имитационного моделирования для GRID-инфраструктуры.

Первая структура универсальная, она может использоваться для любых классов имитационных моделей и состава вычислительных ресурсов (рисунок 1.2). В универсальной структуре выделяют управляющую и моделирующую программы. Имитационные модели включаются в моделирующую программу. Взаимодействие между элементами системы моделирования, расположенными на разных вычислительных ресурсах (ВР), выполняется средствами middleware. Система, построенная на основе этой структуры, может работать как в локальной сети ЭВМ и на территориально распределенных компьютерах, так и на удаленных кластерах. Достоинством структуры является то, что написание имитационных моделей при соблюдении стандартного интерфейса частных с системой моделирования не требует от разработчика знаний о распараллеливании процесса моделирования. Недостаток структурной организации – низкая скорость обмена данными между элементами системы. Если время проведения цикла выполнения частных моделей превышает время передачи данных в среде GRID, применение распараллеливания задачи может оказаться нецелесообразным.

Область применения универсальной структуры ограничивается моделями, у которых частные модели требуют значительные временные ресурсы, а связь частных моделей между собой является слабой. В случае, если частные модели имеют небольшую вычислительную сложность, а их взаимосвязи значительны, необходимо стремиться к минимизации времени работы моделирующей программы и middlewar*e*. В качестве решения этой проблемы предлагается исключение из программного обеспечения, функционирующего на удаленных ресурсах BP_1 – BP_N, моделирующей программы.

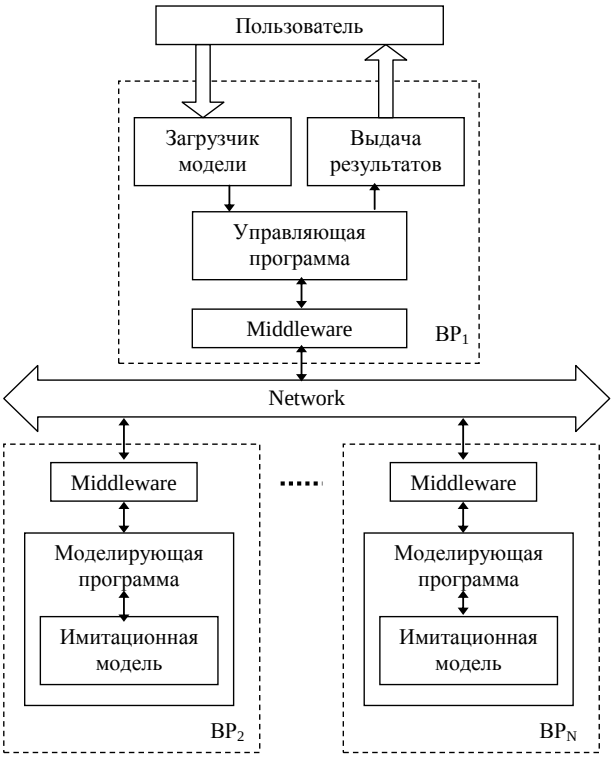


Рисунок 1.2 – Универсальная структура системы имитационного моделирования

В работе [160] предлагается создание контейнера для частной модели, который обладает стандартным интерфейсом как с частной моделью, так и с управляющей программой. С точки зрения объектно-ориентированного программирования (или агентного подхода) реализация выглядит следующим образом: создать базовый класс (объект) – «интерфейс», от которого будут наследоваться все частные модели, содержащий стандартные функции обмена информацией с моделирующей программой и другими частными моделями. Другое решение этой же проблемы – непосредственно внутри частной модели использовать один из стан-

дартных интерфейсов параллельного программирования, например, MPI. Структура системы моделирования, реализующая эту идею, представлена на рисунке 1.3 а.

Создание разнородных распределенных имитационных моделей (сочетающих в себе частные модели, построенные на разных методологических подходах) является более сложным процессом и должно учитывать множество факторов: природу частных моделей (например, дискретные и аналоговые), способы организации моделей (процессный, событийный, транзактный, просмотра активностей), разные механизмы управления модельным временем.

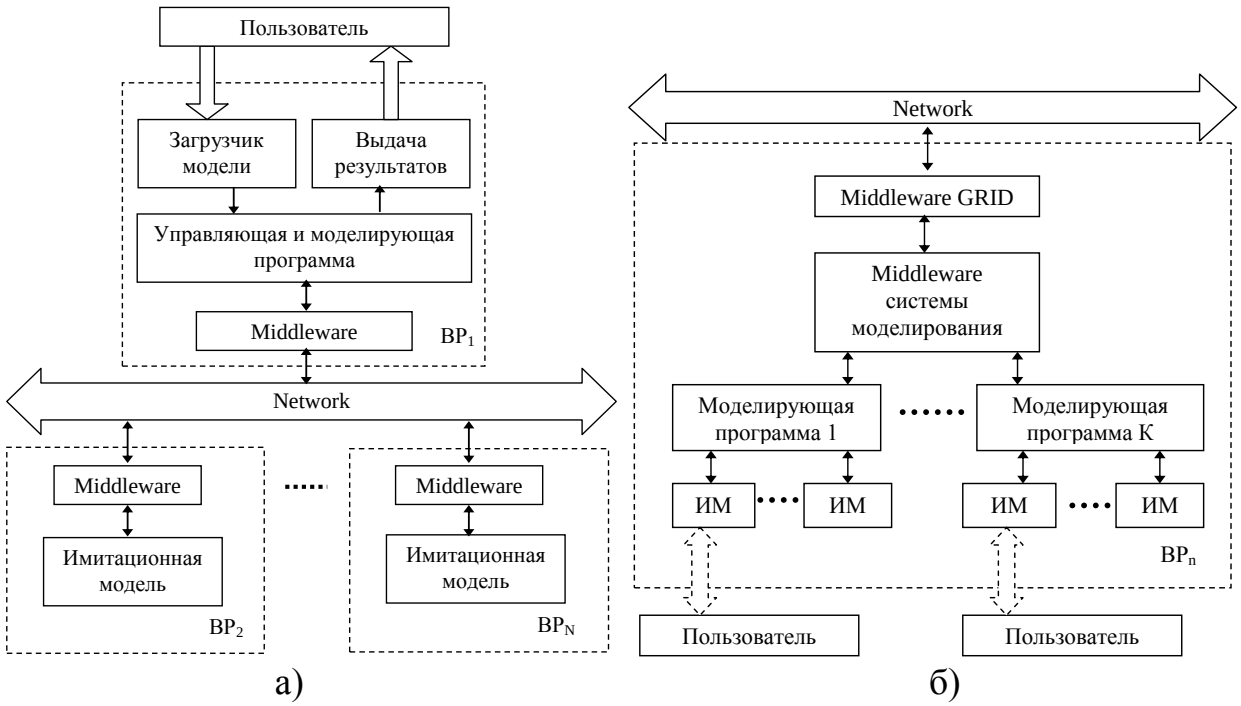


Рисунок. 1.3 – Кластерная (а) и гетерогенная (б) структура распределенной системы имитационного моделирования

Структура распределенной имитационной системы моделирования должна учитывать подобные значимые составляющие имитационной модели:

- вычислительная сложность частных моделей может быть разной. В связи с этим на отдельном ВР могут исполняться несколько частных моделей небольшой сложности и обладающих высокой степенью связности;

- для реализации предідущего пункта в управляющую программу системы

моделирования вводится дополнительный модуль: диспетчер частных моделей, осуществляющий оценку взаимосвязей моделей и их вычислительную сложность;

- для каждого рода частных моделей необходимо наличие своей моделирующей программы;

- для взаимодействия моделирующих программ необходимо создание middleware системы моделирования, согласовывающего работу разнородных моделирующих программ;

- для реализации предыдущего пункта необходимо создание интерфейсов и протоколов взаимодействия, учитывающих специфику имитационного моделирования;

- управление отдельными ВР может осуществляться автоматически (программными или аппаратными средствами) или вручную (пользователями).

Обобщая перечень свойств распределенной системы имитационного моделирования, программное обеспечение, установленное на одном вычислительном ресурсе, приходим к структуре гетерогенной среды распределенного имитационного моделирования (рисунок 1.3 б).

В гетерогенной системе моделирования задания могут корректироваться пользователями во время вычислительного эксперимента. Отдельные ВР могут содержать одну имитационную модель и не содержать управляющих программ. Организация такого рода взаимодействия является сложной технической задачей, работа над которой ведется в настоящее время (например, стандарт HLA, описанный ранее).

Для эффективного применения методов и средств распределенного имитационного моделирования требуется, наряду с формальной моделью имитационной среды, иметь формальную модель GRID-системы. Существующие модели GRID либо ориентированы на решение каких-либо частных задач [161,162], либо имеют неформализованный (поведенческий или статистический) характер [163, 164,165].

1.8 Выводы по анализу предметной области, постановка цели и задач исследования

В результате проведенного анализа на основе обобщения была получена классификация существующих технологий организации имитационного моделирования, которая представлена в приложении Г.

В зависимости от размерности моделируемых объектов или систем, методов моделирования, цели экспериментов и доступных информационных ресурсов, можно выделить три основных технологии имитационного моделирования – последовательная, параллельная и распределенная.

Последовательная технология имитационного моделирования применяется в основном при исследовании простых систем или упрощенных моделей сложных систем, с ограниченным набором элементов и взаимосвязей. Программные средства построения и моделирования при этом рассчитаны на однопроцессорные компьютеры и не требуют использования технологий сетевого обмена данными.

При исследовании сложных систем и процессов, в настоящее время применяют параллельные и распределенные технологии, рассчитанные на использование многопроцессорных и многомашинных аппаратных средств с общей и распределенной памятью. В некоторых случаях, возможно совместное использование как параллельных, так и распределенных технологий.

На основании проведенного анализа, можно сделать вывод о необходимости создания новых и совершенствование существующих технологий имитационного моделирования информационных систем большой размерности с учетом появившихся глобальных вычислительных систем (GRID и облачные системы).

Сформулируем цель диссертационной работы: развитие технологий имитационного моделирования в распределенных вычислительных системах, разработка моделей, методов и информационной технологии анализа распределенных программных моделей GRID-систем, что позволит повысить эффективность распределения ресурсов в процессе распределенного имитационного моделирования.

В соответствие с сформулированной целью для решения единой научной

проблемы в диссертационной работе рассматриваются следующие задачи:

- провести анализ существующих систем распределенного имитационного моделирования и разработать имитационную модель, отражающую динамическое изменение параметров программных моделей: объемов памяти и времени простоя с учетом выбранного метода синхронизации;

- разработать модификации модели для случаев синхронных, консервативных, оптимистических методов синхронизации распределенных имитационных систем моделирования;

- усовершенствовать модель оценки времени выполнения программных моделей с учетом изменений объемов памяти программных моделей и времени время простоя ресурсов;

- на основе созданных моделей разработать методы оценки схем распределения программных моделей на вычислительные ресурсы;

- разработать информационную технологию, алгоритмы и программное обеспечение анализа процессов распределенного имитационного моделирования GRID-систем на основе созданных методов и моделей.

2 РАЗРАБОТКА ФОРМАЛЬНОЙ МОДЕЛИ РАСПРЕДЕЛЕННОЙ ИМИТАЦИОННОЙ СРЕДЫ МОДЕЛИРОВАНИЯ

2.1 Исследование распределенной программной среды моделирования

Целью подраздела является формальное представление распределенной имитационной модели. За основу примем представление имитационной модели, приведенное в [15]. Предлагается следующий алфавит активностей модели:

$$Act = \Phi \cup \Lambda \cup \Delta, (\varphi)_{f=1}^{F} \subseteq \Phi, (\lambda)_{l=1}^{L} \subseteq \Lambda, (\delta)_{d=1}^{D} \subseteq \Delta, \quad (2.1)$$

где Act – алфавит активностей;

Φ – множество внутренних активностей модели;

Λ – множество активностей управления данными модели;

Δ – множество активностей взаимодействия моделей,

F, L, D – определяют количество активностей в соответствующей группе.

Отметим, что такое представление расширяет модель, представленную в [9] множеством активностей управления данными модели, поэтому для сохранения эквивалентности моделей укажем, что множество Λ было выделено из множества Φ по функциональному признаку обработки данных модели, то есть при переходе к модели выполняется $\Lambda \subset \Phi$. Модель в [17] более подробна (мощность алфавита больше), поэтому для установления эквивалентности моделей введем отношения:

$$\{АП, M_T\} \subset \Phi, \{ДМ\} \subseteq \Lambda, \{H\} \subseteq \Delta. \quad (2.2)$$

Таким образом, положения, изложенные в [15,17] будут распространяться и на модель, обладающую алфавитом активностей (2.1).

В имитационной модели выделяют событие (event) e, которое приводит либо к инициализации одной из активностей модели, либо к изменению состояния модели. Также существует понятие процесса P, который является ответом на это событие e. В обозначениях процессной алгебры [16] последовательность

событие→процесс обозначается как $e \rightarrow P$ (за событием e следует выполнение процесса P). В нашем случае непосредственно выполнение процесса реализуется активностями, поэтому будем считать записи $e \rightarrow P$ и $e \rightarrow A$ эквивалентными.

Большинство расширений процессной алгебры включает понятие состояния s объекта (модели, программы, процесса) и понятие перехода объекта из одного состояния (i) в другое (j): $s_i \xrightarrow{A} s_j$. Осуществление перехода происходит под воздействием какой-либо активности A.

Состояние модели s, события e, а также другие элементы модели, которые в программной реализации представлены данными (сегментом, данных, фрагментом памяти и т.п.), будем непосредственно связывать с данными модели dm, то есть $s, e \subset dm$ Действительно, любая программа (в том числе и программная модель) определяется как совокупность данных и кода, который эти данные изменяет в процессе исполнения программы (моделирования). В этом случае понятия изменения состояния модели и изменения данных модели будем считать эквивалентными. Или, другими словами, данные модели являются физической реализацией абстрактного понятия состояния модели. К подобным элементам также относятся транзакты, сообщения, локальные и глобальные переменные, входные и выходные параметры модели т.д. Изменения в модели, приводят к изменению данных под влиянием одной из активностей $dm \xrightarrow{A_i} dm'$.

Таким образом, элементами имитационной модели являются активности A_i и данные (dm_i). Активности A_i осуществляют функционирование на основе данных модели. Рассмотрим более подробно классификацию данных модели.

Выделим среди всей совокупности данных dm_i те, которые доступны только из частной модели (рисунок 2.1). Назовем их локальными данными частной модели dm_i^l. Управляющая программа моделирования (УПМ) выделяет требуемую область памяти данных для активности A_i и передает права на использование этой памяти активностям i-ой частной модели. Все операции с dm_i^l осуществляются исключительно кодом активности из множества A_i. К этой же группе данных можно отнести те данные, которые может использовать УПМ

или мониторинговые программы только для функций чтения, так как они не изменяют состояния частной модели. Обычные dm_i^l размещаются совместно с кодом активности, реализуя, таким образом, свойство инкапсуляции объектно-ориентированного программирования [56].

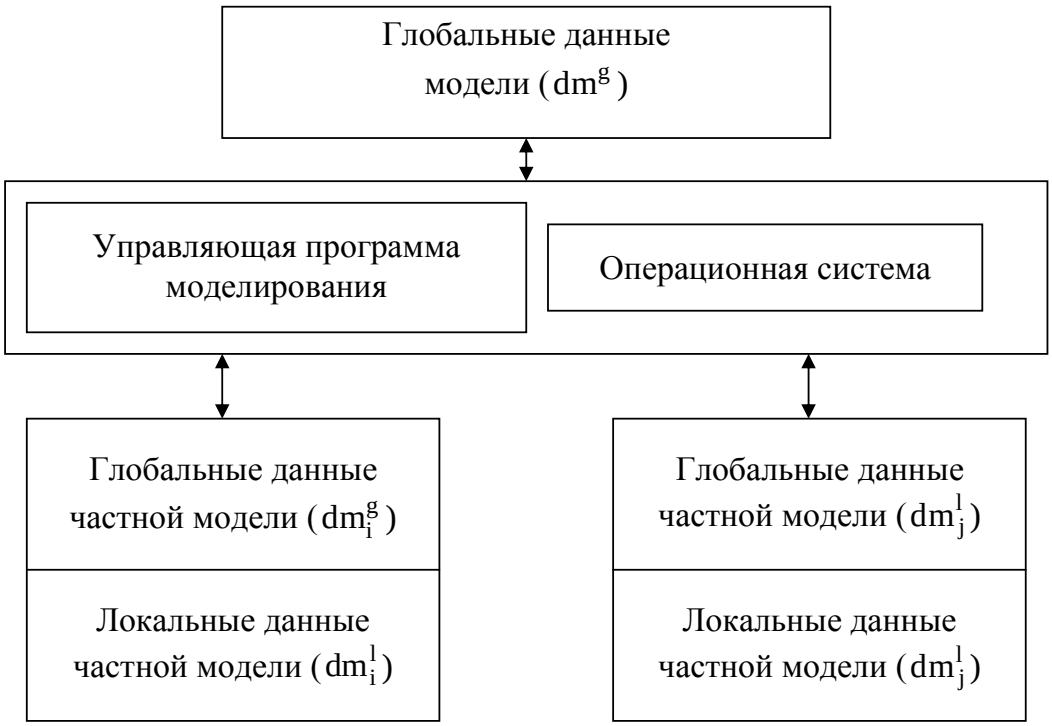


Рисунок 2.1 – Глобальные и локальные данные имитационной модели

Глобальные данные модели используются для реализации двух основных функций:

- хранения параметров внешней (по отношению к частной модели) среды (dm^{gs});

- передаче данных, отвечающих за взаимодействие частных моделей между собой (dm^{gm}).

Управляющая программа моделирования отвечает за реализацию следующих отношений:

$$dm^g = \bigcup_i^n dm_i^g ; dm^g = dm^{gm} \cup dm^{gs} ; dm^{gs} \cap dm^{gm} = \varnothing \qquad (2.3)$$

Нарушение отношений (2.3) считается ошибкой моделирования. Условие не пересечения множеств dm^{gs} и dm^{gm} дает возможность разнести данные,

определяющие состояние частной модели и данные, передаваемые в виде сообщений между частными моделями. Это дает возможность размещения этих данных на разных ресурсах и под управлением разного программного обеспечения (осуществить виртуализацию хранения данных двух типов).

Определим локальные данные частных моделей как данные, существующие только во время выполнения активности. Локальные данные частной модели создаются перед выполнением активности и уничтожаются при ее завершении. Такое определение локальных данных, с одной стороны, соответствует локальным переменным большинства современных систем программирования (локальные данные создаются в стеке подпрограммы на время ее вызова, при условии отсутствия статических локальных переменных), с другой стороны, позволяет модифицировать определение транзакционной модели, введенной в [166], и считать транзакционной моделью ту, которая выполняет все операции с глобальными данными модели через интерфейс менеджера памяти. В результате такого определения исключим из дальнейшего рассмотрения локальные данные частных моделей и подсистем моделирования, считая их префиксными (выполняемыми до текущей активности) и суффиксными (выполняемыми после текущей активности) активностями для любой активности частной модели:

$$\begin{cases} dm_j^g \xrightarrow{A\left(dm_j^l\right).A_i\ .A\left(dm'^l_j\right)} dm'^g_j \\ dm_j^g \cap dm_j^l = \varnothing \\ dm'^l_j \cap dm'^g_j = \varnothing \end{cases}, \qquad (2.4)$$

где A_i – любая активность частной модели;

$A(dm_j^l)$ – активность создания локальных данных для активности частной модели;

$A(dm'^l_j)$ – активность уничтожения локальных данных после выполнения активности частной модели. Как правило, реализация последних двух активностей осуществляется средствами операционной системы.

2.2 Управление состоянием распределенной модели в пространстве модельного времени

В большинстве систем моделирования управление модельным временем осуществляется управляющей программой или одной из активностей частной модели, а поведение имитационной модели состоит в изменении данных модели в зафиксированном, неизменном моменте времени t_k. Следовательно, в один и тот же момент модельного времени существуют два разных состояния данных модели: $dm^{t_k} \xrightarrow{A_i} dm'^{t_k}$. При реализации отката модельного времени из множества состояний модели, зафиксированных в журнале на момент времени t_k, необходимо выбрать одно. В работе [166] приводиться понятие однозначности состояния модели в произвольный момент модельного времени: состояние модели в некоторый момент модельного времени t_k называется однозначным, если с этим моментом времени ассоциируется только одно значение dm^{t_k}.

Утверждение: алгоритмы синхронизации должны осуществлять дамп глобальных данных модели либо до, либо после выполнения всех активностей в момент модельного времени t_k для однозначного определения состояния модели в этот момент времени.

Доказательство: как показано в подразделе 2.1, локальные данные модели не отвечают за состояние модели, а также не хранят данные, передающиеся между частными моделями. Состояние модели полностью определяется глобальными данными модели.

Рассмотрим множество возможных состояний модели в момент времени t_k при условии существования двух активностей, изменяющих состояние модели в этот момент времени:

$$\begin{aligned} dm_1^{gt_k} &\xrightarrow{A_1} dm'^{gt_k}_1, A_1 \in A, dm_1^g \in dm \\ dm_2^{gt_k} &\xrightarrow{A_2} dm'^{gt_k}_2, A_2 \in A, dm_2^g \in dm \end{aligned} \qquad (2.5)$$

Предложены 5 стратегий осуществления дампа памяти.

Стратегия №1: сохранение данных осуществляется перед выполнением

каждой активности реализуя так называемое префиксное действие [20]:

$$\begin{cases} A_1 = A^{save}(t_k).A_1 \\ A_2 = A^{save}(t_k).A_2 \end{cases}$$

В этом случае, согласно (2.5) в журнал будет внесено два разных значения состояния модели: $dm_1^{gt_k}$ и $dm'^{gt_k}_1$. В случае выполнения N активностей, таких возможных значений состояния модели будет соответственно N. В результате, однозначного соответствия состояния модели моменту времени не достигнуто.

Стратегия №2: дамп памяти осуществляется после выполнения каждой активности, реализуя так называемое постфиксное действие:

$$\begin{cases} A_1 = A_1 .A^{save}(t_k) \\ A_2 = A_2 .A_{дамп}(t_k) \end{cases}$$

В этом случае, согласно (2.5) в журнал будет внесено два разных значения состояния модели: $dm_1^{gt_k}$ и $dm'^{gt_k}_1$. В случае выполнения N активностей, таких возможных значений состояния модели будет соответственно N. В результате, однозначного соответствия состояния модели моменту времени не достигнуто.

Стратегия №3: дамп памяти осуществляется до выполнения всех активностей модели в момент времени t_k. Введем два дополнительных состояния модели $dm_0^{gt_k}$ и $dm'^{gt_k}_0$, соответствующие состояниям до и после выполнения всех активностей. Тогда при условии префиксного действия $A^{save}(t_k).(A_1|A_2)$, согласно (2.5) возможны следующие трассы поведения модели: $dm_0^{t_k} \to dm_1^{t_k} \to dm_2^{t_k} \to dm'^{t_k}_0$ или $dm_0^{t_k} \to dm_2^{t_k} \to dm_1^{t_k} \to dm'^{t_k}_0$. Независимо от трассы, будет сохранено только одно состояние модели – $dm_0^{t_k}$. Увеличение количества активностей приведет к увеличению количества трасс, но не повлияет на сохраненное состояние. Следовательно, в данном случае однозначное

соответствие достигнуто.

Стратегия №4: дамп памяти осуществляется после выполнения всех активностей модели в момент времени t_k. Тогда при условии постфиксного действия $(A_1|A_2).A^{save}(t_k)$, согласно (2.5), возможны следующие трассы поведения модели $dm_0^{t_k} \to dm_1^{t_k} \to dm_2^{t_k} \to dm'_0{}^{t_k}$ или $dm_0^{t_k} \to dm_2^{t_k} \to dm_1^{t_k} \to dm'_0{}^{t_k}$. Независимо от трассы, будет сохранено только одно состояние модели – $dm_0^{t_k}$. Увеличение количества активностей приведет к увеличению количества трасс, но не повлияет на сохраненное состояние. Следовательно, в данном случае однозначное соответствие достигнуто.

Стратегия №5: выполнение дампа памяти до и после выполнения активностей $A^{save}(t_k).(A_1|A_2).A^{save}(t_k)$приводит к существованию двух значений состояния модели в один и тот же момент времени: $dm_0^{t_k}$ и $dm'_0{}^{t_k}$. Таким же образом, выполнение дампа памяти до и после выполнения активностей приводит к увеличению количества сохраненных состояний в один и тот же момент модельного времени.

Утверждение доказано. Из данного утверждения следует, что при выполнении неоднократных дампов состояния модели в один и тот же момент времени, порядок выполнения дампов не позволяет решить задачу однозначного состояния модели в фиксированный момент времени. Следовательно, при наличии двух и более дампов памяти в некоторый момент модельного времени t_k не существует однозначности в определении состояния модели в этот момент модельного времени.

2.3 Определение множества активностей управления данными модели

Большинство современных имитационных моделей реализуют свои оригинальные средства управления данными. В работе [10] высказывается предложение создания библиотеки алгоритмов управления состоянием распределенных имитационных моделей. Такой подход сталкивается с трудностями, вызванными разнородностью формального описания различных подходов к моде-

лированию, что показано в разделе 1 диссертационной работы. Один из немногих стандартов в области распределенного имитационного моделирования – HLA, накладывает обязательства на программные модели предоставлять системе моделирования федераты (federates), отвечающие за возврат модели во времени, то есть стандартизованный интерфейс. При этом реализация федерата может быть произвольной (рисунок 2.2). *Ф1, ... , ФN* представляют собой интерфейсные функции, реализуемые федератами. Следует отметить, что HLA стандартизует только интерфейсы федератов с RTI, а не сами реализации. Поэтому корректность исполнения распределенной федерации, т.е. гарантия, что консервативному федерату не будет послано сообщение с временной меткой, меньшей, чем его локальное модельное время, обеспечивается только при условии корректной реализации самого федерата, ответственность за которую несет разработчик федерата.

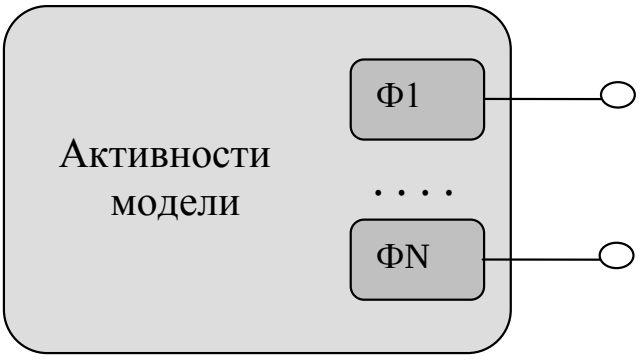


Рисунок 2.2 – Реализация федератов в имитационной модели

На основе представленной выше модели, предлагается новый подход – ввести стандарт на операции с данными модели и реализовать модуль менеджера управления данными модели (рисунок 2.3). Реализация этого действия открывает принципиальную возможность реализовать любой из федератов, управляющих состоянием модели во времени и освободить разработчика модели от необходимости реализации этого интерфейса вручную.

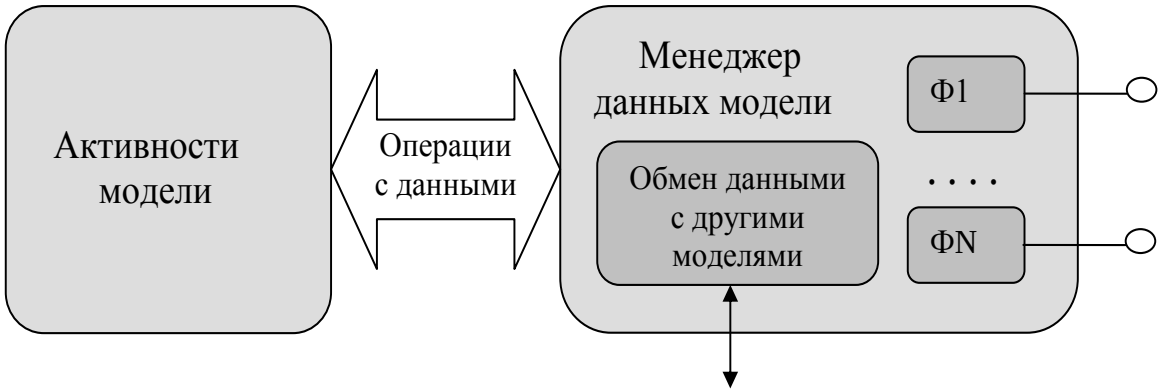


Рисунок 2.3 – Реализация федератов и активностей взаимодействия моделей в менеджере данных

Из множества активностей A^{data}, которые может выполнять менеджер данных модели (множество этих активностей приведено в работе [166]) выберем активность, реализующую сохранение данных модели (дамп памяти) $A^{save}(T_k)$ и активность, позволяющую вернуть состояние модели в один из моментов модельного времени в прошлом $A^{load}(T_k)$:

$$A^{data} = \left\{A^{save}(T_k), A^{load}(T_k)\right\} \tag{2.6}$$

В подразделе 2.4 было доказано, что используя активность сохранения данных модели до или после выполнения поведенческих активностей в фиксированный момент модельного времени можно достигнуть однозначности в определении состояния частных моделей в этот момент модельного времени. Рассмотрим применение менеджера памяти для консервативных и оптимистических алгоритмов синхронизации модельного времени.

2.3.1 Консервативные алгоритмы

Для работы с консервативными федератами RTI имеет интерфейсную функцию *NextEventRequest*() [91]. Консервативный федерат должен иметь следующие интерфейсные функции:

ReflectAttributeValues ();

TimeAdvanceGrant (T).

Когда консервативный федерат завершил все действия, связанные с текущим значением своего локального модельного времени T_k, и готов перейти к выполнению следующего события с временной меткой $T_k^{'}$, он вызывает интерфейсную функцию *NextEventRequest* (запрос следующего события): *NextEventRequest* (T_k'), и ожидает разрешения на посланный запрос.

Инфраструктура RTI, получив вызов *NextEventRequest*, анализирует состояние TSO-очереди. Если TSO-очередь не содержит сообщений с временными метками, меньшими или равными $T_k^{'}$, и RTI гарантирует, что и в дальнейшем не будет таких сообщений, то она вызывает функцию федерата *TimeAdvanceGrant* (разрешение продвижения времени): *TimeAdvanceGrant* (T'_k).

Если TSO-очередь содержит сообщения с временными метками, меньшими или равными $T_k^{'}$, то RTI посылает федерату первое сообщение в очереди. Пусть временная метка этого сообщения T_e ($T_k < T_e < T_k'$), тогда RTI вызывает интерфейсную функцию федерата *TimeAdvanceGrant* (T_e).

Посылка сообщения в стандартном формате осуществляется с помощью вызова интерфейсной функции федерата *ReflectAttributeValues* (передать значения атрибутов).

Получив *TimeAdvanceGrant*, консервативный федерат устанавливает свое локальное модельное время равным $T_k^{'}$ или T_e и выполняет соответствующее событие.

Таким образом, RTI поддерживает консервативную реализацию федерата. Для этого в RTI должен быть реализован один из известных консервативных алгоритмов, например, вычисление LBTS – нижней границы временных меток сообщений. Некоторые реализации RTI требуют, чтобы федераты использовали технику «предсказаний» при посылке сообщений.

Условием корректной реализации федерата является задержка исполнения федерата между вызовом функции *NextEventRequest* и получением разрешения на продвижение модельного времени *TimeAdvanceGrant*. Другим условием корректности исполнения федерата является гарантия, что после исполнения интерфейсной функции *NextEventRequest*(T_k') федерат не пошлет сообще-

ний с временной меткой, равной $T_k^{'}$. Если это ограничение является слишком жестким, то используются другие интерфейсные функции RTI.

Сопоставим вышеприведенным функциям активности для упрощения представления процесса моделирования (выполним переименование):

$$P[f_1]=P[NextEventRequest()/A^{ner}, ReflectAttributeValues/A^{rav}, TimeAdvanceGrant()/A^{tag}]. \quad (2.7)$$

Согласно введенных обозначений введем формальное выражение, отображающее процесс развития частной модели во времени от момента с временной меткой T_k до момента с временной меткой T_e:

$$dm^{T_k} \xrightarrow{\cup A_i} dm'^{T_k} \xrightarrow{A^{ner}(T_k^{'})} dm'^{T_k} \xrightarrow{A^{tag}(T_e)\cdot A^{rav}} dm'^{T_k} \cup dm^{rav}, \quad dm'^{T_k} \cup dm^{rav} = dm^{T_e} \quad (2.8)$$

Предлагается, выполнять активность $A^{save}(T_k^{'})$, сохраняющую состояние модели (сохранение данных модели) непосредственно после завершения выполнения всех поведенческих активностей частной модели в момент времени T_k. Так как к этому моменту времени частная модель определилась со следующим моментом времени, к которому готова перейти, будем указывать в качестве временной метки сохраняемых данных модели значение $T_k^{'}$.

При этом, активность A^{ner} становиться постфиксной активностью для активности $A^{save}(T_k^{'})$ и реализуется автоматически менеджером памяти:

$$dm^{T_k} \xrightarrow{\cup A_i \,.A^{save}(T_k^{'}).A^{ner}(T_k^{'})} dm'^{T_k} \xrightarrow{A^{tag}(T_e).A^{rav}} dm'^{T_k} \cup dm^{rav}, \quad dm'^{T_k} \cup dm^{rav} = dm^{T_e} \quad (2.9)$$

В менеджере памяти в качестве временной метки при вызове активности A^{tag} записывается значение T_e, которое в нашем случае может быть меньшим или равным временной метки $T_k^{'}$ $\left(T_k < T_e < T_k^{'}\right)$. Таким образом, сохранение состояния модели производиться не в конце цикла исполнения активностей с

временной меткой T_k, а в начале цикла исполнения активностей с временной меткой T_e. Активности $A^{tag}(T_e)$и A^{rav} по своей сути приводят к изменению данных глобальных данных модели, то есть изменяют значение планируемого следующего момента времени с $T_k^{'}$ на T_e и изменяют атрибуты следующего события в момент времени T_e. Последние являются частью множества данных модели dm^{T_e} . При активации модели в момент с временной меткой T_e, поведенческие активности модели считывают из глобальных данных модели новые значения модельного времени и атрибутов события в этот момент времени. Таким образом, если консервативный алгоритм не реализует каких-либо специфических действий в интерфейсных функциях федерата, то эти функции можно полностью реализовать в менеджере памяти.

Разобьем все множество активностей на группы по принадлежности одному из объектов моделирования: RTI, менеджеру памяти и частной модели:

$$\begin{cases} RTI: \left\{A^{ner}(T_k)\right\} \\ A_{onm}^{data} = \left\{A^{save}(T_k), A'^{load}(T_k), A^{tag}(T_e), A^{rav}\right\} \\ M: \left\{A^{tag}(T_e) + \varnothing, A^{rav} + \varnothing\right\} \end{cases} \quad (2.10)$$

где "+" – операция альтернативной композиции.

Наличие среди активностей менеджера памяти активности $A^{load}(T_k)$ не является обязательным, так как консервативные алгоритмы не поддерживают откаты модельного времени. Однако, данная активность может быть полезной при реализации функций обеспечения надежности (восстановления) процесса моделирования или в задачах многовариантного расчета.

2.3.2 Оптимистические алгоритмы

Стандарт HLA определяет федераты для оптимистических алгоритмов управления модельным временем в виде двух интерфейсных функции [151]:

ReflectAttributeValues ();

FlushQueueGrant (T).

В момент завершения всех активностей в текущий момент модельного времени T_k, федерат вызывает функцию *FlushQueuetRequest*(T_k), чем сообщает системе RTI (Runtime Infrastructure) о готовности продолжать моделирование и увеличить значение модельного времени. RTI передает оптимистическому федерату очередь сообщений с временными метками TSO (Time Stamp Order), существующую в системе на этот момент времени. Передача осуществляется путем вызова интерфейсной функции *ReflectAttributeValues*(). Далее федерат продвигает свое модельное время на основе этой промежуточной упорядоченной очереди сообщений. Если федерат обнаруживает сообщение, с временной меткой меньшей текущего значения локальных часов, он самостоятельно осуществляет откат, рассылая при необходимости антисообщения вызовом функции RTI *Retract* (). Система RTI время от времени посылает федерату значение GVT (Global Virtual Time) при помощи функции FlushQueueGrant (GVT). Сопоставим описанным функциям активности для упрощения представления процесса моделирования (выполним переименование):

$$P[f_2]=P[FlushQueuetRequest/A^{fqr},ReflectAttributeValues/A^{rav},\ FlushQueueGrant\ /A^{fqg},Retract/A^{retr}\ ]. \quad (2.11)$$

Допустим, что сохранение данных частной моделью осуществляется после выполнения всех активностей в текущий момент времени T_k. Следовательно, с точки зрения логики работы RTI, выполнение активности $A^{save}(T_k)$ из множества (2.6) может являться префиксным процессом к выполнению функции *FlushQueuetRequest(*T_k*)*, или, другими словами, совершение моделью сохранение состояния модели влечет за собой информирование системы RTI о завершении всех действий частной модели в момент модельного времени T_k.

Менеджер данных обязан предоставить RTI функцию *ReflectAttributeValues()*, которая вызывается для передачи частной модели очереди сообщений. При обнаружении сообщения с временной меткой, меньшей текущего значения, менеджер памяти обязан осуществить откат. Представление данного процесса:

$$dm^{T_k} \xrightarrow{\left(A^{save}(T_k).A^{fqr}.A^{rav}\right)} dm^{T_k} \cup dm_{TSO} \xrightarrow{A^{TSO}}$$

$$\xrightarrow{A^{TSO}} \begin{cases} dm^{T_k}, если\ T_{min(TSO)} \geq T_k \\ \xrightarrow{A^{load}\left(T_{min(TSO)}\right)} dm^{T'_{min(TSO)}}, если\ T_{min(TSO)} < T_k \end{cases}, \qquad (2.12)$$

где A^{TSO} – активность менеджера памяти, осуществляющая просмотр очереди сообщений;

dm_{TSO} – сама очередь сообщений;

$T_{min(TSO)}$ – минимальное значение временной метки сообщений в очереди TSO.

Выражение (2.8) показывает, что при соблюдении частной моделью правила выполнения дампа состояния в конце цикла моделирования, менеджер памяти способен совершить откат к состоянию частной модели в конкретный момент в прошлом самостоятельно, без участия самой модели. Обозначение $dm_{T'_{min(TSO)}}$ в выражении (2.6) подчеркивает тот факт, что в момент модельного времени $T_{min(TSO)}$ частная модель могла не совершать действий. В этом случае откат производиться до ближайшего дампа состояния частной модели $T'_{min(TSO)} \leq T_{min(TSO)}$. Активности A^{rav} является активностью модели, причем она может поддерживаться или нет, что не влияет на функциональность менеджера памяти по продвижению модельного времени.

Подводя итог, разобьем все множество активностей на группы по принадлежности одному из объектов моделирования: RTI, менеджеру памяти и частной модели:

$$\begin{cases} RTI: \left\{A^{fqr}, A^{retr}\right\} \\ A^{data}_{onm} = \left\{A^{save}(T_k), A'^{load}(T_k), A^{rav}, A^{TSO}, A^{fqg}\right\} \\ M: \left\{A^{rav} + \varnothing, A^{fqg} + \varnothing\right\} \end{cases} \qquad (2.13)$$

Вызов активности A^{fqg} может быть проигнорирован либо использован менеджером памяти для удаления всех дампов памяти, временная метка кото-

рых меньше GVT (с целью экономии памяти).

Из выражения (2.9) видно, что разработчику частной модели из всего множества федератов, в случае необходимости, надо реализовать только две активности – A^{rav} и A^{fqg}, и то при условии, что они необходимы для обеспечения специфической логики работы всей модели. Остальные активности являются стандартными, либо для RTI, либо для предложенного менеджера памяти.

2.4 Определение множества активностей менеджера памяти, реализующего интерфейсы HLA для обеспечения синхронизации модельного времени

Множество активностей менеджера памяти A^{data} получается путем объединения подмножеств активностей обработки данных модели (2.10) и (2.13) соответствующих активностям, реализующим консервативные и оптимистические федераты:

$$A^{data} = A^{data}_{опт} \cup A^{data}_{конс} = \left\{A^{save}(T_k), A'^{load}(T_k), A^{rav}, A^{tag}(T_e), A^{TSO}, A^{fqg}\right\} \qquad (2.14)$$

Отметим, что пересечение указанных двух множеств не есть пустое множество:

$$A^{data}_{опт} \cap A^{data}_{конс} = \left\{A^{save}(T_k), A'^{load}(T_k), A^{rav}\right\} \qquad (2.15)$$

то есть, ряд активностей используется как в оптимистических, так и в консервативных алгоритмах синхронизации модельного времени.

2.5 Определение множества активностей обмена данными частных моделей

Распределенное имитационное моделирование, в общем случае, подразумевает параллельное исполнение частных моделей, которые в процессе функционирования обмениваются друг с другом некоторой информацией. Это могут быть сообщения (которые поддерживаются наиболее известными интерфейсами параллельного программирования, например, MPI – Massage Passing

Interface [152]), взаимодействие по входными или выходным данным модели, объекты синхронизации при использовании разделяемых ресурсов, объекты сетевого взаимодействия сеансового уровня (сокеты) и т.д.

Так как любая подобная информация программно реализуется посредством данных модели, то введем в множество активностей менеджера памяти дополнительные активности для реализации функций обмена данными между частными моделями. Стандарт HLA не описывает данные элементы имитационных моделей, то есть подразумевается, что эта функциональность реализуется самостоятельно создателями моделей в произвольной форме. Формализация же этой функциональности открывает дополнительные возможности для реализации методов анализа эффективности применения алгоритмов синхронизации распределенных имитационных моделей.

Процесс передачи информации взаимодействия частных моделей содержит два этапа: подготовка данных в области локальных или глобальных данных модели (создание и заполнения буфера данных) и выполнение интерфейсной функции, непосредственно передающей данные с указанием в параметрах функции адреса буфера и объема передаваемых данных.

Введем следующее ограничение: буфер данных *buf* для передачи информации между частными моделями и адресная информация *addr* должны быть создан в глобальных данных модели: $\{buf, addr\} \subset dm^g$.

Введем множество активностей A^s менеджера памяти для реализации пересылки данных между частными моделями:

$$A^s = \left\{A^{send}(buf, addr), A^{asend}(buf, addr), A^{hsend}(buf, addr)\right\} \subset A^{data} \qquad (2.16)$$

где $A^{send}(buf, addr)$ – синхронная передача данных с ожиданием подтверждения завершения процесса передачи информации;

$A^{asend}(buf, addr)$ – асинхронная передача данных без ожидания подтверждения завершения процесса передачи информации;

$A^{hsend}(buf, addr)$ – полусинхронная передача, которая заключается в одновременной передаче всех сообщений поведенческих активностей, накоплен-

ных в фиксированный момент модельного времени.

Перечисленные активности должны иметь соответствующие им активности менеджера памяти на принимающей стороне:

$$A'^{s}\left\{A'^{send}\left(buf,addr\right),A'^{asend}\left(buf,addr\right),A'^{hsend}\left(buf,addr\right)\right\}\subset A^{data} \qquad (2.17)$$

Активности частной модели могут использовать как все активности передачи данных, так и любое их подмножество.

Важно отметить, что операция передачи по сети данных модели включает в себя несколько уровней представления данных (рисунок 2.4).

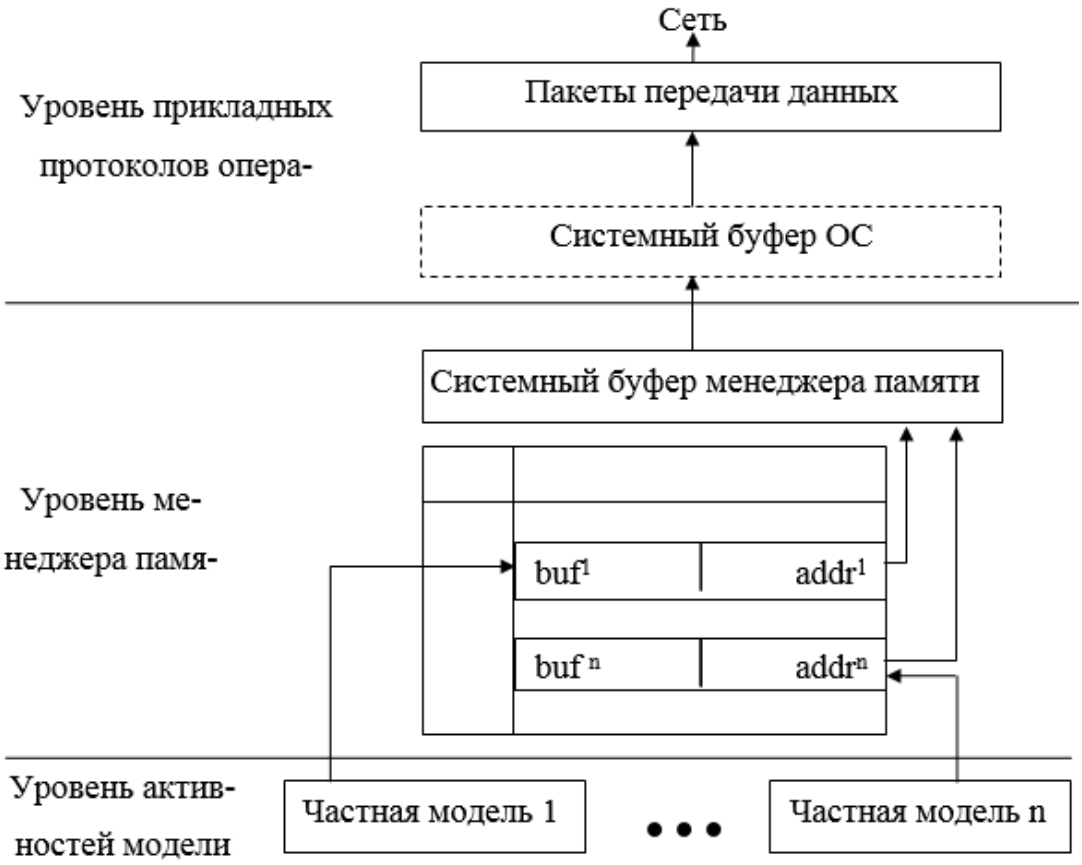


Рисунок 2.4 – Передача информационных данных частных моделей в сети

Частные модели осуществляют подготовку данных в буфере данных, расположенном в адресном пространстве глобальных данных модели. Затем вызываются активности из множества (2.16), осуществляющие копирование данных в системный буфер интерфейса обмена данными по сети (в нашем случае – это менеджер памяти). После чего происходит обращение к функциям (или объек-

там) операционной системы, осуществляющим формирование пакетов (прикладного, представительного или сеансового уровней модели сети).

Перспективным является использование на смежных уровнях одного и того же участка памяти для реализации буфера данных, что даст возможность сократить временные затраты на копирование данных.

В разделе 3 данное представления процесса передачи информации будет отражено в методике оценки затрат памяти на организацию обмена информацией между активностями частных моделей.

2.6 Выводы по разделу

В разделе 2 предложено новое формальное представление распределенной имитационной модели на основе выделения совокупности активностей модели и данных модели, отражающих изменение состояний модели во времени и информационные сообщения между активностями частных моделей. На ее основе показана возможность управления распределенными имитационными моделями.

Доказано утверждение об однозначности определения состояния модели в пространстве модельного времени при реализации различных алгоритмов синхронизации.

Введено понятие менеджера памяти модели. Показана возможность построения на основе введенных активностей консервативных и оптимистических алгоритмов синхронизации распределенных имитационных моделей на основе стандарта HLA, рассмотрены особенности реализации федератов на основе функций менеджера памяти. Показано, что при использовании менеджера памяти для управления состоянием в пространстве модельного времени, возможна автоматическая реализация федератов стандарта HLA и федератов распределенной системы моделирования RTI, поддерживающих консервативные и оптимистические алгоритмы синхронизации, средствами менеджера памяти.

Рассмотрен процесс передачи сообщений между частными моделями с точки зрения управления потоками данных, осуществляемого менеджером памяти при помощи одного из протоколов операционной системы.

Перспективным является использование на смежных уровнях одного и того же участка памяти для реализации буфера данных, что даст возможность сократить временные затраты на копирование данных.

Полученные результаты открывают возможности унификации программных имитационных моделей, ведущие к упрощению и ускорению процесса их построения на основе простого интерфейса, основанного на управлении данными модели, а также создает предпосылки к созданию эффективных методов анализа процессов распределенной имитации.

3 МОДЕЛИ И МЕТОДЫ ИССЛЕДОВАНИЯ РАСПРЕДЕЛЕННЫХ ИМИТАЦИОННЫХ МОДЕЛЕЙ

3.1 Параметры критериев оценивания распределенной имитационной модели

Одним из основных критериев оценивания эффективности модели является стоимость проведения имитационного эксперимента. Часто в основе его определения лежит время имитации. Чем меньше время эксплуатации вычислительных ресурсов, тем меньше стоимость эксперимента.

В подразделе 1.6 приведено описание существующих методов анализа распределенных имитационных моделей. Показано, что большинство методов опираются на ограниченное множество параметров, к которым относят соотношение вычислительной мощности задачи/ресурса, глубина продвижения модельного времени за фиксированное число шагов моделирования и т.п. Во всех описанных методах значительным недостатком является допущение о мгновенной реализации активностей модели, а именно: мгновенной (с точки зрения анализа) считается передача данных между распределенными частными моделями, сохранение состояния модели, откат модели во времени. И если временем выполнения поведенческой активности можно пренебречь (действительно, при любом способе организации имитационного моделирования подпрограмма поведения выполняется за приблизительно одинаковое время), то количество передаваемых, сохраняемых и восстанавливаемых данных, а также среда (аппаратная и программная), обеспечивающая эти операции, в значительной степени влияют на реальное время моделирования.

Предлагается расширить параметры, согласно которым формируются критерии эффективности процесса имитации, и включить в их множество:

- время сохранения состояния модели (дамп памяти) t^s;

- время передачи данных (сообщения) между распределенными моделями t^m;

- время восстановления данных модели (откат модели) $t^{s'}$;

- объем памяти, требуемый для хранения активностей модели V^{act};

- объем памяти данных модели V^{dm};

- объем памяти хранения дампов состояния модели V^{s}.

Для формального определения параметров введем оператор *sizeof(dm)*, определяющий размер памяти *(V, байт)*, необходимой для хранения элемента данных dm_i и оператор *time(A)*, определяющий реальное время выполнения некоторой активности (*T, секунд*). Очевидными являются следующие выражения:

$$\sum_{i=1}^{N} sizeof\left(dm_i\right) = sizeof\left(dm\right), если\ dm_i \cap dm_j = \varnothing, \forall i, j \in \overline{1,N}, i \neq j; \quad (3.1)$$

$$\sum_{i=1}^{N} sizeof\left(dm_i\right) = sizeof\left(dm\right), если\ \exists dm_i \cap dm_j \neq \varnothing, \forall i, j \in \overline{1,N}, i \neq j; \quad (3.2)$$

$$time\left(A_{дамп}\right) \geq time\left(A_{фикс}\left(dm_i\right)\right). \quad (3.3)$$

Выражение (3.1) говорит о том, что если все частные модели проведут сохранение своего состояния, то размер памяти, который потребуется для этого будет равен размеру памяти, необходимому для сохранения состояния всей модели при условии, что данные модели не пересекаются. В противном же случае, эта память будет больше (выражение (3.2)). Выражение (3.3) подчеркивает тот факт, что время, затрачиваемое системой моделирования на выполнение дампа памяти в общем случае больше времени, которое затрачивается на сохранение состояния частной подмодели.

Следует отметить, что операторы *sizeof* и *time* являются доступными в большинстве языков программирования и поддерживаются современными операционными системами. Кроме этого, основные активности по сохранению, изменению и передачи данных моделей согласно результатов, приведенных в разделе 2, выполняются менеджером памяти. Следовательно, эти характеристики могут быть получены программным путем до создания распределенной имитационной модели. Параметры, которые зависят от реализации распределенной имитационной модели (например, время выполнения и размер памяти поведенческих активностей модели) могут быть заданы тремя способами: эвристическим (на основе знания эксперта), экспериментальным, после создания самих

поведенческих подпрограмм либо при помощи автоматических средств анализа программ.

3.2 Разработка имитационной модели процесса распределенного дискретно-событийного моделирования с учетом изменений объемов памяти и времени исполнения

В процессе функционирования распределенная имитационная модель затрачивает процессорное время на исполнение своих активностей. Активности модели могут вызывать активности по работе с данными, которые, в свою очередь, увеличивают процессорное время по обслуживанию частной модели. При этом, на каждом шаге моделирования может произойти увеличение объема памяти, занимаемого частной моделью. С учетом выражения (1.4), которое определяет изменение модельного времени, рассмотрим изменение параметров, учитывающих динамику изменения объемов памяти частных имитационных моделей и реальное время, затрачиваемым на выполнение активностей менеджера памяти по выполнению операций с данными. Система выражений, приведенная ниже является математической моделью изменения указанных параметров в пространстве модельного времени.

При продвижении модельного времени t_i^n n-ой модели на i-ом шаге моделирования происходит сохранение состояния модели, что приводит к росту памяти, занимаемой данными программной частной моделью V_i^n. На операции с памятью затрачивается процессорное время TR_i^n, которое состоит из времени сохранения состояния модели t_i^s, времени передачи сообщения от частной модели n частной модели k (t_i^m). Время t_i^m в общем случае является функцией, зависящей от поведения активности передачи данных δ_i^{nk} и размера передаваемых данных ($buff_i^{nk}$). В свою очередь время, затраченное на выполнение всей имитационной модели на i-ом шаге моделирования ΔTR_i определяется как максимум среди суммы времен исполнения поведенческой активности φ_i^n и време-

ни выполнения операций с данными TR_i^n.

В результате, получаем математическую модель, описывающую динамику изменения параметров распределенной имитационной модели, учитывающую изменения объемов памяти и реальное время исполнения модели:

$$\begin{cases} TR_0 = 0; \\ TR_0^n = 0; \\ \Delta TR_i^n = \begin{cases} t_i^s, если\, t_i^n = T_i \\ 0, если\, t_i^n \neq T_i \end{cases} + \sum_{k=1}^{N} a^{nk} \cdot t_i^m(\delta_i^{nk}, sizeof(buff_i^{nk})); \\ TR_{i+1}^n = TR_i^n + \Delta TR_i^n; \\ TR_{i+1} = TR_i + \max_{n=\overline{1,N}}(time(\varphi_i^n) + \Delta TR_i^n) + TR_i^M; \\ V_0^n = sizeof(\Lambda^n \cup \Delta^n); \\ V_0 = \sum_{n=1}^{N} V_0^n; \\ \Delta V_i^n = \begin{cases} V_i^{ns}, если\, t_i^n = T_i \\ 0, если\, t_i^n \neq T_i \end{cases}; \\ V_{i+1}^n = V_i^n + \Delta V_i^n; \\ V_{i+1} = V_i + \sum_{n=1}^{N} \Delta V_i^n; \\ T_0 = 0; \\ T_{i+1} = \min_{n=\overline{1,N}}(t_{i+1}^n); \\ n = \overline{1,N}. \end{cases} \quad (3.4)$$

где TR_i – реальное время, затраченное на выполнение i шагов процесса моделирования всей имитационной модели;

TR_i^n – реальное время, затраченное n-ой частной моделью на i-м шаге моделирования на работу с данными;

t_i^s – время сохранения состояния модели на i-м шаге моделирования;

t_i^m – время передачи данных;

$time\left(\varphi_i^n\right)$ – время исполнения поведенческой активности φ_i^n;

a^{nk} – вероятность посылки сообщения от частной модели n к частной модели k;

N – количество частных моделей;

TR_i^M – время работы служб системы моделирования на i-ом шаге моделирования;

t_i^n – локальное модельное время n-ой частной модели на i-ом шаге моделирования;

T_i – глобальное модельное время на i-ом шаге моделирования;

ξ_i^n – значение времени, характеризующее внутреннюю работу процессов между двумя соседними шагами моделирования;

V_i^n – объем памяти, занимаемый данными n-ой частной моделью на i-ом шаге моделирования;

V_i^{ns} – объем дампа памяти для n-ой частной модели на i-ом шаге моделирования;

ΔV_i^n – приращение объема памяти n-ой частной модели на на i-ом шаге моделирования;

V_i – объем памяти, занимаемой всей распределенной имитационной моделью на i-ом шаге моделирования;

TR_0, TR_0^n, t_0^n, V_0^n, V_0 – начальные значения соответствующих параметров.

Параметры T_i, V_i, TR_i вычисляются независимо друг от друга. Следовательно, возможна реализация параллельного вычисления этих параметров. Кроме того, отсутствие количественных данных о каком-либо члене выражения (3.4) (например, времени, которое характеризует внутреннюю работу процессов между двумя соседними шагами моделирования), повлечет за собой невозможность вычисления только одного из них.

3.3 Разработка модификации модели процесса распределенного дискретно-событийного моделирования под управлением консервативных алгоритмов синхронизации

Особенностью консервативных алгоритмов синхронизации модельного времени распределенных имитационных моделей является недопущение продвижения модельного времени частных имитационных моделей при вероятности появления события с меньшим значением временной метки. Следовательно, при стандартной реализации этих алгоритмов, отсутствует необходимость в выполнении дампа памяти частных моделей. Изменение объемов памяти, вызванное алгоритмическими изменениями параметров модели, как правило, отсутствует, либо незначительно (по сравнению с размерами самой программной модели). В таком случае, в выражение (3.4) не учитываются параметры V_i^{ns} (объем памяти для сохранения состояния модели) и t_i^s (время сохранения модели), а выражение (3.4) примет вид:

$$\begin{cases} TR_0 = 0; \\ TR_0^n = 0; \\ \Delta TR_i^n = \sum\limits_{k=1}^{N} a^{nk} \cdot t_i^m(\delta_i^{nk}, sizeof(buff_i^{nk})); \\ TR_{i+1}^n = TR_i^n + \Delta TR_i^n; \\ TR_{i+1} = TR_i + \underset{n=1,N}{max}(time(\varphi_i^n) + \Delta TR_i^n) + TR_i^M; \\ V_0^n = sizeof(\Lambda^n \cup \Delta^n); \\ V_0 = \sum\limits_{n=1}^{N} V_0^n; \\ V_{const} = V_0; \\ T_0 = 0; \\ T_{i+1} = \underset{n=1,N}{min}(t_{i+1}^n) \\ n = \overline{1,N}. \end{cases} \quad (3.5)$$

Реализация некоторых целей эксперимента, например, многовариантный анализ или обеспечение ветвления алгоритмов анализа в прошлом, могут потребовать сохранения состояния программных моделей. В этом случае, указанные параметры могут быть оставлены, но условие их включения на определенных шагах моделирования должно определяться логикой эксперимента априорно, до начала моделирования.

В консервативных алгоритмах частные модели простаивают, ожидая разрешения на продвижение модельного времени. Вследствие этого, ресурс, выделенный для обслуживания этой частной модели, простаивает. Некоторые алгоритмы распределения ресурсов учитывают данный факт при балансировке нагрузки.

Добавим к математической модели процесса имитации выражение, учитывающее время простоя частной модели:

$$\begin{cases} TP_0^n = 0 \\ TP_{i+1}^n = \begin{cases} TP_i^n, если\ t_i^n = T_i \\ TP_i^n + \max\limits_{n'}\left(time\left(\varphi_i^{n'}\right) + \Delta TR_i^{n'}\right), n' = \overline{1,N} / n' = n, если\ t_i^n \neq T_i \end{cases} \end{cases} \quad (3.6)$$

где TP_i^n – время простоя n-ой частной модели на момент достижения i-го шага моделирования;

n' – индекс множество активностей модели, за исключением частной модели с индексом n.

Отметим, что при реализации вычисления данного выражения, поиск максимума можно вести на множестве частных моделей, активированных на i-ом шаге моделирования.

Таким образом, получена математическая модель функционирования распределенной имитационной модели с консервативным алгоритмом синхронизации частных моделей (выражение (3.5)) с учетом простоя вычислительных ресурсов, на которых произведено распределение имитационной модели (выражение (3.6)).

3.4 Разработка модификации модели процесса распределенного дискретно-событийного моделирования под управлением оптимистических алгоритмов синхронизации

В общем случае, для анализа оптимистических алгоритмов, можно использовать систему выражений (3.4). Однако, ряд модификаций оптимистических алгоритмов приводят к ее изменению. Особенностью оптимистических алгоритмов, описанной в пункте 2.3.2 является возможность откатов моделей в прошлое, для организации которого необходимо сохранение состояний модели (выполнение дампов памяти). Это приводит к постоянному росту памяти в моменты активизации частной модели. Выделим из выражения (3.4) часть, отвечающую за рост объема памяти:

$$\begin{cases} \Delta V_i^n = \begin{cases} V_i^{ns}, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases} \\ V_{i+1}^n = V_i^n + \Delta V_i^n \\ V_{i+1}^n = V_i + \sum_{n=1}^{N} \Delta V_i^n \end{cases}, \qquad (3.7)$$

При большом количестве шагов моделирования увеличение объема памяти приводит к переполнению физической или виртуальной памяти вычислительного ресурса.

Так, при использовании наиболее известного оптимистического алгоритма, разработанного Джефферсоном [96] (Time Warp), в момент получения частной моделью события, имеющего временную отметку меньшую, чем уже обработанные события, частная модель выполняет откат и обрабатывает эти события повторно в хронологическом порядке. Откатываясь назад, частная модель восстанавливает состояние, которое было до обработки событий (используются контрольные дампы памяти) и отказывается от сообщений, отправленных «откаченными» событиями. Следовательно, для частной модели наблюдается постоянный рост памяти за счет выполнения дампа памяти и формирования но-

вых сообщений. Во время отката необходимо не только вернуть состояние частной модели, что соответствует загрузке одного из контрольных дампов памяти, но и очистить память, занимаемую дампами памяти, созданными после контрольного.

Механизм антисообщений в алгоритме Time Warp позволяет уничтожать в памяти необработанные сообщения, созданные после контрольного дампа памяти, либо, если частная модель-получатель уже обработала его, активирует процесс отката для частной модели-получателя.

Результатом реализации этого алгоритма является постоянное изменение объемов памяти частной модели. Для отражения этих процессов предлагается модифицировать выражение (3.6) следующим образом:

$$\begin{cases} V_0^n = sizeof\left(\Lambda^n \cup \Delta^n\right); \\ V_0 = \sum\limits_{n=1}^{N} V_0^n; \\ \Delta V_i^n = \begin{cases} V_i^{ns} + V_i^{nm} - \sum\limits_{j=1}^{M^n} \tau_j V_{ij}^{nm}, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases}; \\ V_{i+1}^n = V_i^n + \Delta V_i^n - \sum\limits_k \Delta V_k^n, \forall k \in [q,i]; \\ V_{i+1}^n = V_i + \sum\limits_{n=1}^{N} \Delta V_i^n, \end{cases} \tag{3.8}$$

где V_i^{nm} – объем сообщений (или антисообщений), возникающих в результате выполнения активностей частной модели на i-м шаге моделирования и помещенных в очередь сообщений;

M^n – количество сообщений в локальной очереди сообщений n-ой частной модели;

V_{ij}^{nm} – объем памяти, занимаемый j-ым сообщением в этой очереди;

$\tau_j \in [0,1]$ – бинарный признак удаления сообщения из очереди;

q – шаг моделирования, до состояния которого частная модель осуществ-

ляет откат ($q < i$); индекс k перечисляет все удаляемые дампы памяти, совершенные для данной частной модели между шагами моделирования i и q.

Признак удаления сообщения из очереди τ_j будет равен 1 в следующих случаях:

- при удалении обработанного сообщения из очереди, то есть если временная метка сообщения равна T_i и сообщение обслужено частной моделью, для которого это сообщение предназначалось;

- при удалении сообщений во время отката частной модели для всех сообщений, чья временная метка больше временной метки, соответствующей откату.

Не смотря на то, что в выражении (3.7) параметр V_{i+1}^n уменьшается в случае отката модельного времени, в динамике, при увеличении GVT этот параметр растет в силу того, что в памяти сохраняются все дампы памяти, выполненные до этого момента времени (GVT).

В связи с этим, предложено ряд модификаций оптимистических алгоритмов синхронизации, ограничивающих рост памяти.

Для Time Warp систем, для которых отмечается чрезмерное «забегание» вперёд частных моделей, что ведет к серьёзным затратам памяти и длинным откатам, применяют ограничения этих «забеганий». Вводятся «перемещаемые» в модельном времени окна, определяемые как временной интервал *[GVT, GVT+T_w]*, где T_w – это параметр, задаваемый пользователем либо вычисляемый в процессе имитации. Частные модели обрабатывают только те события, временные метки которых находятся в данном временном интервале. Ограничение на продвижение модельного времени частной имитационной модели приводит к простою вычислительного ресурса, что обсуждалось при рассмотрении консервативных алгоритмов синхронизации распределенных имитационных моделей. В этом случае становиться возможным применения выражения (3.6) для оценки простоя вычислительного ресурса. В этом случае можно использовать выражение (3.6) с модифицированным условием, учитывающим существование временных окон:

$$\begin{cases} TP_0^n = 0; \\ TP_{i+1}^n = \begin{cases} TP_i^n, если \left(t_i^n = T_i \text{ или } t_i^n GVT + T_w\right) \\ TP_i^n + \max\limits_{n'}\left(time\left(\varphi_i^{n'}\right) + \Delta TR_i^{n'}\right), n' = \overline{1,N} / n' = n, если\ t_i^n \neq T_i \end{cases} \end{cases}. \quad (3.9)$$

Еще одна модификация метода заключается в задержке отправки сообщения до того момента, когда появляется гарантия, что она не состоится позже отката (GVT продвигается до модельного времени, на которое было запланировано событие, что исключает необходимость в антисообщениях и исключаются каскадированные откаты). Известны подходы, использующие «просмотр назад» (lookback), технику прямой отмены, которая иногда используется для срочной отмены некорректных сообщений. Чтобы учесть данные модификации, предлагаются математические выражения, учитывающие как возрастание объемов памяти (в случае выполнения дампа памяти), так и уменьшение объемов памяти за счет удаления дампов, сделанных до момента времени, определяемого GVT.

Существуют ряд путей решения проблемы больших затрат памяти для хранения дампов памяти. Наиболее известные из них: выполнение отката с той целью, чтобы освободить память; сохранение текущего состояния реже, чем сохранение после каждого события; задание периодов сохранения в начале моделирования, освобождение памяти, занятой векторами состояний, даже в том случае, если их временная метка больше, чем GVT, и т.д.

Учитывая возможность реализации вышеперечисленных способов управления памятью, предлагается следующая обобщенная математическая модель, отражающая динамику изменения объемов памяти (как увеличение используемой памяти, так и ее уменьшение в ситуациях, когда система моделирования гарантирует отсутствие откатов во время, предшествующее GVT) при реализации распределенных имитационных моделей, использующих оптимистические алгоритмы синхронизации:

$$\begin{cases} V_0^n = sizeof\left(\Lambda^n \cup \Delta^n\right); \\ V_0 = \sum_{n=1}^{N} V_0^n; \\ \Delta V_i^n = \begin{cases} V_i^{ns} + V_i^{nm} - \sum_{j=1}^{M^n} \tau_j V_{ij}^{nm}, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases}; \\ \Delta V'^n_i = \sum_{j=1}^{i} V_j^n, \left(\forall j,\ j < i_{GVT}\ или\ j > i'_n\right); \\ V_{i+1}^n = V_i^n + \Delta V_i^n - \sum_k \Delta V_k^n - \Delta V_i^{'n}, \forall k \in [q,i]; \\ V_{i+1} = V_i + \sum_{n=1}^{N} \Delta V_i^n - \Delta V_i^{'n}, \end{cases} \quad (3.10)$$

где $\Delta V_i^{'n}$ – объемы памяти, освобождаемые *n*-ой частной моделью при передвижении нижней границы GVT;

i_{GVT} – шаг моделирования, на котором была достигнута нижняя граница GVT;

i'_n – шаг моделирования, до временной метки которого осуществляется откат частной модели n.

В случае реализации отката удаляются все дампы памяти, выполненные частной моделью между шагами моделирования, временные метки которых больше значения модельного времени, до которого осуществляется откат. В этом случае, вычисление значения освобождаемой памяти *n*-ой частной модели $\Delta V_i^{'n}$ предлагается производить согласно следующему выражению:

$$\Delta V_i^{'n} = \sum_{j=1}^{i} V_j^n, \forall j, \tau_j > \tau_{i'}, \quad (3.11)$$

где τ – временная метка дампа памяти, то есть уменьшение объема памяти для *n*-ой частной модели при реализации отката модельного времени определяется как сумма объемов дампов памяти частной модели, временная метка которых τ_j больше временной метки $\tau_{i'}$ дампа памяти частной модели, выпол-

ненного на шаге моделирования i', до которого осуществляется откат.

При продвижении нижней границы модельного времени GVT объем освобождаемой памяти определяется как сумма объемов дампов памяти, временная метка которых меньше значения GVT, и будет равен

$$\Delta V_i^{'n} = \sum_{j=1}^{i} V_j^n, \forall j, \tau_j > \tau_{GVT}, \quad (3.12)$$

то есть из совокупности дампов памяти частной модели будут удалены все выполненные дампы памяти n-ой частной модели, временная метка τ_j которых меньше нижней границы глобального модельного времени τ_{GVT}.

Обобщенная математическая модель функционирования распределенной имитационной модели под управлением оптимистических алгоритмов синхронизации должна учитывать описанные наиболее известные алгоритмы, а также дополнительные временные параметры и параметры объемов памяти, появление которых обуславливается применением этих алгоритмов. Отметим, что эти параметры должны учитываться моделью, но при их отсутствии (невозможности их получения в процессе имитационного моделирования), не должны влиять на другие параметры модели.

В разделе 4 будет показано, что при отсутствии априорной информации о некоторых из приведенных параметров, их возможно получить уже во время проведения экспериментов, используя стандартные средства мониторинга и удаленного администрирования, входящие в состав современных сетевых операционных систем.

С учетом вышеизложенного, предлагается обобщенная математическая модель функционирования распределенной имитационной модели под управлением оптимистических алгоритмов синхронизации, учитывающая выражения (3.8-3.12):

$$
\begin{cases}
TR_0 = 0; \\
TR_0^n = 0; \\
\Delta TR_i^n = \begin{cases} t_i^s, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases} + \sum_{k=1}^{N} a^{nk} \cdot t_i^m(\delta_i^{nk}, sizeof(buff_i^{nk})); \\
TR_{i+1}^n = TR_i^n + \Delta TR_i^n; \\
TR_{i+1} = TR_i + \underset{n=1,N}{max}(time(\varphi_i^n) + \Delta TR_i^n) + TR_i^M + \underset{j=1,N}{max}\left(time\left(\Delta V'^j_i\right)\right); \\
TP_0^n = 0; \\
TP_{i+1}^n = \begin{cases} TP_i^n, если (t_i^n = T_i\ или\ t_i^n > GVT + Tw); \\ TP_i^n + \underset{n'}{max}(time(\varphi_i^{n'}) + \Delta TR_i^{n'}), n' = \overline{1,N} / n' = n, если\ t_i^n \neq T_i \end{cases}; \\
V_0^n = sizeof(\Lambda^n \cup \Delta^n); \\
V_0 = \sum_{n=1}^{N} V_0^n; \\
\Delta V_i^n = \begin{cases} V_i^{ns} + V_i^{nm} - \sum_{j=1}^{M^n} \tau_j V_{ij}^{nm}, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases}; \\
\Delta V_i'^n = \sum_{j=1}^{i} V_j^n, (\forall j, j < i_{GVT} \vee j > i'_n); \\
V_{i+1}^n = V_i^n + \Delta V_i^n - \sum_k \Delta V_k^n - \Delta V_i'^n, \forall k \in [q,i]; \\
V_{i+1} = V_i + \sum_{n=1}^{N} \Delta V_i^n - \Delta V_i'^n; \\
T_0 = 0; \\
T_{i+1} = \underset{n=1,N}{min}(t_{i+1}^n); \\
n = \overline{1,N},
\end{cases}
\tag{3.13}
$$

где $\underset{j=1,N}{max}\left(time\left(\Delta V'^j_i\right)\right)$ – время, затрачиваемое системой моделирования на процедуру отката или удаления дампов памяти, выполненных ранее временной метки GVT: определяется как максимальное время удаления дампов памяти среди частных локальных моделей. На практике данный параметр может фиксироваться самой системой моделирования, специальными мониторинговыми программами, либо средствами операционной системы. Подчеркнем тот факт,

что при отсутствии возможности мониторинга одного из параметров, все остальные могут быть вычислены (нет взаимной зависимости этих параметров).

В данном подразделе была предложена математическая модель функционирования распределенной имитационной модели с оптимистическими алгоритмами синхронизации частных моделей (выражение 3.13), которая учитывает временные затраты и затраты памяти на выполнение сохранений состояний частных моделей (дампы памяти), реализацию откатов, обслуживание потоков сообщений и антисообщений, наличие временных окон оптимистической синхронизации.

3.5 Усовершенствование модели оценки времени выполнения программных моделей с учетом изменения объемов памяти программных моделей и времени простоя ресурсов

Обобщим результаты, полученные в подразделах 3.2-3.4 и дополним известную математическую модель оценки времени выполнения программных моделей (1.4) оценочными параметрами изменения объемов памяти, объемов данных, передаваемых между программными моделями, временем простоя ресурсов (3.4, 3.5, 3.6, 3.13). При этом отметим, что выражения, входящие в обобщенную модель могут меняться в соответствии с выражениями 3.7-3.12 в зависимости от возможности определения некоторых параметров (априорно или в процессе моделирования) и выбранного метода синхронизации частных программных моделей (синхронного, консервативного или оптимистического). В результате, получаем обобщенную модели оценки параметров выполнения программных моделей с учетом изменения объемов памяти программных моделей, объемов данных, передаваемых между программными моделями и времени простоя ресурсов:

$$
\begin{cases}
t_0^n = 0; \quad TR_0 = 0; \quad TR_0^n = 0; \\
t_{i+1}^n = \begin{cases} t_i^n + \xi_i^n, если\, t_i^n = T_i \\ t_i^n, если\, t_i^n \neq T_i \\ t_{i'}^n, если\, \exists i'_n \end{cases}; \\
\Delta TR_i^n = \begin{cases} t_i^s, если\, t_i^n = T_i \\ 0, если\, t_i^n \neq T_i \end{cases} + \sum_{k=1}^{N} a^{nk} \cdot t_i^m(\delta_i^{nk}, sizeof(buff_i^{nk})); \\
TR_{i+1}^n = TR_i^n + \Delta TR_i^n; \\
TR_{i+1} = TR_i + \underset{n=1,N}{max}(time(\varphi_i^n) + \Delta TR_i^n) + TR_i^M + max(\underset{j=1,N}{time}(\Delta V_i'^j)); \\
TP_0^n = 0; \\
TP_{i+1}^n = \begin{cases} TP_i^n, если (t_i^n = T_i \text{ или } t_i^n > GVT + Tw); \\ TP_i^n + \underset{n'}{max}(time(\varphi_i^{n'}) + \Delta TR_i^{n'}), n' = \overline{1,N} / n' = n, если\, t_i^n \neq T_i \end{cases}; \\
V_0^n = sizeof(\Lambda^n \cup \Delta^n); V_0 = \sum_{n=1}^{N} V_0^n; \\
\Delta V_i^n = \begin{cases} V_i^{ns} + V_i^{nm} - \sum_{j=1}^{M^n} \tau_j V_{ij}^{nm}, если\, t_i^n = T_i \\ 0, если\, t_i^n \neq T_i \end{cases}; \\
\Delta V_i'^n = \sum_{j=1}^{i} V_j^n, (\forall j, j < i_{GVT} \vee j > i'_n); \\
V_{i+1}^n = V_i^n + \Delta V_i^n - \sum_k \Delta V_k^n - \Delta V_i'^n, \forall k \in [q,i]; \\
V_{i+1} = V_i + \sum_{n=1}^{N} \Delta V_i^n - \Delta V_i'^n; \\
T_0 = 0; T_{i+1} = \underset{n=1,N}{min}(t_{i+1}^n); \\
n = \overline{1,N}.
\end{cases}
\tag{3.14}
$$

Таким образом, одновременно с изменением модельного времени частных программных моделей t_i^n, выражение (3.14) дает возможность оценить реальное время моделирования TR_i^n, время простоя вычислительного ресурса TP_i^n, изменение объемов памяти на *i*-ом шаге моделирования ΔV_i^n и объем памяти, занимаемый *n*-ой моделью V_i^n.

3.6 Разработка алгоритма процесса функционирования распределенной имитационной модели

На основе математических моделей, приведенных в подразделах 3.2-3.4 разработана алгоритмическая модель процесса функционирования распределенной имитационной модели, целью реализации которой является вычисление основных параметров критериев оценивания распределенной имитационной модели, к которым относятся:

- i – номер шага моделирования;

- T_i – глобальное модельное время, достигнутое моделью на i-ом шаге моделирования;

- TR_i – реальное (астрономическое) время, затраченное системой моделирования на выполнение i шагов модели;

- TR_i^n реальное (астрономическое) время, затраченное системой моделирования на выполнение i шагов n-ой частной модели;

- V_i – объем виртуальной памяти, занимаемый моделью по достижению i-го шага моделирования;

- V_i^n – объем виртуальной памяти, занимаемый n-ой частной моделью по достижению i-го шага моделирования.

Для упрощения представления алгоритма введем вектор $\overline{\alpha}$, бинарный элемент α_m которого определяет наличие (отсутствие) определенного параметра распределенной имитационной модели или является признаком модификации алгоритма синхронизации, который реализован в данной распределенной имитационной модели: $\overline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_M\}, \alpha_m \in \{0,1\}$.

Введем следующие значения элементов вектора:

- $\alpha_1 = 1$, если требуется вычислять временные параметры функционирования распределенной имитационной модели (TR_i^n, TR_i), и $\alpha_1 = 0$, если временные параметры вычислять не нужно;

- $\alpha_2 = 1$, если требуется вычислять изменения объемов памяти при функ-

ционировании распределенной имитационной модели (V_i, V_i^n), и $\alpha_2 = 0$, если параметры объемов памяти вычислять не нужно;

- $\alpha_3 = 1$, если используется оптимистический алгоритм синхронизации частных моделей, и $\alpha_3 = 0$, если консервативный алгоритм;

Приведем алгоритм процесса функционирования распределенной имитационной модели:

- шаг 1: инициализация и загрузка параметров модели;

- шаг 2: установление начальных значений следующим вычисляемым параметрам модели: $TR_0 = 0,\ T_0 = 0, TR_0^n = 0, TP_0^n = 0, t_0^n = 0,\ \left(n = \overline{1,N}\right)$;

- шаг 3: если $\alpha_2 = 1$ вычисляем начальные объемы памяти, занимаемые моделью: $\begin{cases} V_0^n = sizeof\left(\Lambda^n \cup \Delta^n\right); \\ V_0 = \sum\limits_{n=1}^{N} V_0^n; \end{cases}$

- шаг 4: начало цикла шага моделирования, $i=0$;

- шаг 5: если $\alpha_1 = 0$, переход к шагу 13 алгоритма;

- шаг 6: инициализируем цикл текущего номера частной модели $n=1$;

- шаг 7: вычисляем время, затрачиваемое на выполнение n-ой частной модели на i-ом шаге моделирования:

$$\Delta TR_i^n = \begin{cases} t_i^s, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases} + \sum_{k=1}^{N} a^{nk} \cdot t_i^m\left(\delta_i^{nk}, sizeof\left(buff_i^{nk}\right)\right);$$

- шаг 8: вычисляем реальное время, затраченное на выполнение n-ой частной модели за все шаги моделирования, начиная с 0-го до i-го включительно: $TR_{i+1}^n = TR_i^n + \Delta TR_i^n$;

- шаг 9: сохраняем вычисленные значения для i-го шага моделирования;

- шаг 10: переходим к следующей частной модели $n=n+1$;

- шаг 11: если все модели не перебраны ($n \le N$), переходим к шагу 7;

- шаг 12: вычисляем реальное время, затраченное на выполнение всей модели за все шаги моделирования, начиная с 0-го до i-го включительно:

$$TR_{i+1} = TR_i + \max_{n=1,N}\left(time\left(\varphi_i^n\right) + \Delta TR_i^n\right) + TR_i^M ;$$

- шаг 13: если $\alpha_2 = 0$, переход к шагу 25 алгоритма;

- шаг 14: инициализируем цикл текущего номера частной модели *n=1*;

- шаг 15: если $\alpha_3 = 1$, переход к шагу 19 алгоритма;

- шаг 16: вычисляем приращение объема памяти *n*-ой частной модели на *i*-ом шаге моделирования: $\Delta V_i^n = \begin{cases} V_i^{ns}, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases}$;

- шаг 17: вычисляем общий объем памяти *n*-ой частной модели и всей модели после i-го шаге моделирования: $V_{i+1}^n = V_i^n + \Delta V_i^n$;

- шаг 18: переход к шагу 23 алгоритма;

- шаг 19: вычисляем приращение объема памяти *n*-ой частной модели на *i*-ом шаге моделирования:

$$\Delta V_i^n = \begin{cases} V_i^{ns} + V_i^{nm} - \sum_{j=1}^{M^n} \tau_j V_{ij}^{nm}, если\ t_i^n = T_i \\ 0, если\ t_i^n \neq T_i \end{cases};$$

- шаг 20: вычисляем уменьшение объема памяти частной модели за счет удаления дампов памяти при откатах и продвижении глобального модельного времени: $\Delta V'^n_i = \sum_{j=1}^{i} V_j^n, \left(\forall j,\ j < i_{GVT} \vee j > i'\right)$;

- шаг 21: вычисляем общий объем памяти n-ой частной модели после i-го шаге моделирования: $V_{i+1}^n = V_i^n + \Delta V_i^n - \sum_k V_k^n - \Delta V'^n_i, \forall k \in [q,i]$;

- шаг 22: переходим к следующей частной модели *n=n+1*;

- шаг 23: если все модели не перебраны ($n \le N$), переходим к шагу 15;

- шаг 24: вычисляем общий объем памяти модели после *i*-го шага моделирования: $V_{i+1} = V_i + \sum_{n=1}^{N} \Delta V_i^n - \begin{cases} \Delta V'^n_i, если\ \alpha_3 = 1 \\ 0, если\ \alpha_3 = 0 \end{cases}$;

- шаг 25: определяем для каждой модели время будущего события:

$$t_{i+1}^{n} = \begin{cases} t_i^n + \xi_i^n, если\ t_i^n = T_i \\ t_i^n, если\ t_i^n \neq T_i \\ t_{i'}^n, если\ \exists i'_n \end{cases};$$

- шаг 26: продвигаем модельное время: $T_{i+1} = \underset{n=\overline{1,N}}{min}\left(t_{i+1}^{n}\right)$;

- шаг 27: увеличиваем значение шага моделирования $i=i+1$;

- шаг 28: если не достигнуто условие достижения конца моделирования, переходим к шагу 5 алгоритма;

- шаг 29: фиксация номера шага окончания моделирования $I = i$;

- шаг 30: формирование отчетов по результатам моделирования;

- шаг 31: конец алгоритма.

Выходной информацией, полученной в результате выполнения алгоритма, является кортеж параметров оценивания распределенной имитационной модели:

$$O = < I, \overline{T_i}, \overline{TR_i}, \overline{TR_i^n}, \overline{TP_i^n}, \overline{V_i}, \overline{V_i^n} >, i = \overline{1,N} \qquad (3.15)$$

Отметим, что кортеж содержит вектора параметров, отражающих их изменение во времени. Элементы векторов с индексом $i = I$ являются значениями соответствующих параметров на момент окончания процесса моделирования. При этом существует возможность получения значения любого из приведенных параметров на i-м шаге моделирования или в одно из дискретных моментов модельного времени T_i. Процедура получения данных в последнем случае заключается в нахождении индекса i для момента времени T_i (или ближайшего значения из вектора $\overline{T_i}$) и выбора из других векторов значений с индексом i.

В данном подразделе приведен обобщенный алгоритм процесса функционирования распределенной имитационной модели под управлением консервативных и оптимистических алгоритмов синхронизации.

3.7 Разработка метода оценки возможности осуществления распределения частных моделей на основе динамического изменения объемов виртуальной памяти

Значение объема свободной виртуальной памяти вычислительного ресурса является ограничивающим параметром для распределения частных моделей. Действительно, вычислительный ресурс не может выполнять частную модель (совокупность частных моделей), объем которой превышает оперативную (виртуальную) память ресурса без дополнительных средств виртуализации. Данный факт учитывается многими методами распределения ресурсов и является ограничительным условием варианта распределения частных моделей по ресурсам.

Разработанная модель функционирования распределенной имитационной модели позволяет учитывать не только статические характеристики занимаемой памяти частной модели в момент распределения, но и динамические, возникающие в процессе функционирования модели. В большинстве случаев, в процессе имитации объем памяти, занимаемый частной моделью, растет за счет дампов памяти создания новых сообщений, внутренних объемов динамической памяти. В последнем случае может возникнуть ситуация переполнения виртуальной памяти ресурса, что приведет к отказу ресурса для системы моделирования.

Пусть имеется R вычислительных ресурсов с объемами свободной памяти $VR_r, r = \overline{1,R}$, на которые надо распределить N частных моделей $SM_n, n = \overline{1,N}$. В отличие от большинства известных методов распределения, будем считать, что количество ресурсов не равно количеству частных моделей и на один ресурс можно распределить несколько частных моделей. Обозначим через K множество возможных распределений частных моделей:

$$\begin{array}{l} K_k : SM \to VR \ \ или \ \ K_k : sm_m^k \to VR_r^m, m = \overline{1,M}, \ M \le N, \\ sm_m^k \subset SM, sm_m^k \cap sm_m^k = \varnothing, \forall (i,j), \ i \in [1,N], \ \ j \in [1,N], \ i \ne j \end{array}. \qquad (3.16)$$

Задачу оценки качества распределения частных моделей на основе динамического изменения объемов виртуальной памяти сформулируем следующим образом: распределение K_k является допустимым, если объем памяти V_{sm_m},

занимаемый любым подмножеством частных моделей sm_m, на любых этапах моделирования меньше объема памяти r-го ресурса VR_r, на который распределено множество частных моделей sm_m.

Обозначим через u индекс элемента подмножества sm_m:

$$\mathrm{sm_m} = \bigcup_{u=1}^{U_m} \mathrm{sm_{mu}},\ \mathrm{U_m} \le \mathrm{N}. \tag{3.17}$$

Тогда память, занимаемая множеством частных моделей sm_m, с учетом выражений (3.4), (3.5), (3.13) на момент начала моделирования:

$$V_{sm_m} = \sum_{u=1}^{U_m} V_0^u\ . \tag{3.18}$$

где V_0^u – объем памяти частной модели из множества sm_m. $V_0^u \in O$ (выражение 3.15).

Выражение 3.18 может быть использовано только в том случае, если объемы памяти частных моделей не изменяются в процессе моделирования. В случае динамического изменения объемов памяти частных моделей предлагается использовать следующее выражение:

$$V_{sm_m} = \max_{i=1,I} \sum_{u=1}^{U_m} V_0^u \tag{3.19}$$

где $V_i^u \in \overline{V_i^n}$ из выражения (3.15).

Отметим, что некоторые алгоритмы распределения ресурсов используют другую модификацию выражения (3.19):

$$V_{sm_m} = \sum_{u=1}^{U_m} \max_{i=1,I} \left(V_i^u\right), \tag{3.19}$$

то есть объем памяти, требуемый для множества частных моделей sm_m, определяется как сумма максимальных объемов памяти частных моделей, входящих

в данное множество. Это выражение может быть использовано, однако оно ухудшает оценку качества распределения частных моделей. Действительно, пусть подмножество sm_m содержит две частные модели, каждая из которых имеет только один максимум значений объемов памяти, причем на разных шагах моделирования V_i^1 и V_j^2, $i \neq j$. В таком случае, на соответствующих шагах моделирования, объемы памяти, занимаемые вторыми моделями, будут меньше максимума: $V_j^1 < V_i^1, V_i^2 < V_j^2$. В таком случае, выражение (3.19) даст оценку памяти как $V_i^1 + V_j^2$, значение которой больше реальных объемов памяти, занимаемых частными моделями на *i*-ом и *j*-ом шагах моделирования. Такая оценка может в дальнейшем исключить из рассмотрения один из возможных вариантов распределения ресурсов, хотя реальные затраты памяти не соответствуют рассчитанным. Выражение (3.18) в этом случае учитывает реальные значения объемов памяти для всего множества частных моделей на всех шагах моделирования.

Основываясь на выражении (3.18) сформулируем условие возможности распределения K_k:

$$\forall sm_m^k \rightarrow VR_r, V_{sm_m^k} \leq VR_r^m. \qquad (3.21)$$

Если условие (3.20) не выполняется хотя бы для одного из подмножеств sm_m^k, распределение K_k невозможно, так как приведет к переполнению памяти на одном из ресурсов, что приведет к сбою процесса моделирования.

Сформулируем этапы реализации метода оценки возможности осуществления распределения частных моделей на основе анализа динамического изменения объемов виртуальной памяти.

Этап 1: на основании схем назначения K, полученных от системы распределения ресурсов, для каждой *k*-ой схемы назначения, выделяются подмножества программных моделей sm_m^k, распределенных на один ресурс R_m. Условием включения программной модели в данное подмножество является ее принадлежность к *m*-му ресурсу.

Этап 2: задать начальные параметры распределенных имитационных моделей согласно файлу-описания задания *Z* и шагов 2-3 алгоритма, приведенного в подразделе 3.6.

Этап 3: согласно рассмотренному выше алгоритму (подраздел 3.6) произвести вычисление кортежа (3.15) для всех реализаций распределения программных моделей.

Этап 4: вычислить $V_{sm_m^k}$ согласно выражениям (3.19) и (3.20) для всех подмножеств частных моделей в зависимости от выбранного алгоритма синхронизации программных моделей.

Этап 5: проверить выполнение условия (3.21) для всех возможных распределений частных моделей. Для этого, необходимо в цикле перебрать все элементы вектора sm_m, и сравнить значения совокупного объема памяти всех программных моделей, распределенных на данный ресурс V_{sm_m}, с объемом памяти вычислительного ресурса VR_m.

Этап 6. Исключить все распределения, для которых условие (3.21) не выполняется хотя бы для одного из подмножеств sm_m^k. В результате, получаем новое подмножество $K' = K \setminus K_k, \forall K_k \to \exists\left(sm_m^k \to VR_r\right) V_{sm_m^k} > VR_r^m$.

Предложенный метод может быть использован для оценки качества полученного множества возможных распределений или до начала поиска оптимального распределения для исключения некоторых подмножеств вариантов распределений. Последнее применение метода позволит не только исключить тупиковые варианты распределения, но и сократить время работы алгоритмов распределения.

3.8 Разработка метода получения оценок простоя ресурсов для различных методов синхронизации распределенных имитационных моделей

Большинство методов распределения имитационных моделей на вычислительные ресурсы не учитывают простой ресурсов, связанный с синхронизацией частных моделей в едином пространстве модельного времени. Если одним

из существенных критериев оценки эффективности является стоимость использования вычислительных ресурсов, то значительный простой ресурсов оказывает существенное влияние на этот критерий. В большей степени это проявляется в случае использования консервативных алгоритмов синхронизации распределенных имитационных моделей, так как основной их идеей является запрещение продвижения модельного времени, то есть, по сути, приостановка моделирования, до момента синхронизации всех моделей.

Использование каждого *j*-го ресурса из множества элементов среды имитационного моделирования, на который распределена *i*-я частная модель, характеризуется некоторой стоимостью c_j. Величина c_j характеризует стоимость использования ресурса в единицу времени. Существует множество методов определения этого параметра, но для пользователя, как правило, стоимость является исходными данными, выставляемыми владельцем ресурсов.

Целью применения данного метода является получение критериев оценки эффективности использования алгоритмов синхронизации распределенных имитационных моделей, исходя из стоимости использования ресурсов.

Предлагаемый метод оценки простоя ресурсов основан на выражении (3.6), предложенном в подразделе 3.3 для консервативных алгоритмов или выражении (3.9) для оптимистических алгоритмов. Метод содержит следующие этапы.

Этап 1: задать исходные параметры распределенных имитационных моделей согласно файлу описания задания (*Z*) и системы выражений (3.14) для параметров $TR_0 = 0,\ T_0 = 0, TR_0^n = 0, TP_0^n = 0, t_0^n = 0,\ V_0^n, V_0\ \left(n = \overline{1,N}\right).$

Этап 2: согласно рассмотренному выше алгоритму (подраздел 3.4) произвести вычисление векторов $\overline{TR_i^n}$ и $\overline{TP_i^n}$ для всех реализаций распределенных имитационных моделей, характеризующие реальное время работы ресурсов по обслуживанию частных моделей и время простоя ресурсов. Значения TR_I^n и TP_I^n характеризуют конечное состояние параметров на момент останова процесса имитационного моделирования.

Этап 3: вычислить для каждого ресурса *j*-го стоимость использования в

режиме исполнения частных моделей: $CR_j = TR_I^n \cdot c_j$, где n – номер модели, распределенной на *j*-ый ресурс.

Этап 4: вычислить для каждого *j*-го ресурса стоимость использования в режиме простоя: $CP_j = TP_I^n \cdot c_j$.

Этап 5: вычислить суммарную стоимость использования *j*-го ресурса: $C_j = CR_j + CP_j$.

Этап 6: вычислить суммарную стоимость использование ресурсов, а также суммарные стоимости использования ресурсов в режиме выполнения модели и в режиме простоя: $C = \sum_{j=1}^{R} C_j$, $CR = \sum_{j=1}^{R} CR_j$, $CP = \sum_{j=1}^{R} CP_j$.

Этап 7: вычислить коэффициенты использования ресурсов как отношение времени обслуживания частных моделей к общему времени моделирования: $\rho_j = \frac{TR_I^n}{TR_I^n + TP_I^n}$.

Этап 8: вычислить коэффициенты простоя ресурсов как отношение времени простоя ресурса к общему времени моделирования: $\varsigma_j = \frac{TP_I^n}{TR_I^n + TP_I^n} = 1 - \rho_j$.

Этап 9: полученный в результате выполнения пунктов 1-8 кортеж параметров использовать для оценки качества применения алгоритмов синхронизации для распределенной имитационной модели.

$$CC = < \overline{CR_j}, \overline{CP_j}, \overline{C_j}, CR, CP, C, \overline{\rho_j}, \overline{\varsigma_j} > \qquad (3.22)$$

Данный метод предполагает получение параметров оценивания качества распределения на основе стоимости использования ресурсов и может быть использован для выделения из всего множества возможных алгоритмов синхронизации и множества возможных распределений задач по ресурсам наиболее оптимального (с наименьшей стоимостью или наименьшими значениями стоимости простоя).

Более сложные алгоритмы распределения ресурсов (например, алгоритмы

динамические распределения) могут учитывать оценки простоя ресурсов для параллельного использования ресурса в других вычислительных процессах или для совмещения на одном ресурсе нескольких частных моделей.

3.9 Модификация метода сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени

С помощью приведенных выше моделей можно оценить модельное время, которое достигнет распределенная имитационная модель за фиксированное количество шагов моделирования, что позволяет оценить эффективность применения различных алгоритмов синхронизации и стратегий моделирования, реализованных в распределенной имитационной модели. Данный критерий (продвижение модельного времени) используется для оценки выбранного алгоритма синхронизации распределенных имитационных моделей, например, в работах [11-15].

Приведем этапы модифицированного метода сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени:

Этап 1: получить от системы распределения ресурсов множество возможных схем назначения K (3.16) программных моделей на доступные вычислительные ресурсы.

Этап 2: задать исходные параметры распределенных имитационных моделей согласно файлу описания задания (Z) и системы выражений (3.14) для параметров $TR_0 = 0,\ T_0 = 0, TR_0^n = 0, TP_0^n = 0, t_0^n = 0,\ V_0^n, V_0\ \left(n = \overline{1,N}\right)$.

Этап 3: согласно рассмотренному выше алгоритму (подраздел 3.6) произвести вычисление элементов кортежа (3.15) для всех реализаций распределенных имитационных моделей (с использованием консервативных и оптимистических методов синхронизации).

Этап 4: используя метод оценки распределения частных моделей на основе динамического изменения объемов виртуальной памяти (подраздел 3.7) выбрать схему назначения для программных моделей с оптимистической синхро-

низацией.

Этап 5: используя метод оценки простоя ресурсов при выполнении частных моделей (подраздел 3.8) выбрать схему назначения для программных моделей с консервативной синхронизацией.

Этап 6: выбрать из двух схем назначения (полученных на этапах 4 и 5) реализаций распределенных имитационных моделей ту модель, для которой T_I (модельное время на последнем I-ом шаге) является максимальным.

Полученная, в результате применения метода, схема распределения позволит выполнить моделирование с меньшим реальным временем моделирования. Кроме этого, полученные в результате применения метода значения модельного времени T_I для всех схем распределения ресурсов можно использовать в качестве отдельного критерия или в качестве параметра совокупного критерия эффективности распределенной имитационной модели.

В отличие от существующего метода сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени, модифицированный метод за счет применения методов оценки распределения частных моделей на основе динамического изменения объемов виртуальной памяти и метод оценки простоя ресурсов при выполнении частных моделей (этапы 4 и 5 рассмотренного метода) убрать из рассмотрения те схемы назначения, для которых не выполняются ограничения по использованию памяти и простоя вычислительных ресурсов.

Этапы 4 и 5 возможно реализовать за счет предварительного проведения имитационного моделирования (этапы 2 и 3).

3.10 Выводы по разделу

В данном разделе были предложены ряд параметров, согласно которым можно оценить схемы распределения программных моделей по вычислительным ресурсам (подраздел 3.1). Данные параметры могут использоваться самостоятельно, либо учитываться в обобщающих критериях оценивания.

Была предложена имитационная модель процесса распределенного дискретно-событийного моделирования с учетом изменений объемов памяти и

времени исполнения (подраздел 3.2). Рассмотрены частные случаи модификации модели при использовании консервативных (подраздел 3.3) и оптимистических (подраздел 3.4) алгоритмов синхронизации, которые учитывают временные характеристики выполнения распределенной имитационной модели и затраты виртуальной памяти процесса моделирования.

На основе разработанных моделей в подразделе 3.5 предложена усовершенствованная модель оценки времени выполнения программных моделей с учетом изменения объемов памяти программных моделей и времени простоя ресурсов.

На основе разработанных моделей разработан алгоритм процесса функционирования распределенной имитационной модели при использовании консервативных и оптимистических алгоритмов синхронизации (подраздел 3.6).

На основе математической и алгоритмических моделей предложено ряд методов оценки схем распределения программных моделей с учетом разных алгоритмов синхронизации (подразделы 3.7-3.9), а именно: метод оценки возможности осуществления распределения частных моделей на основе динамического изменения объемов виртуальной памяти (подраздел 3.7), метод получения оценок простоя ресурсов для различных методов синхронизации распределенных имитационных моделей (подраздел 3.8), модификация метода сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени (подраздел 3.9).

Следует отметить, что полученные результаты могут быть использованы в ряде других методов анализа распределенных имитационных моделей для разных алгоритмов синхронизации, рассмотрение которых выходит за рамки данной работы. Кроме этого, они могут использоваться в методах статического и динамического распределения ресурсов в задачах распределенного имитационного моделирования.

4 ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ РАСПРЕДЕЛЕННОГО ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ И ЕЕ ВНЕДРЕНИЕ В СИСТЕМУ МОДЕЛИРОВАНИЯ GRASS

Предложенные в разделе 3 математические и имитационные модели описывают, при помощи количественных характеристик, поведение распределенных программных моделей. Данные модели и методы позволяют разработать информационную технологию распределенного имитационного моделирования с включением в нее новой информационной технологии исследования программных моделей и схем назначения, которая в отличие от существующих технологий, позволяет получить оценочные характеристики объемов оперативной памяти, время простоя процессора и продвижение модельного времени, что дает возможность выбора схемы назначения, которая обеспечивает меньшее время моделирования при ограничениях на память и простой ресурсов. В разделе 4 описывается разработанные информационные технологии и их имплементация в систему GRASS распределенного имитационного моделирования GRID-систем, а также проведенные эксперименты по применению предложенных моделей, методов и информационных технологий.

4.1 Разработка информационной технологии распределенного имитационного моделирования с исследованием программных моделей и выбором схемы назначения

Согласно определению, данному в ГОСТ 34.003-90, информационная технология представляет собой приемы, способы и методы применения средств вычислительной техники при выполнении функций сбора, хранения, обработки, передачи и использования данных.

Применительно к системам распределенного имитационного моделирования объектами данных выступают исходные данные к моделированию (конфигурационные файлы и программные модули), которые должны быть переданы на вычислительные ресурсы, данные, передаваемые между программными

моделями во время моделирования, а также файлы результатов, которые должны быть собраны и переданы пользователю после (или во время) моделирования. Разработанная информационная технология, приведенная ниже, также создает новую информацию о программных моделях, которую необходимо сохранять в базе данных, хранящих информацию о проведенных экспериментах, и обеспечить доступ к ней программным модулям системы моделирования, реализующим методы оценки схем назначения.

На основе моделей и методов анализа распределенных имитационных моделей, приведенных разделе 3, разработана технология анализа распределенных имитационных моделей, которая включает следующие этапы.

Этап 1: получение задания на моделирование от потребителя в виде совокупности конфигурационных, исполняемых, информационных файлов:

$$n_i = \{ Z^i, \bigcup_{j=1}^{N} Pr_j^i, D_j^i, W^i \} \tag{4.1}$$

где Z – конфигурационный файл описания задания;

$\bigcup_{j=1}^{N} Pr_j^i$ – множество программних моделей, каждая из которых требует для своего выполнения отдельного вычислительного ресурса (компьютера);

D – файл (файлы) входных данных, определяющий начальное состояние программных моделей;

W – имя файла (имена файлов), в которые записываются результаты моделирования. Конфигурационный файл задания Z содержит информацию о программных моделях, требуемых ресурсах и т.п., например,

$$Z^i = \{ ar_i^z, os_i^z, pc_i^z, ps_i^z, ms_i^z, dc_i^z, pr_i, ca_i, bw_i, rt_i, ... \} \tag{4.2}$$

где ar – архитектура процессора;

os – операционная система;

pc – требуемая производительность процессоров;

ps – быстродействие процессоров;

ms – объем оперативной памяти;

dc – требуемый объем винчестера;

ca – коэффициент связности задач в задании;

pr – приоритет задания;

bw – пропускная способность канала связи для организации обмена между программными моделями;

rt – время выполнения задания и т.п.

Производится параллельный запуск этапов 2 и 3.

Этап 2: на основе конфигурационного файла задания *Z* проходит инициализация априорных данных о распределенных программных моделях и размещение задания в очереди ожидания распределения. Далее переход к этапу 5.

Этап 3: проверка наличия в базе данных информации о предыдущих запусках модели (согласно атрибутам *Z*), полученной от подсистемы анализа модели и собранной в результате мониторинга предыдущих процессов имитационного моделирования. Если данные присутствуют в базе данных, а происходит загрузка информации из базы данных и переход к этапу 5, иначе – переход к этапу 4.

Этап 4: запуск программных моделей, моделирующих процессы имитации, исследование поведения моделей с различными методами синхронизации (консервативными и оптимистическими). Результатом является получение вектора, определяемого выражением (3.15).

Этап 5: применения стандартных методов распределения ресурсов с целью получения множества допустимых схем распределения ресурсов.

Решается задача распределения программных моделей Pr_j^i на доступные ресурсы $\bigcup_{m=1}^{M} R_m$, где

$$Rj = \left\{ ar_m^r, os_m^r, pc_m^r, ps_m^r, ms_m^r, dc_m^r, bw_m^r \right\} \tag{4.3}$$

Результатом является множество доступных схем назначения, где каждому программному компоненту ставиться в соответствие вычислительный ре-

сурс $P = \bigcup_k \left\{ Pr_j^i \to R_m, \forall j = \overline{1,N} \right\}_k$ (k-количество допустимых схем назначения). Эффективность распределения определяется минимизацией времени имитации или стоимостью использования ресурсов.

Этап 6: применение методов анализа распределенных программных моделей. В разделе 3 рассмотрены три метода: метод сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени, метод оценки возможности распределения частных моделей на основе динамического изменения объемов виртуальной памяти, метод получения простоя ресурсов при использовании разных методов синхронизации распределенных имитационных моделей. Кроме того, существует возможность использования новых методов, подключаемых параллельно. Целью выполнения этапа является нахождение схемы назначения, имеющей минимальное время выполнения с выполнением ограничений на объемы памяти, занимаемые программными моделями и времени простоя вычислительных ресурсов.

В конце этапа производится сохранение информации, полученной в результате применения методов оценки схем распределения, в базе данных.

Этап 7: вызов брокера ресурсов, реализующего физическое распределение локальных программных моделей на вычислительные ресурсы согласно выбранной схемы распределения.

Этап 8: использование средств мониторинга системы моделирования и операционной системы для получения динамических параметров процесса имитации и сохранение их в базе данных. Сохранение и установление связей в базе данных между априорными параметрами эксперимента, с одной стороны, и полученными в результате применения технологии анализа и сбора статистических данных от систем мониторинга среды исполнения программных моделей, с другой стороны.

Этап 9: передача результатов выполнения задания пользователю в виде множества файлов-результатов *W*. Анализ информации, полученной в результате процесса моделирования (параллельного исполнения программных моделей) производится пользователем и не входит в перечень вопросов, рассмотренных в

данной работе.

Таким образом, предлагаемая информационная технология распределенного имитационного моделирования включает в себя девять этапов, среди которых оригинальными этапами являются этап моделирования процессов распределенной имитации, который исследует поведение моделей с различными методами синхронизации (консервативными и оптимистическими), а также этап применения методов сравнения схем назначения (этапы 4 и 6). Информация, которая формируется на выходе этих этапов, учитывается на этапе распределения программных моделей по доступным вычислительным ресурсам.

4.1.1 Параллельные и последовательные этапы разработанной технологии

Представим разработанную технологию анализа, указав этапы технологии в привязке к диаграммы действий, отразив последовательные и параллельные этапы (рисунок 4.1).

Применение третьего и четвертого этапа возможно параллельно с выполнением второго этапа, который реализует размещение и передачу исходных файлов на ресурсах, обслуживающих распределенную систему моделирования и пятого этапа, который генерирует множество схем назначения стандартными методами. В качестве стандартных методов применяются методы равномерного заполнения ресурсов на основе вычислительной сложности и размеров программных модулей, методы случайного распределения и метод backfill.

Полученные схемы назначения являются входными данными для шестого этапа. Второй частью входных данных для шестого этапа являются результаты исследования программных моделей и данные о предыдущих экспериментов, которые хранятся в базе данных.

Если эксперимент повторяется без изменения конфигурационных файлов и множества программных моделей (4.1) и в базе данных находится выбранная ранее схема назначения, этапы исследования программных моделей могут быть пропущены. Если, при этом, множество доступных вычислительных ресурсов совпадает с предыдущим экспериментом, то может быть выключен шестой этап разработанной информационной технологии.

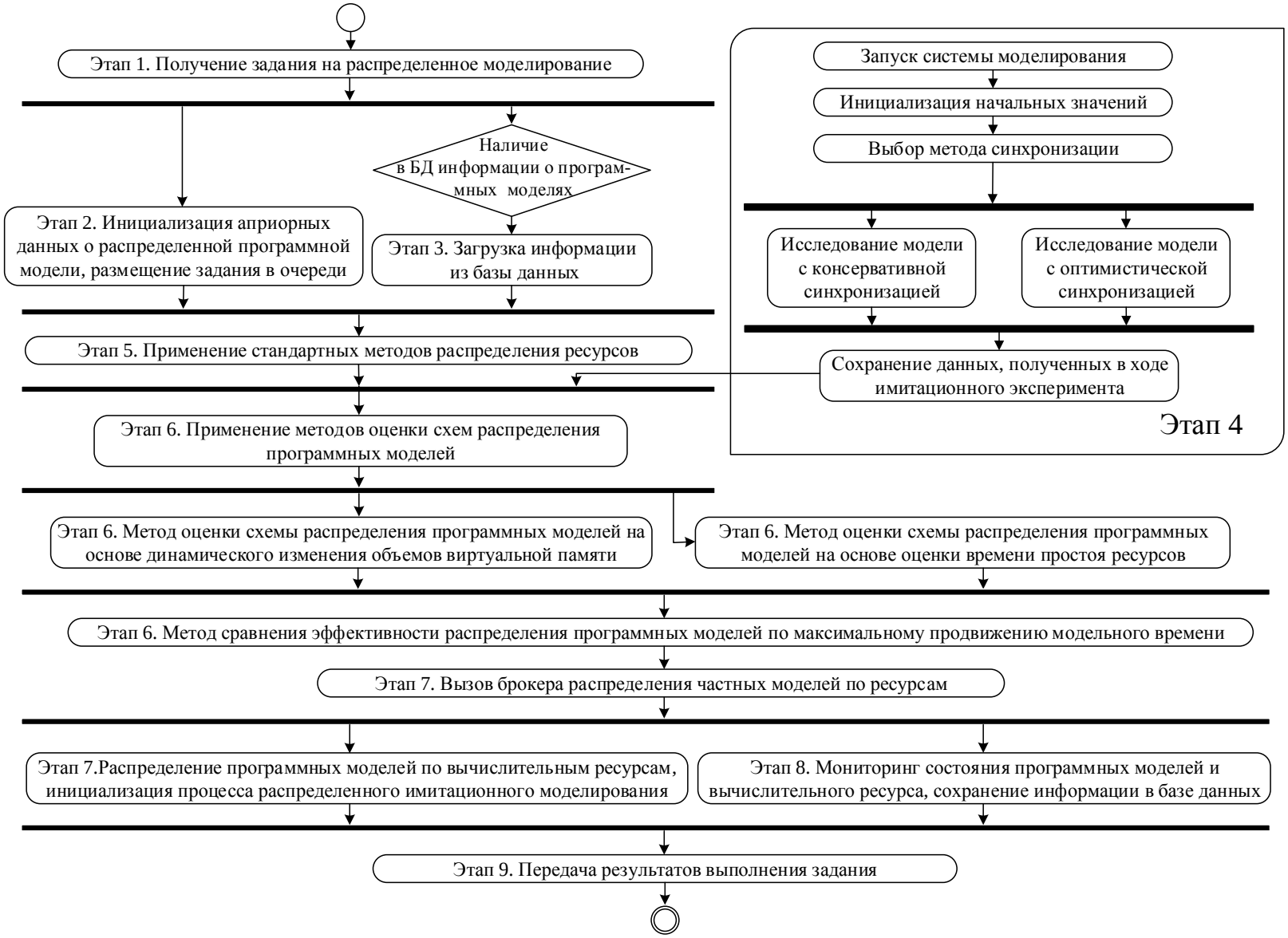


Рисунок 4.1 – Диаграмма действий информационной технологии анализа распределенных программных моделей

Из приведенной диаграммы видно, что реализация шестого этапа возможна с параллельным выполнением двух методов исследования программных моделей: метода оценки возможности осуществления распределения частных моделей на основе динамического изменения объемов виртуальной памяти (метод описан в подразделе 3.7) и метода получения оценок простоя ресурсов для различных методов синхронизации распределенных имитационных моделей (метод описан в подразделе 3.8).

Также параллельно проводится инициализация седьмого и восьмого этапов, то есть запуск и управление распределенным имитационным моделированием осуществляется параллельно со службами удаленного мониторинга распределенных вычислительных ресурсов и каналов обмена данным в распределенной системе моделирования.

Предложенная диаграмма действий фактически является алгоритмом, определяющим последовательность выполнения этапов разработанной информационной технологии с возможностью выделения на ней параллельных и последовательных этапов.

4.1.2 Основные отличия разработанной информационной технологии распределенного имитационного моделирования

Основными отличиями разработанной информационной технологии являются наличие четвертого и шестого этапов, которые позволяют на основе имитационного моделирования и применения методов оценки схем назначения с учетом объемов оперативной памяти, времени простоя ресурсов и методов синхронизации, сократить время моделирования и количество используемых ресурсов.

В результате применения информационной технологии, система распределения ресурсов получает доступ к новой информации (параметрам процесса имитации), что отражено на рисунке 4.2.

Как видно из рисунка, предлагаемая информационная технология позволяет получить новый информационный продукт, в основе которого лежат полученные характеристики (параметры) процесса распределенной имитации, эле-

ментный состав которых приведен в кортеже (3.15). Параметры размещаются в базе данных одновременно с параметрами эксперимента, определяемыми элементами кортежа (4.1) и статическими и динамическими характеристиками вычислительных ресурсов и каналов передачи данных (кортеж 4.3), полученных от систем мониторинга вычислительных ресурсов. Получение этой информации дает основание говорить о применении новой информационной технологии исследования (анализа) программных моделей. Данная технология будет описана в подразделе 4.1.3.

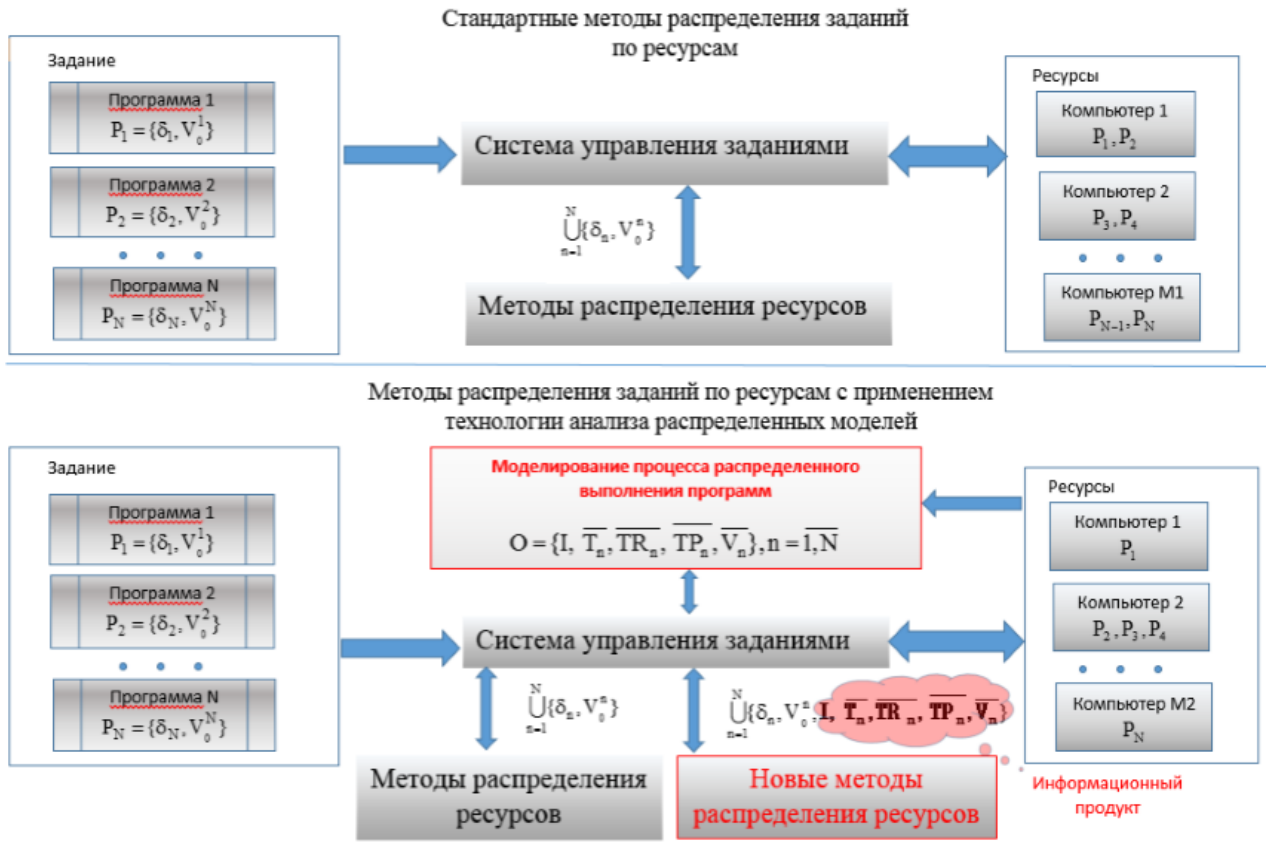


Рисунок 4.2 – Сравнение стандартной и предлагаемой технологии распределения программных моделей по вычислительным ресурсам

Полученная в результате использования новой технологии информация может использоваться не только методами оценки схем назначения, приведенными в данной работе (раздел 3), но и новыми методами распределения ресурсов. В этом случае процесс распределения задания на распределенное имитационное моделирование может быть значительно ускорен.

В предложенной технологии распределенного имитационного моделиро-

вания впервые использована разработанная модель процесса распределенного дискретно-событийного моделирования, которая, в отличие от существующих, позволила получить данные о динамическом изменении объемов памяти, потоков данных в локальных моделях, времени простоя, алгоритмы синхронизации модельного времени, что дало возможность выполнить установленные ограничения на объем оперативной памяти и минимизировать время простоя вычислительных ресурсов при оптимистических и консервативных методах синхронизации программных моделей. Также была использована усовершенствованная модель оценки времени выполнения программных моделей, которая, в отличие от существующей, учитывает увеличение объемов памяти программных моделей и объемы передаваемых данных между программными моделями, время простоя ресурсов, что позволило оценить время выполнения программных моделей для различных схем распределения ресурсов с учетом ограничений на объем памяти и суммарное время простоя вычислительных ресурсов.

В предложенной информационной технологии впервые использованы методы оценки схемы распределения локальных программных моделей по вычислительным ресурсам, которые на основе характеристик динамического изменения объемов их сегментов данных, дампов памяти состояний моделей и оценок простоев ресурсов, выбирают приемлемые схемы распределения ресурсов, что позволяет уменьшить время выполнения и количество вычислительных ресурсов при реализации оптимистичных и консервативных методах синхронизации программных моделей.

В следующих подразделах будет показано практическое использование предложенной информационной в распределенном имитационном моделировании GRID-систем, которая, в отличие от существующих, выбирает схемы распределения ресурсов с максимальным ходом модельного времени при ограничениях на объем оперативной памяти, времени простоя и выбирают способ синхронизации программных моделей, что позволяет повысить эффективность процесса распределенного имитационного моделирования путем уменьшения времени моделирования и количества вычислительных ресурсов.

4.1.3 Разработка информационной технологии исследования распределенных программных моделей

Информационная технология исследования распределенных программных моделей разработана для реализации четвертого этапа информационной технологии распределенного имитационного моделирования. Технология включает следующие этапы.

Этап 1: запуск программы, моделирующей поведение частных программных моделей. Инициализация начальных значений на основе файла задания (4.1) и (4.2).

Этап 2: выполнение алгоритма процесса функционирования распределенной имитационной модели (приведен в подразделе 3.6) с применением консервативного алгоритма синхронизации программных моделей (подраздел 2.3.1).

Этап 3: выполнение алгоритма процесса функционирования распределенной имитационной модели с применением оптимистического алгоритма синхронизации программных моделей (подраздел 2.3.2).

Этап 4: сохранение данных (кортеж 3.15), полученных в ходе моделирования, в базе данных с сохранением идентификатора эксперимента, атрибутами которого выступают элементы множества задания (4.1).

Таким образом, предложенная информационная технология исследования программных моделей содержит четыре этапа и позволяет получить и сохранить в базе данных информацию о статических и динамических характеристиках программных моделей. Данная информация является входной для шестого этапа информационной технологии распределенного имитационного моделирования, предложенной в подразделе 4.1.

Распределение функций системы распределенного имитационного моделирования, при внедрении разработанной информационной технологии представлены на функциональной модели (рисунок 4.3). В отличие от существующих технологий распределенного имитационного моделирования, добавлены следующие функциональные элементы:

- A6 – обеспечивающий выбор из базы данных информации о предыдущих запусках модели;

- A7 – формирующий дополнительные схемы распределения методами равномерного и случайного распределения;

- A8 – осуществляющий запуск программных модулей анализа;

- A9 – моделирующий поведение распределенной программной системы моделирования для разных методов синхронизации;

- A10, A11 и A12 – применяющих методы оценки схем распределения;

- A3 – выбирает схему распределения ресурсов на основе информации от A10-A12.

Брокер ресурсов (A4) на основе переданной схемы назначения p_k, используя сервисы передачи данных операционной системы и системы моделирования, осуществляет передачу файлов задания на удаленные вычислительные ресурсы.

На рисунке 4.3 показана взаимосвязь разработанных функциональных элементов с уже существующими в системе GRASS:

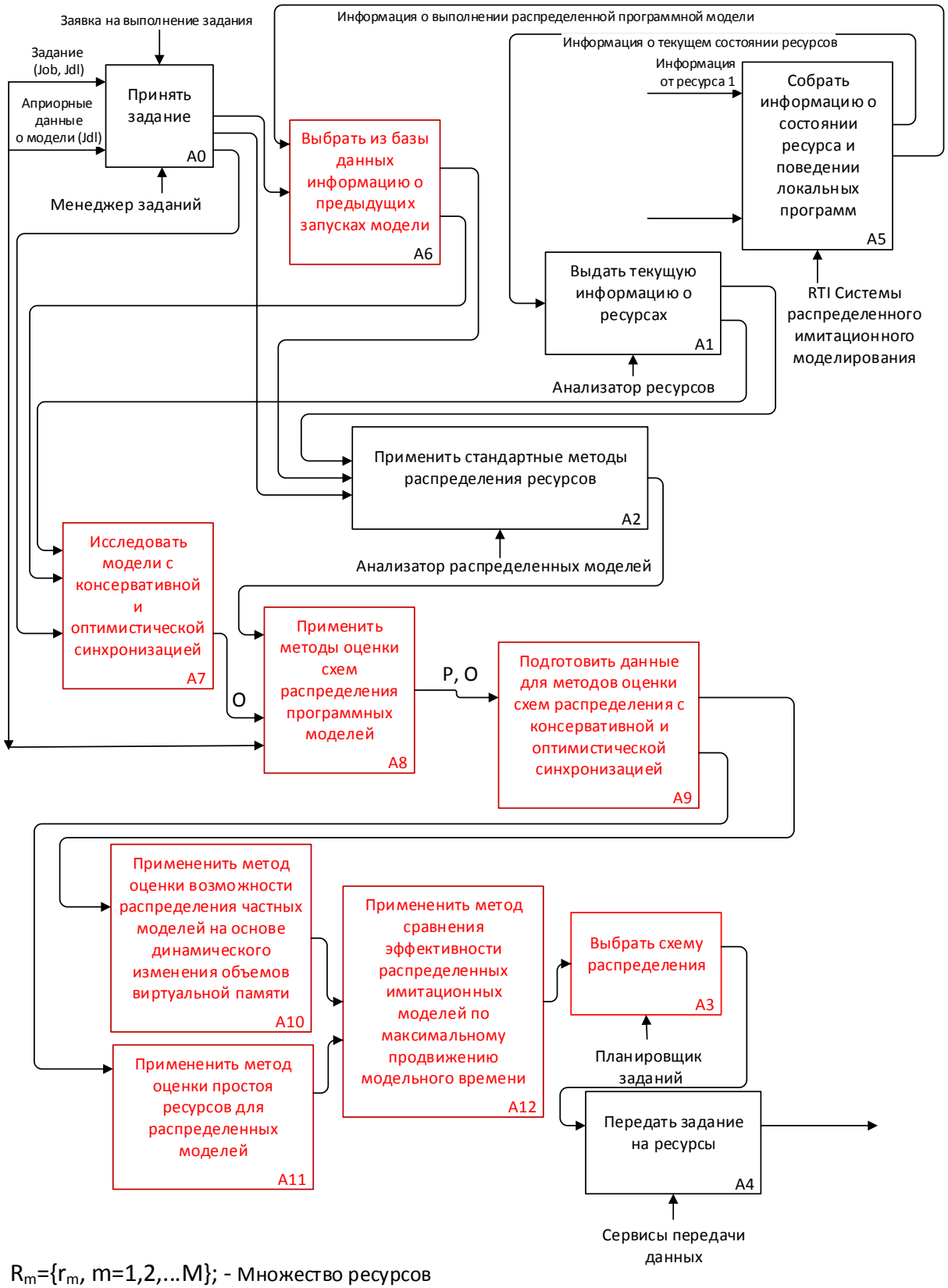
- A0 – принимает задание от пользователя в виде множества файлов (описания задания программных моделей на языке jdl, самих программных моделей и конфигурационных файлов), элементы которого соответствуют рассмотренному кортежу (4.1);

- A1 – производит удаленных мониторинг ресурсов с целью определения основных характеристик состояния вычислительного ресурса в текущий момент времени;

- A2 – генерирует множество вариантов схем распределения ресурсов стандартными и модифицированными методами;

- A3 – выбирает схему распределения ресурсов (модифицируется в предлагаемой технологии распределенного имитационного моделирования);

- A4 – осуществляет передачу программных моделей и сопутствующих файлов на выделенные ресурсы согласно схемы назначения, запуск программных моделей и инициализацию RTI, поддерживающей обмен данными и синхронизацию программных моделей.



$R_m=\{r_m, m=1,2,...M\}$; - Множество ресурсов

$P=\{p_k, k=1,2,...K\}$; - Множество схем распределения

$$O=<I, \overline{T_i}, \overline{TR_i}, \overline{TR_i^n}, \overline{TP_i^n}, \overline{V_i}, \overline{V_i^n}>, i=\overline{1,N}$$

Рисунок 4.3 – Функциональная модель информационной технологии исследования программных моделей

Предложенная технология была реализована и интегрирована в существующее программное обеспечение, построенное на базе распределенной имитационной системы моделирования GRID-инфраструктур – GRASS (GRID Advanced Simulation System), которая описана подробно в следующих подразделах.

4.1.4 Объекты представления и индексирование данных в предложенных информационных технологиях

Применение разработанных информационных технологий подразумевает создание, хранение, передачу и использование данных о программных моделях, вычислительных ресурсах, каналах передачи данных и имитационных экспериментах. Для эффективного взаимодействия подсистем, обеспечивающих процесс распределенного имитационного моделирования, необходимо создание единого протокола обмена данными и формата представления данных внутри системы моделирования.

Для представления данных использована объектная модель представления документа XML (Extensible Markup Language), который позволяет структурировать информацию разного типа с одной стороны, и имеет значительную поддержку программными шаблонами взаимодействия с существующими базами данных с другой.

В качестве первичных объектов представления данных выступает множество:

$$Q =< Q_Z , Q_R , Q_O > \qquad (4.4)$$

где Q_Z – множество данных о задании Z, соответствующее множеству (4.1) или множеству (4.2);

Q_R – множество данных о вычислительных ресурсах (4.3);

Q_O – множество данных, полученных в результате применения технологии анализа распределенных программных моделей, которые определяются кортежем (3.15).

Формирование данных множества (4.4) до создания объектной XML модели предлагается осуществлять согласно продукционным правилам, которые используются автоматически или под управлением пользователя. В отличие от объектных технологий формирования набора данных, применение продукционных правил не привязывается к строго ограниченному формату, при этом обеспечивая простой механизм задания правил формирования множества сохраняемых данных. После определения множества данных, которые подлежат сохранению, происходит автоматическое создание объекта в формате XML и сохранение его в базе данных.

Хранение продукционных правил также возможно в базе данных в виде реляционных таблиц, содержащих номер правила, область применения правила (к элементу множества Q или элементу подмножеств Q_Z, Q_R, Q_O), процедура (программный элемент) применения правила, процедура создания XML объекта:

$$(a); F(q); Pr_q(q \Rightarrow q'); Pr_{XML}(q'), \tag{4.5}$$

где a – номер продукционного правила;

$F(q)$ – функция определения применимости правила F к элементу данных $q, q \in Q$;

$Pr_q(q \Rightarrow q')$ – процедура приведения сохраняемого элемента множества Q в единый формат хранения данных системы моделирования;

$Pr_{XML}(q')$ – процедура создания объекта данных в формате XML.

В реально существующей системе возможна ситуация существования двух продукционных правил для одного и того же элемента q. Во избежание конфликтов применения правил или существования в базе данных двух и более копий, соответствующих одному и тому же элементу правил, в разработанной технологии существует ограничение: возможно существование только одного правила для каждого элемента данных. В связи с этим, каждый элемент данных q при регистрации в системе моделирования получает уникальный идентификатор (4.5).

В подразделе 4.1 отмечалось, что данные, описывающие задания и вычислительные ресурсы могут быть количественными (например, вычислительная сложность или количество процессоров) и качественные (тип операционной системы, наличие внешней памяти у ресурса). Так как в формировании множества входных данных Q принимает участие пользователь, то возможно внесение ошибочных или некорректных данных, которые не распознаются процедурой приведения данных Pr. Поэтому, во всех процедурах приведения предусмотрено исключение, которое фиксирует несовпадение форматов данных и, в случае невозможности приведения, не создает объект XML, соответствующий этой информации. Отсутствие данного объекта в базе данных, соответствующей текущему эксперименту, определяется отсутствием объекта с атрибутом (4.5).

Множество процедур приведения и создания объекта XML, таким образом, может быть проиндексировано при помощи атрибута (4.5):

$$Pr = < (Pr_q^a , Pr_{XML}^a) > \quad (4.6)$$

Практическая реализация указанных процедур предлагается в виде реализации плагинов на языке JavaScript.

В множестве Q_R целесообразно выделить два подмножества: Q_{Rs} – статические характеристики о ресурсе, которые собираются в результате мониторинга вычислительных ресурсов вне зависимости от заданий на моделирование и Q_{Rd} – параметры, собираемые от системы мониторинга во время проведения эксперимента.

Элементы множества (4.4) имеют логическую связь. Так формирование множества Q_O происходит на основе информации из множества Q_Z, и соответствует определенному эксперименту, проводимому в системе моделирования. Изменение количественных и качественных данных из множества Q_Z (значительное изменение условий имитационного эксперимента), может привести к значительному изменению параметров, характеризующих процесс моделирования (множество (3.15)). Наконец, элементы множества Q_O и схема распределе-

ния ресурсов оказывают сильное влияние на значения элементов множества Q_{Rd}.

В связи с этим, предлагается индексировать данные множества (4.4) атрибутом $b - Q^b$, который назовем атрибутом эксперимента. Данный атрибут позволяет связать исходные данные задания с множеством данных, полученных в результате моделирования и данными, получаемыми от системы мониторинга ресурсов.

При работе с системой моделирования пользователь выполняет следующие действия: формирует исходные данные эксперимента (кортеж 4.1), передает их системе моделирования, система выполняет моделирование и передает результаты моделирования пользователю. Пользователь анализирует, полученные в результате эксперимента, данные, корректирует исходные данные и повторяет эксперимент. Такой цикл повторяется многократно. При этом изменения в исходных данных могут быть незначительными или затрагивать большое количество элементов множества (4.1). Если фиксировать информацию о всех экспериментах, то объемы данных в базе могут быть значительны, а соответственно может увеличиваться время поиска соответствующей записи в базе данных.

Основываясь на том факте, что повторение экспериментов с незначительными изменениями может осуществляться уже с помощью исследованной совокупностью программных моделей и схемы назначения, предлагается ввести понятие настраиваемого коэффициента близости, определяющего наиболее подходящее множество Q^b, которое может быть использовано для выбора схемы назначения.

$$K_b = K^Z \cdot K^{Pr} \cdot K^D \cdot K^W \tag{4.7}$$

где K^Z – коэффициент близости параметров конфигурационного файла описания задания,

K^{Pr} – коэффициент близости множества программних моделей, каждая из которых требует для выполнения отдельного вычислительного ресурса,

K^D – коэффициент близости множества файлов входных данных, определяющих начальное состояние программных моделей,

K^W – коэффициент близости множества файлов, в которые записываются результаты моделирования.

В простейшем случае предлагается использовать следующее выражение для любого из коэффициентов близости:

$$K_b^Y = \begin{cases} 1, если\, Q^Y = Q_b \\ 0, если\, Q^Y \neq Q_b \end{cases} \quad (4.8)$$

то есть, коэффициент близости равен 1, если состав множества элементов описания текущего эксперимента совпадает с элементами множества описания эксперимента, сохраненного в базе данных, и равен нулю, если состав множеств отличается.

Анализ произведения (4.7) показывает, что интегральный коэффициент близости K_b равен 1 , если все коэффициенты близости равны 1 и элемент базы данных Q^b можно использовать для выбора схемы назначения. Если хотя бы один из коэффициентов равен 0, то интегральный коэффициент K_b равен 0 и запись с индексом b не может быть использована для выбора схемы назначения.

Дальнейшее развитие в определении коэффициента близости видится в задании продукционных правил его определения. При этом необходимо учитывать два направления. Первое связано с наличием количественных и качественных параметров в качестве элементов множества Q. Качественные параметры (например, в множестве Z, могут задавать тип операционной системы или архитектуру процессора) могут быть основаны на предикатном выводе. Для сравнения количественных параметров можно использовать математические и алгоритмические методы. Второе направление связано с составом программных моделей, которые могут меняться как в количественном составе, так и в качественном (например, в случае существования нескольких программных моделей, реализующих один и тот же модуль системы).

В связи с этим, в разработанной технологии предлагается ввести возможность задания продукционных правил, определяющих коэффициенты близости множества параметров текущего и предыдущих экспериментов (сохраненных в базе данных) на основе подключаемых программных модулей. Индексация правил может быть сквозная целочисленная (начиная с 1) или при помощи атрибутов подмножества данных:

$$(Y); F(Q); Pr_Y((Q, Q^Y) \Rightarrow K^Y), \tag{4.9}$$

где Y – атрибут продукционного правила, соответствующий подмножеству данных множества Q;

$F(Q)$ – функция определения применимости правила F к множеству данных текущего эксперимента Q (при значительном изменении условий и состава модулей вычислительного эксперимента, правило может быть пропущено);

$Pr_Y((Q, Q^Y) \Rightarrow K^Y)$ – процедура определения коэффициента близости K^Y.

Интегральный коэффициент близости подсчитывается по формуле (4.7). В приведенном варианте, множество возможных значений коэффициента близости включает два значения К={0,1}. Однако, не исключается возможность создания более сложных методов его определения с большим количеством возможных значений (в том числе и непрерывного значения коэффициента близости).

Каждый из первичных объектов представления данных (4.4) представляет собой объектную модель в XML формате. В примере 4.1 показано оформление задания пользователя на основе выражения 4.1.

```
<?xml version="1.0" encoding="utf-8"?>
<Task>
    <Z>
     <processor > Intel Core i3 </ processor >
     <operation system> Scientific Linux <ver> 4.0</ver> </ operation
system>
     <RAM> 1200 </RAM>
     <HDD> 3000 </HDD>
     <priority> 1 <priority>
          <parameter pluginsDirectory =
```

```
"//WS39–2/C:/simulation/plugins/"/>
            …
    </Z>
    <program models>
            <plugin name="AlgorithmLoader" filename="algorithm_loader">
                    <parameter TasksCount="10"/>
                    <parameter Algorithm="RandomDistributionAlgorithm"/>
            </plugin>
            <plugin name = "DataManager" filename = "data_manager" />
            <plugin name = "Queue" filename = "queue" />
            <plugin name = "RandomDistributionAlgorithm" filename =
"rda"/>
            <plugin name = "ResourcesController" filename = "rc" />
       <plugin name = "SimpleResourcesManager" filename = "srm" />
       <plugin name = "SimpleTasksGenerator" filename = "stm"/>
</program models>
<Data>
     <file = "Task stream" filename = "Matrix.dat" />
</Data>
<Result>
     <file = "results" filename = "out.res"/>
<Result >
</Task>
```

Пример 4.1 – Оформление задания пользователя

В приведенном фрагменте XML документа раздел Task определяет описание задания, которое ставиться на моделирование, раздел program models – множество программных моделей, с указанием имен исполняемых файлов. Сами файлы размещаются на сетевом диске, путь к актуальному каталогу которого указывается с помощью параметра *pluginsDirectory = "//WS39–2/C:/simulation/plugins/"* в подразделе задание (идентифицируется тегом <*Z*>).

Файлы исходных данных задаются в разделе <*Data*>. В приведенном примере в качестве исходного файла данных указан файл *"Matrix.dat"*, который будет передан на все вычислительные ресурсы, на которые будут распределены программные модели. Существует возможность привязать конкретный файл данных к отдельной программной модели. В таком случае, система моделирования передаст файл данных только на тот вычислительный ресурс, на который будет распределена программная модель:

```
<file = "Task stream" filename = "Matrix.dat" user = "DataManager"/>.
```

Если программная модель с именем *"DataManager"* не будет найдена, то файл *"Matrix.dat"* будет проигнорирован, а в журнал системы моделирования

будет внесена соответствующая запись.

Аналогичным образом можно привязать файл результатов к программной модели:

```
<file = "results" filename = "out.res" user = "DataManager"/>.
```

В этом случае, система моделирования по окончанию эксперимента будет искать файл результатов с именем "out.res" только на том вычислительном ресурсе, на который была распределена программная модель *"DataManager"*.

Более подробно с примером файла описания эксперимента можно ознакомиться в приложении В.

4.2 Разработка и интеграция программного обеспечения в состав системы моделирования GRASS

Программное обеспечение, реализованное в рамках диссертационной работы можно разбить на две части. Первая из них реализует алгоритмы анализа распределенных имитационных моделей и описана в подразделе 4.2.1 Вторая – распределенная система моделирования GRASS, которая использовалась для проведения экспериментов по оценке результатов, полученных в результате диссертационной работы.

Система моделирования GRASS описана в подразделе 4.2.2 Кроме этого, ее структурой, функциональность и особенность использования прошла апробацию [24].

4.2.1 Программные средства анализа распределенных имитационных моделей

После запуска, программа переходит в состояние ожидания ввода параметров модели, что соответствует получению сообщения WM_COMMAND от компонента edit – IDC_STEP1_OK. Программа считывает значение, введенное пользователем с помощью функции *GetEditValue()* и, если оно попадает в интервал от 0 до 100, то программа переходит на следующий шаг, в противном

случае выдает сообщение о некорректности введенных данных.

На втором шаге пользователь должен ввести стартовые параметры для каждой модели. Программа ожидает сообщения от одного из трёх компонентов edit, каждый из которых отвечает за заполнение одного из начальных параметров для каждой из модели.

Сообщение IDC_INPUT_MEMORY инициализирует диалоговое окно InputMemory, в котором пользователю предложено будет ввести объем памяти изначально занимаемый каждой из частных моделей. Создается n edit (где *n* – количество частных моделей) и после того как пользователь всех их заполнит и нажмет ОК произойдёт считывание параметров. Если пользователь заполнит не все поля, то программа выдаст соответствующее сообщение. Считывание происходит с помощью функции *GetDlgItemText()*, а с помощью функции *atoi()* считаные строковые значения преобразуются в целочисленные, которые затем записуются в массив и учитываются при рассчетах выходных параметров. Для удобства ввода параметров и сохранения их в отдельные файлы с целью повторного использования, был разработан модуль Input v0.1. Внедрение данного модуля позволяет пользователю ввести параметры в удобном редакторе с множеством подсказок, или выбрать файл с уже сохраненными параметры, что существенно экономит время при вводе параметров и делает программу более гибкой.

После заполнения всех первоначальных параметров программа переходит на третий шаг, в стадию моделирования. При нажатии на кнопку «Старт» приходит сообщение IDC_START которое запускает сам процесс моделирования. Запускается три потока: *CalcualteRT*, *CalcualteVT*, *Syncronize*. Первый поток служит для вычисления реального времени затраченное на работу всей системы в целом на i-ом шаге моделирования. Второй поток вычисляет реальное время, затраченное частной моделью, глобальное модельное временя, объем данных частной модели и всей системы в целом. Третий поток служит для синхронизации процесса моделирования и записи дампа. Для синхронизации использовался объект синхронизации *Event*.

Сохранение состояния выходных параметров происходит следующим об-

разом. Когда оба потока, которые выполняют расчет: реального времени, модельного времени и размера затраченной памяти, завершают текущую итерацию в действие вступает «поток синхронизации». Основной задачей, которого является добавление выходных параметров к дампу памяти. Дамп памяти представляет файл с расширением .xml. Каждый новый элемент имеет следующую структуру.

```
<Iteration>
    <RealTime size="1.24" param="msec" />
    <ModelTime size="0.57" param="msec" />
    <MemorySize size="73" param="kb" />
</Iteration>
```

После завершения своей основной задачи «синхронизирующий поток» запускает приостановленные потоки и завершает свою работу, тем самым запуская циклическую роботу подсчета параметров. Программа предусматривает остановку выполнения моделирования в любой момент времени. После этого, пользователь может посмотреть графики для каждого из параметров. Графики строятся на основе файла дампа и отображают информацию о параметре с начала запуска программы и до остановки моделирования. По оси *X* расположена шкала шагов моделирования, а по оси *Y* шкала для текущего параметра (время, размер памяти, и т.д).

Компонент вывода графиков позволяет маштабировать изображение (при помощи mouse wheel), что позволяет пользователю более детально проанализировать поведение параметров за определнный период моделирования.

Математическая часть проекта состоит из следующих функций:

- SaveStateModel() – функция, времени сохранения состояния модели на ім шаге;

- TimeWorkModel() – функция, времени работы *i*-ой частной модели;

- TimeWorkSystem() – функция, времени работы системы между двумя шагами моделирования;

- CalculationRealTime() – функция, вычисления реального времени затраченого *i*-ой частной моделью;

- CalculationGlobalTime() – функция, вычисления глобального модельного

времени;

- CalculationStepVn() – функция, вычисления шага прироста обьема памяти модели;

- CalculationVn() – функция, вычисления объема данных *i*-ой частной модели;

- VolumeDamp() – функция возвращающая объем дампа памяти;

- SendMesBetweenModels() – функция, возвращающает вероятность посылки сообщения от i-ой частной модели к остальным частным моделям умноженное на время посылки сообщения.

Оконные формы интерфейсного модуля ввода параметров модели представлены в приложении Б (рисунок Б.1).

Модуль Input v0.1 выполняет следующие функции:

- ввод параметров всех типов, с различными режимами;

- сохранение параметров в файл;

- загрузка и редактирование уже имеющихся параметров;

- создание новой таблицы для ввода с возможностью выбора: режима, параметра и размера.

Запустить модуль можно двумя способами. Первый способ: зайти в папку, где установлена программа, и выбрать бинарный файл Input.exe. В этом случае тип и режим ввода пользователь должен выбрать сам, после чего создать таблицу для ввода нажатием кнопки «создать».

Для проверки корректности выбранных файлов модуль Input v0.1 анализирует заголовочную часть файла на соответствие определенным требованиям, в случае несоответствия программа показывает сообщение, данное введение существенно повышает безопасность работы с файлами.

Второй способ: в основном окне программы выбрать необходимые параметр и нажать на кнопку «ручной ввод» (рисунок Б.2).

При таком входе в модуль, программа сама установит необходимый режим, параметр и размер ввода, затем создаст таблицу с необходимым количеством ячеек (размер таблицы зависит от количества моделей, выбранных на первом шаге моделирования в главной программе и режима). В дальнейшем

планируется ввести в модуль комплекс подсказок и ограничений в зависимости от выбранных параметров для ещё более удобной работы.

Рассмотрим инструкция пользователя по вводу информации.

Интерфейс пользователя представляет собой оконное приложение (рисунок Б.3). Для выполнения задачи моделирования пользователю нужно пройти три этапа моделирования.

На первом шаге пользователь задает количество моделей. На втором этапе вводится начальные параметры для каждой модели:

- объем памяти, занимаемый данными n-ой частной моделью;

- вероятность посылки сообщения от одной частной модели к другой частной модели;

- время передачи данных.

После этого осуществляется переход на этап моделирования. Для запуска этого процесса нужно нажать кнопку «Старт», после чего пользователь может наблюдать изменения выходных параметров. Для того, чтобы увидеть изменения этих параметров на всём этапе моделирования следует нажать кнопку «Стоп», после чего будут доступны графики по каждому из параметров. Нажав на любой из графиков можно наблюдать как изменялись эти параметры. При нажатии снова на кнопку «Старт» процесс моделирования продолжится. Вся история изменения выходных параметров на каждом шаге моделирования записывается в файл data.xml находящийся в каталоге с программой в папке data.

4.2.2 Система имитационного моделирования GRASS

Система моделирования GRASS имеет модульную архитектуру (рисунок 4.4). Она состоит из ядра и динамически подключаемых модулей (плагинов). Каждый модуль выполняет свою узкоспециализированную задачу, обращаясь при необходимости к другим модулям системы. Ядро предоставляет средства межмодульного взаимодействия, а также обеспечивает начальную загрузку и конфигурацию системы.

Каждый модуль имеет уникальный строковый идентификатор (имя, ID). Он также предоставляет набор интерфейсов взаимодействия с ним для других

компонентов системы. Каждый интерфейс модуля также имеет имя и может быть реализован любым количеством модулей. Таким образом, для получения какого-либо интерфейса модуля необходимо знать его имя и имя интерфейса взаимодействия.

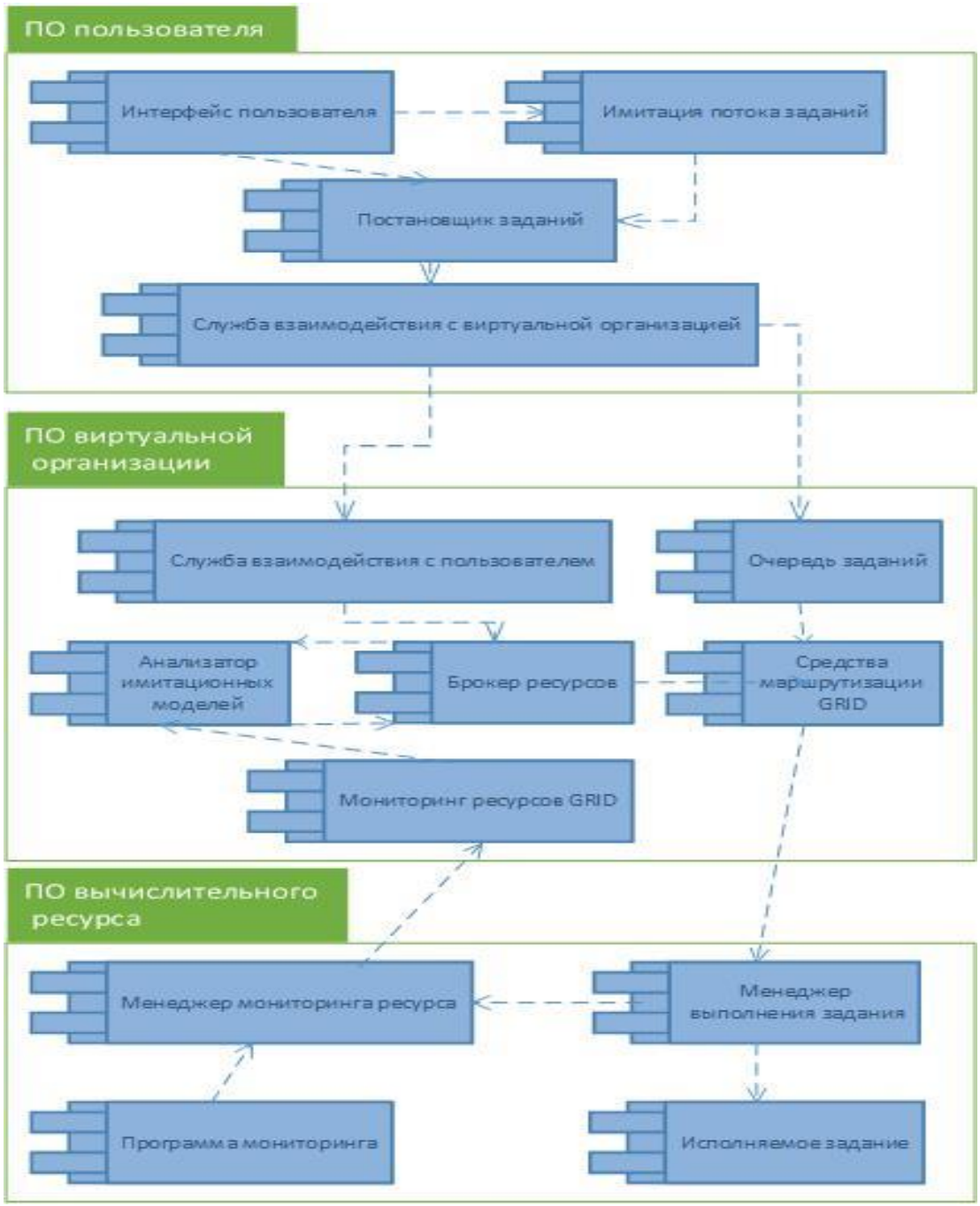


Рисунок 4.4 – Компонентная модель распределенной системы моделирования GRASS

Модуль в системе GRASS представляет собой динамически подключаемую библиотеку (dynamic linked library (dll) – в ОС семейства Microsoft

Windows, shared objects (so) – в UNIX-подобных ОС), которая реализует фабричный метод (factory method), создающий экземпляр класса модуля. Класс модуля реализует интерфейс *Framework::IPlugin*, что позволяет универсально работать с ним, не учитывая особенности реализации и тот адаптер, который согласует его работу.

Этот интерфейс включает в себя базовые операции, которые обязан предоставить системе каждый модуль:

- инициализация интерфейса;
- получение имени модуля;
- получение интерфейса с заданным именем;
- инициализация (необязательно);
- завершение работы (необязательно).

Компонентная диаграмма программной системы распределенного моделирования GRASS представлена на рисунке 4.4. Программное обеспечение системы GRASS разбито на три группы: программное обеспечение пользователя, виртуальной организации, вычислительного ресурса. Каждая группа обычно представляет собой множество программ на одном вычислительном ресурсе. В качестве ресурса выступает, как правило, отдельный компьютер.

Каждый интерфейс, предоставляемый модулем, также должен быть унаследован от стандартного класса *Framework::IPluginInterface*, который позволяет унифицировать все интерфейсы модулей в системе. Основным запросом к нему является получение имени в виде строкового идентификатора. Общая структурная схема отношений и взаимодействия модулей, их интерфейсов и ядра показана на рисунке 4.5.

На диаграмме также представлены классы примера реализации основных интерфейсов подключаемого модуля: *Example::Plugin* – главный класс реализации плагина, Example::IExample – один из его интерфейсов.

Запрос интерфейса одного модуля другим выглядит следующим образом. Запрашивающий модуль вызывает метод ядра *getPluginInterface()*, передавая ему в качестве аргументов имя модуля и имя его интерфейса.

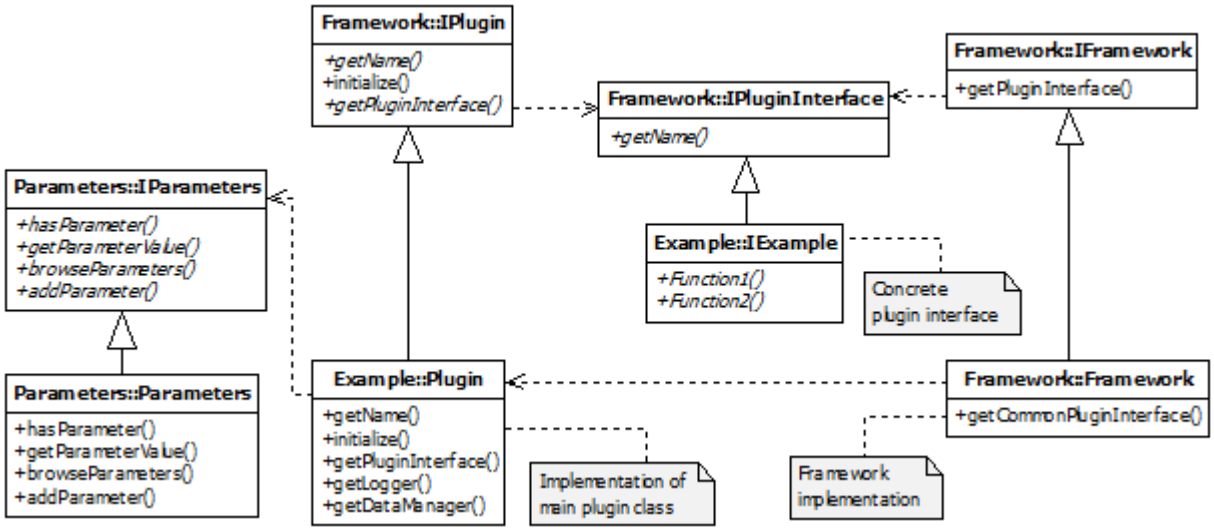


Рисунок 4.5 – Структурная схема взаимодействия модулей системы и ядра

Ядро проверяет наличие модуля с заданным именем и его состояние. Если модуль не найден или уже была неудачная попытка загрузить его, возвращается ошибка. Если это первое обращение к модулю, и он еще не был загружен, выполняется его загрузка в память, инициализация и конфигурирование.

Если загрузка прошла успешно, ядро вызывает метод *getPluginInterface()* и получает требуемый интерфейс модуля по имени, если он поддерживается.

При успешном запросе ядро выполняет проверку соответствия имени полученного интерфейса и возвращает его, выполняя преобразование к требуемому типу C++.

Получение доступа к интерфейсам других плагинов происходит через главный интерфейс ядра *Framework::IFramework*. Он реализуется классом *Framework::Framework*, который осуществляет обработку конфигурационного файла и начальную загрузку плагинов. Интерфейс *Parameters::IParameters*, а также его реализация Parameters::Parameters предназначены для передачи подключаемым модулям начальных параметров, заданных в конфигурационном файле.

Рассмотрим преимущества предложенной модульной архитектуры системы моделирования.

Гибкость настройки. Система может быть запущена с различным количественным и качественным составом модулей. Например, одно и то же прило-

жение без перекомпиляции может быть запущено в графическом или консольном режиме в зависимости от установленных модулей визуализации. Аналогичным образом можно запускать локальную или сетевую версию приложения, интерактивную версию или версию, которая запускается из командной строки и генерирует отчет. Разумеется, набор модулей должен быть достаточным для выполнения поставленной задачи, т. е. если какой-либо критически необходимый модуль отсутствует, корректная работа системы в данной конфигурации будет невозможна.

Простота модификации. Изменение технологий, разработка новых методов и алгоритмов, а также исправление ошибок в уже готовых реализациях неизбежно приводят к модификации существующих систем. С использованием модульной архитектуры, для обновления системы, достаточно лишь заменить реализацию старого компонента на новую или исправленную, поддерживающую прежние интерфейсы. При этом нет необходимости вносить изменения в остальные компоненты системы.

Упрощение разработки отдельных компонентов. Используя модульную архитектуру, можно легко реализовать один из основных принципов программирования – декомпозицию задачи на подзадачи, каждая из которых реализуется отдельным модулем. Соответственно, различные модули могут быть реализованы разными разработчиками независимо друг от друга после составления спецификации интерфейсов межмодульного взаимодействия. Кроме того, в большой системе важной также является возможность перекомпиляции одного модуля без сборки других, что значительно экономит время и силы разработчиков.

Упрощение тестирования. Модульная архитектура позволяет легко производить unit-тестирование как отдельно взятого модуля или группы, так и всей системы в целом. Для этого следует реализовать соответствующие тестовые модули, содержащие исходные данные, результаты тестирования, а также механизмы сравнения полученных данных с эталонными, и сконфигурировать систему таким образом, чтобы она использовала их. Кроме того, могут быть разработаны служебные тестовые модули, содержащие общий часто используемый

код. Более того, можно также сравнивать между собой результаты работы различных реализаций одного и того же логического элемента (алгоритма) для проверки корректности работы обоих.

Возможность иметь несколько реализаций одного и того же модуля. Модульная архитектура идеально подходит для сравнения различных реализаций одного и того же логического элемента. Это может быть алгоритм защиты, распределения общей нагрузки, расчета какой-либо величины или реализация функциональности для конкретной платформы. Имея несколько модулей, реализующих одну и ту же логику, мы можем поочередно запустить их на одних и тех же данных за счет конфигурации состава системы и, сравнив результаты работы, сделать выводы об их эффективности. Кроме того, модульная архитектура легко позволяет динамически во время исполнения выбирать наиболее эффективный модуль в зависимости от исходных данных или предоставленных ресурсов.

Возможность повторного использования кода. Модульная архитектура позволяет использовать одни и те же наработки в различных проектах, реализованных на основе одинаковых или совместимых ядер. Это могут быть часто используемые компоненты, такие как системы ведения журналов, логов, статистики, менеджеры памяти, компоненты графического интерфейса пользователя (GUI), модули интернационализации и т.д.

Ленивая загрузка – одно из существенных преимуществ модульной системы, которое позволяет экономить оперативную память компьютера, а также время запуска приложения. Это означает, что модуль не будет загружен до тех пор, пока не возникнет необходимость в функциональности, предоставляемой им. Хотя большинство модулей системы следуют этому правилу, возникает необходимость загрузки некоторых модулей при старте системы, чтобы начать выполнять какую-либо работу, вызывая в дальнейшем другие компоненты.

Для создания гибкой модульной системы необходима возможность простого изменения набора плагинов и их параметров без модификации ядра или самих библиотек модулей. Для этого в GRASS используется система конфигурационных файлов на основе XML. Основной конфигурационный файл под-

ключаемых модулей plugins.xml имеет вид, показанный в приложении В (листинг В.1).

Он описывает взаимосвязи между именами модулей в системе и названиями файлов их библиотек. Кроме того, он позволяет задать параметры, передаваемые модулям при загрузке, которые могут быть использованы для задания режима работы или инициализации внутренних значений. Некоторые из этих параметров могут также задавать путь к другим файлам, содержащим более детальную информацию о настройках для конкретного модуля.

4.2.3 Разработка подсистемы визуализации данных распределенной имитационной системы моделирования GRID

В современных программных системах одним из основных элементов является интерфейс пользователя, который оказывает сильное влияние на скорость освоения приложений, время выполнения прикладных задач, стоимость программных разработок, производительность отдельных сотрудников и целых организаций. В последнее время появилось множество научных публикаций в этой предметной области [153,154]. Наиболее актуальной и обсуждаемой является построение визуального интерфейса распределенных и параллельных систем [155,156], ярким представителем которой является GRID-инфраструктура [157].

В данном подразделе приведено описание модели визуального интерфейса для распределенной имитационной системы моделирования GRID-инфраструктуры – GRASS (GRID Advanced Simulation System), разрабатываемой в Харьковском национальном университете радиоэлектроники [40, 146].

Рассмотрим состав системы визуализации информации, получаемой в процессе функционирования системы распределенного имитационного моделирования GRASS.

В распределенной имитационной модели GRID функциональность визуализации данных не является непосредственно частью структуры системы, а наряду с другими подсистемами выделена в отдельные модули: «Windows

Manager» и множество модулей-адаптеров.

Модуль «Windows Manager» представляет собой основу подсистемы визуализации, которая при запуске системы в графическом режиме создает главное окно, а также предоставляет методы, позволяющие другим компонентам системы регистрировать собственные графические элементы, панели инструментов и пункты меню. «Windows Manager» также занимается размещением и группировкой этих графических элементов внутри главного окна.

Модули-адаптеры, являютщиеся вспомогательными компонентами системы, создают графические элементы, регистрируют их в главном окне с помощью модуля «Windows Manager» и обеспечивают взаимодействие графического интерфейса пользователя с основными модулями системы, а также своевременное обновление данных, предоставляемых этими модулями, в главном окне.

Как и любой модуль в системе, «Windows Manager» поддерживает и реализует интерфейс *Framework::IPlugin* [146]. Вся логика работы менеджера окон расположена непосредственно в классе *WindowsManager*, с которым связан класс плагина (plug-in) отношением агрегирования [158].

Система конфигурируется таким образом, что при запуске в графическом режиме плагин менеджера окон загружается автоматически, создавая при этом главное окно и ожидая регистрации графических элементов другими плагинами. Класс Plugin – основной класс, описывающий данный модуль в системе распределенного имитационного моделирования GRID. Данный модуль реализует интерфейс IPlugin, описание методов которого представлено ниже. Метод *initialize()* вызывается при запуске системы и загрузке данного модуля. Все действия, которые необходимо выполнить до непосредственного начала работы модуля, описаны в этом методе.

Для получения имени модуля необходимо вызвать метод *getName()*. Метод *getPluginInterface()* в свою очередь позволяет получить один из интерфейсов модуля по его имени.

Класс *WindowsManager* наследуется от класса *QMainWindow* библиотеки

Qt, а также предоставляет интерфейс, описанный в *IWindowsManager* и содержащий нижеприведенные методы.

Метод *registerWidget()* позволяет любому другому модулю системы зарегистрировать в главном графическом окне системы собственный элемент. Ответственность за расположение этого графического элемента ложится на *WindowsManager*, в то время как обновление состояния должен производить модуль-адаптер.

Методы *registerMenu()* и *registerToolBar()* предоставляют возможность другим модулям системы добавлять в главное окно собственные пункты меню и собственные панели инструментов соответственно. Как и в случае с регистрацией графического элемента, обработка событий для добавляемых меню/панелей осуществляется модулем, инициирующим регистрацию.

Таким образом, *WindowsManager* играет пассивную роль и занимается лишь правильным отображением и размещением графических элементов в главном окне, а инициировать процесс добавления этих элементов должен непосредственно каждый модуль в отдельности.

4.2.4 Разработка модулей-адаптеров для подключения новых модулей к системе GRASS

Модули-адаптеры используются для того чтобы снизить связанность (coupling) системы и повысить связность (cohesion) каждого отдельно взятого модуля, следуя правилу эффективной разработки «low coupling – high cohesion». В стандартной ситуации каждому модулю системы помимо выполнения своей основной работы пришлось бы «заботиться» и о выводе информации на экран оператора системы. С добавлением нового функционала сложность модуля растет экспоненциально, то есть уровень связности падает, а уровень связанности наоборот возрастает вследствие корреляции этих двух параметров, что приводит к затруднению дальнейшей модернизации и сопровождения системы.

Модули-адаптеры, призванные решить эту проблему, представляют собой дополнительную прослойку между графической и аналитической

подсистемами, то есть играют роль связующего звена.

Для каждого модуля, чьи данные необходимо выводить в графическом режиме работы системы, создается дополнительный модуль-адаптер, который так же, как и все, наследует и реализует интерфейс *IPlugin*. При этом сам модуль-адаптер не предоставляет собственного интерфейса, так как всю работу выполняет на этапе загрузки. Таким образом, система должна быть сконфигурирована так, чтобы все необходимые модули-адаптеры автоматически загружались при старте.

Каждый модуль-адаптер может иметь различное внутреннее устройство, однако должен выполнять как минимум одну функцию из трех возможных: инициировать регистрацию графического элемента, инициировать добавление пунктов меню и/или инициировать добавление панели инструментов.

Все регистрируемые компоненты хранятся внутри самого модуля-адаптера, поэтому типичная реализация модуля, выполняющего все три функции, подразумевает наличие внутреннего класса *Widget*, а также набора действий, инкапсулированных в классах *QAction*, на основе которых могут создаваться как пункты меню, так и кнопки панелей инструментов.

При старте системы и последующей инициализации модуля-адаптера вызывается его конструктор, который обеспечивает создание всех необходимых графических компонентов и их регистрацию в менеджере окон, после чего за их отображение отвечает «Windows Manager». Обновление данных в уже зарегистрированном и отображаемом графическом элементе производит модуль-адаптер. Связь адаптера с модулем, предоставляющим данные, производится на основе паттерна (pattern) проектирования «Наблюдатель» (Observer). Согласно этому паттерну, модуль, предоставляющий данные, реализует метод *addListener()*, а модуль-адаптер реализует метод notify() и подписывается на события, происходящие в основном модуле. Соответствующая диаграмма классов приведена на рисунке 4.6.

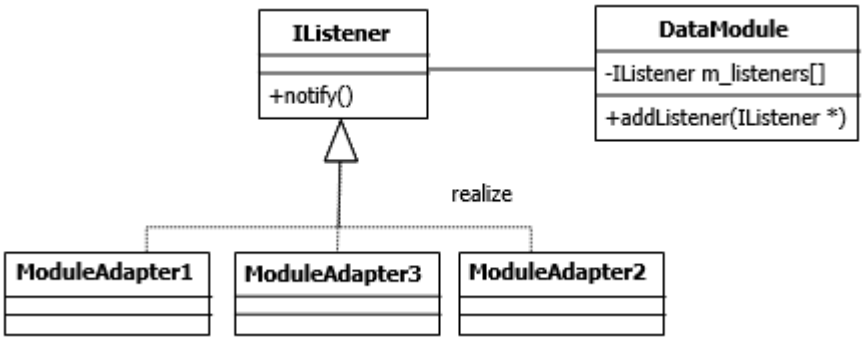


Рисунок 4.6 – Диаграмма отношений основного модуля с модулями-адаптерами на основе паттерна проектирования «Наблюдатель»

Опишем схему работы: модуль-адаптер подписывается на события, в предоставляющем данные модуле, посредством вызова метода *addListener()*, где в качестве «наблюдателя», или «слушателя» указывает себя. Этот модуль сохраняет слушателя в соответствующей коллекции, и когда данные обновляются, он вызывает для всех зарегистрированных в коллекции слушателей (модулей-адаптеров) метод *notify()*. Оповещенный модуль-адаптер повторно считывает данные из основного модуля через открытый интерфейс.

Такой подход позволяет не только присоединять несколько модулей-адаптеров к одному основному модулю, позволяя выводить информацию от одного и того же модуля в различном виде, но и присоединять один модуль-адаптер к нескольким основным модулям, чтобы выводить общую информацию, поступившую от каждого из них.

4.3 Экспериментальные исследования применения методов анализа распределенных имитационных моделей

4.3.1 Технические характеристики использованного оборудования и программного обеспечения, критерии сравнения теоретических и экспериментальных результатов исследований

В качестве аппаратной среды для проведения экспериментов выбрана локальная сеть персональных компьютеров следующей конфигурации: 10 персональных компьютеров.

Для сравнения теоретических числовых характеристик, оценивающих

функционирование распределенных имитационных моделей, с экспериментально полученными, будем использовать значение среднего арифметического относительной погрешности и значение среднеквадратического отклонения [167]. Пусть x_i – вычисленное i-е значение некоторого параметра X, $x_i^{'}$ – значение этого же параметра, полученное экспериментальным путем. Абсолютная погрешность в этом случае будет равна $\left|x_i^{'} - x_i\right|$, а относительная погрешность определяется формулой $\frac{\left|x_i^{'} - x_i\right|}{x_i^{'}}$. Значение среднего арифметического относительной погрешности $m_{\Delta x}$ и среднеквадратическое отклонение теоретических оценок от экспериментальных σ_x будем определять по формулам

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{N}\left(x_i^{'} - x_i\right)^2}{N}}, \tag{4.10}$$

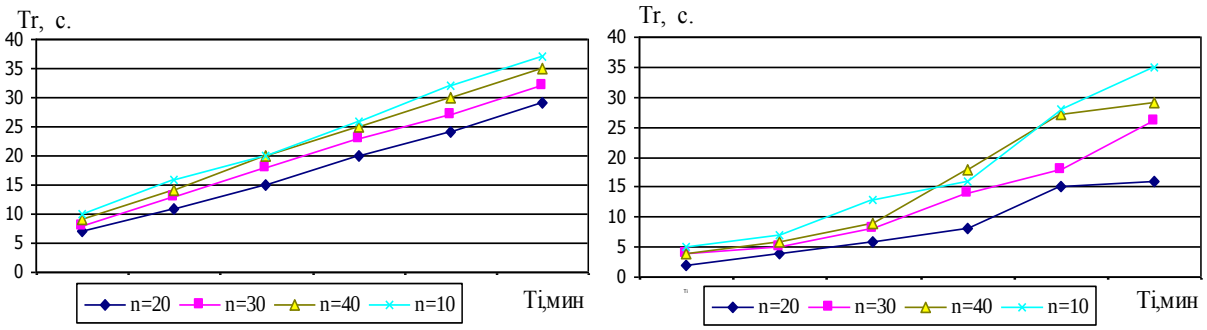
$$m_{\Delta x} = \frac{\sum_{i=1}^{N}\frac{\left|x_i^{'} - x_i\right|}{x_i^{'}}}{N}, \tag{4.11}$$

где N – количество полученных значений характеристики.

4.3.2. Применения метода анализа имитационных моделей по продвижению модельного времени

Как указывалось в разделе 3.5, для сравнения эффективности имитационных моделей часто используется критерий наибольшего продвижения модельного времени за одно и тоже количество шагов моделирования или за одно и тоже реальное время функционирования модели. Выполним выборочный анализ полученных результатов.

На рисунке 4.7 представлены результаты моделирования и измерений реальных временных затрат с использованием консервативных и оптимистических алгоритмов синхронизации.



а) б)

Рисунок 4.7 – Зависимость реального времени моделирования от значения модельного времени при использовании а) консервативных, б) оптимистических алгоритмов синхронизации

Анализируя результаты моделирования можно сделать следующие выводы. Для приведенных параметров модели более эффективно использовать оптимистические алгоритмы синхронизации, которые, в целом, дают лучшие значения времени моделирования. Нелинейности времени исполнения моделей при оптимистической синхронизации связаны с неравномерной вычислительной нагрузкой, обусловленной откатами модельного времени. Рост времени исполнения моделей в консервативных алгоритмах носит линейный характер, а, значит более прогнозируемый.

Интересные результаты показывает анализ времени моделирования при увеличении количества выделяемых вычислительных ресурсов. Так, при изменении количества ресурсов в интервале $n=[5,20]$ для моделируемой системы наблюдается значительный рост производительности, который замедляется в интервале $n=[5,25]$, а при значениях более 30 ресурсов наблюдается обратное снижение производительности, что фиксируется как теоретическими, так и экспериментальными результатами. Объяснение этому лежит в том, что при значительном количестве ресурсов повышается время, которое тратиться на синхронизацию большего количества моделей, увеличивается время, связанное с увеличением трафика модели и системы моделирования.

В ходе экспериментов, было замечено, что время исполнения модели также зависит от алгоритма распределения ресурсов перед моделированием. На

рисунке 4.7 можно видеть результаты анализа времени продвижения моделей при различных алгоритмах синхронизации и методах распределения ресурсов.

Анализ показывает значительную зависимость продвижения модельного времени от выбранного алгоритма распределения ресурсов. Так, алгоритмы распределения, основанные на учете производительности и трафика показывают улучшение эффективности до 2 раз. Таким образом, проведение данного анализа способно значительно повысить эффективность использования дорогостоящих вычислительных ресурсов.

Также был проведен сравнительный анализ результатов, полученных во время моделирования, и выдаваемых системой управления кластером. На рисунках 4.8, 4.9 показаны графики времени исполнения модели для разных алгоритмов синхронизации при равном количестве вычислительных ресурсов (*n=10*).

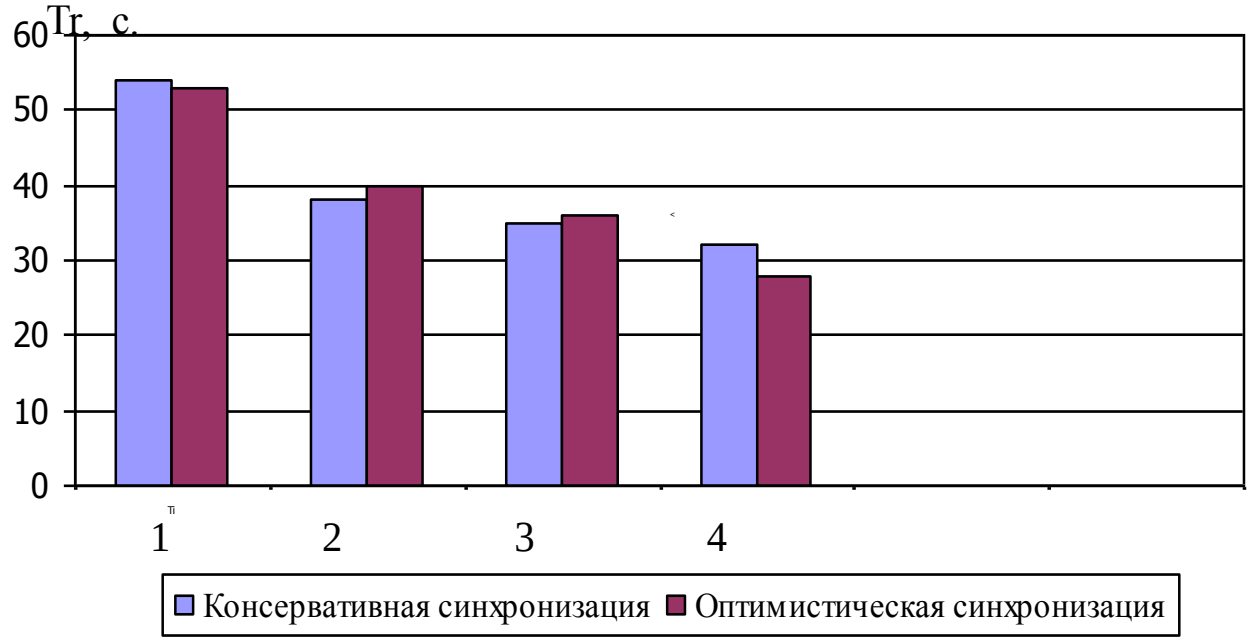


Рисунок 4.8 – Зависимость реального времени моделирования от выбранного метода синхронизации и метода распределения ресурсов: 1 – случайное распределение, 2 – на основе производительности, 3 – на основе сетевого обмена, 4 – комбинированного метода

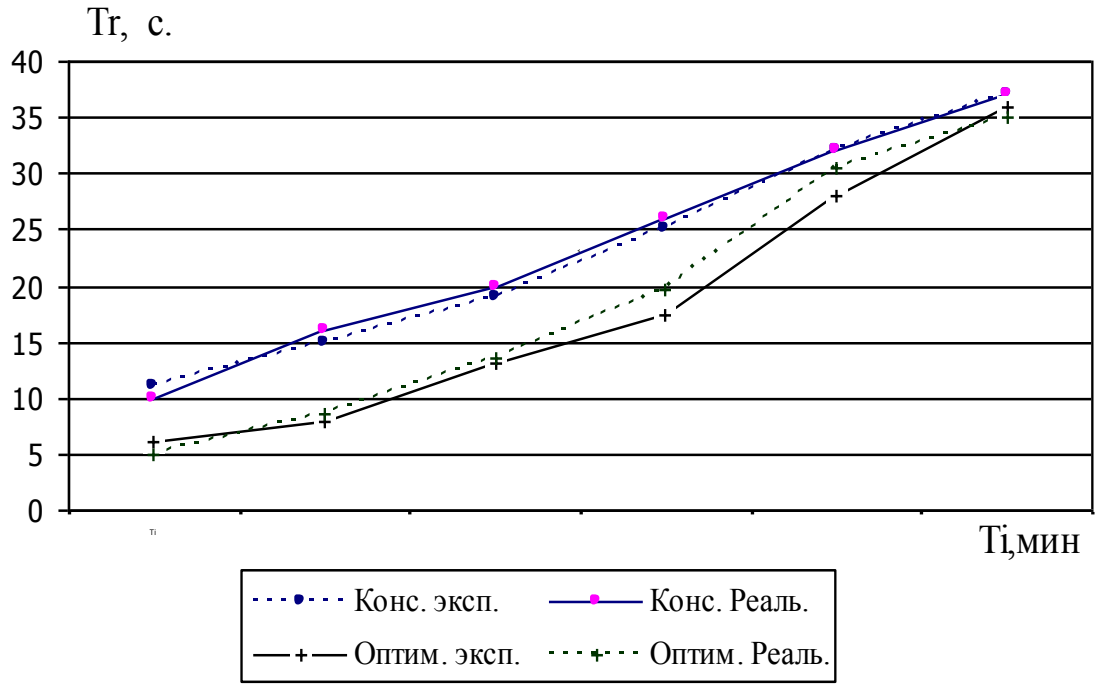


Рисунок 4.9 – Экспериментальные и теоретические зависимости времени имитации при консервативных и оптимистических методах синхронизации

Статистический анализ сравнения теоретических и экспериментальных данных показал следующие значения параметров для всех выборок, приведенных в приложении В: для консервативных алгоритмов – σ_x = 0.8 с, $m_{\Delta x}$ = 0.5с, для оптимистических – σ_x = 2,3 с, $m_{\Delta x}$ = 1.7 с. Анализ приведенных параметров показывает, что погрешность теоретических значений оценки времени моделирования составляет для консервативных алгоритмов 2,3%, для оптимистических – 3,4%. Таким образом, более точные оценки получены для консервативных алгоритмов. Это вызвано тем, что при оптимистических алгоритмах не учитывается ряд активностей операционной системы, которые выполняются как следствие работы специфических действий оптимистических алгоритмов: динамическое перераспределение памяти при сохранении состояний частных моделей и откатах, создание потоков сообщений при возвратах модельного времени и.т.д.

В целом, погрешности теоретических результатов удовлетворяют задачам анализа имитационных моделей по продвижению модельного времени.

4.3.3 Применение метода оценки возможности осуществления распределения частных моделей на основе динамического изменения объемов виртуальной памяти

В качестве конфигурации модели для исследования применения данного метода были выбраны оптимистические алгоритмы синхронизации. Это объясняется тем, что в консервативных алгоритмах, как правило, изменение объемов памяти во время имитации незначительно. В оптимистических алгоритмах же происходит постоянное изменение объемов памяти, связанное с постоянным сохранением состояний модели и реализацией откатов модельного времени.

На рисунке 4.10 приведены примеры зависимостей объемов памяти частных моделей во времени, полученных в ходе моделирования.

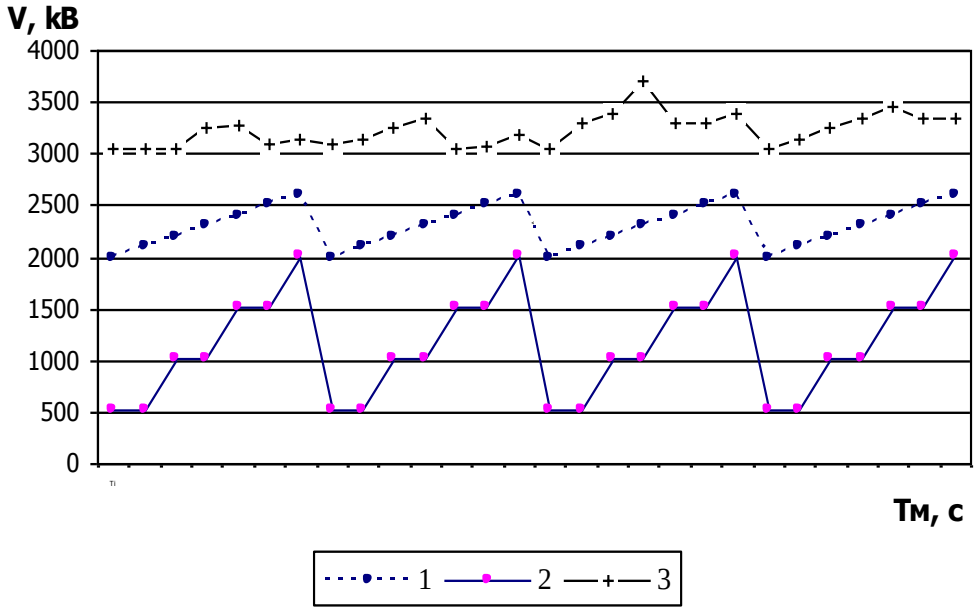


Рисунок 4.10 – Зависимость объемов памяти частных моделей

Как видно из рисунка, анализ изменения объемов памяти частных моделей позволяет определить, как количественные параметры, так и характер изменений. Так графики 1 и 2 носят периодический характер. В рамках временного окна память этих моделей возрастает, причем рост частной модели 1 носит нелинейный, а 2-ой – линейный характер. Изменения частной модели 3 носят асинхронный нелинейный характер. Характер изменений памяти модели 3 поз-

воляет сделать вывод, что во время ее исполнения часто происходит откат модельного времени (уменьшение объемов памяти между моментами продвижения временного окна).

Полученные в ходе анализа данные позволяют также оценить реальный максимум объем памяти частной модели, который проявляется в динамике, в отличие от заявленных в задании объемов, носящих априорный характер, и часто указываются приблизительно. Так, например, для первой модели (рисунок 4.7) начальный (заявленный) объем памяти составляет 500 кбайт, а во время моделирования динамически увеличивается до 2000 кбайт.

Оценка максимальных объемов памяти частных моделей является ограничивающим параметром при поиске подходящих для моделирования ресурсов. Кроме этого, она может быть использована в сложных статических и динамических алгоритмах планирования и распределения ресурсов.

4.3.4 Применение метода оценок простоя ресурсов для алгоритмов синхронизации распределенных имитационных моделей

При распределенном моделировании существует проблема простоя ресурсов, вызванная неоднородностью частных моделей или неоднородностью ресурсов. Параллельное исполнение частных моделей неизбежно приводит к ситуации, когда часть моделей вынуждена ожидать в точке синхронизации другие частные модели. Иногда простои ресурсов значительны. Решениями данной проблемы являются методы статического и динамического распределения (перераспределения) ресурсов.

Однако данные методы требуют знания дополнительных параметров о модели, которые отсутствуют в языках описания заданий, а также сложны для оценивания разработчиками моделей. Предложенный в разделе 3.7 метод позволяет оценить простои частных моделей.

Простой ресурсов наиболее проявляется при применении синхронных и консервативных алгоритмах синхронизации, которые ограничивают возможности частных моделей по продвижению модельного времени. Система анализа распределенных имитационных моделей позволяет получить зависимости вре-

мени простоя ресурсов в пространстве физического времени моделирования.

На рисунке 4.11 представлены несколько графиков такой зависимости. В качестве функции представлено суммарное время простоя ресурсов для разных алгоритмов синхронизации.

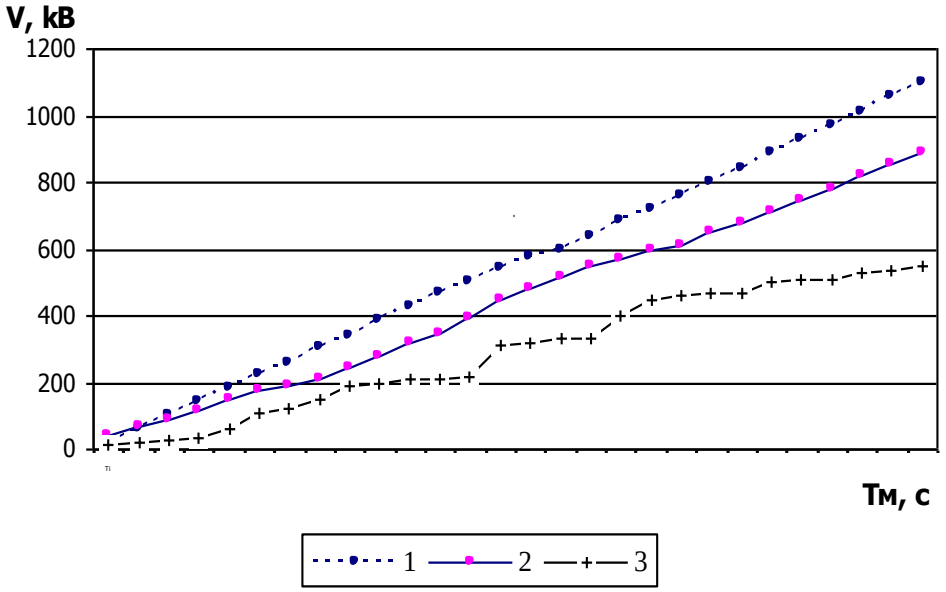


Рисунок 4.11 – Зависимости суммарного времени простоя ресурсов для разных алгоритмов синхронизации

Время простоя ресурсов в динамике является возрастающей функцией для всех моделей. Первый график отражает поведение синхронных моделей. Время простоя для них является функцией, близкой к линейной, величина простоя наибольшая среди всей примененных алгоритмов синхронизации. Применение более сложных алгоритмов синхронизации приводит к уменьшению времени простоя ресурсов, но может увеличить общее время эксперимента за счет времени работы алгоритмов синхронизации. Так, консервативные алгоритмы (график 2) также имеют возрастающий характер времени простоя, однако, за счет дополнительных затрат процессорного времени и сетевого трафика на синхронизирующие сообщения, показывают, в общем, меньшее значение времени простоя в сравнении с синхронными алгоритмами. Оптимистические алгоритмы, в силу наличия значительных вычислительных затрат на синхронизацию и откаты, а также возможности моделирования частных моделей «вперед», приводят к самым малым значениям времени простоя, носящим нерегулярный

характер.

Следует отметить, что само по себе время простоя не является конечным критерием оценки эффективности применения того или иного метода синхронизации. Действительно, возможна ситуация, когда время простоя минимально, но физическое время, затраченное на моделирование фиксированного количества шагов распределенной модели меньше. Таким образом, полученные зависимости для времени простоя должны учитываться в обобщенных критериях эффективности.

Перспективным также является использование полученной информации о временах простоев в алгоритмах первоначального и динамического (в процессе исполнения программных моделей) распределения ресурсов. При этом открывается возможность размещать на некоторых вычислительных ресурсах несколько частных моделей, имеющих высокие значения времени простоя на одних и тех же интервалах модельного времени.

4.3.5 Исследование эффективности применения технологии распределенного имитационного моделирования с использованием методов оценки схем распределения ресурсов

Анализ времени, затраченного при проведении описанных выше (подраздел 4.3) четырех имитационных экспериментов приведен на рисунке 4.12, а результаты исследований основных параметров, участвующих в оценке эффективности выбранного варианта распределения – в таблице 4.1. Из диаграммы видно, что при простых экспериментах (малый вычислительный объем или небольшое значение времени моделирования) применение технологии анализа не является рациональным. Остальные эксперименты показали эффективность применения технологии, что проявилось в уменьшении времени проведения экспериментов в случае применения разработанных методов для оценки вариантов распределения ресурсов. В процессе работы не разработан универсальный критерий, позволяющий обобщить оценочную информацию о различных вариантах распределения ресурсов, что дало бы возможность создать универсальный метод распределения ресурсов.

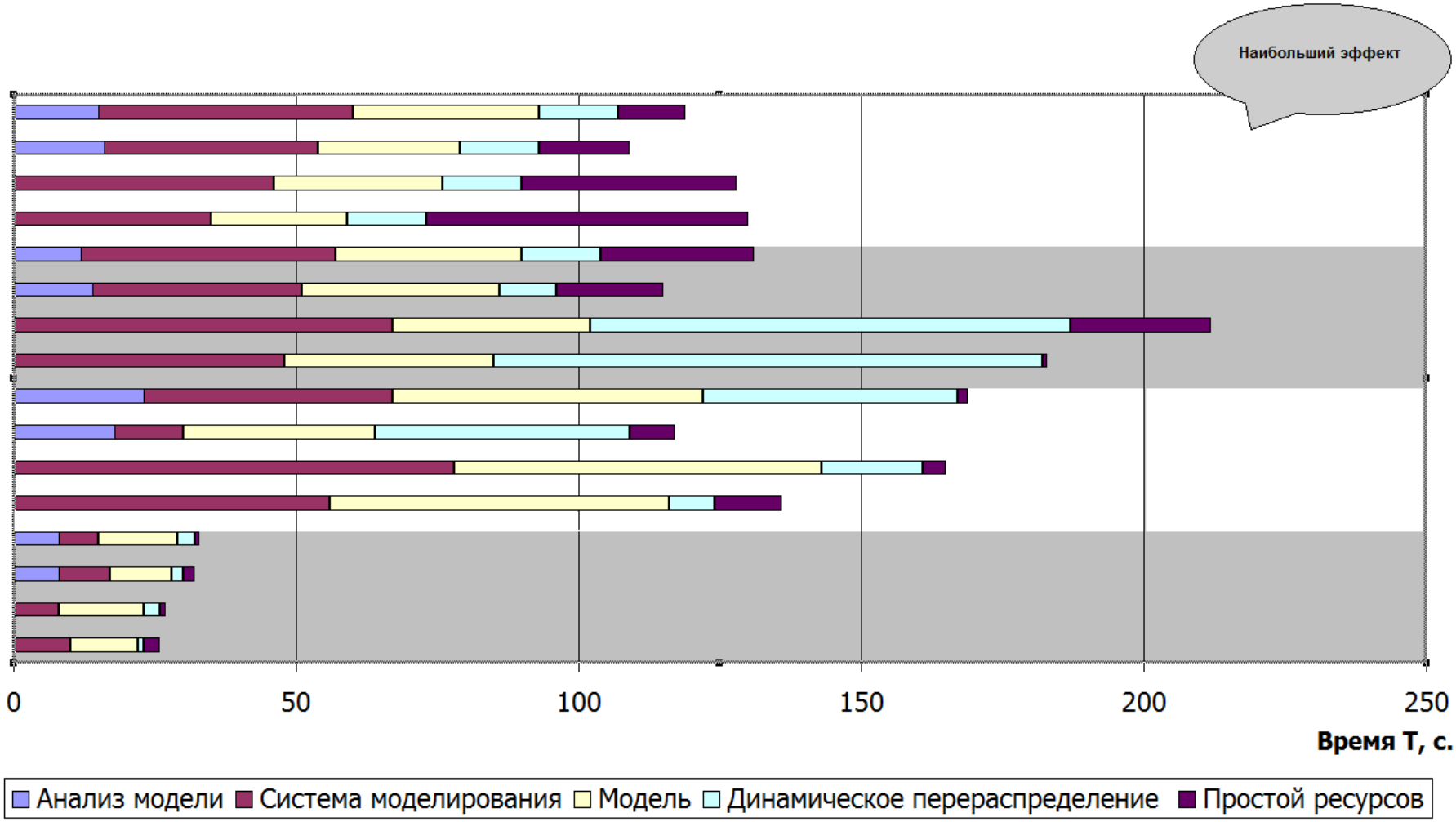


Рисунок 4.12 – Результаты применения разработанной технологии анализа при моделировании различных потоков заданий распределенного имитационного моделирования в GRID-системах

Таблица 4.1 – Сравнительная таблица применения методов распределения программных моделей по доступным ресурсам

Размерность эксперимента				Метод случайного распределения			Метод равномерного распределения (на основе вычислительной сложности программных моделей)			Метод обратного заполнения (backfill) (на основе оценки времени выполнения программных моделей)			С применением разработанных методов		
N	M	T_M, мин	Синхр	TR, с	TP, с	%Vk	TR, с	TP, с	%Vk	TR, с	TP, с	%Vk	TR, с	TP, с	%Vk
10	40	100	К	28	14	0	42	12	0	44	8	0	48	14	0
40	40	100	К	96	52	3	86	48	0	72	36	12	70	34	0
40	40	100	О	120	35	12	87	38	6	78	15	18	74	10	3
200	40	100	К	562	241	3	480	260	7	470	215	10	420	209	3
200	40	100	О	460	202	14	470	253	10	462	207	13	402	198	7
200	488	500	К	390	160	4	320	140	0	301	138	0	280	125	0
200	488	500	О	390	160	4	320	140	0	301	138	0	280	125	0
600	488	500	К	1538	456	14	1340	401	12	1280	390	10	989	123	0
600	488	500	О	1567	315	45	1215	201	38	1275	170	41	779	43	8

4.4 Выводы по разделу

Предложена оригинальная архитектура распределенной имитационной системы моделирования GRID-систем, основанная на подключаемых модулях. Архитектура обладает рядом преимуществ: гибкость настройки, простота модификации, упрощение разработки, тестирования, поддержки и обновления системы.

В работе предложена архитектура визуального интерфейса для распределенной имитационной системы моделирования GRID-инфраструктуры – GRASS. Описанное решение позволило выделить визуализацию данных в отдельную подсистему, уменьшить связанность модулей в системе и увеличить их внутреннюю связность. Одним из основных достоинств архитектуры является возможность подключения новых модулей со своими оригинальными функциями визуализации без перекомпиляции всего проекта. Гибкость разработанной архитектуры позволяет расширять или модифицировать модули визуализации с целью проведения более сложных экспериментов.

Проведено ряд экспериментов, позволяющих проверить работоспособность системы моделирования GRASS под управлением разных алгоритмов синхронизации. Система анализа имитационной модели подтвердила работоспособность методов анализа. Сравнение количественных характеристик, полученных экспериментальным и модельным путем показала, что экспериментальная погрешность методов не превышает 4% .

Проведено ряд експериментів, які дозволили перевірити працездатність та ефективність запропонованої технології аналізу програмних моделей у системі моделювання GRASS під управлінням різних алгоритмів синхронізації. Аналіз результатів експериментів показав, що впровадження інформаційної технології аналізу розподілених програмних моделей дозволив зменшити час моделювання та кількість обчислювальних ресурсів у середньому на 36% та 28% відповідно. Одночасно з цим зменшилися час простою ресурсів у середньому на 19% та кількість конфліктів пам’яті на 54%.

Перспективы развития системы состоят в создании модулей, реализующих модели элементов GRID-систем, описанных в [9].

Полученные результаты могут быть полезными разработчикам программного обеспечения GRID-систем, специалистам в области разработки распределенного программного обеспечения и создания имитационных моделей.

ВЫВОДЫ

Наличие новых информационно-вычислительных инфраструктур уровня GRID приводит к пересмотру теории имитационного моделирования с точки зрения эффективного использования распределенных гетерогенных архитектур при проведении имитации. Информационные GRID-технологии и технологии облачных вычислений становиться инструментом проведении имитационных экспериментов. Применение распределенного имитационного моделирования позволяет значительно расширить класс реализуемых имитационных моделей за счет использования большого количества мощных удаленных вычислительных ресурсов.

Анализ литературы в области имитационного моделирования показал отставание в развитии теории имитационного моделирования от стремительного развития распределенных вычислительных систем, обусловленное появлением мощных вычислительных ресурсов и быстродействующих коммуникационных сетей.

В результате диссертационной работы получили развитие технологии имитационного моделирования в распределенных вычислительных системах, разработаны модели, методы и информационн технология анализа распределенных программных моделей, что позволило уменьшить время моделирования и необходимое для моделирования количество вычислительных ресурсов.

В процессе выполнения работы для решения единой научной проблемы были получены следующие научные результаты:

1. Разработаны модели предметной области на основании процессного представления частных моделей как программных элементов вычислительной среды, которые в отличии от существующих, учитывают динамическое изменение объемов памяти, время простоя, что позволяет удовлетворить установленные ограничения на объем оперативной памяти и минимизировать время простоя вычислительных ресурсов для разных условий синхронизации программных моделей.

2. Разработаны модификации моделей для случаев синхронных, консер-

вативных, оптимистических методов синхронизации распределенных имитационных моделей, которые учитывают методы синхронизации модельного времени и архитектуру локальных моделей стандарта HLA, что позволяет оценить время выполнения программных моделей для разных схем распределения ресурсов с учетом ограничений на объем памяти и сумарное время простоя вычислительных ресурсов.

3. Разработан метод оценки возможности распределея частных моделей на основе динамического изменения объемов виртуальной памяти, что позволяет учитывать изменение объемов памяти, особенно в случае оптимистических алгоритмов синхронизации программных моделей, уменьшить количество вычислительных ресурсов, учесть ограничения на объем памяти и уменьшить время моделирования.

4. Разработан метод получения оценок простоя ресурсов для консервативных алгоритмов синхронизации распределенных имитационных моделей, что позволяет оценить время выполнения программных моделей для разных схем распределения ресурсов с учетом времени простоя вычислительных ресурсов.

5. Улучшен метод сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени, что в отличии от существующего, учитывает возможность изменения объема памяти программных моделей и объемов данных, которые передаются между программными моделями, а также время простоя ресурсов, что позволяет оценить время выполнения программных моделей для разных схем распределения ресурсов с учетом ограничений на объем пасяти и время простоя вычислительных ресурсов.

6. Получили дальнейшее развитие методы распределения програмных моделей по вычислительным ресурсам в распределенном имитационном моделировании GRID-систем, которые в отличии от существующих, позволяют осуществлять распределение ресурсов по максимальному продвижению модельного времени при ограничениях на объем оперативной памяти, время простоя и выбрать способ синхронизации программных моделей, что обеспечивает

повышение эффективности процеса распределенного имитационного моделирования путем уменьшения времени моделирования и количество вычислительных ресурсов.

В качестве практической значимости полученных теоретических результатов диссертационной работы можно отметить улучшение качества анализа распределенных имитационных моделей, работающих под управлением разных методов синхронизации. В частности, практические решения теоретических исследований заключаются в следующем: увеличено количество параметров, получаемых в процессе анализа распределенных имитационных моделей, что дает возможность улучшения методов распределения ресурсов, алгоритмов управления заданиями на моделирования в распределенных информационных системах; разработаны и реализованы алгоритмы и процедуры анализа распределенных имитационных моделей; разработана процедура сравнения эффективности распределенных имитационных моделей по максимальному продвижению модельного времени; разработана процедура оценки возможности распределения частных моделей на основе динамического изменения объемов виртуальной памяти; разработана процедура получения оценок простоя ресурсов для алгоритмов синхронизации распределенных имитационных моделей.

Полученные теоретические результаты реализованы при разработке программного обеспечения имитационного моделирования информационных систем.

Достоверность полученных практических результатов диссертационной работы подтверждена экспериментальными исследованием созданного программного обеспечения, результатами моделирования реальных информационных систем (акты внедрения прилагаются в приложении к диссертации).

Направления теоретических и практических исследований диссертационной работы целесообразно развивать в области создания и использования распределенных имитационных систем моделирования.

ПЕРЕЧЕНЬ ССЫЛОК

1. Шеннон Р. Имитационное моделирование систем – искусство и наука: пер. с англ. / Р. Шеннон. – М. : Мир, – 1978. – 417 с.

2. Максимей И. В. Имитационное моделирование на ЭВМ / И. В. Максимей. – М. : Радио и связь, 1988. – 222 с.

3. Моисеев Н. Н. Математические задачи системного анализа / Н. Н. Моисеев. – М. : Наука, 1981. – 488 с.

4. Бусленко Н. П. Моделирование сложных систем / Н. П. Бусленко. – М. : Наука, 1978. – 400 с.

5. СЛЭНГ – система программирования для моделирования дискретных систем / В. М. Глушков, Л. А. Колиничеико, Т. П. Марьяиович, В. М. Москаленко, М. А. Сахнюк. – К. : изд-во ин-та кибернетики АН УССР, 1969. – 413 с.

6. Программные средства моделирования непрерывно-дискретных систем / В. М. Глушков, В. В. Гусев, Т. П. Марьяиович, М. А. Сахнюк. – К. : Наук. думка. – 152 с.

7. Прицкер А. Введение в имитационное моделирование и язык СЛАМ II: пер. с англ. / А. Прицкер – М. : Мир, 1987. – 644 с.

8. Gordon G. System Simulation / Geoffrey Gordon. – 2nd ed. – Upper Saddle River, NJ : Prentice-Hall, 1978. – 324 p.

9. Окольнишников В. В. Разработка средств распределенного имитационного моделирования для многопроцессорных вычислительных систем : дис. … д-ра техн. наук : 05.13.18 «Математическое моделирование, численные методы и комплексы программ» / В. В. Окольнишников. – Новосибирск, 2006 – 227 с.: 71 07-5/433.

10. Окольнишников В. В. Представление времени в имитационном моделировании / В. В. Окольнишников // Вычислительные технологии. Сибирское отделение РАН. – 2005. №5. – С. 57–80.

11. Вознесенская Т. В. Математическая модель алгоритмов синхронизации времени для распределенного имитационного моделирования / Т. В. Вознесенская // «Программные системы и инструменты»: Тематический сборник фа-

культета ВМиК МГУ им. Ломоносова / под ред. Л. Н. Королева – М. : МАКС Пресс, – 2000. №1. – С. 56–66.

12. Вознесенская Т. В. Оценка эффективности оптимистических алгоритмов синхронизации времени для распределенного имитационного моделирования / Т. В. Вознесенская // Интеллектуальные многопроцессорные системы : тезисы докладов Международной конференции, 1–5 сентября 1999 г. – Таганрог, Россия, 1999. – С. 66.

13. Вознесенская Т. В. Математическая модель для анализа производительности распределенных систем имитационного моделирования / Т. В. Вознесенская // Искусственный интеллект (Донецк), 2002. – № 2. – С. 74-78.

14. Вознесенская Т. В. Математическая модель алгоритмов синхронизации времени для распределенного имитационного моделирования / Т. В. Вознесенская // Повышение эффективности методов и средств обработки информации: материалы V Всероссийской научно-технической конференции, 12–15 мая 1997 г. – Тамбов, Россия, 1997. – С. 193.

15. Волк М. А. Методы и средства распределенного имитационного моделирования электронных систем : дис. … канд. техн. наук : 01.05.02 «Математическое моделирование и вычислительные методы» / М. А. Волк ; Харьк. техн. ун-т радиоэлектроники. – Харьков, 1999. – 189 с. : ил. – Библиогр.: с. 154–163.

16. Волк М. А. Применение процессной алгебры для формального описания систем имитационного моделирования / М. А. Волк, С. А. Олищук, Р. Н. Гридель // Радиоэлектроника и молодеж в XXI веке : материалы 13-го междунар. молодеж. форума, 30 марта – 1 апр. 2009 г. – Харков: ХНУРЭ, 2009. – Ч.2.– С. 202.

17. Волк М. А. Процессное представление состояний распределенных имитационных моделей с учетом специфики их программной реализации / М. А. Волк // Сборник научных трудов «Вестник НТУ «ХПИ» : Информатика и моделирование. – Х., 2009. – №13. – С. 23–33.

18. Хоар Ч. Взаимодействующие последовательные процессы: пер. с англ. / Ч. Хоар. – М. : Мир, 1989. – 264 с., ил.

19. Hoare C. A. R. Communicating sequential processes / C. A. R. Hoare //

Communications of the ACM. – 1978. – Vol. 21, № 8. – P. 666–677.

20. Hoare C. A. R. A model for communicating sequential processes / C. A. R. Hoare // On the Construction of Programs / ed. R. M. McKeag, A. M. Macnaghten. – Cambridge University Press, 1980. – P. 229–254.

21. Milner R. A Calculus of Communicating Systems / R. Milner. – Berlin ; Heidelberg : Springer-Verlag, 1980. – 174 p. – (Lecture Notes in Computer Science ; vol. 92).

22. Алгоритмічна модель процесу розподіленої імітації для технології аналізу розподілених імітаційних моделей / М. О. Волк, Р. М. Грідель, С. Н. Саранча, Д. А. Гавриш // Східно-Європейський журнал передових технологій. – Х., 2013. – № 2 (66). – С. 32–36.

23. Волк М. А. Методы распределения ресурсов для GRID-систем / М. А. Волк, Т. В. Филимончук, Р. Н. Гридель // Зб. наук. пр. Харк. ун-ту Повітр. Сил. – Х., 2009. – Вип. 1 (19). – С. 100–104.

24. Волк М. А. Архитектура имитационной модели GRID-системы, основанная на подключаемых модулях / М. А. Волк, Р. Н. Гридель, А. С. Горенков // Системи обробки інформації. – Х., 2010. – Вип. 1 (82). – С. 17–20.

25. Волк М. А. Архитектура подсистемы визуализации данных распределенной имитационной системы моделирования GRID / М. А. Волк, Р. Н. Гридель, А. С. Олендаренко // АСУ и приборы автоматики. – Х., 2010. – Вып. 150. – С. 89–92.

26. Анализ распределенных имитационных моделей с консервативными алгоритмами синхронизации / М. А. Волк, Р. Н. Гридель, М. А. Филимончук, Муаз Ал Шиблак // Зб. наук. пр. Харк. ун-ту Повітр. Сил. – Х., 2012. – Вип. 1 (30). – С. 95–98.

27. Волк М. А. Анализ распределенных имитационных моделей с оптимистическими алгоритмами синхронизации / М. А. Волк, Р. Н. Гридель, Муаз Ал Шиблак // Системи обробки інформації. – Х., 2013. – Вип. 1 (108). – С. 35–40.

28. Формализация процессов распределенной имитации информационных систем / М. А. Волк, Р. Н. Гридель, Т. В. Филимончук, Муаз Ал Шиблак // Си-

стеми обробки інформації. – Х., 2013. – Вип. 4 (111). – С. 89–93.

29. Волк М. А. Технологии синхронизации распределенных компьютерных моделей / М. А. Волк, Р. Н. Гридель, Муаз Ал Шиблак // Информатика, математическое моделирование, экономика : сб. науч. ст. по итогам Третьей Междунар. науч.-практ. конф., 24–26 апр. 2013 г. – Смоленск, 2013. – Т. 1. – С. 54–59.

30. Волк М. А. Элементы распределенной имитационной модели GRID-систем / М. А. Волк, Р. Н. Гридель, К. Ю. Дъяченко // Системный анализ и информационные технологии : материалы XI Междунар. науч.-техн. конф., 26–30 мая 2009 г. – К. : НТУУ «КПИ», 2009. – С. 422.

31. Гридель Р. Н. Расширение методов анализа алгоритмов синхронизации времени для распределенного имитационного моделирования в GRID-системах / Р. Н. Гридель // Радиоэлектроника и молодежь в XXI веке : материалы 14-го междунар. молодеж. форума, 18–20 марта 2010 г. – Х. : ХНУРЭ, 2010. – Ч. 2. – С. 212.

32. Волк М. А. Архитектура имитационной модели GRID-системы, основанная на подключаемых модулях / М. А. Волк, А. С. Горенков, Р. Н. Гридель // Проблемы информатики и моделлирования : материалы 9-й междунар. науч.-техн. конф., 26–28 ноября. 2009 г. – Х. : НТУ «ХПИ», 2009. – С. 42.

33. Волк М. А. Имитационная система моделирования GRID-инфраструктуры GRASS / М. А. Волк, Р. Н. Гридель, Т. В. Филимончук // Системный анализ и информационные технологии : материалы XII Междунар. науч.-техн. конф., 25–29 мая 2010 г. – К. : НТУУ «КПИ», 2010. – С. 359.

34. Волк М. А. Метод анализа распределенных имитационных моделей / М. А. Волк, Р. Н. Гридель, Муаз Ал Шиблак // Інформаційні технології в навігації і управлінні: стан та перспективи розвитку : матеріали 5–6 лип. 2010 р. – К. : ДП «ЦНДІ НіУ», 2010. – С. 46.

35. Гридель Р. Н. Применение процессной алгебры для формального описания систем имитационного моделирования / Р. Н. Гридель, С. А. Олищук // Радиоэлектроника и молодежь в XXI веке : материалы 13-го междунар. молодеж. форума, 30 марта – 1 апр. 2009 г. – Х. : ХНУРЭ, 2009. – Ч. 2. – С. 202.

36. Волк М. А. Анализ моделей потоков заданий для моделирования рабочей нагрузки в GRID-системах / М. А. Волк, Р. Н. Гридель, В. В. Зозуля // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матеріали першої наук.-техн. конф., 13–14 груд. 2010 р. – Х. : ДП «ХНДІ ТМ» ; К. : ДП «ЦНДІ НіУ», 2010. – С. 75.

37. Волк М. А. Методы анализа распределенного имитационного моделирования с учетом сетевого трафика / М. А. Волк, Р. Н. Гридель, Муаз Ал Шиблак // Інформаційні технології в навігації і управлінні: стан та перспективи розвитку : матеріали другої міжнар. наук.-техн. конф., 16–17 лип. 2011 р. – К. : ДП «ЦНДІ НіУ», 2011. – С. 45.

38. Волк М. А. Расширенный анализ распределенных имитационных моделей с оптимистическими и консервативными алгоритмами синхронизации / М. А. Волк, Р. Н. Гридель // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матеріали другої наук.-техн. конф., 15–16 груд. 2011 р. – К. : ДП «ЦНДІ НіУ» : КДАВТ ; Х. : ДП «ХНДІ ТМ», 2011. – С. 36.

39. Волк М. А. Менеджер памяти распределенных имитационных моделей / М. А. Волк, Р. Н. Гридель, Муаз Ал Шиблак // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матеріали третьої міжнар. наук.-техн. конф., 11–12 квіт. 2013 р. – Полтава : ПНТУ ; Білгород : НДУ «БілДУ» ; Х. : ДП «ХНДІ ТМ» ; К. : НТУ ; Кіровоград : КЛА НАУ, 2013. – С. 34.

40. Волк М. А. Технология синхронизации в распределенном имитационном моделировании / М. А. Волк, Р. Н. Гридель, Муаз Ал Шиблак // Внедрение перспективных микропроцессорных систем железнодорожной автоматики и средств телекоммуникаций на базе цифровизации : материалы 26-й междунар. науч.-практ. конф., 23–28 сент 2913 г., Алушта, Украина. – Алушта, 2013. – С. 63.

41. Волк М. А. Информационная технология анализа распределенных имитационных моделей / М. А. Волк, Р. Н. Гридель, Муаз Ал Шиблак // Проблеми інформатизації : матеріали першої міжнар. наук.-техн. конф., 19–20 груд.

2013 р. – Черкаси : ЧДТУ ; Київ : ДУТ ; Тольятті : ТДУ ; Полтава : ПНТУ, 2013. – С. 29.

42. Волк М. А. Информационные технологии обеспечения распределенного имитационного моделирования / М. А. Волк, Р. Н. Гридель // Проблеми інформатизації : матеріали другої міжнар. наук.-техн. конф., 12–13 квіт. 2014 р. – К. : ДУТ ; Полтава : ПНТУ ; Катовице ; Париж ; Білгород : НДУ «БДУ» ; Черкаси : ЧДТУ ; Харків : ХНДІТМ, 2014. – С. 56.

43. Автоматизация проектирования вычислительных систем. Языки, моделирование и базы данных / ред. Брейер М. : пер. с англ. Е. Е. Махова, В. Г. Меркулов, О. Ф. Мясин. – М. : Мир, 1979. – 463 с. : ил

44. Мур Дж. Экономическое моделирование в Microsoft Excel, 6-е изд.: пер. с англ. / Дж. Мур, Л. Уэдерфорд и др. – М. : Издательский дом «Вильямс», 2004. – 1024 с.

45. Analog Computer Simulation Using Spreadsheets / K. Y. Kabalan, A. El-Hajj, H. Diab, S. Fakhreddine // Simulation. – 1997. – Vol. 68, № 2. – P. 101–106.

46. Ажогин В. В. Моделирование на цифровых, аналоговых и гибридных вычислительных ЭВМ / В. В. Ажогин, М. З. Згуровский. – К. : Вища шк. Головное изд-во, 1983. – 280 с.

47. Земляк Є. М. Автоматизоване моделювання неперервних процесів і систем : автореф. дис. д-ра техн. наук : 05.13.07 «Автоматизація технологічних процесів і виробництв», 05.13.16 «Застосування обчислювальної техніки та математичних методів у наукових дослідженнях» / Є. М. Земляк ; Київський політехнічний інститут. – К. : КПІ, 1993. – 32 с.

48. Технология системного моделирования / Е. Ф. Абрамчук, А. А. Вавилов, С. В. Емельянов и др. под общ. ред. С. В. Емельянова. – М. : Машиностроение ; Берлин : Техник, 1988. – 520 с. : ил.

49. Брейер М. Автоматизация проектирования вычислительных систем. Языки, моделирование и базы данных / М. Брайер. – М. : Мир, 1979. –463 с.

50. Roumeliotis M. HDL Modeling for Process Oriented Simulation / M. Roumeliotis, J. Armstrong // Computer Hardware Description Languages and their Application : proceedings of the IFIP WG 10.2 8th International Conference,

April 27–29, 1987, Amsterdam, the Netherlands. – North-Holland, 1987. – P. 289–297.

51. Шрайбер Т. Дж. Моделирование на GPSS / Т. Дж. Шрайбер. – М. : Маширостроение, 1980. – 592 с.

52. Томашевский В. Имитационное моделирование в среде GPSS / В. Томашевский, Е. Жданова. – М. : Бестселлер, 2003 – 416 с.

53. GPSS – прошло 40 лет: выбранные перспективы [Электронный ресурс] / Т. Дж. Шрайбер, С. Кокс, Дж. О. Хенриксен и др. // Труды конференции Winter Simulation Conference – 2001 (WSC-2001), 9–12 дек. 2001 г. – Режим доступа : www. URL: http://www.gpss.ru/paper/wsc2001/print.html. – Загл. с экрана.

54. Якимов И. М. Развитие методов и систем имитации в СССР и России [Электронный ресурс] / И. М. Якимов, В. В. Девятков. – Режим доступа : www. URL: http://www.gpss.ru/paper/dev_yak/index_w.html. – Загл. с экрана.

55. Лычкина Н. Н. Опыт применения GPSS в Государственном Университете Управления [Электронный ресурс] / Н. Н. Лычкина. – Режим доступа : www. URL: http://www.gpss.ru/paper/lychkina/print_w.html. – Загл. с экрана.

56. Object-Oriented Analysis and Design with Applications / Grady Booch, Robert A. Maksimchuk, Michael W. Engel et al. – 3rd ed. – Addison-Wesley Professional, 2007. – 720 p.

57. Дал У. Симула-67 / У. Дал, Б. Мюрхауг, К. Нюгорд. – М. : Мир, 1969. – 100 с.

58. Андрианов А. Н. Программирование на языке Симула-67 / А. Н. Андрианов, С. П. Бычков, А. И. Хорошилов. – М. : Наука, 1985. – 370 с.

59. Pegden C. D. Future Directions in Simulation Modeling / C. D. Pegden // Proceedings of the 37th conference on Winter simulation (WSC'05), December 4–7, 2005, Orlando, FL, USA. – IEEE Computer Society, 2005.

60. Bonabeau E. Agent-based modeling: methods and techniques for simulating human systems / E. Bonabeau // Proceedings of the National Academy of Sciences of the USA. – 2002. – Vol. 99, suppl. 3. – P. 7280–7287.

61. O'sullivan D. Agent-based models and individualism: Is the world agent-based? / D. O'sullivan , M. Haklay // Environment and Planning A. – 2000. – Vol.

32, № 8. – P. 1409–1425.

62. Shoham Y. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations / Yoav Shoham, Kevin Leyton-Brown. – New York : Cambridge University Press, 2009. – 521 p.

63. Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation / ed. Ron Sun. – New York : Cambridge University Press, 2006. – 442 p.

64. Бусленко Н. П. Лекции по теории сложных систем / Н. П. Бусленко, В. В. Еалашников, И. Н. Коваленко. – М. : «Советское радио», 1973. – 440 с.

65. Рыков В. В. Два подхода к декомпозиции сложных иерархических систем. Агрегативные системы / В. В. Рыков // Автоматика и телемеханика. – М., 1997. – №12. – С. 140–149.

66. Petri C. A. Introduction to general net theory / C. A. Petri // Proc. Advanced Course on General Net Theory, Processes and Systems / ed. W. Brauer. – London : Springer-Verlag, 1980. – P. 1–20. – (Lecture Notes in Computer Science ; vol. 84).

67. Петренко А.І. GRID-технології в науці і освіті // Системний аналіз та інформаційні технології: матеріали 9-ої Міжнародної конференції, Київ, 2007. – С. 138–140.

68. Волк М. А. Структурная организация поведенческого имитационного моделирования в grid / М. А. Волк // Системи обробки інформації. – Х., 2007. – Вип. 9 (67). – С.41–45.

69. Казаков Ю. П. Об организации распределенного имитационного моделирования / Ю. П. Казаков, Р. Л. Смелянский // Программирование. – 1994. – №2. – С. 45–63.

70. Ferscha A. Parallel and distributed simulation of discrete event systems / A. Ferscha // Parallel and Distributed Computing Handbook / ed. Albert Y. Zomaya. – McGraw-Hill, 1996. – P. 1003–1041.

71. Cermele M. Adaptive Load Balancing of Distributed SPMD Computations: A Transparent Approach : Tech. Rep. DISP-RR-97.02, 1997 / M. Cermele, M. Colajanni, S. Tucci // Future Generation Computer Systems. – 2000. – Vol. 16. – P. 571–584.

72. Fujimoto R. M. Parallel and Distributed Simulation Systems / R. M. Fujimoto. – New York : Wiley-Interscience, 2000. – 324 p.

73. Fujimoto R. M. Parallel and distributed simulation systems / R. M. Fujimoto // Proceedings of the 33rd conference on Winter simulation (WSC'01), December 9–12, 2001, Arlington, VA, USA. – Washington, DC : IEEE Computer Society, 2001. – P. 147–157.

74. Mitra D. Analysis and optimum performance of two message-passing parallel processors synchronized by rollback / D. Mitra, I. Mitrani // Performance’84. – Amsterdam, Netherlands : North-Holland, Elsevier Science Publishers, 1984. – P. 35–50.

75. Bryant R. E. Simulation of Packet Communications Architecture Computer Systems : Technical Report MIT-LCS-TR-188 / R. E. Bryant. – Cambridge, MA, 1977.

76. Chandy K. M. Distributed simulation: a case study in design and verification of distributed programs / K. M. Chandy, J. Misra // IEEE Transactions on Software Engineering. – 1979. – Vol. SE–5, № 5. – P. 440–452.

77. Chandy K. M. Asynchronous Distributed Simulation via a Sequence of Parallel Computations / K. M. Chandy, J. Misra // Communications of the ACM. – 1981. – Vol. 24, № 4. – P. 198–205.

78. Jha V. Simultaneous events and lookahead in simulation protocols / V. Jha, R. Bagrodia // ACM Transactions on Modeling and Computer Simulation. – 2000. – Vol. 10, № 3. – P. 241–267.

79. Misra J. Distributed discrete-event simulation / J. Misra // ACM Computing Surveys. – 1986. – Vol. 18, № 1. – P. 39–65.

80. Sokol L. M. MTW: experimental results for a constrained optimistic scheduling paradigm / L. M. Sokol, B. K. Stucky // Proceedings of the SCS Multiconference on Distributed Simulation. – 1990. – Vol. 22, № 1. – P. 169–173.

81. Rao D. M. Unsynchronized parallel discrete event simulation / D. M. Rao, N. V. Thondugulam, P. A. Wilsey // Proceedings of the 30th conference on Winter simulation (WSC'98), December 13–16, 1998, Washington, DC, USA. – Los Alamitos, CA : IEEE Computer Society, 1998. – P. 1563–1570.

82. Rao D. M. An ultra-large-scale simulation framework / D. M. Rao, P. A. Wilsey // Journal of Parallel and Distributed Computing. – 2002. – Vol. 62, № 11. – P. 1670–1693.

83. Das S. R. Adaptive protocols for parallel discrete event simulation / Samir Roo Das // Journal of the Operational Research Society. – 2000. – Vol. 51. – P. 385–394.

84. Altuntas B. A framework for adaptive synchronization of distributed simulations / B. Altuntas, R. A. Wysk // Proceedings of the 2004 Winter simulation conference, December 5–8, 2004, Washington, DC, USA. – Washington, DC : IEEE Computer Society, 2004. – P. 371–377.

85. Meyer R. A. Path lookahead: a data flow view of pdes models / R. A. Meyer, R. L. Bagrodia // Proceedings of the 13th ACM Workshop on Parallel and Distributed Simulation (PADS’99), May 1–4, 1999, Atlanta, GA, USA. – Washington, DC : IEEE Computer Society, 1999. – P. 12–19.

86. Xiao Z. Scheduling critical channels in conservative parallel simulation / Z. Xiao, B. Unger // Proceedings of the 13th ACM Workshop on Parallel and Distributed Simulation (PADS’99), May 1–4, 1999, Atlanta, GA, USA. – Washington, DC : IEEE Computer Society, 1999. – P. 20–28.

87. Fujimoto R. M. Exploiting temporal uncertainty in parallel and distributed simulations / R. M. Fujimoto // Proceedings of the 13th ACM Workshop on Parallel and Distributed Simulation (PADS’99), May 1–4, 1999, Atlanta, GA, USA. – Washington, DC : IEEE Computer Society, 1999. – P. 46–53.

88. Beraldi R. Exploiting temporal uncertainty in time warp simulations / R. Beraldi, L. Nigro // Proceedings of the 4th Workshop on Distributed Simulation and Real-Time Applications (DS-RT'00). – Washington, DC : IEEE Computer Society, 2000. – P. 39–46.

89. Loper M. Exploiting temporal uncertainty in process-oriented distributed simulations / M. Loper, R. M. Fujimoto // Proceedings of the 2004 Winter simulation conference, December 5–8, 2004, Washington, DC, USA. – Washington, DC : IEEE Computer Society, 2004. – P. 395–400.

90. Ayani R. A parallel simulation scheme based on the distance between ob-

jects / R. Ayani // Proceedings of the SCS Multiconference on Distributed Simulation, March 28–31, 1989, Tampa, Florida. – Society for Computer Simulatio, 1989. – Vol. 22. – P. 113–118. – (SCS Simulation Series).

91. Lubachevsky B. D. Efficient distributed event-driven simulations of multiple-loop networks / B. D. Lubachevsky // Communications of the ACM. – 1989. – Vol. 32, № 1. – P. 111–123.

92. Cai W. An algorithm for distributed discrete-event simulation — "carrier null message" approach / W. Cai, S. J. Turner // Proceedings of the SCS Multiconference on Distributed Simulation, January 17–19, 1990, San Diego, California. – Society for Computer Simulatio, 1990. – P. 3–8. – (SCS Simulation Series).

93. Lamport L. Concurrent reading and writing of clocks / L. Lamport // ACM Transactions on Computer Systems. – 1990. – Vol 8, № 4. – P. 305–310.

94. Raynal M. Logical time: capturing causality in distributed systems / M. Raynal, M. Slnghal // IEEE Computer. – 1996. – Vol. 29, № 2. – P. 49–56.

95. Baldony R. Fundamentals of distributed computing: a practical tour of vector clock systems [Electronic resource] / R. Baldony, M. Raynal // IEEE Distributed Systems Online. – 2002. – Mode of access : www. URL: http://csdl.computer.org/comp/mags/ds/2002/02/o2001.pdf. – Title from the screen.

96. Jefferson D. Virtual time / D. Jefferson // ACM Transactions on Programming Languages and Systems. – 1985. – Vol. 7, № 3. – P. 404–425.

97. Mattern F. Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation / F. Mattern // Journal of Parallel and Distributed Computing. – 1993. – Vol. 18, № 4. – P. 423–434.

98. Mattern F. Algorithms for distributed termination detection / F. Mattern // Distributed Computing. – 1987.– Vol. 2. – P. 161–175.

99. Preiss B. R. Memory management techniques for time warp on a distributed memory machine / B. R. Preiss, W. M. Loucks // Proceedings of the 9th workshop on Parallel and distributed simulation, June 13–16, 1995, Lake Placid, NY, USA. – Washington, DC : IEEE Computer Society, 1995. – P. 30–39.

100. Lin Y. B. Selecting the checkpoint interval in time warp simulations / Y. B. Lin, B. R. Preiss // Proceedings of the 7th Workshop on Parallel and Distribut-

ed Simulation, May 16–19, 1993, San Diego, CA, USA. – New York, NY, 1993. – P. 3–10.

101. Palaniswamy A. C. An analytical comparison of periodic checkpointing and incremental state saving / A. C. Palaniswamy, P. A. Wilsey // Proceedings of the 7th Workshop on Parallel and Distributed Simulation, May 16–19, 1993, San Diego, CA, USA. – New York, NY, 1993. – P. 127–134.

102. Jefferson D. R. Virtual time II: storage management in distributed simulation / D. R. Jefferson // Proceedings of the 9th annual ACM symposium on Principles of distributed computing (PODC'90), August 22–24, 1990, Quebec City, QC, Canada. – New York, NY, 1990. – P. 75–89.

103. Lin Y. B. Optimal memory management for time warp parallel simulation / Y. B. Lin, B. R. Preiss // ACM Transactions on Modeling and Computer Simulation. – 1991. – Vol. 1, № 4. – P. 283–307.

104. Chen G. Lookback: a new way of exploiting parallelism in discrete event simulation / G. Chen, B. K. Szymanski // Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS'02), May 12–15, 2002, Arlington, VA, USA. – Washington, DC : IEEE Computer Society, 2002. – P. 153–162.

105. Chen G. Four types of lookback / G. Chen, B. K. Szymanski // Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS'03), June 10–13, 2003, San Diego, CA, USA. – Washington, DC : IEEE Computer Society, 2003. – P. 3–10.

106. Fujimoto R. M. Time warp on a shared memory multiprocessor / R. M. Fujimoto // Transactions of the Society for Computer Simulation. – 1989. – Vol. 6, № 3. – P. 211–239.

107. Zhang J. L. The dependence list in time warp / J. L. Zhang, C. Tropper // Proceedings of the 15th Workshop on Parallel and Distributed Simulation (PADS’2001), May 15–18, 2001, Lake Arrowhead, CA, USA. – Washington, DC : IEEE Computer Society, 2001. – P. 35–45.

108. Hussam M. Soliman Ramadan. Throttled Lazy Cancellation in Time Warp Parallel Simulation / Hussam M. Soliman Ramadan // SIMULATION. – 2008. – Vol. 84, № 2–3. – P. 149–160.

109. Rajan R. Dynamically switching between lazy and aggressive cancellation in a Time Warp parallel simulator / R. Rajan, P. A. Wilsey // Proceedings of the 28th Annual Simulation Symposium. – Washington, DC : IEEE Computer Society, 1995. – P. 22.

110. Wu Yue. Cancellation strategy in rollback mechanism / Wu Yue, Li Jun-hong, Yang Hong-bin // Journal of Shanghai University. – 2005. – Vol. 9, № 6. – P. 501–505.

111. Das S. R. Adaptive memory management and optimism control in time warp / S. R. Das, R. M. Fujimoto // ACM Transactions on Modeling and Computer Simulation. – 1997. – Vol. 7, № 2. – P. 239–271.

112. Carothers C. D. Efficient optimistic parallel simulation using reverse computation / C. D. Carothers, K. S. Perumalla, R. M. Fujimoto // ACM Transactions on Modeling and Computer Simulation. – 2000. – Vol. 9, № 3. – P. 224–253.

113. IEEE Std 1278.1–1995. IEEE Standard for Distributed Interactive Simulation – Application Protocols. – New York, NY : Institute of Electrical and Electronics Engineers, Inc., 1996.

114. Mille D. C. SIMNET: the advent of simulator networking / D. C. Mille, J. A. Thorpe // Proceedings of the IEEE. – 1995. – Vol. 83, № 8. – P. 1114–1123.

115. Wilson A. L. The aggregate level simulation protocol: an evolving system / A. L. Wilson., R. M. Weatherly // Proceedings of the Winter Simulation Conference, December 1994, Lake Buena Vista, FL. – IEEE Computer Society, 1994. – P. 781–787.

116. Wei Wang. Research on framework of reliability interactive simulation analysis based on HLA / Wei Wang, Huilin Liu // Proceedings of the 8th International Conference on Reliability, Maintainability and Safety (ICRMS 2009), July 20–24, 2009, Chengdu, China. – IEEE Computer Society, 2009. – P.201–204.

117. IEEE Std 1516.2–2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT) Specification. – New York, NY : Institute of Electrical and Electronics Engineers, Inc., 2001. – 130 p.

118. IEEE Draft Standard for Modeling and Simulation (M&S) High Level

Architecture (HLA) – Federate Interface Specification. – New York, NY : Institute of Electrical and Electronics Engineers, Inc., 2009.

119. IEEE Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules. – New York, NY : Institute of Electrical and Electronics Engineers, Inc., 2009.

120. IEEE Std P1516. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules. – New York, NY : Institute of Electrical and Electronics Engineers, Inc., 2000.

121. IEEE Std P1516.2. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT) Specification. – New York, NY : Institute of Electrical and Electronics Engineers, Inc., 2001.

122. IEEE Std P1516.3. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federation Development and Execution Process. – New York, NY : Institute of Electrical and Electronics Engineers, Inc., 2000.

123. AGI Licenses MAK Technologies' VR-Link For HLA And DIS Compliance In STK [Electronic resource] // Business Wire. – December 04, 2006. – Mode of access : http://www.businesswire.com/news/home/20061204005250/en/AGI-Licenses-MAK-Technologies-VR-Link-HLA-DIS#.Vh9_LkBQnEQ. – Title from the screen.

124. Milner R. A complete inference system for a class of regular behaviours / R. Milner // Journal of Computer and System Sciences. – 1984. – Vol. 28, № 3. – P. 439–466.

125. Milner R. A calculus of mobile processes / R Milner, J Parrow, D Walker // Information and Computation. – 1992. – Vol. 100, № 1. – P. 1–77.

126. Milner R. Communicating and mobile systems: the Pi-calculus / R. Milner. – New York, NY : Cambridge University Press, 1999. – 159 p.

127. Bergstra J. A. Fixed point semantics in process algebra : preprint / J. A. Bergstra, J. W. Klop // Stichting Mathematisch Centrum. Informatica. – 1982. – № IW 206/82. – P. 1–21.

128. Handbook of Process Algebra / eds. J. A. Bergstra, A. Ponse, S. A. Smolka. – Amsterdam : Elsevier Science, 2001.

129. Aceto L. Some of my favorite results in classic process algebra [Electronic resource] : Technical Report NS-03-2 / Luca Aceto. – BRICS, 2003. – 24 p. – Mode of access : www.brics.dk/NS/03/2/BRICS-NS-03-2.pdf. – Title from the screen.

130. Baeten J. C. M. Over 30 years of process algebra: Past, present and future / J. C. M. Baeten // Process Algebra : Open Problems and Future Directions, PA '03, July 21–25, 2003, Bologna, Italy / eds. L. Aceto, Z. Ésik, W. J. Fokkink, A. Ingólfsdóttir. – Amsterdam : Elsevier, 2003. – P. 7–12. – (BRICS notes series).

131. Park D. M. R. Concurrency and automata on infinite sequences : Technical Report / D. M. R. Park // Proceedings of the 5th GI Conference on Theoretical Computer Science. – Berlin : Springer-Verlag, 1981. – P. 167–183. – (Lecture Notes in Computer Science ; vol. 104).

132. R. J. van Glabbeek. Branching time and abstraction in bisimulation semantics / R. J. van Glabbeek, W. P. Weijland // Journal of the ACM. – 1996. – Vol. 43, № 3. – P. 555–600.

133. R. J. van Glabbeek. The linear time – branching time spectrum II: the semantics of sequential systems with silent moves / R. J. van Glabbeek // Proceedings CONCUR’93 / ed. E. Best. – Berlin : Springer-Verlag, 1993. – P. 66–81. – (Lecture Notes in Computer Science ; vol. 715).

134. R. J. van Glabbeek. What is branching time semantics and why to use it? / R. J. van Glabbeek // Bulletin of the EATCS. – 1994. – Vol. 53. – P. 190–198.

135. R. J. van Glabbeek. The linear time – branching time spectrum I: the semantics of concrete, sequential processes / R. J. van Glabbeek // Handbook of Process algebra. – New York, NY : Elsevier Science Inc., 2001. – P. 3–100.

136. Verification on infinite structures / O. Burkart, D. Caucal, F. Moller, B. Steffen // Handbook of Process algebra. – New York, NY : Elsevier Science Inc., 2001. – P. 545–623.

137. Groote J. F. Algebraic process verification / J. F. Groote, M. A. Reniers // Handbook of Process algebra. – New York, NY : Elsevier Science Inc., 2001. – P. 1151–1208.

138. Engberg U. Systems with label passing : Technical Report / U. Engberg,

M. Nielsen // DAIMI Report Series. – 1986. – Vol. 15, № 208.

139. Wischik L. New directions in implementing the π-calculus [Electronic resource] / LucianWischik. – 2002. – Mode of access : www. URL: http://www.fairdene.com/picalculus/implementing-pi-c.pdf. – Title from the screen.

140. Hansson H. Time and Probability in Formal Design of Distributed Systems : PhD thesis / H. Hansson. – University of Uppsala, 1991.

141. Lowe G. Probabilities and Priorities in Timed CSP : PhD thesis / G. Lowe. – University of Oxford, 1993.

142. Hillston J. A Compositional Approach to Performance Modelling : PhD thesis / J. Hillston. – Cambridge University Press, 1996.

143. Baeten J. C. M. Axiomatizing probabilistic processes: ACP with generative probabilities / J. C. M. Baeten, J. A. Bergstra, S. A. Smolka // Information and Computation. – 1995. – Vol. 121, № 2. – P. 234–255.

144. Нестеренко Б.Б., Новотарский М.А. Алгебра процессов для моделирования параллельных асинхронных вычислительных структур // Электронное моделирование. – 2006. – Т.28, №4. – С. 47-64.

145. Нестеренко Б. Б. Моделирование сложных дискретных систем / Б. Б. Нестеренко, М. А. Новотарский // Математичні машини і системи. – 2007, №3-4. – С. 111–121.

146. Нестеренко Б. Б. Определение адекватности моделей сложных дискретных систем / Б. Б. Нестеренко, М. А. Новотарский // Математичні машини і системи. – 2008. – №2. – С. 3–13.

147. Боев В. Д. Моделирование систем. Инструментальные средства GPSS World / В. Д. Боев. – СПб. : БХВ-Петербург. – 2004. – 368 с.

148. Питерсон Дж. Теория сетей Петри и моделирование систем: пер. с англ. / Дж. Питерсон. – М. : Мир, 1984. – 264 с.

149. Юдицкий С. А. И. А. Мурадян. Метод анализа конфигураций организационных систем на сетях Петри / С. А. Юдицкий, И. А. Мурадян // Управление большими системами. – М. : ИПУ РАН, 2007. – №16 – С. 163–170.

150. Бандман М. К. Территориально-производственные комплексы: прогнозирование процесса формирования с использованием Сетей Петри /

М. К. Бандман, О. Л. Бандман, Т. Н. Есикова. – Новосибирск. : Наука, 1990. – 303 с.

151. Крэйн М. Введение в регенеративный метод анализа моделей / М. Крэйн, О. Лемуан. – М. : Наука, 1982. – 104 с.

152. Калиниченко Д. В. Методы и средства прогнозирования времени выполнения последовательных программ / Д. В. Калиниченко, А. П. Капитонова, Н. В. Ющенко // Методы математического моделирования. – Москва : ВМК МГУ, 1997.

153. Миков А.И. Программные средства оптимизации распределенного имитационного эксперимента / А. И. Миков, Е. Б. Замятина, А. А. Козлов // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: труды Всероссийской суперкомпьютерной конференции, 21–26 сентября 2009 г. – М. : Изд-во МГУ, 2009. – 524 с.

154. Райтер Р. Распределенное имитационное моделирование дискретно-событийных систем / Р. Райтер, Ж. Вальран. – М. : Мир, ТИИЭР, 1989. – т. 77, №1. – С. 245-262.

155. Ladyzhensky Y. V. Software system for event-driven logic simulation / Y. V. Ladyzhensky, Y. V. Popoff // Proceedings of the IEEE EWDWT, Odessa, September 15–19, 2005. – Odessa, 2005. – P. 119–122.

156. Ладыженский Ю. В. Математическая модель динамического алгоритма продвижения времени для распределенного логического моделирования цифровых систем / Ю. В. Ладыженский, Г. А. Тесленко // Наукові праці Донецького національного технічного університету. Серія: Інформатика, кібернетика та обчислювальна техніка. – Донецк, 2008. – №9. – С. 55-62.

157. Петренко А.І. GRID технології в науці і освіті / А. І. Петренко // Системний аналіз та інформаційні технології : матеріали IX Міжнародної науково-технічної конференції, 15-19 травня 2007 р. – К. : НТУУ «КПІ», 2007. – С. 22–23.

158. Mascarenhas E. ParaSol: A multithreaded system for parallel simulation based on mobile threads / E. Mascarenhas, F. Knop, R. Vernon // Proceedings of the

27th conference on Winter simulation (WSC'95), December 3–6, 1995, Arlington, VA, USA. – Washington, DC : IEEE Computer Society, 1995.

159. Діденко Д. Г. Агент реплікації в розподіленій дискретно-подійній системі імітаційного моделювання OpenGPSS / Д. Г. Діденко // Інтелектуальні системи прийняття рішень та прикладні аспекти інформаційних технологій: матеріали міжнародної наукової конференції, 2006. – С. 264–266.

160. Диденко Д. Г. Взаимодействие агентов в распределенной дискретно-событийной системе имитационного моделирования OpenGPSS / Д. Г. Диденко // Имитационное моделирование. Теория и практика : сб. докл. 3-й всерос. науч.-практ. конф. по имитационному моделированию и его применению в науке и промышленности (ИММОД–2007), 17–19 окт. 2007 г. – Санкт-Петербург, 2007. – С. 272–278.

161. Шелестов А. Ю. Методи, моделі і технології аналізу та створення Grid-систем для задач дослідження Землі : дис. … д-ра техн. наук: 05.13.06 «Інформаційні технології» / А. Ю. Шелестов ; НАН України. Міжнар. наук.-навч. центр ЮНЕСКО інформ. технологій та систем. – К., 2008. – 406 с.

162. Ефимов С. Н. Модели и алгоритмы формирования grid-систем для структурно-параметрического синтеза нейросетевых моделей / С. Н. Ефимов, В. С. Тынченко // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева. – 2008. – №4(21). – С. 18-22.

163. Кузнецов Г. В. Модель обчислення загальної метрики довіри для підвищення рівня інформаційної безпеки регіонального сегменту грід / Г. В. Кузнецов, А. О. Бубнов // Системний аналіз та інформаційні технології : матеріали XI Міжнародної нуково-технічної конференції, 26-30 травня 2009 р. – К.: ННК «ІПСА» НТУУ «КПІ», 2009. – С. 435.

164. Atsuko Takefusa. Bricks: A performance evaluation system for scheduling algorithms on the grids / Atsuko Takefusa // JSPS workshop on applied information technology for science. – 2001. – № 1.

165. GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing [Electronic resource]. – Mode of access : www. URL: http://www.buyya.com/gridsim/. – Title from the

screen.

166. Волк М. А. Журнализация состояний программных распределенных мооделей и ее использование в оптимистических алгоритмах синхронизации / М. А. Волк // Збірник наукових праць Харківського університету Повітряних Сил. – Х., 2010, Вип. 1 (23). – С. 104–107.

167. Гмурман В. Е. Теория вероятностей и математическая статистика / В. Е. Гмурман. – М. : Высш. школа, 2000. – 469 с.

ПРИЛОЖЕНИЕ А

Акты о внедрении результатов диссертационной работы

ЗАТВЕРДЖУЮ
Харківський національний університет радіоелектроніки

61166, м. Харків,
пр. Леніна, 14

Проректор з наукової роботи

професор _____ М.І. Сліпченко
"16" 05 2013 р.

ЗАТВЕРДЖУЮ
ТОВ "Альвіс"

02154, м. Київ,
буль. Русанівський, 7

Директор

_____ І.В. Жученко
"15" травня 2013 р.

АКТ

впровадження результатів дослідження

Ми, комісія у складі директора І.В. Жученко, головного конструктора О.А. Кухаренко, провідного інженера ЦІСТ ХНУРЕ М.А. Керносова склали даний акт за результатами дисертаційних досліджень аспіранта кафедри електронно-обчислювальних машин (ЕОМ) Харківського національного університету радіоелектроніки (ХНУРЕ) Гріделя Ростислава Миколайовича, спрямованих на розробку методів та моделей розподіленого імітаційного моделювання інформаційних систем, які розробляються компанією ТОВ "Альвіс".

Розроблені методи базуються на консервативних та оптимістичних методах синхронізації розподілених імітаційних моделей інформаційних систем. Запропоновані моделі та методи дозволяють провести багатоваріантний аналіз імітаційних схем моделювання.

Отримані результати демонструють підвищення ефективності етапу моделювання складних інформаційних систем, що розробляються. Вагомою перевагою розроблених методів аналізу моделей елементів систем є відсутність необхідності модифікації вже розроблених моделей для середовища розподіленого моделювання.

Результати досліджень можуть бути використані на етапах вибору методів імітаційного моделювання, побудови розподілених моделей, вибору алгоритму синхронізації та проведення розподіленої імітації на гетерогенних обчислювальних ресурсів.

Комісія перевірила практичну реалізацію запропонованих методів та програмних засобів і підтверджує їх працездатність та ефективність.

Даний документ не є підставою для фінансових розрахунків, премій та інших видів винагород.

Головний конструктор
ТОВ "Альвіс" О.А. Кухаренко

Директор
ТОВ "Альвіс" І.В. Жученко

Провідний інженер ЦІСТ ХНУРЕ,
к.т.н. М.А. Керносов

ЗАТВЕРДЖУЮ
Харківський національний університет радіоелектроніки

61166, м. Харків,
пр. Леніна, 14

Проректор з наукової роботи

професор М.І. Сліпченко
„16" 05 2013 р.

ЗАТВЕРДЖУЮ
ТОВ «ВЄНТА»

01011,м. Київв, вул. Панаса Мирного, 14

Директор

О.В. Александрова
„10" 05 2013 р.

АКТ

впровадження результатів дослідження

Ми, комісія у складі директора О.В. Александрова, головного спеціаліста Н.Е Погиба, провідного інженера ЦІСТ ХНУРЕ М.А. Керносова склали даний акт за результатами дисертаційних досліджень аспіранта кафедри електронно-обчислювальних машин (ЕОМ) Харківського національного університету радіоелектроніки (ХНУРЕ) Гріделя Ростислава Миколайовича, спрямованих на розробку методів аналізу імітаційних моделей інформаційних систем, використаних при побудові інформаційної системи ТОВ «ВЄНТА».

За допомогою запропонованої технології розподіленого імітаційного моделювання було проведено декілька експериментів з різними варіантами побудови інформаційної системи лікарні, вибрані оптимальні конфігурації обладнання та програмного забезпечення. Зафіксовано поступове скорочення часу проведення експериментів на задачах однієї розмірності за рахунок автоматизованих методів аналізу програмних моделей та перерозподілу обчислювальних ресурсів.

Комісія перевірила практичну реалізацію запропонованої технології моделювання та програмних засобів і підтверджує їх працездатність та ефективність.

Даний документ не є підставою для фінансових розрахунків, премій та інших видів винагород.

Головний спеціаліст
ТОВ «ВЄНТА» _____ Н.Е. Погиба

Директор
ТОВ «ВЄНТА» _____ О.В. Александрова

Провідний інженер ЦІСТ ХНУРЕ,
к.т.н. _____ М.А. Керносов

ПРИЛОЖЕНИЕ Б

Оконные формы программ анализа распределенных имитационных моделей и системы распределенного имитационного моделирования GRASS

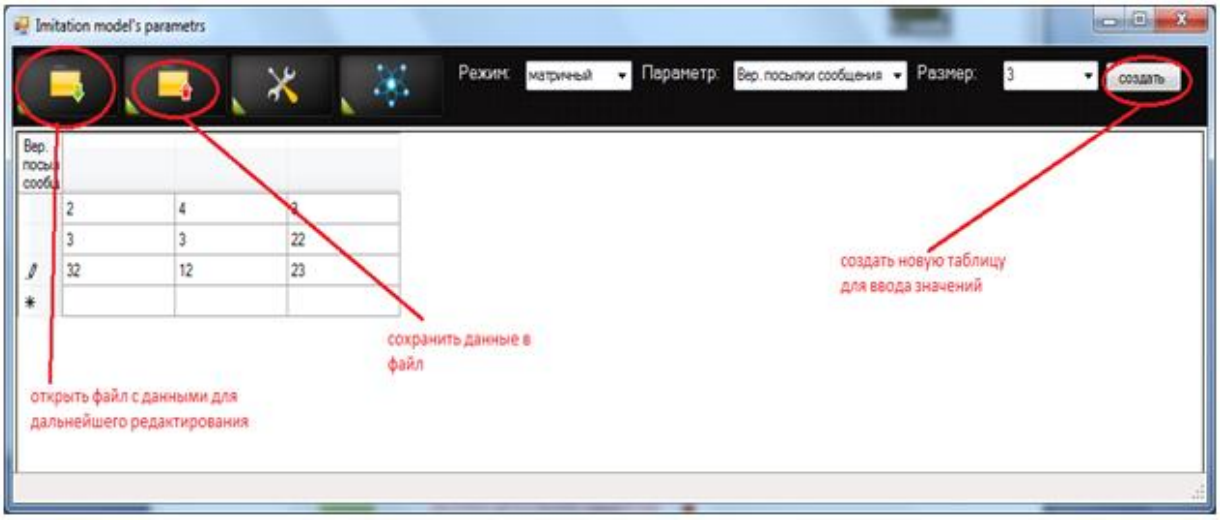


Рисунок Б.1 – Интерфейс модуля Input v0.1

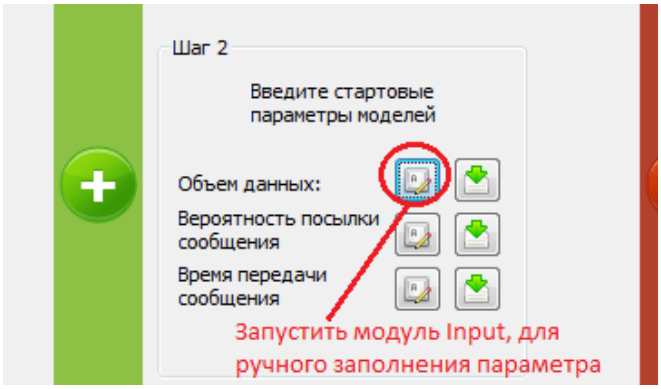


Рисунок Б.2 – Запуск модуля Input, с окна главной программы

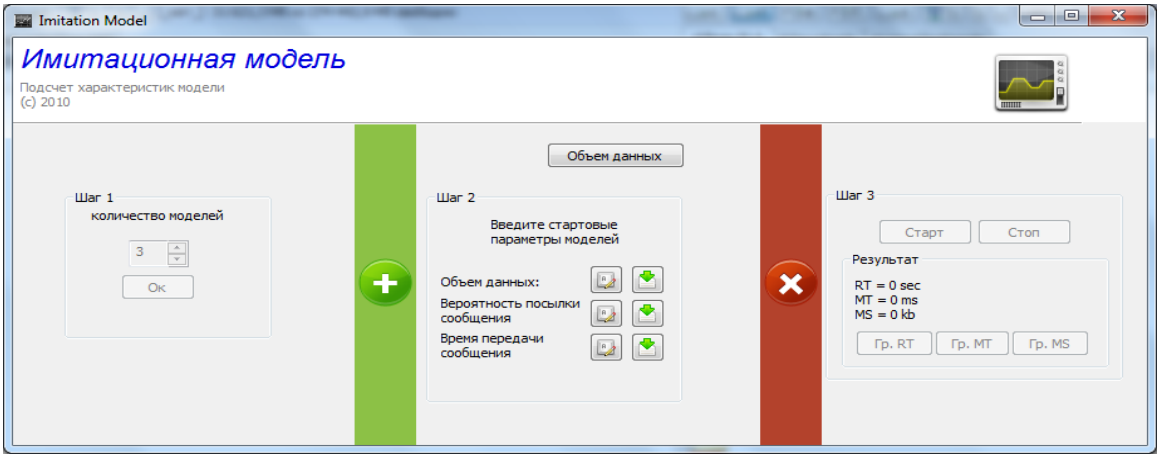


Рисунок Б.3 – Интерфейс главного окна программы

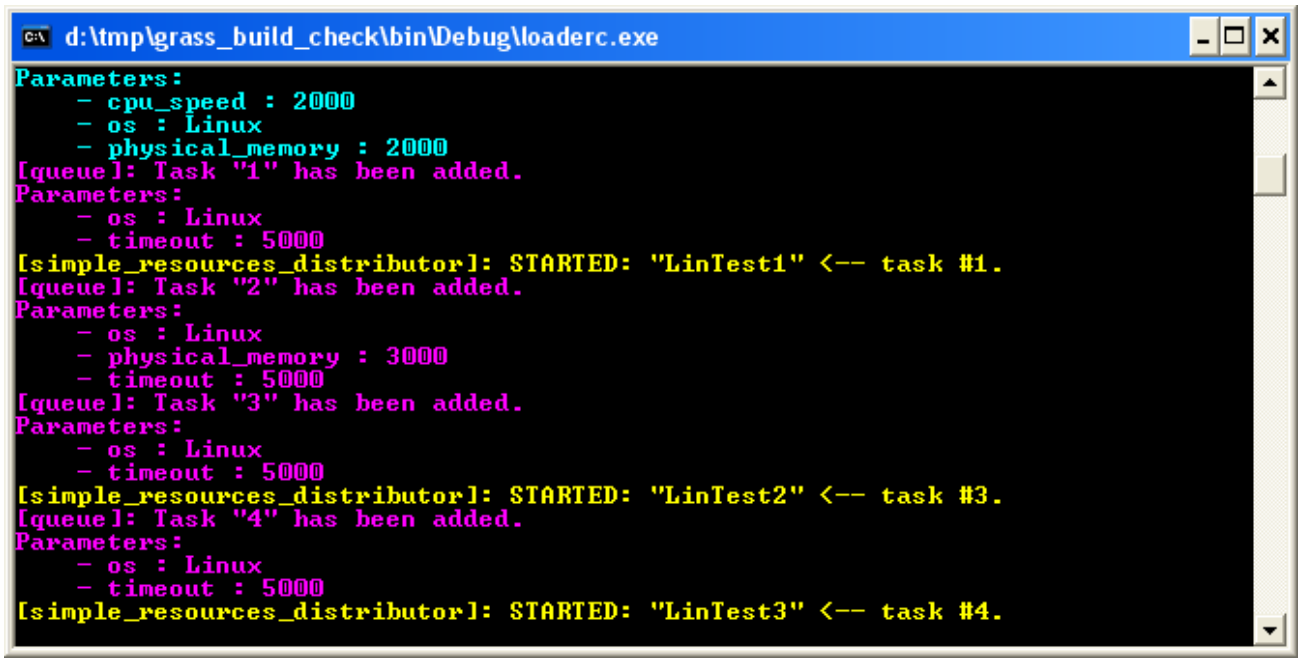


Рисунок Б.4 – Консольный интерфейс системы GRASS

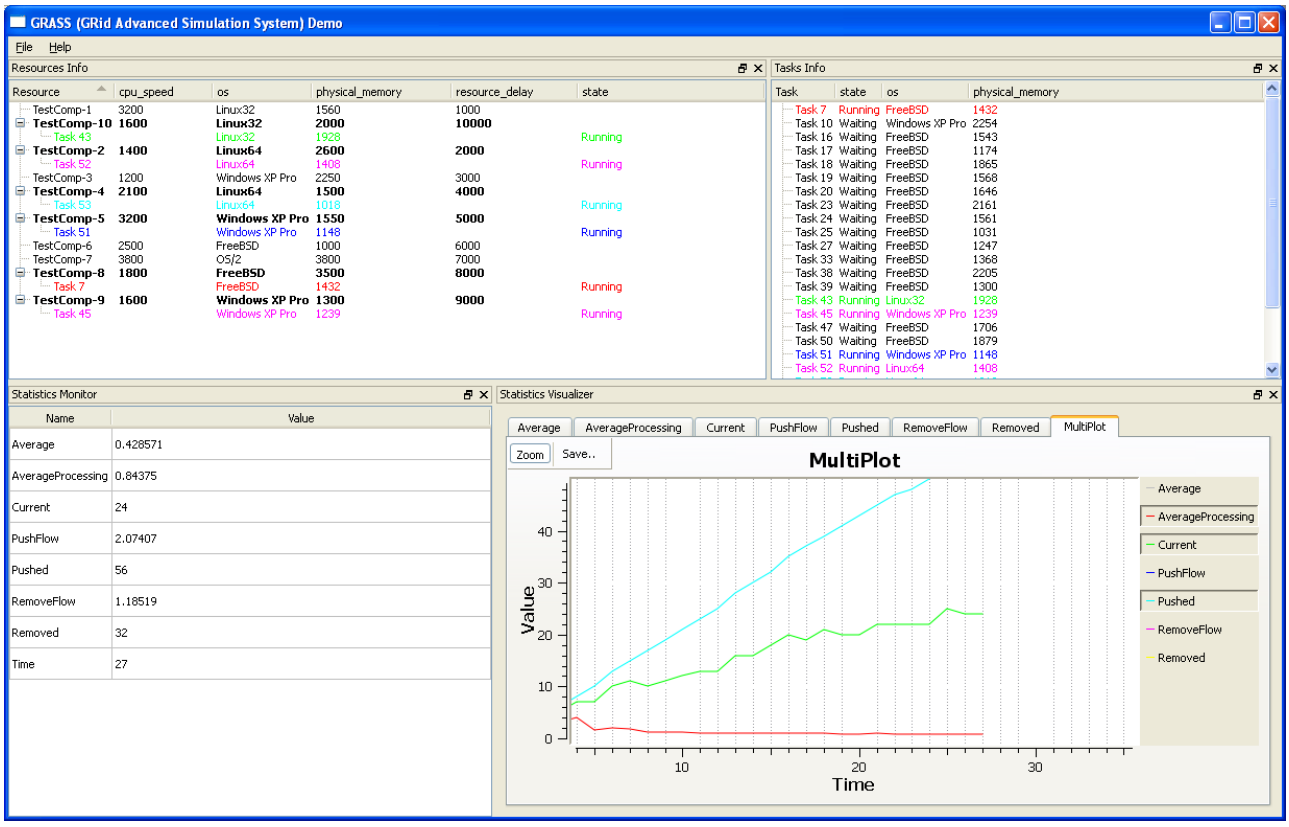


Рисунок Б.5 – Графический интерфейс системы GRASS

ПРИЛОЖЕНИЕ В

Листинг В.1 Вид основного конфигурационного файла системы

```
<?xml version="1.0" encoding="utf-8"?>
<framework>
    <config>
            <parameter pluginsDirectory = "./plugins/"/>
    </config>
    <plugins>
            <plugin name="AlgorithmLoader" filename="algorithm_loader"
     loadatstartup="yes" >
                    <parameter TasksCount="10"/>
                    <parameter Algorithm="RandomDistributionAlgorithm"/>
            </plugin>
            <plugin name = "DataManager" filename = "data_manager" />
            <plugin  name="Example"  filename="example"  loadatstartup="no"
>
                    <parameter StringToPrint="Hello, world!"/>
                    <parameter Times="3"/>
            </plugin>
            <plugin name = "Logger" filename = "logger">
                    <parameter LogDirectory = "./log/"/>
                    <parameter MessageFormat
                        ="[$(DateTime)][$(MessageLevel)]:          $(Mes-
sageText)"/>
                    <parameter GlobalMessageLevel = "debug"/>
                    <parameter SectionsMessageLevel = ""/>
                    <parameter StdOutEcho = "all"/>
            </plugin>
            <plugin name = "Queue" filename = "queue" />
            <plugin  name  =  "RandomDistributionAlgorithm"  filename  =
"rda"/>
            <plugin name = "ResourcesController" filename = "rc" />
            <plugin  name  =  "SimpleResourcesManager"  filename  =  "srm"
loadatstartup =
     "yes" >
                    <parameter InputFile="./config/resources_data.xml"/>
                    <parameter                                  ModelConfig-
File="./config/resources_data_model.xml"/>
            </plugin>
            <plugin  name  =  "SimpleTasksGenerator"  filename  =  "stm"
loadatstartup =
     "yes">
                    <parameter ModelFile="./config/model.xml"/>
            </plugin>
    </plugins>
</framework>
```

ПРИЛОЖЕНИЕ Г

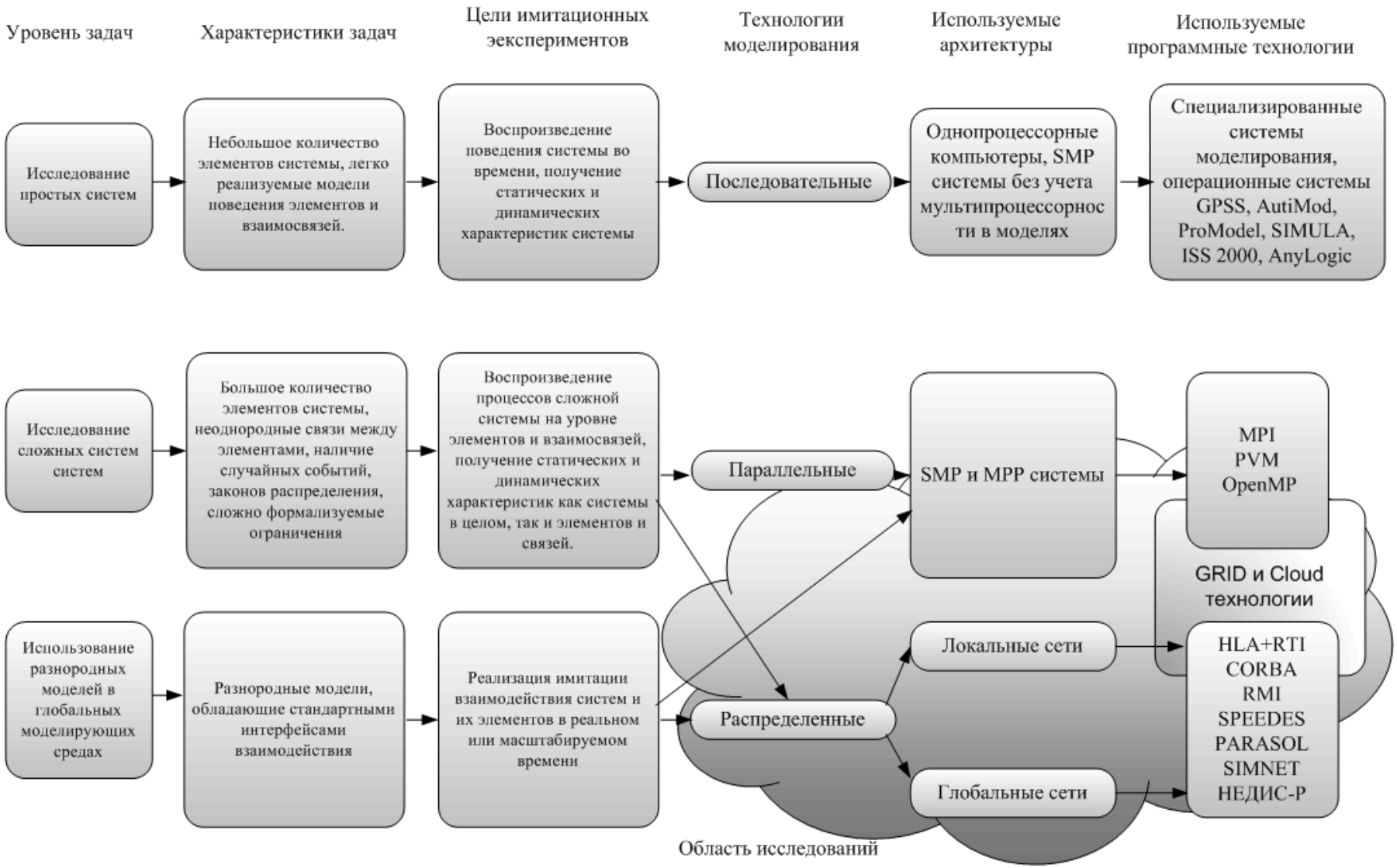


Рисунок Г.1 – Технологии имитационного моделирования