РЕФЕРАТ

Отчет 154 с., 30 рис., 14 табл., 27 источников, 5 прил.

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ, РЕСУРС-ДЕЙСТВИЕ-ОПЕРАЦИЯ, МОДЕЛЬ, РАСПРЕДЕЛЕННЫЙ, МОДУЛЬ, ТИП РЕСУРСА, РЕСУРС, ПЕРЕДАЧА ДАННЫХ, CORBA, УДАЛЕННЫЙ ВЫЗОВ.

Объектом разработки является распределенная многомодельная система дискретного имитационного моделирования на основе РДО. Ресурс, действие, операция (РДО) – программный комплекс, предназначенный для имитационного моделирования сложных дискретных систем с целью проведения их анализа и синтеза.

Цель работы – создание на основе РДО распределенной системы имитационного моделирования.

При создании системы проведены исследования, целью которых являлось установить оптимальный алгоритм изменения состояния ресурсов в системе для уменьшения загрузки сети.

В результате работы создана распределенная многомодельная система дискретного имитационного моделирования на основе РДО, которая позволяет нескольким распределенным по сети комплексам РДО связываться между собой и передавать данные о типах ресурсов, ресурсах и их состоянии в любой момент времени, таким образом, реализован современный «модульный подход» к построению моделей сложных дискретных систем.

Эффективность модульного подхода при моделировании заключается в возможности отойти от ограничений связанных с выполнением имитационной модели на однопроцессорном компьютере, возможности повторного использования готовых компонентов модели в различных комбинациях, упрощения и ускорения отладки и разработки моделей сложных систем.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, СИМВОЛОВ И СПЕЦИАЛЬНЫХ ТЕРМИНОВ С ИХ ОПРЕДЕЛЕНИЕМ

ARIS	Architecture of Integrated Information Systems (методология и программный продукт для моделирования бизнес-процессов компании)
CORBA	Common Object Request Broker Architecture (Общая архитектура брокера объектных запросов)
OMG	Object Management Group (Группа управления объектами)
ORB	Object Request Broker (Брокер объектных запросов)
UML	Universal Modeling Language (Универсальный язык моделирования)
БД (БЗ)	База данных (База знаний)
ВДТ	Видеодисплейные терминалы
ИМ	Имитационная модель
ОС	Операционная система
ПП	Программный продукт
ПО	Программное обеспечение
ПЭВМ	Персональная электронно-вычислительная машина
РДО	Ресурс Действие Операция – система имитационного моделирования
СДС	Сложная дискретная система
ТЗ	Техническое задание
ЭП	Эскизный проект

ВВЕДЕНИЕ

Разработка распределенной многомодельной системы дискретного имитационного моделирования на основе РДО является актуальной задачей развития РДО. В настоящее время в системе не существует механизмов, обеспечивающих совместную работу нескольких моделей в рамках одной распределенной.

При распределенном моделировании в отличие от последовательного моделирования первичной единицей является не объект, а так называемый логический процесс. Логический процесс – это последовательная подмодель, структура которой аналогична структуре дискретной модели. Каждый логический процесс имеет собственный набор объектов и собственную управляющую программу. Логический процесс имеет собственный локальный список событий и собственные часы локального модельного времени. Логические процессы взаимодействуют исключительно с помощью передачи сообщений. [1]

Современный модульный (распределенный) подход к построению имитационных моделей и моделированию обусловлен следующими преимуществами по отношению к «монолитному»:

1) Возможность отойти от ограничений, связанных с выполнением ИМ на однопроцессорном компьютере.
2) Возможность многоразового использования готовых компонентов модели в различных комбинациях при компоновке различных ИМ.
3) Возможность модернизации готовых ИМ на модульном уровне без изменения всех составляющих компонентов.
4) Перспективная возможность создания систем поддержки принятия решений – систем реального времени, которые могут представлять собой множество модулей (программ управления оборудованием, программ контроля и диагностики и т.д.), связанных с системой

ИМ посредством специфицированных интерфейсов, позволяющих выполнять моделирование и принимать пользователям соответствующие решения с учетом реального состояния объектов внешней среды.

5) Перспективная возможность создания многоагентных систем (МАС) на основе распределенных систем имитационного моделирования.

Этих причин вполне достаточно для того, чтобы понять актуальность реализации распределенной системы на основе РДО и приступить к ее разработке.

В дипломном проекте приведены в порядке следования разделы предпроектного исследования, концептуального, технического и рабочего проектирования распределенной многомодельной системы дискретного имитационного моделирования на основе РДО, исследовательская и огранизационно-экономическая части, а также раздел, касающийся техники безопасности и условий жизнеобеспечения.

1 Предпроектное исследование

1.1 Назначение программного комплекса РДО

Задачи системного анализа и синтеза объектов различной природы и назначения часто решаются с использованием имитационных моделей. Эти модели позволяют исследовать динамические аспекты поведения сложных дискретных систем и процессов. Имитация в частности позволяет выполнить анализ функционирования объекта, прогнозирование, организационное управление, поддержать принятие решений при проектировании и управлении.

RAO-studio является средством имитационного моделирования, позволяющим воспроизводить на ЭВМ динамику объекта, принятие решений сложной системой управления, и даже моделировать деятельность человека при принятии решений. В основе имитатора лежит РДО-метод формализации знаний о дискретных системах и процессах. Знания представляются в форме модифицированных продукционных правил. При этом сохраняются такие достоинства продукционных систем, как универсальность, гибкость и наличие формальных механизмов логического вывода. Традиционные продукционные правила являются частным случаем модифицированных, поэтому в имитационную модель легко могут быть включены, например, экспертные системы.

Язык описания объектов, алгоритмов управления и задач в RAO-studio это по существу язык представления знаний. Он требует от пользователя лишь знаний в предметной области, а не в программировании. Пользователь описывает ресурсы, правила функционирования, требуемые показатели и анимационные кадры

непосредственно в терминах предметной области, не прибегая при этом к представлению своей системы в терминах какого-либо известного метода (системы очередей, сети Петри, автоматы) или языка типа SLAM-II, ARENA, SIMPLE++ и других. Это резко повышает гибкость, мощность и наглядность. РДО - язык высокого уровня, использующий символические имена, арифметические и логические выражения и функции, генераторы псевдослучайных чисел, модифицированные и простые продукции.

Основные элементы RAO-studio - это модифицированная продукционная система и аппарат событий. Действия инициируются системой вывода, а нерегулярные события специальным блоком. При имитации состояние системы изменяется в соответствии с описанием нерегулярного события либо действия, которое началось или завершилось. После любого изменения состояния, т.е. при каждом событии, вызывается система вывода. Она просматривает в базе знаний продукционные правила и проверяет по предусловиям, могут ли начаться какие-либо действия. При нахождении таких действий инициируются события их начала.

Продукционная система, блок имитации нерегулярных событий и аппарат ведения событий совместно осуществляют построение имитационной модели процесса. На основании анализа результатов имитации вычисляются требуемые показатели функционирования системы.

Система трассировки выводит подробную информацию о событиях в специальный файл, который затем обрабатывается для детального анализа процесса. Система анимации отображает на экране во время имитации поведение моделируемого объекта.

RAO-studio может быть применен для создания имитационных моделей, систем планирования, игр и тренажеров, экспертных систем

реального времени и гибридных систем, включающие экспертные системы, имитационные модели и алгоритмы оптимизации.

Непрерывные процессы также могут быть описаны, поскольку формулы интегрирования переменных состояния можно записать в виде модифицированных продукций. [2]

1.2 Функции программного комплекса

При выполнении работ, связанных с созданием и использованием ИМ в среде РДО, пользователь оперирует следующими основными понятиями:

Модель – совокупность объектов языка РДО, описывающих какой-то реальный объект, собираемые в процессе имитации показатели, кадры анимации и графические элементы, используемые при анимации, результаты трассировки.

Прогон – это единая неделимая точка имитационного эксперимента. Он характеризуется совокупностью объектов, представляющих собой исходные данные и результаты, полученные при запуске имитатора с этими исходными данными.

Проект – один или более прогонов, объединенных какой-либо общей целью. Например, это может быть совокупность прогонов, которые направлены на исследование одного конкретного объекта или выполнение одного контракта на имитационные исследования по одному или нескольким объектам.

Объект – совокупность информации, предназначенной для определенных целей и имеющая смысл для имитационной программы. Состав объектов обусловлен РДО-методом, определяющим парадигму представления СДС на языке РДО.

Объектами исходных данных являются:

- типы ресурсов (с расширением .rtp);

- ресурсы (с расширением .rss);
- образцы операций (с расширением .pat);
- операции (с расширением .opr);
- точки принятия решений (с расширением .dpt);
- константы, функции и последовательности (с расширением .fun);
- кадры анимации (с расширением .frm, .bmp);
- требуемая статистика (с расширением .pmd);
- прогон (с расширением .smr).

Объекты, создаваемые РДО-имитатором при выполнении прогона:

- результаты (с расширением .pmv);
- трассировка (с расширением .trc). [4]

На данный момент РДО реализует следующие основные функции, которые представлены на 1 листе дипломного проекта (Функциональная структура РДО (AS IS). Нотация ARIS):

1) Создание модели на языке РДО:
   - создание основных объектов (*.rtp, *.rss, *.pat, *.opr, *.smr);
   - создание объектов данных и функций (*.fun);
   - создание объектов вывода (*.frm, *.pmd, *.bmp).
2) Проведение экспериментов:
   - изменение параметров системы в процессе моделирования;
   - генерация случайных чисел.
3) Вывод результатов моделирования:
   - анимация объектов модели;
   - вывод необходимых показателей;
   - построение графиков;
   - трассировка изменений объектов модели.

Эти функции также изображены ниже (см. Рисунок 1).

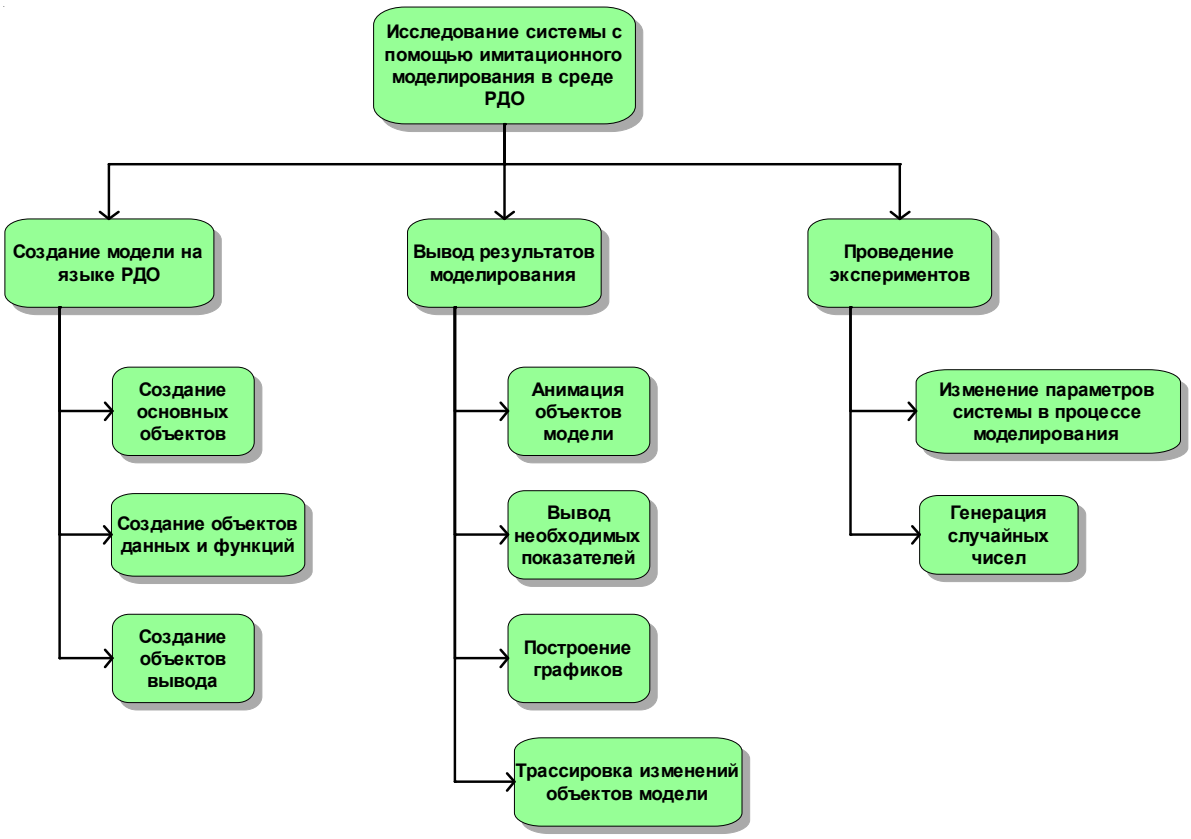

Рисунок 1 – Функциональная структура РДО (AS IS). Нотация ARIS.

1.3 Предпосылки создания распределенной системы

Для иллюстрации предпосылок создания распределенной многомодельной системы дискретного имитационного моделирования на основе РДО рассмотрим следующий пример постановки задачи на моделирование (представлен на 1 листе дипломного проекта – Пример постановки задачи на моделирование технологического процесса изготовления деталей). Предположим, что существует машиностроительное предприятие, производящего изделия в соответствии с номенклатурой (Н) по программе выпуска (П).

С целью исследования загрузки и производительности оборудования, производственных участков и цехов решается задача имитационного моделирования технологического процесса изготовления деталей.

Исходными данными для моделирования являются следующие:

1) Производственная система состоит из 4 цехов: склад, цеха механической и термической обработки деталей и сборочный цех.

2) Для каждого цеха известны параметры функционирования, достаточные для описания логики взаимодействия между объектами.

3) Для каждого цеха известны объекты, принимающие участие во взаимодействии и их параметры.

4) Известна последовательность прохождения заготовок (деталей) по цехам (последовательность схематично отображена стрелками – см. Рисунок 2).

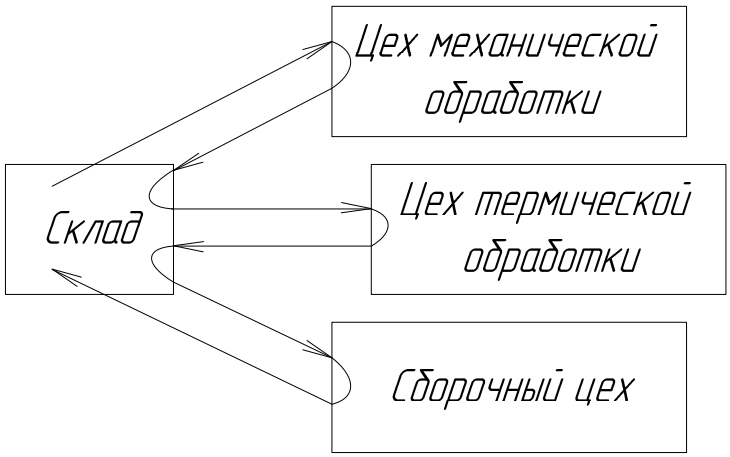

Рисунок 2 – Схема движения заготовок по цехам.

Рассмотрим возможные пути решения задачи моделирования.

При традиционном подходе к моделированию необходимо создавать «монолитную» модель, описывая работу каждого цеха в одной модели. Таким образом, база данных этой модели будет содержать информацию обо всех типах ресурсов и ресурсах, представленных в системе. Аналогично база знаний (образцы операций и операции) будет одной для производственной системы и будет описывать логику функционирования всех цехов. Структура имитационной модели на РДО при использовании этого подхода схематично изображена ниже (см. Рисунок 3).

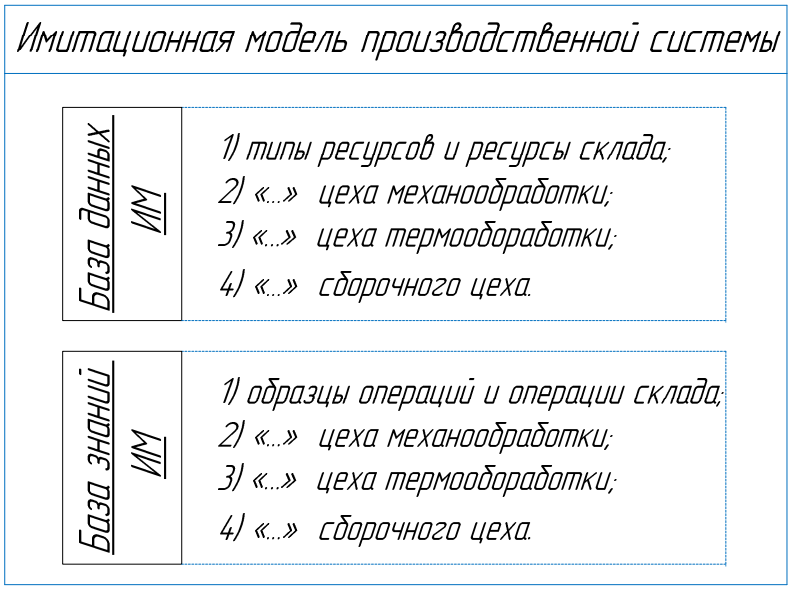

Рисунок 3 – Схема структуры имитационной модели при традиционном подходе к моделированию.

В другом случае при создании распределенной модели на РДО будет создано четыре отдельных имитационных модели, каждая из которых будет реализовывать законченный функционал каждого модуля (цеха) производственной системы (отдельная БД и БЗ для каждого цеха). Общими для всех цехов ресурсами в этом случае будут детали, которые изначально будут описаны в имитационной модели склада. Когда деталь будет иметь определенное состояние в соответствии с выбранной классификацией, она будет активизировать работу соответствующей модели (модуля) – так будет реализовано моделирование технологического процесса обработки деталей. Структура имитационной модели на РДО при использовании этого подхода схематично изображена ниже (см. Рисунок 4).

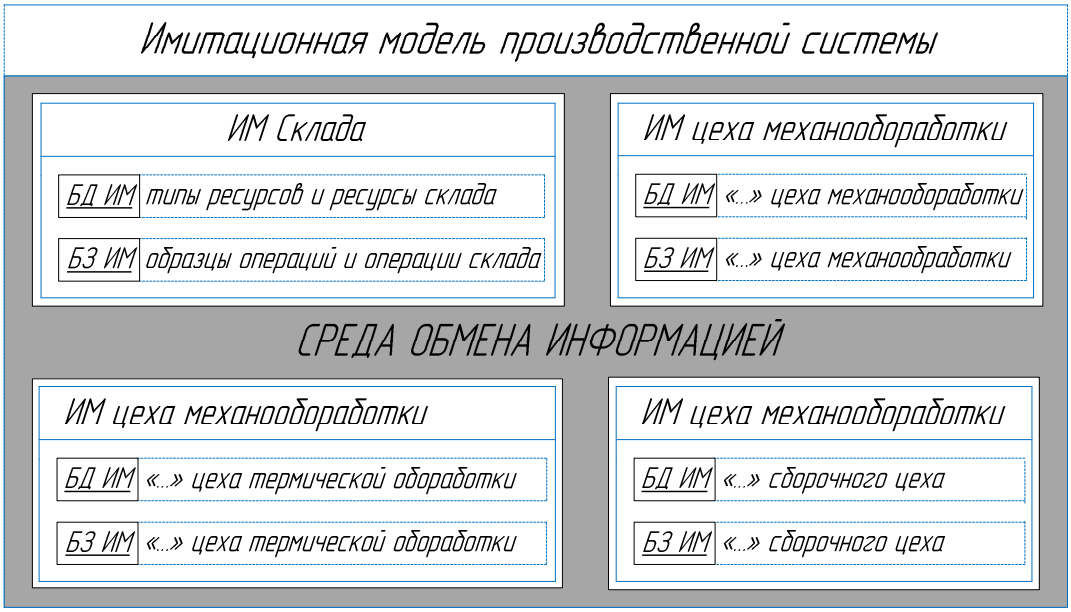

Рисунок 4 – Схема структуры имитационной модели при модульном подходе к моделированию.

Преимущества модульного подхода к моделированию (предпосылки реализации) очевидны и заключаются в следующем:

1) Появляется возможность отойти от ограничений, связанных с выполнением ИМ на однопроцессорном компьютере, т.е. ускорить вычислительный процесс за счет распараллеливания вычислений на отдельные ПЭВМ.
2) Ускоряется процесс разработки моделей за счет:
   - параллельной разработки моделей, как модулей распределенной модели;
   - упрощения разработки, отладки и модернизации на модульном уровне без затрагивания остальных составных частей распределенной модели (уменьшение кода, описание только функционала законченной части технологического процесса);
   - использования ранее разработанных моделей цехов;

3) Появляется возможность избежать дублирования информации за счет использования ранее описанных типов ресурсов и ресурсов во вновь создаваемой модели модуля (возможно использование единого хранилища типов ресурсов и ресурсов).

1.4 Выводы по предпроектному этапу

Рассмотренная выше задача моделирования демонстрирует прекрасный пример применения распределенной многомодельной системы дискретного имитационного моделирования на основе РДО и четко поясняет предпосылки ее создания.

Задача взаимодействия моделей в режиме выполнения представляет наибольший интерес с точки зрения моделирования и может быть решена за счет создания в РДО программного интерфейса, отвечающего за взаимодействие с другими моделями на РДО (изменение состояния ресурсов).

В любом случае требуется программирование РДО как распределенного приложения, которое влечет за собой изучение и применение соответствующих технологий.

Так как на текущий момент времени существует большое количество технологий распределенного программирования, есть все основания полагать, что задача создания распределенной многомодельной системы дискретного имитационного моделирования на основе РДО может быть успешно решена. Остается определиться с выбором технологии, изучить ее положения, сформулировать цели и концепцию разработки, затем разработать соответствующие алгоритмы функционирования системы на техническом этапе проектирования и приступить к рабочему этапу реализации системы.

2 Концептуальное проектирование

2.1 Цели разработки системы

Возвращаясь к ранее рассмотренному примеру задачи на моделирование технологического процесса обработки деталей, следует сформулировать основные цели разработки распределенной системы. Они представлены на диаграмме целей (нотация ARIS), изображенной на 2 листе дипломного проекта (см. Рисунок 5).

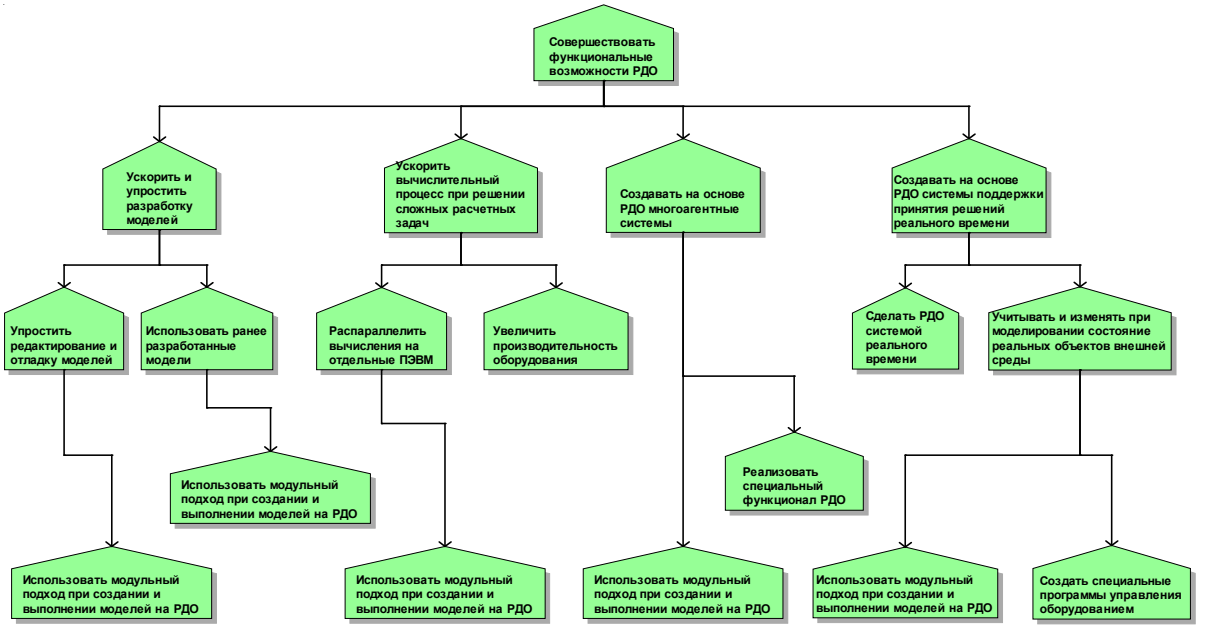

Рисунок 5 – Дерево целей разработки распределенной системы.

Так как реализация модульного подхода (распределенного) при создании моделей на РДО приводит к совершенствованию функциональных возможностей РДО во многих аспектах, именно «Совершенствовать функциональные возможности РДО» является главной целью разработки распределенной системы.

Подцелями являются следующие:

1) Ускорить и упростить разработку моделей за счет:

- упрощения редактирования и отладки моделей (параллельно производится на модульном уровне без изменения всех составляющих компонентов);
- использования ранее разработанных моделей (или их компонентов, таких как типы ресурсов и ресурсы) в новых распределенных моделях.

2) Ускорить вычислительный процесс при решении сложных расчетных задач за счет:
   - распараллеливания вычислений на отдельные ПЭВМ;
   - увеличения производительности оборудования (в данной работе не является приоритетом).
3) Перспективная возможность создания на основе РДО многоагентных систем (для создания многоагентных систем потребуется реализация специального функционала РДО, о котором не ведется речи в данной работе, однако, реализация модульного подхода является необходимым условием функционирования многоагентных систем на основе РДО).
4) Перспективная возможность создания на основе РДО систем поддержки принятия решений (представляют собой множество модулей (программ управления оборудованием, программ контроля и диагностики и т.д.), связанных с системой ИМ посредством специфицированных интерфейсов, позволяющих выполнять моделирование и принимать пользователям соответствующие решения с учетом реального состояния объектов внешней среды). Следует заметить, что эта цель, как и предыдущая, не является приоритетной в данной работе, так как для ее достижения (как видно из дерева целей), кроме создания интерфейсов взаимодействия между распределенными компонентами (модулями РДО), необходимо наделить РДО

дополнительным функционалом, касающимся поддержания режима реального времени. Так же потребуется создание специальных программ взаимодействия с внешней средой (управление конкретным оборудованием) для каждой конкретной системы поддержки принятия решений.

Как видно из диаграммы большинство этих подцелей будут решены при использовании модульного подхода при создании и выполнении имитационных моделей на РДО. Таким образом, для достижения приоритетных для данной работы целей необходимо создать интерфейсы обмена данными для РДО и решить вопросы синхронизации модельного времени в распределенной системе.

2.2 Новая функциональность РДО

В этом случае достижения поставленных приоритетных целей разработки, каждая отдельная система РДО будет рассматриваться как модуль (один из множества компонентов распределенной модели) системы и будет реализовывать следующие функции:

1) Создание модели на языке РДО, как модулей распределенной модели:
   - создание основных объектов модуля (*.rtp, *.rss, *.pat, *.opr, *.smr);
   - создание объектов данных и функций модуля (*.fun);
   - создание объектов вывода модуля (*.frm, *.pmd, *.bmp);
   - использование готовых моделей или их компонентов.
2) Проведение экспериментов с распределенной моделью:
   - использование баз данных других модулей распределенной модели;

- изменение параметров распределенной системы (распределенных и «внутренних» ресурсов) в процессе моделирования;
- моделирование с учетом состояния реальных объектов внешней среды (при реализации вышеуказанного дополнительного необходимого функционала РДО);
- генерация случайных чисел.

3) Вывод результатов распределенного моделирования:

- анимация объектов модели;
- вывод необходимых показателей;
- построение графиков;
- трассировка изменений объектов модели.

На 2 листе дипломного проекта (см. Рисунок 6) приведена функциональная структура комплекса РДО (нотация ARIS), как модуля распределенной системы имитационного моделирования.

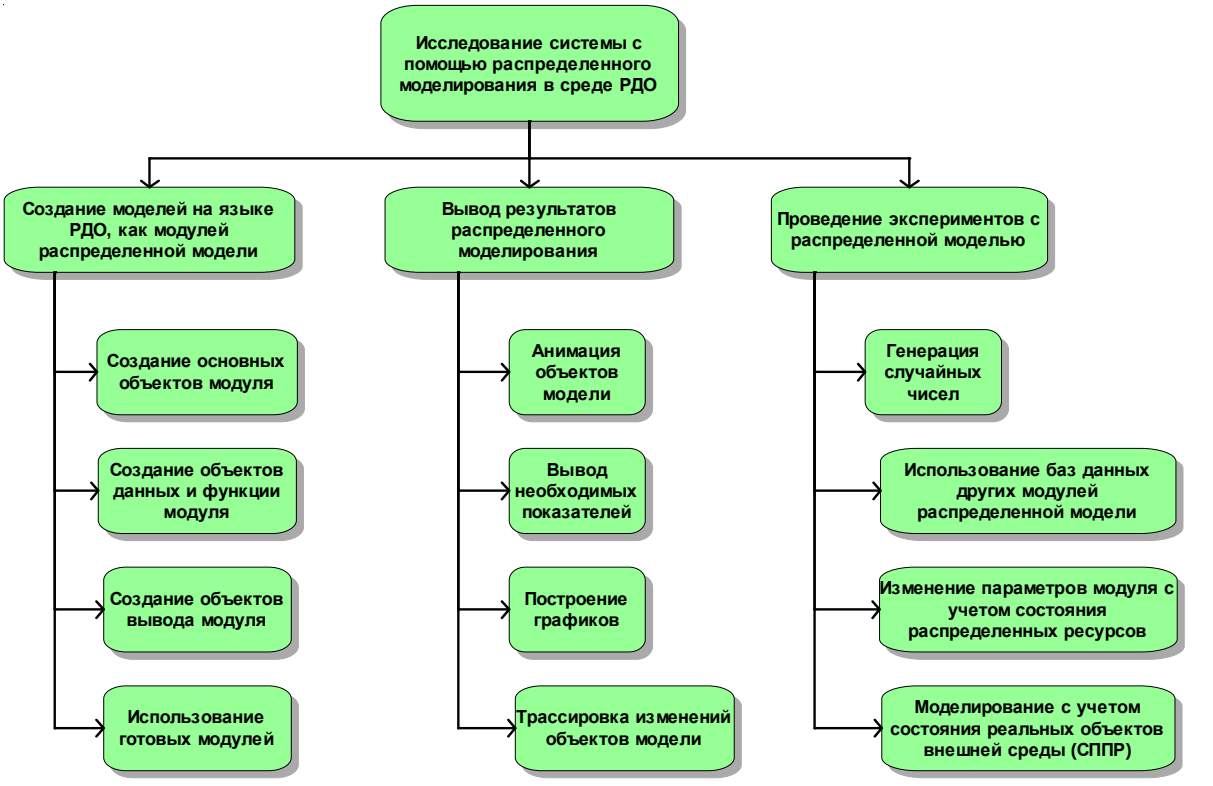

Рисунок 6 – Функциональная структура РДО (TO BE). Нотация ARIS.

2.3 Выбор технологии реализации распределенной системы

Для реализации транспортного уровня распределенной многомодельной системы дискретного имитационного моделирования на основе РДО мы будем использовать технологию программирования распределенных приложений – CORBA. Причина такого выбора заключается в том, что CORBA является открытой, платформонезависимой и независимой от языка программирования развитой концепцией программирования распределенных систем в отличие от других подобных технологий (RMI, EJB, DCOM и т.д. [5]).

Платфомонезависимость и многоязыковость могут сыграть решающую роль при построении на основе РДО системы поддержки принятия решений и разработки ее модулей под UNIX-подобные операционные системы на других (более удобных для разработчиков) языках программирования (например, Java).

Для нашего приложения CORBA будет обеспечивать передачу необходимых нам данных по сети (за счет вызова методов распределенных объектов). Подробнее основные понятия технологии рассмотрены ниже.

2.3.1 Технология распределенного программирования CORBA

2.3.1.1 Общие сведения

CORBA (Common Object Request Broker Architecture) является спецификацией создания распределенных программных систем [1316], развиваемой консорциумом OMG (Object Management Group. Лидерами OMG являются фирмы Oracle, IBM, Sun. Цель OMG - достижение «быстрого роста технологии объектов» через развитие архитектуры управления объектами OMA (Object Management Architecture). OMA является концептуальным базисом, на котором основаны спецификации OMG.

CORBA развивается как система открытых стандартов с 1989 г. Первый итоговый документ OMG был опубликован в 1991г.

На основе CORBA развивается компонентная модель объектов CCM (CORBA Component Model). Целью этого проекта является стандартизация процессов разработки распределенных систем, специализированных по отраслевому признаку - системы управления производством, финансами и т.п.

Отличительными характеристиками CORBA являются:

1) описание всех интерфейсов на языке IDL (Interface Definition Language), разработанном OMG;
2) наличие связующего программного обеспечения (ПО), включающего ядро ORB (Object Request Broker) и много-численные стандартные службы CORBA. В настоящее время существует более 15 таких служб, из которых самыми распространенными являются службы именования, управления жизненным циклом и событиями;
3) реальная интероперабельность, выражающаяся в независимости от языка программирования и вычислительной платформы.

Существует большой выбор реализаций CORBA от разных производителей - от коммерческих до свободно распространяемых (часто не менее удачных). Это давно уже позволяет заниматься практической разработкой распределенных систем на основе этой технологии. Каждый конкретный комплект CORBA предоставляет разработчику набор готовых утилит (IDL компилятор, ядро ORB и набор CORBA сервисов), что позволяет сосредоточиться на разработке прикладной составляющей распределенной системы.

Область применения CORBA чрезвычайно широка - от чисто исследовательских проектов (например, из области распределенных вычислений, изучения характеристик компьютерных сетей, построения

моделей сложных технических систем и т.д.) до решения проблем уровня управления производством и финансами предприятия.

Исследовательские проекты в области информационных технологий, включающие использование CORBA, финансируются Европейским Сообществом, начиная с середины 90-х годов (проекты EDISON, SEDRES, DSE).

Наиболее заметными продуктами, построенными на базе CORBA, являются системы управления предприятием фирмы Oracle.

Параллельно постоянно совершенствуются комплекты реализации CORBA. К наиболее известным относятся комплекты VisiBroker, IONA (ORBIX, ORBacus) и TAO.

Поддержка технологии CORBA встроена во многие известные инструменты разработки: Builder C++, JBuilder, Forte for Java и Forte for C++ Internet Edition.

Отображение спецификации CORBA (фактически речь идет об отображении языка IDL) существуют для следующих языков программирования: C, C++, Java, Smalltalk, COBOL, Ada.

Отображения языка определяют, как можно обращаться к объектам через ORB, используя конкретный язык. Отображение включает формирование IDL определений для типов данных и процедур используемого языка, структуру исключений и прочие важные детали, необходимые программисту-разработчику.

В соответствии со спецификацией, ограничения по аппаратному исполнению и операционной системе в CORBA отсутствуют.

Интероперабельность CORBA подразумевает, что клиентские фрагменты распределенной системы могут вызывать серверные фрагменты, независимо от того, на каких (разумеется, поддерживаемых) языках программирования они написаны, а также на базе каких именно реализаций ORB развернуты эти фрагменты.

2.3.1.2 Архитектура

Архитектура CORBA в настоящее время может считаться эталонной для построения распределенных систем. Она полностью соответствует правилам объектного стиля разработки программ и, фактически, обобщает его в направлении компонентного подхода, где проводится четкое разделение между понятиями: интерфейс, объект (реализующий интерфейс на уровне исходного кода на языке высокого уровня) и программный компонент (реализующий объект на уровне исполняемого кода).

Описание CORBA начинают с описания объектной модели, используемой в стандарте. Прежде чем определять архитектуру взаимодействия объектов, необходимо определить архитектуру самих объектов. Объекты могут посылать и принимать сообщения от других объектов и изменять при этом внутреннее состояние. Передача сообщения полностью аналогична удаленным вызовам методов объектов. Сообщение при этом содержит идентификатор целевого объекта и параметры запроса (как и в случае «обычных методов» возможен вызов без параметров или без возвращаемого значения). Объектная модель OMG полностью разделяет интерфейс объекта и его реализацию. Более того, сама модель имеет дело только с интерфейсами объектов, при этом термины интерфейс объекта и его тип являются синонимами. Такое разделение позволяет абстрагироваться от конкретного языка реализации объектов.

Для описания интерфейсов OMG разработала специальный язык IDL (Interface Definition Language). Основные понятия технологии объектного программирования - инкапсуляция, наследование и полиморфизм,- поддерживаются синтаксисом IDL.

Реализация распределенной системы на основе технологии CORBA, изображена на 4 листе дипломного проекта.

Связующее ПО основано на реализациях брокера объектных запросов ORB (Object Request Broker). Главными частями ORB являются:

1) реализация общего протокола межсетевого взаимодействия GIOP (General Inter-Object Protocol) протокола;
2) реализация GIOP, специфическая для конкретного типа сетей; обязательной в настоящее время, считается реализация для сетей TCP/IP, называемая IIOP (Internet Inter-Object Protocol);
3) реализация Объектного Адаптера, выполняющего важные функции администрирования созданных CORBA-объектов;
4) реализация стандартного интерфейса ORB (активация и регистрация объектов; получение и разрешение объектных ссылок; поддержка динамических вызовов методов объектов, тип которых заранее не известен, и т.д. Всего - около 40 методов);

Объект является базовым элементом архитектуры CORBA. Функциональность объекта полностью определяется интерфейсом, который он реализует. При этом содержание термина реализация в точности соответствует терминологии современных объектно-ориентированных языков программирования. На этапе написания исходного кода на объектно-ориентированных языках (C++, Java) IDL-интерфейсы преобразуются в классы, а CORBA-объекты - в экземпляры этих классов.

2.3.1.3 Основные понятия архитектуры CORBA

CORBA-объект - виртуальное понятие: он представляет собой нечто, расположенное на брокере объектных запросов, посылающее запросы к другим CORBA-объектам - серверным объектам и получающее запросы от других CORBA-объектов - клиентов. CORBA-объект не имеет физической реализации, например, его нельзя записать на дискету. Обращение к нему осуществляется по объектной ссылке (object reference), которая в отличие от самого объекта является вполне реальной и представляет собой

последовательность символов, которую, например, можно хранить в файле.

На этапе реализации CORBA-объект как бы раздваивается. А именно, в серверном фрагменте кода программист имеет дело с обычным объектом - экземпляром класса, который является потомком некоторого специального класса - скелетона. В клиентском фрагменте кода используется объект с тем же интерфейсом, который является потомком другого специального класса - стаба интерфейса. Именно в этом смысле следует понимать часто встречающееся в литературе не совсем точные высказывания типа «клиентский объект вызывает серверный объект». На уровне описания архитектуры системы такая «раздвоенность» отсутствует [5].

Ниже (см. Рисунок 7) показана связь между большинством понятий архитектуры CORBA, которые описаны в этом разделе.

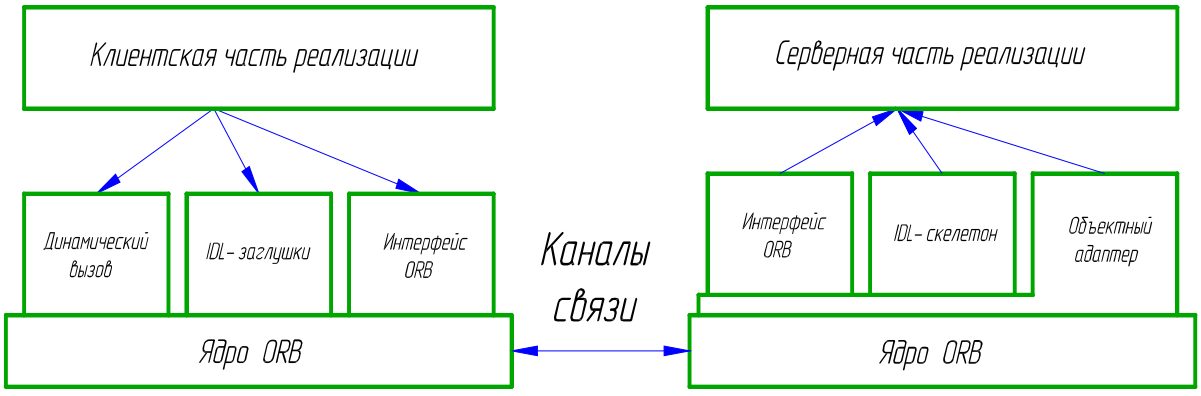

Рисунок 7 - Общая архитектура брокера объектных запросов (CORBA)

Сервант – серверная программа, написанная на каком-либо из языков программирования и выполняющая CORBA-объект. Это может быть набор функций, представляющих состояние объекта или реализации определенных классов серверного приложения (на C++ или Java). Сервант написан на том же программном языке, что и приложение, и является программной реализацией объекта CORBA.

Необходимо отделять CORBA-объекты от сервантов. По своей ленивой виртуальной природе CORBA-объект не способен ответить на запрос клиента, так что этим занимается как раз сервант. С другой стороны, CORBA-объекты могут иметь состояние, в то время как серванты не обязательно имеют его. Например, состояние CORBA-объекта может храниться в БД. В этом случае соответствующий сервант занимается извлечением и модификацией объекта – записи базы данных, но сам состояния не имеет.

Скелетон – серверная программа, которая связывает сервант с объектным адаптером, позволяя объектному адаптеру перенаправлять запросы к соответствующему серванту. В языке Си скелетон – набор указателей на функции серванта, в C++ скелетон – родительский класс для всех классов сервантов. При статических методах вызова скелетон формируется при компиляции IDL кода. При динамических – не используется. [7]

Портируемый (переносимый) объектный адаптер POA (Portable Object Adapter). В архитектуре CORBA адаптер представляет собой элемент, обеспечивающий создание и реализацию серверных объектов. До CORBA 2.1 в качестве такого элемента был определен, т.н. основной объектный адаптер BOA (Basic Object Adapter), поддержка которого регламентировалась для всех ORB. Спецификации основного адаптера были неполными, и разработчики создавали свои, весьма различные версии. POA был впервые определен в CORBA 2.2. Он полностью интегрирован с другими спецификациями технологии, и является одним из основных усовершенствованных элементов серверной части в CORBA 3. Основной адаптер просто исключен из CORBA. Конечно, разработчики ORB продолжают его поддерживать в целях обратной совместимости уже разработанных систем.

Идентификатор объекта (object ID) - уникальное имя объекта внутри его объектного адаптера. Он представляет собой последовательность байт, которая ассоциируется с объектом в момент его создания. Обеспечивается либо программным приложением, либо POA. Идентификатор объекта не обязан быть уникальным во всем остальном мире и даже для сервера.

Уникальной характеристикой объекта является *объектная ссылка IOR (Interoperable Object Reference).* Как правило, идентификатор объекта является частью этой объектной ссылки. Клиент при обращении к целевому объекту использует именно IOR. Есть еще объектный ключ (тоже часть объектной ссылки), который используется протоколом взаимодействия брокеров объектных запросов GIOP для идентификации конечной точки связи. Именно там содержатся параметры, позволяющие взаимодействовать с объектом по сети. Например, IP-адрес, номер порта [5].

2.3.2 Алгоритм разработки приложений

Получив представление об архитектуре CORBA, мы можем построить алгоритм работы над системой РДО в дипломном проекте при решении поставленных задач.

Он включает в себя полную последовательность действий по разработке распределенного приложения, начиная с «установки и настройки программного обеспечения, реализующего идеи CORBA» и заканчивая «работой с распределенным приложением в рамках достижения поставленной цели».

Данный алгоритм в добавление к последовательности концептуальных шагов содержит также блоки условий для того, чтобы получить наглядное представление об обработке исключительных ситуаций в процессе разработки клиент-серверного приложения.

Алгоритм представлен на 3 листе дипломного проекта (Алгоритм разработки распределенного приложения на основе CORBA) содержит следующую последовательность действий:

1) Необходимо установить и настроить программное обеспечение, реализующее функционал CORBA. В соответствии с документацией поставщика потребуется провести настройку операционной системы и среды программирования.

2) Выполнение, если необходимо, анализа проектируемого приложения с целью его упрощения за счет деления на подсистемы, реализуемые объектами на разных компьютерах и/или разными программистами.

3) Создание IDL-описаний интерфейсов, с помощью которых клиентские программы смогу обращаться к серверным приложениям, на которых будут реализованы необходимые клиенту функции или методы.

4) Трансляция IDL-описаний интерфейсов в файлы-исходники с помощью IDL-компилятора необходима для получения с помощью стандартных средств ПО, реализующего идеи CORBA, IDL-заглушек и IDL-скелетонов, необходимых для реализации серверной и клиентской программы.

5) Наличие ошибок при создании файлов означает, что необходимо локализовать ошибки либо на уровне описания интерфейсов, либо на уровне адекватной работы ПО, реализующего идеи CORBA. В любом случае нужно вернуться на этап алгоритма, указанный соответствующей связью.

6) Реализация серверного приложения с использованием всех файлов-исходников, сгенерированных IDL-компилятором на основании файла с описаниями интерфейсов.

7) Реализация клиентского приложения с использованием файлов-заглушек, сгенерированных IDL-компилятором на основании файла с описаниями интерфейсов.

8) Оценка функциональности распределенного приложения и отладка ошибок с возвращением на этапы алгоритма разработки распределенных приложений в соответствии со связями.

9) После отладки последовательный запуск сервера и клиента, работа с распределенным приложением в рамках достижения поставленной цели.

Алгоритм разработан в соответствии с ГОСТ 19.701-90 [9] на основе информации, полученной из [11].

2.4 Структура системы для верхнего уровня

Плакат «Структура и уровни взаимодействия компонентов распределенной много модельной системы дискретного имитационного моделирования на основе РДО» представлен на 4 листе дипломного проекта. Целью этого материала является выявление различных логических уровней взаимодействия компонентов распределенной системы, а также компонентов и взаимодействий внутри каждого отдельного модуля, что очень важно для понимания концепции создаваемой системы.

Хотелось бы подчеркнуть, что эта структура и дальнейшие диаграммы, разработанные в дипломном проекте, рассматриваются с точки зрения клиент-серверной архитектуры в рамках одного запроса, т.е. есть клиентская часть одного из компонентов распределенного приложения обращается к серверной части одного из компонентов того же распределенного приложения.

Надо понимать, что компонент, рассматриваемый как клиент, в то же самое время может быть участником другого клиент-серверного

взаимодействия и выполнять при этом функции сервера. Аналогично, компонент, рассматриваемый на диаграмме, как сервер, может быть клиентом, принимающим участие в другом клиент-серверном взаимодействии (описывается аналогичной структурной схемой, только роли компонентов различны).

Итак, рассматриваются два компьютера (*Компьютер 1* и *Компьютер 2* на схеме), связанные между собой физическими каналами связи, что показано двусторонней стрелкой взаимодействия. На каждом из них функционирует система имитационного моделирования РДО (*РДО 1* и *РДО 2* на схеме) с запущенными имитационными моделями соответственно цеха механической обработки деталей и склада (отдела логистики и складирования).

Каждая из этих моделей реализована в программном комплексе РДО и может самостоятельно выполнять поставленные задачи. Выделим информационный поток между моделями, как поток верхнего уровня (наиболее абстрактный с точки зрения реализации уровень обмена информацией о значения параметров ресурсов). Примерами, информационно обмена на этом уровне является передача данных (состояния параметров ресурсов) о готовности деталей (заготовок) со склада поступить на обработку, о состоянии обработки деталей, о готовности склада принять обратно обработанные детали, данные об отказах оборудования и т.д.

Другой информационный поток – обмен между модулями РДО по сети (удаленный вызов процедур) – реализуется компонентами CORBA (информация в виде последовательности байтов (пакетов) передается по сети).

Важнейшей составляющей системы является программный интерфейс РДО-CORBA, который реализует внутрикомпонентное взаимодействие вышеуказанных информационных потоков.

В дипломном проекте рассматривается реализация этого интерфейса с позиции достижения следующих целей:

1) Организация совместной работы нескольких моделей на РДО в рамках одной распределенной модели.
2) Возможность получения и использования информации о распределенных ресурсах и их типах при компиляции модели на РДО.
3) Возможность получения информации о состоянии распределенных ресурсов в процессе моделирования.
4) Возможность изменения состояния распределенных ресурсов в процессе моделирования.

Чтобы осуществить вышеуказанные цели, потребуется реализовать эффективную передачу необходимых данных по сети между различными имитационными моделями, которая включает реализацию интерфейса РДО-CORBA.

Как было ранее отмечено, для реализации транспортного уровня распределенной многомодельной системы дискретного имитационного моделирования на основе РДО выбрана технология программирования распределенных приложений – CORBA. Для приложения CORBA будет обеспечивать передачу необходимых приложению данных по сети (за счет вызова методов распределенных объектов).

Однако сама по себе технология является лишь средством достижения поставленных целей. Сама логика работы распределенного приложения будет реализована на уровне интерфейса взаимодействия РДО-CORBA. Выполняемые функции клиентской и серверной части распределенной системы ИМ на основе РДО (которые необходимо реализовать) указаны на 4 листе дипломного проекта.

2.5 Функциональная модель работы системы

Для пояснения работы распределенной многомодельной системы имитационного моделирования на основе РДО разработана функциональная модель. Она необходима в качестве исходных данных для разработки программного обеспечения, т.к. по ней можно проследить алгоритмы функционирования системы, преобразование данных и объектов в процессе ее работы и т.д.

Необходимо заметить, что функциональная модель работающей системы составлена с учетом того, что выполнены все стадии разработки распределенного приложения на основе архитектуры CORBA и существуют все, получаемые в процессе их выполнения соглашения, документы, алгоритмы и программы.

Выделим основную функцию системы – см. Рисунок 8: *Выполнение моделирования на отдельном модуле РДО в рамка распределенной модели*.

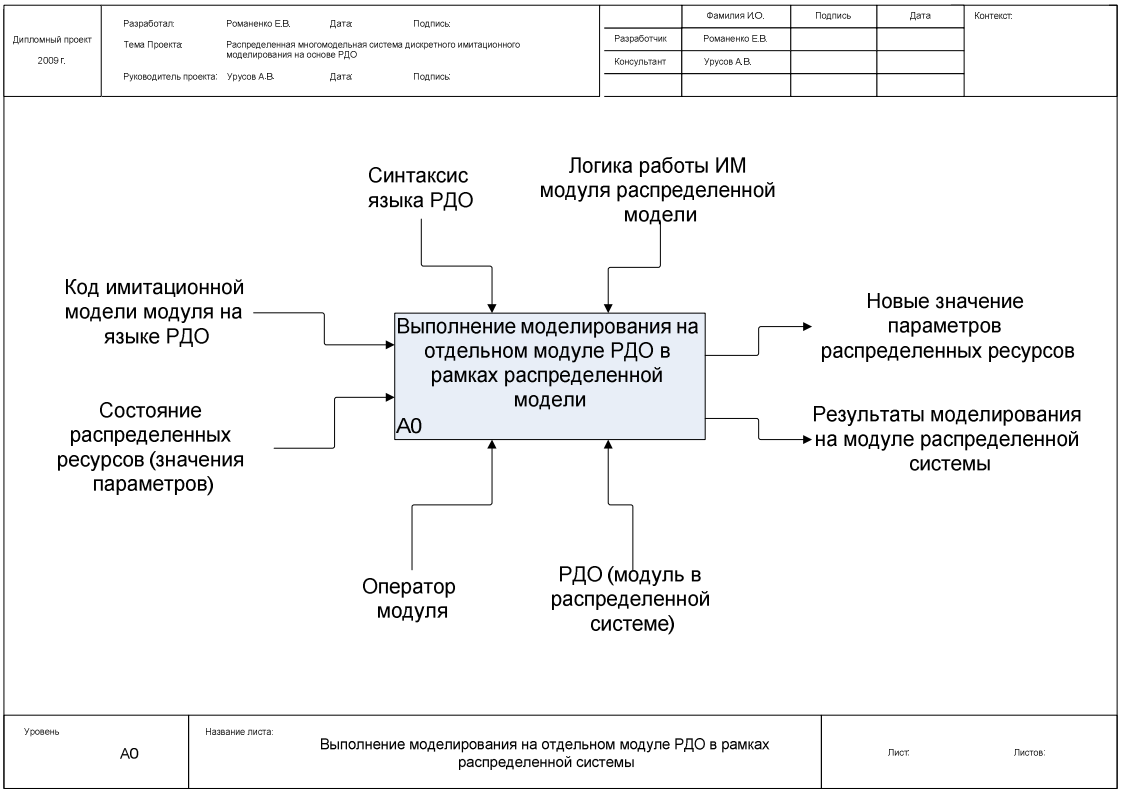

Рисунок 8 – Функциональная модель работы системы.

Входами модели являются:

- код имитационной модели модуля на языке РДО;
- состояние распределенных ресурсов (значения параметров необходимы для учета состояния при моделировании).

Выходами являются:

- новые значения параметров распределенных ресурсов (в результате выполнения модели изменяется состояние распределенных ресурсов);
- результаты моделирования на модуле распределенной системы (графики, анимация, трассировка показателей, вывод необходимых показателей).

Управление осуществляется посредством:

- синтаксиса языка РДО (разработка кода имитационной модели и перевод в соответствующие объекты);
- логика работы имитационной модели модуля распределенной модели (в соответствии с поведением и функциями моделируемой системы или ее части).

Следующие механизмы принимают участи в моделировании:

- программный комплекс РДО, рассматриваемый как модуль в распределенной системе;
- оператор модуля РДО.

В результате декомпозиции основной функции появляются следующие (функциональная модель приведена на 5 листе дипломного проекта):

1) *Компиляция моделей с учетом распределенных ресурсов и их типов*. Сюда включены запросы от данного модуля к внешним моделям о сборе информации о распределенных типах ресурсов и ресурсах, добавление их в «свою» базу данных и, собственно,

компиляция модели с созданием списка событий, действий и операций.

2) *Выбор следующего элемента из списка событий, действий, операций для текущего модельного времени с учетом состояния распределенных ресурсов.* Эта функция выполняет последовательную проверку выполнимости очередного события, действия или операции в зависимости от состояния собственных и указанных распределенных ресурсах.

3) *Выполнение выбранного события, действия или операции с возможным изменением параметров распределенных ресурсов.* Изменение состояния системы (параметров) ресурсов производится, как для локальной базы данных модели, так и для распределенных баз данных.

4) *Обработка результатов моделирования, анимация и подсчет статистики.* В эту функцию также входит перевод модельного времени.

Разработанная функциональная модель позволяет представить поведенческие аспекты распределенной многомодельной системы дискретного имитационного моделирования на основе РДО на концептуальном уровне.

2.6 Синхронизация модельного времени в распределенной системе

Отдельным вопросом концепции является синхронизация модельного времени в распределенной системе. Этот вопрос является наукоемким и достаточно объемным для детального обсуждения в данной работе, поэтому здесь рассматриваются только базовые понятия о

синхронизации модельного времени и предложен концептуальный механизм его реализации.

В ИМ имеются три понятия времени: физическое, модельное и процессорное. Физическое время относится к моделируемой системе. Модельное время – воспроизведение физического времени в модели. Процессорным временем будем называть время выполнения имитационной модели на компьютере.

Соотношение физического и модельного времени определяется спецификой моделируемой системы и задается коэффициентом временного масштаба – это диапазон физического времени, принимаемый за единицу модельного времени.

Сущностью ИМ является продвижение модельного времени при выполнении модели и выполнение событий, связанных с определенными значениями модельного времени. Событие в модели – это программный модельный образ значимого, с точки зрения разработчика модели, изменения в моделируемой системе.

Основной задачей ИМ является правильное отображение порядка и временных отношений между изменениями в моделируемой системе на порядок выполнения событий в модели. Если не рассматривать «одновременные» события, т.е. выполняющиеся в один и тот же момент модельного времени, то это требование правильного отображения порядка изменений в моделируемой системе означает, что события в модели должны выполняться в хронологическом порядке в модельном времени.

Процессорное время выполнения модели зависит от частоты изменений в моделируемой системе за моделируемый период физического времени, а также от объема вычислений, связанных с выполнением событий. Процессорное время явно не связано с модельным временем.

Таким образом, управление временем с системах ИМ является наиболее интересной, сложной, наукоемкой проблемой [1].

Применительно к задачам данной работы, предлагается рассматривать следующую концепцию синхронизации модельного времени в распределенной многомодельной системе дискретного имитационного моделирования на основе РДО (см. Рисунок 9).

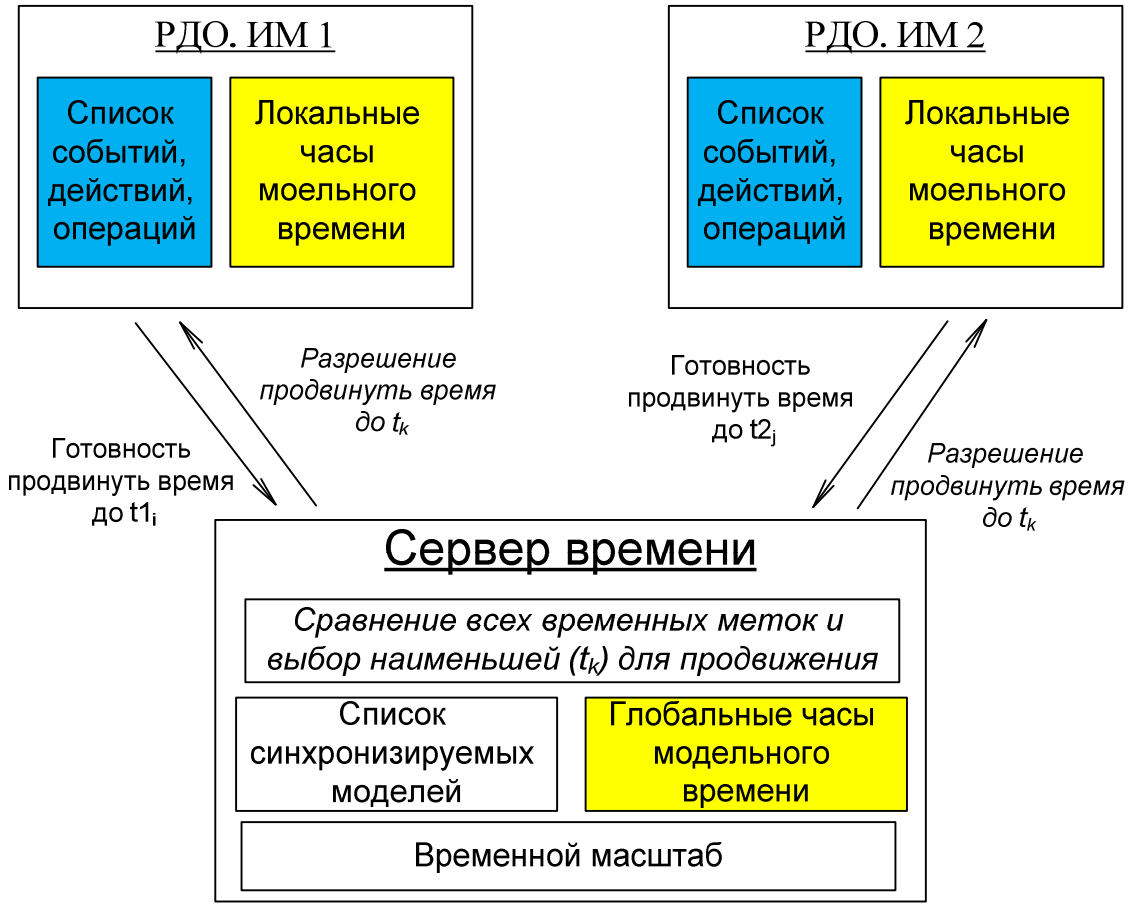

Рисунок 9 - Схема синхронизации модельного времени в распределенной системе.

В сети, где развернуты компоненты распределенной модели, запущен исполнимый файл, который представляет собой сервер времени, единственный для всей многокомпонентной модели.

Пользователь настраивает на сервере список моделей, которые необходимо синхронизировать, а также задает временной масштаб моделирования.

Сервер в неявном виде ведет глобальные часы модельного времени, которые запускаются в момент, когда все модели из списка обращаются к

нему с готовностью продвинуть свое локальное время на указанную в запросе величину. В момент, когда от всех моделей поступают подобные сообщения, сервер выбирает наименьшую из меток и разрешает всем моделям продвинуть свое локальное модельное время на эту минимальную величину. Затем сервер вновь ждет сообщений от всех моделей с новыми метками продвижения времени, а дальше процесс сравнения повторяется.

При такой организации процесса синхронизации, требование правильного отображения порядка изменений в моделируемой системе выполняется, что означает, что события в модели выполняются в хронологическом порядке в модельном времени.

Таким образом, межмодельное взаимодействие, которое является основным предметом обсуждения в данной работе, выполняется всегда в строго определенные моменты модельного времени, и соответственно его можно в дальнейшем рассматривать как отдельный законченный функционал системы, реализация которого не влияет на механизм синхронизации модельного времени.

2.7 Подготовка и проведение имитационных экспериментов с распределенными моделями

Рассмотрим, как изменятся этапы имитационного эксперимента в случае выполнения распределенного моделирования.

Процесс моделирования начинают с определения цели разработки модели, на основе которой затем устанавливаю границы системы и необходимый уровень детализации моделируемых процессов. Выбранный уровень детализации должен позволять абстрагироваться от неточно определенных из-за недостатка информации аспектов функционирования реальной системы. В описание системы, кроме того, должны быть включены критерии эффективности функционирования системы и оцениваемые альтернативные решения, которые могут рассматриваться

как часть модели или как ее входы. Оценки же альтернативных решений по заданным критериям эффективности рассматривают как выходы модели. Обычно оценка альтернатив требует внесения изменений в описание системы и, следовательно, перестройки модели. Поэтому на практике процесс построения модели является итеративным.

Этапы имитационного эксперимента представлены ниже (см. Рисунок 10) [4].

Рисунок 10 – Этапы имитационного эксперимента.

При выполнении распределенного моделирования сами этапы имитационного эксперимента не изменяются, также остается

итеративность процесса построения модели. Однако, на этапе формализации моделируемой системы, стоит выделить функционально-законченные подсистемы в моделируемой системе, которые в последствии будут запрограммированы как отдельные модули одной распределенной модели.

Кроме того, на этапе формализации моделируемой системы может быть установлено, что функционально-законченная часть вновь создаваемой модели уже ранее была создана, как компонент другой модели. В этом случае возможно его повторное использование (доработка и использование). Как говорилось ранее, итеративный процесс совершенствования (отладки) моделей как компонентов распределенной модели ускоряется за счет параллельной доработки моделей, а так же за счет того, что изменения вносятся на модульном уровне без затрагивания (если в этом нет необходимости) всех компонентов распределенной модели.

До этапа проведения имитационных экспериментов каждый модуль распределенной модели должен сообщить системе, состояние ресурсов какой внешней модели его интересует (в соответствии с логикой работы модели указывают разработчики при создании объектов прогона).

Проведение экспериментов проходит в обычном режиме, однако нельзя забывать, что при моделировании учитываются состояния распределенных ресурсов, поэтому связанные модели (имеющие в базе данных целевые ресурсы) должны быть также откомпилированы и запущены. Последовательность запуска моделей на исполнение зависит от логики функционирования моделируемой системы и устанавливается на этапе планирования имитационных экспериментов.

2.8 Разработка технического задания

Техническое задание разработано в соответствие с ГОСТ 19.201-78 (Единая система программной документации. Техническое задание. Требования к содержанию и оформлению) [8].

2.8.1 Введение

Распределенная многомодельная система дискретного имитационного моделирования на основе РДО имеет обширную область применения в сфере моделирования поведения реальных систем различной сложности, в частности при исследовании существующих систем, проектировании новых систем, организации и проведения обучения на различных моделях. Перспективным направлением использования системы является создание на ее основе территориально-распределенных систем поддержки принятия решений.

2.8.2 Основания для разработки

1) Разработка ведется на основании следующих документов:
   - Задание на выполнение дипломного проекта.
   - Календарный план на выполнение дипломного проекта.
2) Документы утверждены «09» февраля 2009 года.
3) Тема дипломного проекта:

   Распределенная многомодельная система дискретного имитационного моделирования на основе РДО.

2.8.3 Назначение разработки

Функциональное и эксплуатационное назначение связано с организацией процесса моделирования на РДО с учетом использования при моделировании распределенных типов ресурсов, которые описаны в имитационных моделях, открытых в других средах РДО в пределах локальной сети.

Основными целями моделирования с учетом распределенных ресурсов являются:

- Организация совместной работы нескольких моделей на РДО в рамках одной распределенной модели.
- Возможность получения информации о состоянии распределенных ресурсов в процессе моделирования.
- Возможность изменения состояния распределенных ресурсов в процессе моделирования.
- Возможность получения и использования информации о распределенных ресурсах и их типах при компиляции модели на РДО.

Принципиально новое функциональное назначение системы также связано с возможностью создания на ее основе территориально-распределенных систем поддержки принятия решений. Подобные системы на производстве могут представлять собой множество модулей (программ управления оборудованием, программ контроля и диагностики и т.д.), связанных с системой РДО посредством специфицированных интерфейсов, позволяющих выполнять моделирование и принимать соответствующие решения с учетом реального состояния объектов внешней среды.

2.8.4 Требования к программе или программному изделию

2.8.4.1 Требования к функциональным характеристикам

Требования к составу выполняемых функций реализуемой системы заключаются в возможности использования системой РДО при моделировании распределенных типов ресурсов и ресурсов, которые описаны в имитационных моделях на других системах РДО в пределах локальной сети.

2.8.4.2 Требования к надежности

Основное требование к надежности направлено на поддержание в исправном и работоспособном состоянии вычислительной сети, в рамках которой функционирует система (необходимы правильные сетевые настройки компьютеров и надежные каналы связи между ними).

2.8.4.3 Условия эксплуатации

Эксплуатация распределенного приложения должна проводиться на нескольких компьютерах в пределах локальной вычислительной сети с необходимыми настройками и физическими каналами связи, позволяющими проводить бесперебойный информационный обмен между компьютерами. Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости.

Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением при следующих климатических условиях:

- температура окружающей среды – от 15 до 30 градусов С;
- относительная влажность воздуха - от 30% до 80%;
- атмосферное давление - от 630 мм. р.с. до 800 мм. р.с.

2.8.4.4 Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 256 Мб;
- объем жесткого диска не менее 20 Гб;
- микропроцессор с тактовой частотой не менее 400 МГц;
- монитор не менее 15” с разрешением от 800*600 и выше;

Компьютеры должны быть подключены к локальной вычислительной сети с пропускной способностью каналов не менее 10 Мбит/с.

2.8.4.5 Требования к информационной и программной совместимости

Данная система должна работать под управлением операционных систем Windows NT, Windows 2000, Windows, XP и Windows 2003 Server. Программы, управляющие работой оборудования и/или его диагностикой, в составе системы поддержки принятия решений на основе РДО могут быть реализованы на языках C++ и Java и могут функционировать под управлением UNIX-подобных операционных систем и использовать специфицированные средства связи (API) с программным комплексом РДО.

2.8.4.6 Требования к маркировке и упаковке

Не предъявляются.

2.8.4.7 Требования к транспортированию и хранению

Не предъявляются.

2.8.4.8 Требования к программной документации

Не предъявляются.

2.8.5 Технико-экономические показатели

Расчет экономической эффективности разработанного приложения не является целью дипломного проектирования, однако возможный экономический эффект может быть достигнут за счет следующих преимуществ системы:

1) При разработке новых моделей появится возможность интеграции ранее разработанных моделей (компонентов моделей) и их

многократного переиспользования в различных комбинациях в качестве готовых модулей вновь разрабатываемых моделей.

2) Модульный подход позволит упростить разработку и отладку моделей сложных систем, а также создавать территориально распределенные системы поддержки принятия решений на основе РДО (принципиально новое функциональное назначение).

3) Повысится производительность процесса моделирования (в связи с возможностью отойти от ограничений, связанных с выполнением имитационной модели на однопроцессорном компьютере).

2.8.6 Стадии и этапы разработки

Состав, содержание и сроки выполнения работ по созданию системы в соответствии с календарным планом на выполнение дипломного проекта приведены ниже (Таблица 1).

Таблица 1 - Стадии и этапы разработки системы.

Стадии	Этапы работ	Срок выполнения
1. Предпроектное исследование	Предпроектное исследование РДО с точки зрения перспективы создания распределенной системы, анализ и выбор технологии ее реализации	25.02.09
2. Концептуальное проектирование	Разработка концепции реализации системы, изучение выбранной технологии программирования распределенных систем (основных понятий, алгоритмов и методов), разработка информационной модели передаваемых данных	15.03.09
	Разработка технического задания на систему	20.03.09
3. Техническое	Разработка алгоритмов функционирования системы	01.04.09

проектирование	(серверной и клиентской частей), структуры программных средств	
	Описание интерфейсов распределенных объектов и генерация необходимых файлов с исходными кодами	05.04.09
4. Рабочее проектирование	Программная реализация функционала системы	05.05.09
	Разработка тестового примера распределенной модели и апробирование системы	10.05.09
5. Организационно-экономическая часть	Расчета затрат на разработку системы	15.05.09
6. Техника безопасности и условия жизнеобеспечения	Требования безопасности при работе с системой, типовой расчет зануления ПЭВМ	20.05.09

2.8.7 Порядок контроля и приемки

Контроль и приемка распределенного приложения должны осуществляться в процессе проверки функциональности (апробирования) системы имитационного моделирования на тестовом примере распределенной модели в соответствии с требованиями к функциональным характеристикам системы.

2.8.8 Приложения

1) Документы, используемые при разработке приведены в списке использованных источников.
2) Используемое при разработке программное обеспечение:
   - Microsoft Visual Studio 2005.
   - Исходные коды и исполнимое ПО omniORB-4.1.2
   - Исходные коды системы РДО.

3 Техническое проектирование

Для описания технических аспектов реализации функционала распределенной многомодельной системы дискретного имитационного моделирования на основе РДО на этапе технического проектирования была использована нотация UML.

Унифицированный язык моделирования (UML) является стандартным инструментом для создания "чертежей" программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем.

UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. Это очень выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к ее разработке и последующему развертыванию. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования.

Несмотря на свои достоинства, UML - это всего лишь язык; он является одной из составляющих процесса разработки программного обеспечения, и не более того. [10].

3.1 Варианты использования системы

Диаграммы прецедентов представляют собой один из пяти типов диаграмм, применяемых в UML для моделирования динамических аспектов системы. Каждая такая диаграмма показывает множество прецедентов, актеров и отношения между ними.

Диаграммы прецедентов имеют большое значение для визуализации, специфицирования и документирования поведения элемента. Они облегчают понимание систем, подсистем или классов, представляя взгляд

извне на то, как данные элементы могут быть использованы в соответствующем контексте. Кроме того, такие диаграммы важны для тестирования исполняемых систем в процессе прямого проектирования и для понимания их внутреннего устройства при обратном проектировании [10].

Разработанная диаграмма вариантов использования системы представлена на 6 листе дипломного проекта и ниже (см. Рисунок 11).

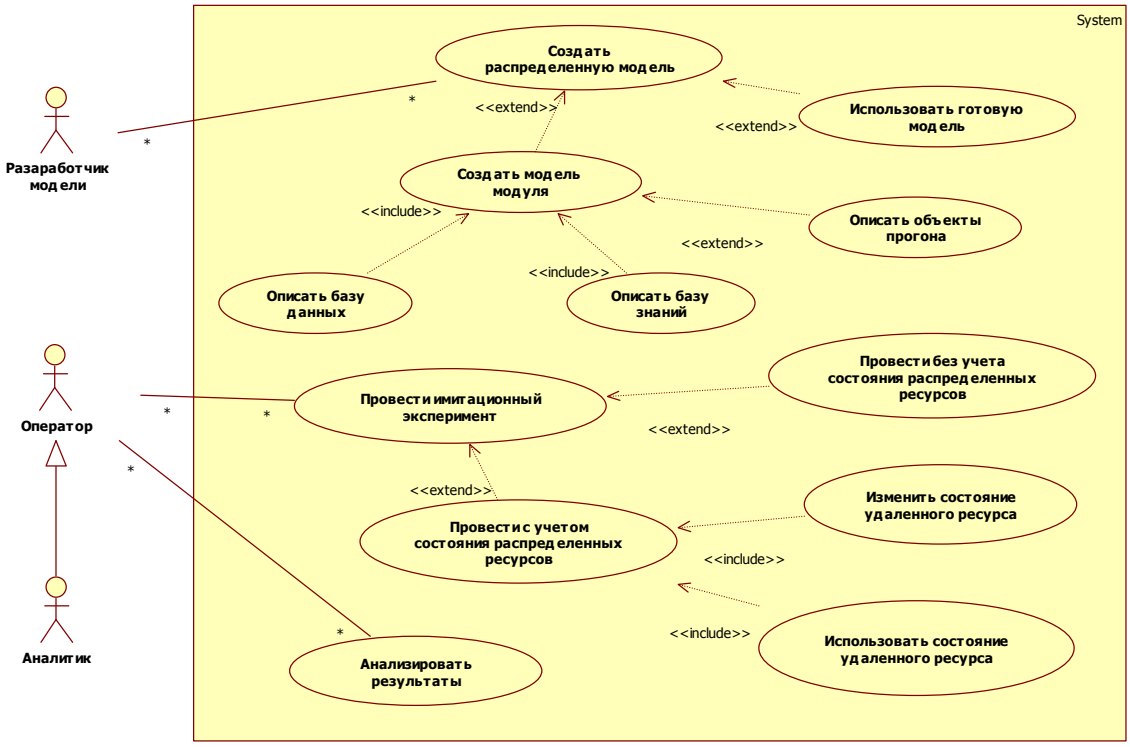

Рисунок 11 - Диаграмма вариантов использования системы. Нотация UML.

Как видно из диаграммы, варианты использования системы соответствуют ее функциям. В данной работе детально проработан вариант использования системы – проведение имитационных экспериментов с учетом состояния распределенных ресурсов на этапе компиляции модели на РДО.

3.2 Разработка диаграммы классов

На 6 листе дипломного проекта и ниже (см. Рисунок 12) представлена диаграмма классов разрабатываемой системы.

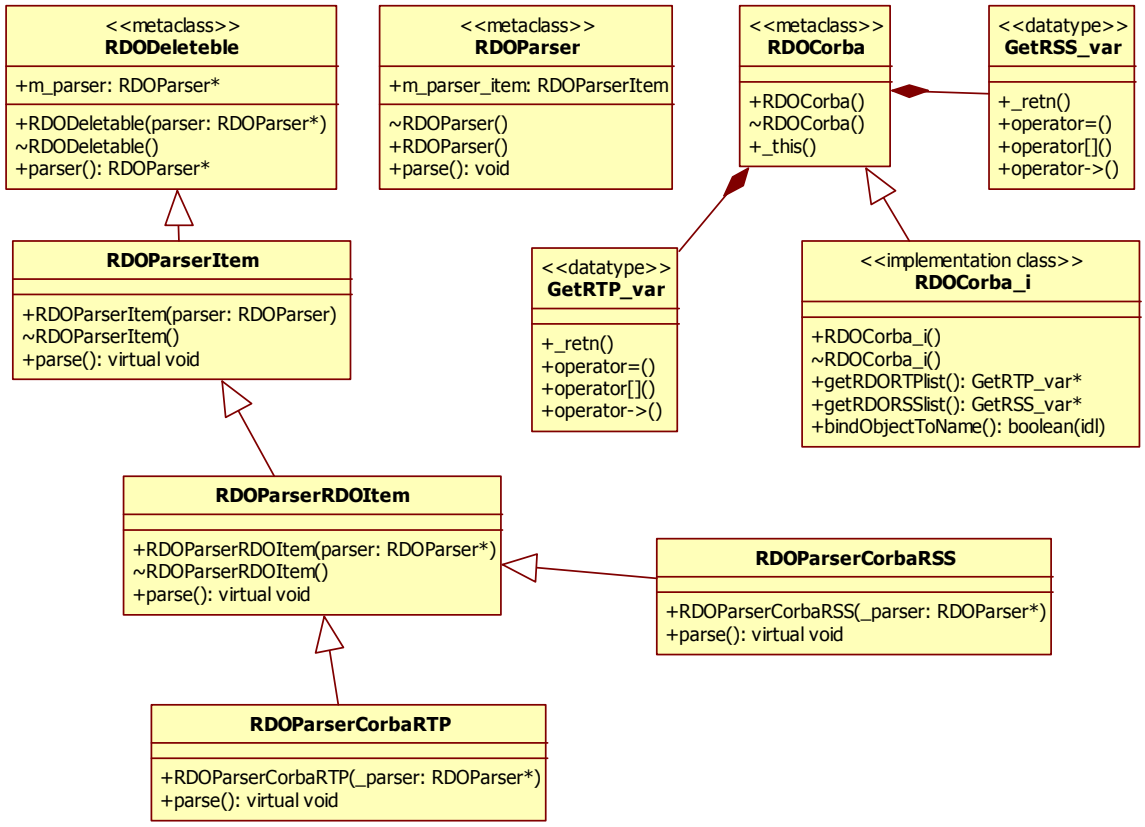

Рисунок 12 - Диаграмма классов. Нотация UML.

Здесь представлены только те классы, которые были созданы или модернизировались при разработке системы. В данном случае классы RDOParserCorbaRTP и RDOParserCorbaRTP представляют собой клиентскую часть реализации системы. Методы parse() этих классов реализуют следующий функционал:

1) формирование и отправку запросов на получение информации о распределенных ресурсах и их типах;
2) добавление распределенных ресурсов и их типов в базу данных «клиентской» модели.

Серверная часть реализации представляет собой «implementation class» RDOCorba_i – класс реализации методов распределенных объектов, методы которого реализуют следующий функционал:

1) сбор и передача информации о типах распределенных ресурсов, имеющихся в базе данных «серверной» модели, заинтересованным приложениям по их удаленным запросам;
2) сбор и передача информации о распределенных ресурсах, имеющихся в базе данных «серверной» модели, заинтересованным приложениям по их удаленным запросам.

Классы «data type» GetRTP_var и GetRSS_var представляют собой типы данных, экземпляры которых предназначены для хранения полной информации соответственно о типах ресурсов и ресурсах модели. Объекты этих типов данных предназначены для передачи по сети.

3.3 Функционирование системы в процессе работы с распределенной моделью

Для понимания алгоритма функционирования системы в процессе межмодельного взаимодействия в системе разработана диаграмма деятельности в нотации UML. Она представлена на 7 листе дипломного проекта и ниже (см. Рисунок 13).

Диаграммы деятельности, как правило, применяются, чтобы промоделировать последовательные (а иногда и параллельные) шаги вычислительного процесса. С помощью диаграмм деятельности можно также моделировать жизнь объекта, когда он переходит из одного состояния в другое в разных точках потока управления. Диаграммы деятельности описывают переходы от одной деятельности к другой. В конечном итоге деятельность (Activity) сводится к некоторому действию, которое составлено из атомарных вычислений, приводящих к изменению состояния системы или возврату значения.

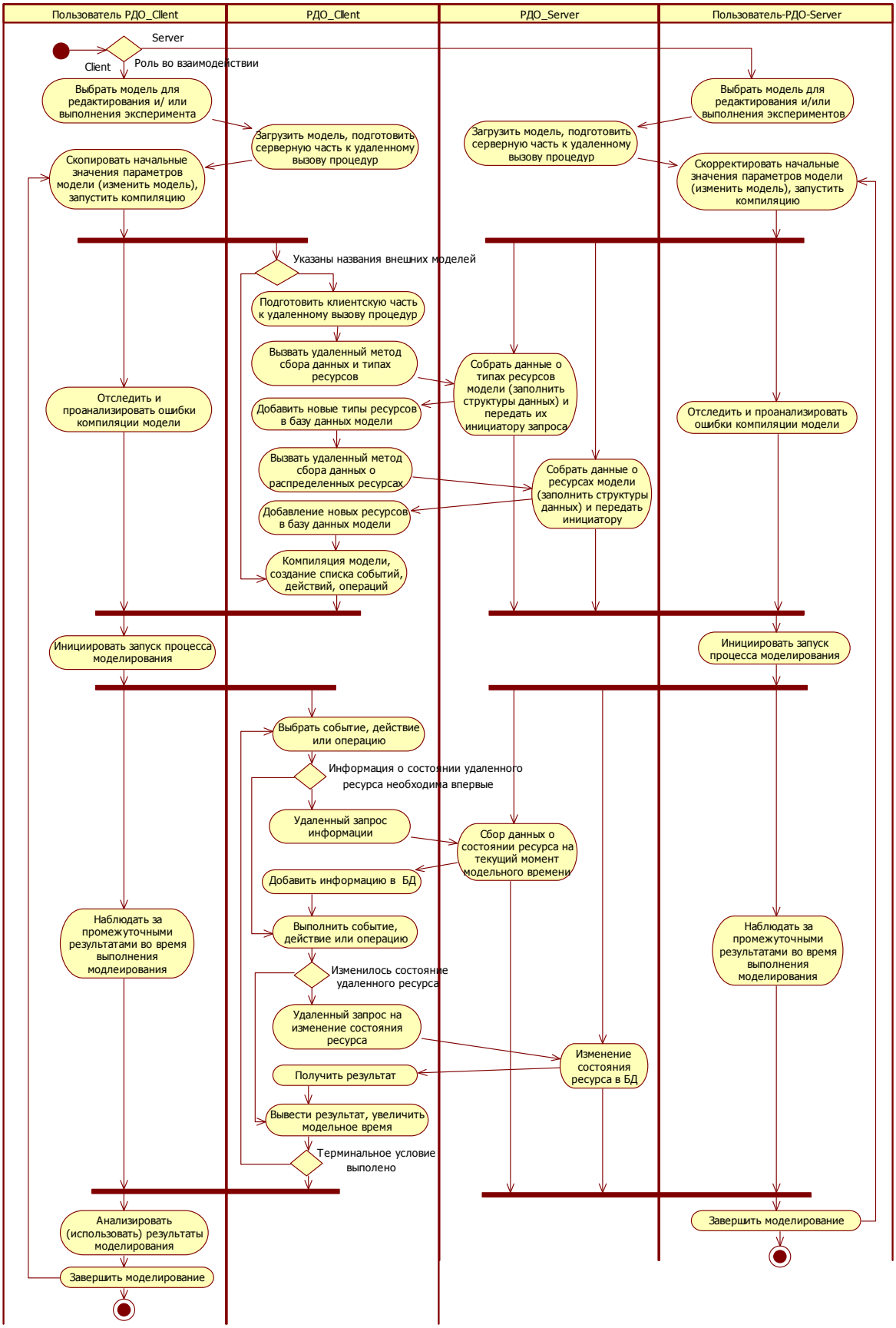

Рисунок 13 - Диаграмма действия. Нотация UML.

На диаграмме представлены действия системы, как в режиме компиляции, так и непосредственно в режиме выполнения модели, в зависимости от выполняемой во взаимодействии роли (клиент или сервер). В данной работе в соответствии с целями достижения практического результата реализация указанного функционала только на этапе компиляции модели является основной задачей.

3.4 Последовательность добавления распределенных ресурсов и их типов в модель на РДО

Для анализа временной упорядоченности событий при программной реализации вышеуказанного функционала системы была разработана диаграмма последовательности в нотации UML (см. Рисунок 14 и лист 8 дипломного проекта). На диаграмме изображено множество объектов и посланные или принятые ими сообщения.

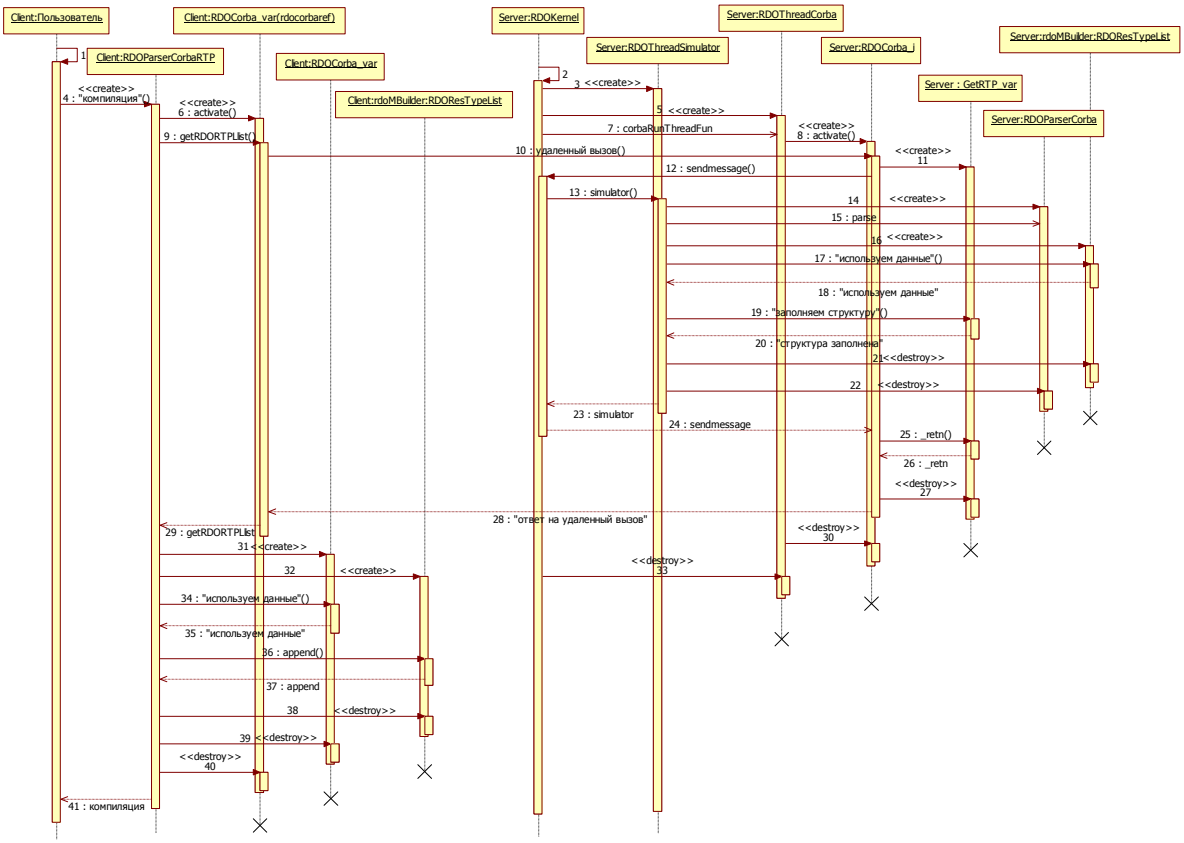

Рисунок 14 - Диаграмма последовательности. Нотация UML.

Диаграмма детально поясняет последовательность работы системы на этапе компиляции модели на РДО с учетом распределенных типов ресурсов (для ресурсов диаграмма отличается только именами объектов). Принадлежность объектов к серверной и клиентской частям реализации системы показана соответствующими ключевыми словами.

Сообщения 1-8 отвечают за подготовку клиент-серверного взаимодействия. Сообщениями 9-29 реализован вызов и отработка удаленного метода по сбору и передаче информации об удаленных типах ресурсов. Сообщениями 31-41 реализовано добавление в собственную модель копий распределенных типов ресурсов на основе полученной о них информации.

3.5 Структура программных средств

3.5.1 Диаграмма компонентов

Для пояснения структурной зависимости компонентов системы РДО и выделения модернизируемых компонентов принято решение о разработке соответствующей диаграммы в нотации UML. Диаграммы компонентов применяются для моделирования статического вида системы с точки зрения реализации (см. Рисунок 15 и лист 9 дипломного проекта).

Поясняя базовый функционал компонентов, следует заметить, что:

1) rdo_kernel реализует ядровые функции системы и не изменяется при разработке системы.
2) RAO-studio.exe реализует графический интерфейс пользователя и не изменяется при разработке системы.
3) rdo_repository управляет потоками данных внутри системы и отвечает за хранение и получение информации.
4) rdo_simulator управляет процессом моделирования на всех его этапах. Он осуществляет координацию и управление

компонентами rdo_runtime и rdo_parser, а так же является интерфейсом имитатора. В рамках данного компонента реализована серверная часть распределенного приложения, в файле rdosimwin.cpp описана реализация интерфейсов (IgetRTP, IgetRSS) удаленного объекта класса RDOCorba_i, более подробно описанных выше.

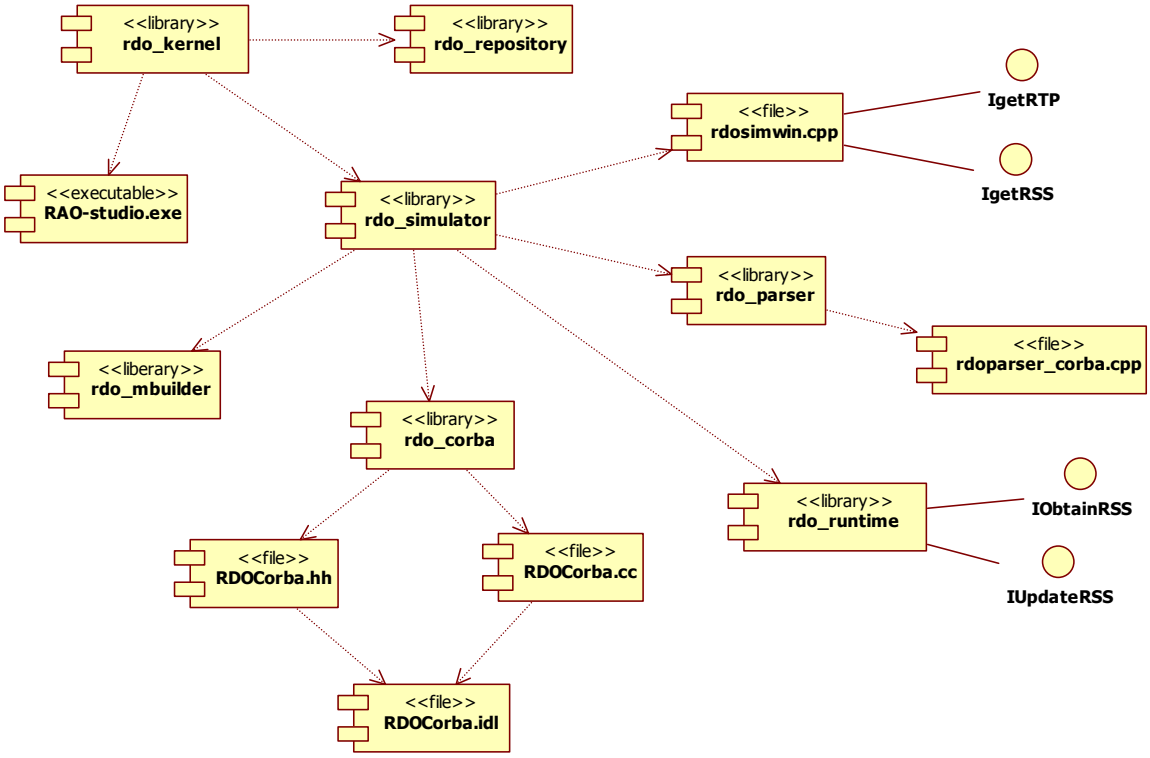

Рисунок 15 - Диаграмма компонентов. Нотация UML.

5) rdo_parser производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО. В рамках данного компонента реализована клиентская часть распределенного приложения, в файле rdoparser_corba.cpp описана подготовка, вызов методов удаленных объектов (отвечающих за сбор необходимой информации) и добавление копии распределенных ресурсов в собственную модель на основе полученной информации в режиме компиляции модели.

6) rdo_mbuilder реализует функционал, используемый для программного управления типами ресурсов и ресурсами модели.
7) rdo_corba реализует функционал технологии CORBA для системы. Разрабатывается для распределенной системы и зависит от описания интерфейсов удаленных объектов (RDOCorba.idl).
8) rdo_runtime отвечает за непосредственное выполнение модели, управление базой данных и базой знаний. Для этого компонента предусмотрены интерфейсы (функции сервера в распределенной системе), используемые в режиме моделирования для получения или изменения информации о состоянии ресурсов по удаленным запросам клиентов (так же реализованных в rdo_runtime) в процессе выполнения распределенной имитационной модели. Однако их практическая реализация не предусмотрена целями данной работы.

3.5.2 Диаграмма развертывания

Для пояснения размещения компонентов системы на аппаратных средствах разработана соответствующая диаграмма UML (см. Рисунок 16 и лист 9 дипломного проекта).

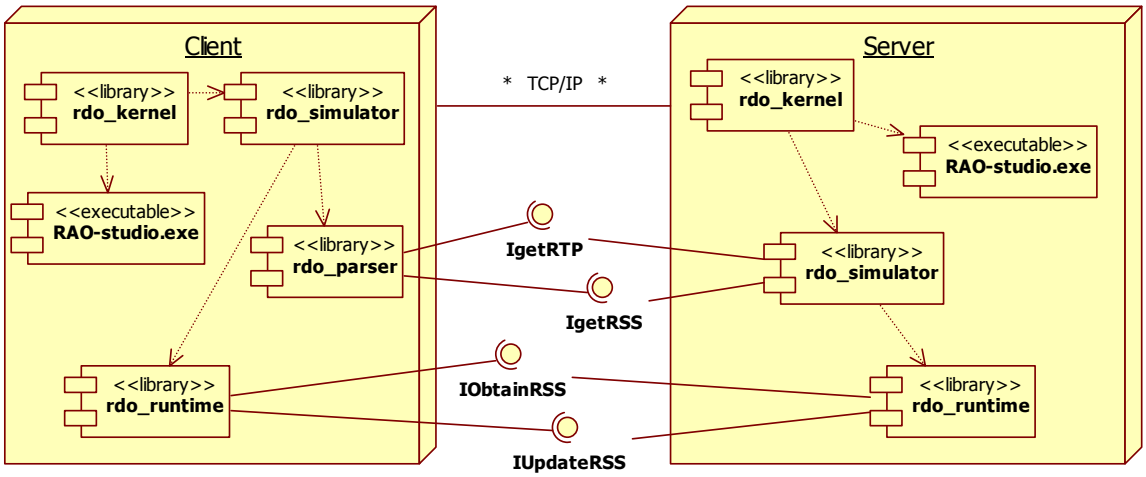

Рисунок 16 - Диаграмма развертывания. Нотация UML.

Диаграммы развертывания применяются для моделирования статического вида системы с точки зрения развертывания.

Из диаграммы видно, каким образом взаимодействуют компоненты системы посредством разрабатываемых интерфейсов. Так же следует заметить, что система функционирует под ОС Windows в сети TCP/IP (показаны те компоненты системы, которые участвуют в межмодельном взаимодействии).

3.6 Алгоритмы подготовки к вызову удаленных методов

3.6.1 Алгоритм подготовки серверной части системы

Для реализации программного кода распределенного приложения необходимо составить алгоритм его работы, причем, отдельно серверного и клиентского приложения. Сначала необходимо составить алгоритм для сервера. В соответствии с документацией на omniORB-4.1.2 [14], спецификацией CORBA [6, 12, 15] и идеей функционирования приложения алгоритм будет следующим (приведен на 10 листе дипломного проекта):

1) Для начала работы необходимо инициализировать брокер объектных запросов, чтобы можно было обращаться к его функциональности в дальнейшем.

2) Необходимо получить ссылку на портируемый объектный адаптер.

3) По полученной ссылке нужно сгенерировать объект poa, к функциональности которого необходимо будет обращаться для управления объектами.

4) Далее требуется создание серванта – серверной подпрограммы, выполняющей CORBA-объект указанного типа.

5) Чтобы пользоваться функциональностью серванта, а соответственно CORBA-объекта, необходимо активизировать созданный сервант.

6) Необходимо теперь получить объектную ссылку на CORBA-объект, реализуемый сервантом, чтобы в дальнейшем связать данный CORBA-объект с определенным контекстом имен.

7) Связываем объектную ссылку на CORBA-объект с контекстом имен «RDO_context\RDO» и именем модели, открытой в данный момент в РДО, model_name, используя которые, клиентская программа сможет получить объектную ссылку реализованного в серверной части приложения CORBA-объекта.

8) Далее должен следовать запуск менеджера POA, который и будет работать с CORBA-объектами.

9) Далее требуется активизировать менеджер POA для запуска обработки клиентских запросов.

10) Также после активизации менеджера POA необходимо сообщить ORB, что он готов принимать запросы клиентов.

11) Далее следует цикл обработки клиентских запросов, который является основной целью работы распределенного приложения.

12) По окончанию работы распределенного приложения работа CORBA в качестве «промежуточного слоя» прекращается.

По такому алгоритму должно в должно работать серверное приложение, чтобы обеспечить указанную в техническом задании функциональность распределенного приложения. Алгоритм представлен на одном из листов дипломного проекта.

3.6.2 Алгоритм подготовки клиентской части системы

В соответствии с документацией на omniORB-4.1.2 [14], спецификацией CORBA [6, 12, 15] и идеей функционирования

приложения алгоритм работы клиентского приложения будет следующим (приведен на 10 листе дипломного проекта):

1) Для начала работы необходимо инициализировать брокер объектных запросов, чтобы можно было обращаться к его функциональности в дальнейшем.

2) Далее необходимо получить ссылку на CORBA-объект необходимого удаленного объекта, для чего используется Сервис именования. Необходимая нам ссылка привязана к контексту «root\RDO1».

3) Теперь для того чтобы пользоваться функциональностью удаленного объекта, необходимо сгенерировать сначала объектную ссылку по ссылке на CORBA-объект.

4) Далее следует выполнение запроса о получении информации обо всех необходимых распределенных типах ресурсов по именам внешних моделей, указанных с помощью ключевого слова external_model во вкладке SMR.

5) Когда запрос успешно обработан и в списке структур для хранения информации о типах ресурсов содержится вся необходимая информация, можно приступать к добавлению новых типов ресурсов в данной среде РДО по полученной информации. Этот процесс начинается со сбора информации о собственных типах ресурсов.

6) Выполняется процесс добавления типов ресурсов.

7) При отсутствии дальнейших клиентских запросов в зависимости от необходимости пользователя брокер объектных запросов деактивируется и клиентское приложение завершается.

По этому алгоритму должно работать клиентское приложение, чтобы обеспечить указанную в техническом задании функциональность распределенного приложения.

3.7 Информационная модель передаваемых данных

Разработка и анализ информационной модели ресурсов РДО необходимы прежде всего для того, чтобы определиться со структурой данных, передаваемой по сети и необходимой для хранения информации о типах ресурсов. Для решения этой задачи было проведено дополнительное обследование системы и документации на систему РДО [2, 3]. В результате была построена информационная модель ресурсов РДО в нотации IDEF1x (см. Рисунок 17 и лист 11 дипломного проекта).

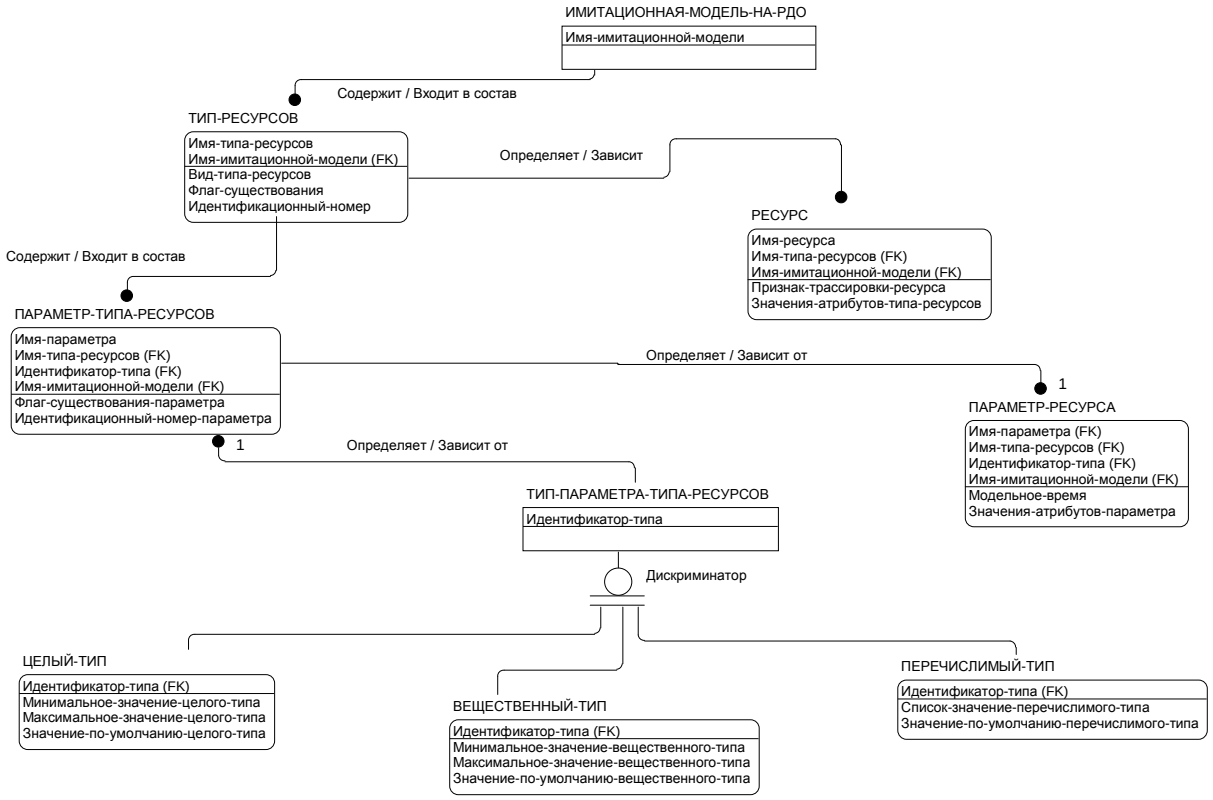

Рисунок 17 - Информационная модель ресурсов РДО. Нотация Idef1x.

База данных имитационной модели на РДО в информационном аспекте представляется собой набор сущностей (объектов) с множеством атрибутов (свойств).

Для решения задач рабочего этапа проектирования необходима информационная модель ресурсов. Она содержит описания атрибутов типов ресурсов, ресурсов и их параметров. На основании этой информации

была построена информационная модель данных, передаваемых по сети, (в нотации IDEF1x) которая также представлена на 11 листе дипломного проекта (см.Рисунок 18). Она включает данные из информационной модели ресурсов РДО и дополнительно содержит структуры данных и поля, необходимые для программной реализации интерфейсов распределенных объектов.

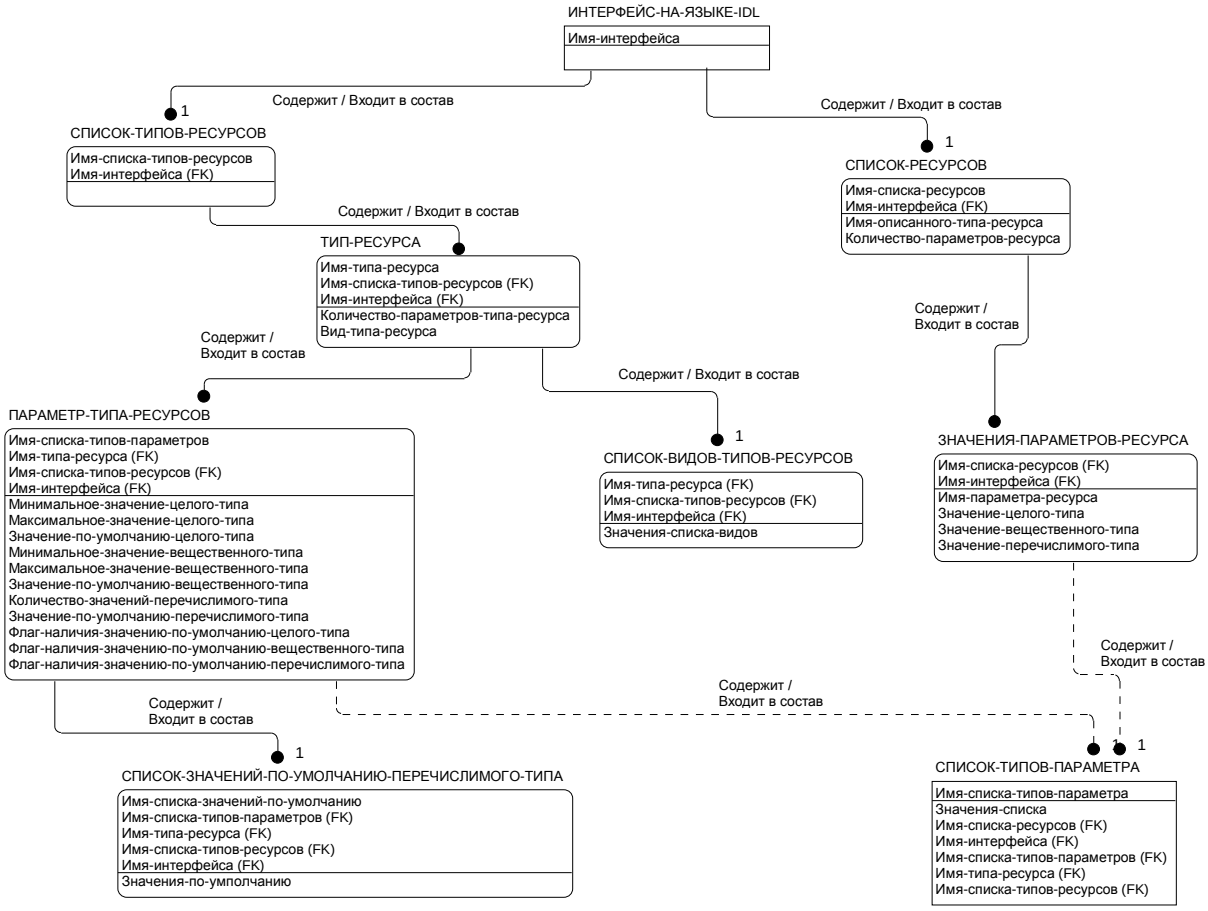

Рисунок 18 - Информационная модель данных, передаваемых по сети. Нотация idef1x.

В дальнейшем на основании этой модели будет разработан интерфейс удаленного объекта, описание которого будет содержать структуры данных, предназначенные для хранения информации о типах ресурсов и параметрах модели на РДО. Эти структуры данных будут передаваться при вызове удаленных методов, которые будут осуществлять сбор необходимой информации о распределенных типах ресурсов.

4 Рабочее проектирование

В соответствии с практическими целями дипломного проекта на этапе рабочего проектирование рассматривается реализация следующей законченной части функционала системы: использование распределенной базы данных ресурсов для компиляции модели на РДО. Последовательность действий на данном этапе по достижению поставленной цели, включая разработку распределенной модели и апробирование системы, приведены ниже.

4.1 Подготовка ПО, реализующего функционал ORB

В соответствии с алгоритмом разработки приложений на основе CORBA необходимо подготовить и настроить программного обеспечение, реализующее функционал выбранной технологии.

Было рассмотрено несколько возможных вариантов такого программного обеспечения. С учетом критерия бесплатного использования для собственных разработок, а также поддержки языка C++, было выделено три варианта ПО для использования в качестве транспотрной среды распределенной системы дискретного имитационного моделирования на основе РДО:

- продукт Orbacus фирмы IONA (http://web.progress.com);
- продукт omniORB компании Apasphere Ltd (http://omniorb.sourceforge.net/);
- продукт TAO компании Distributed Object Computing (DOC) Group (http://www.dre.vanderbilt.edu/).

Среди этих продуктов путем субъективной оценки был выделен omniORB, как наиболее доступный, документированный, постоянно поддерживаемый и обновляемый продукт. Еще одним критерием в пользу него стала простота его установки, настройки и начала использования.

Была выбрана предпоследняя доступная версия программы (как наиболее стабильная) - omniORB -4.1.2 - для операционной системы Windows с учетом использования для программирования приложений языка C++ (Visual C++ 8.0). Для того чтобы получить архив с исходными коды программы, необходимо было посетить сайт http://omniorb.sourceforge.net/index.html и загрузить их, используя соответствующие ссылки.

Далее была настроена конфигурация устанавливаемой системы в соответствии с требованиями файла README.win32.txt, затем настроены переменные окружения в ОС Windows в соответствии с требованиями того же файла.

Первоначальной проверкой функционирования установленного программного обеспечения необходимо было запустить на выполнение с консоли ОС Windows команды:

cd C:\omniORB-4.1.2\bin\x86_win32

omniidl –h (для вывода справки об использовании idl-компилятора).

Если справочное сообщение было выдано на консоль, значит idl-компилятор запускается, и можно приступать к следующему этапу разработки приложения в соответствии с алгоритмом разработки.

В случаях возникновения ошибок приходилось обращаться к указаниям файла README.win32.txt для уточнения факторов, оказывающих влияние на процесс компиляции и установки.

После выбора Naming Service в качестве механизма передачи объектных ссылок на CORBA-объекты, возник вопрос с его установкой и запуском, который был решен с помощью документов README.win32.txt и omniNames.pdf (из состава документации поставщика ПО). Была произведена настройка реестра Windows в соответствии с инструкцией, созданы необходимые для адекватной работы Naming Service переменные среды.

Комплексный анализ разрабатываемого приложения был проведен на концептуальном и техническом этапе дипломного проекта, поэтому дальнейшие действия состояли в описании интерфейса удаленных объектов и программной реализации целевого функционала.

4.2 Описание интерфейса и генерация исходных файлов

Следующий этап разработки приложения – описание интерфейсов распределенных объектов, которое осуществляется в соответствии с правилами IDL (Interface Definition Language – язык описания интерфейсов). В соответствии с ранее разработанной информационной моделью данных, передаваемых по сети, при использовании [12], [14] и [15], был описан интерфейс RDOCorba, который включил список структур данных, предназначенных для хранения всей информации параметрах типов ресурсов, самих ресурсов и несколько методов. Их функциональность сводится к получению информации о необходимых распределенных типах ресурсов и ресурсах в режиме компиляции модели на РДО (в соответствии целями, поставленными на рабочий этап разработки системы).

Листинг файла RDOCorba.idl с подробными комментариями по содержимому приведен в приложении 1.

Далее необходимо в консоли ОС Windows перейти в папку с созданным idl-описанием интерфейса и выполнить следующую команду:

Omniidl -bcxx RDOCorba.idl,

с помощью которой idl-компилятор транслирует файл idl-описаний интерфейсов в файлы-исходники (RDOCorba.cc и RDOCorba.hh), необходимые для реализации распределенного приложения.

Эти файлы включены в отдельную статическую библиотеку rdo_corba.

4.3 Реализация серверного фрагмента кода

Реализацию серверной части приложения будем осуществлять в соответствии с разработанным ранее алгоритмом и последовательностью добавления распределенных ресурсов и их типов в модель. Листинг файла реализации серверной части приложения с комментариями приведен в приложении 2. Укрупненно последовательность действий при разработке серверной части описана ниже.

1) Был создан класс реализации удаленного интерфейса:

```
class RDOCorba_i: public POA_rdoParse::RDOCorba.
```

2) Класс имеет два абсолютно виртуальных метода в соответствии с описанными ранее интерфейсами, предназначенных соответственно для получения информации обо всех имеющихся на этом сервере (описанных в модели) типах ресурсов и ресурсах:

```
virtual rdoParse::RDOCorba::GetRTP*
    getRDORTPlist(::CORBA::Long& rtp_count);

virtual rdoParse::RDOCorba::GetRSS*
    getRDORSSPlist(::CORBA::Long& rss_count);
```

3) Класс имеет статический метод, который реализует связывание удаленного объекта с именем модели:

```
static CORBA::Boolean bindObjectToName(
    CORBA::ORB_ptr orb, CORBA::Object_ptr objref);
```

4) Реализация виртуальных методов сводится к посылу сообщений симулятору и инициированием вызовов методов симулятора, которые непосредственно отвечают за сбор данных о типах ресурсов и ресурсах:

```
void RDOThreadSimulator::corbaGetRTP(
    rdoParse::RDOCorba::GetRTP_var& my_rtpList );

void RDOThreadSimulator::corbaGetRSS(
    rdoParse::RDOCorba::GetRSS_var& my_rssList );
```

Эти методы выполняют компиляцию модели на РДО с последующим заполнением информацией объектов, являющихся входными параметрами.

5) Далее, если это не было сделано ранее, настроим проект для работы с CORBA в соответствии с документацией на ПО [14], добавив в соответствующие проекты необходимые библиотеки, настроив пути к библиотекам и подключаемым файлам. Запустим службу Naming Service, выполнив команду с консоли Windows:

   omninames –always –start;

6) Чтобы обеспечить функциональность приложения, необходимо создать отдельный поток, в котором будет работать брокер объектных запросов, обслуживая заявки клиентов в непрерывном цикле, который можно будет перезапустить только при открытии новой модели в среде РДО:

```
unsigned int RDOThreadCorba::corbaRunThreadFun(
    void* param )
```

7) Действия, выполняемые потоком, соответствуют алгоритму подготовки сервера к вызову методов удаленных объектов:

   1. Для начала работы необходимо инициализировать брокер объектных запросов, чтобы можно было обращаться к его функциональности в дальнейшем:

```
int argc = 0;
g_orb = CORBA::ORB_init(argc, NULL);
```

   2. Необходимо получить ссылку на портируемый объектный адаптер:

```
CORBA::Object_var obj = g_orb->
   resolve_initial_references("RootPOA");
```

   3. По полученной ссылке нужно сгенерировать объект poa, к функциональности которого необходимо будет обращаться для управления объектами:

```
PortableServer::POA_var poa =
```

```
PortableServer::POA::_narrow(obj);
```

4. Далее требуется создание серванта – серверной подпрограммы, выполняющей CORBA-объект указанного типа:

```
RDOCorba_i* myrdocorba = new RDOCorba_i();
```

5. Чтобы пользоваться функциональностью серванта, а соответственно CORBA-объекта, необходимо активизировать созданный сервант::

```
PortableServer::ObjectId_var myrdocorbaid =
    poa->activate_object(myrdocorba);
```

6. Необходимо теперь получить объектную ссылку на CORBA-объект, реализуемый сервантом, чтобы в дальнейшем связать данный CORBA-объект с определенным контекстом имен:

```
obj = myrdocorba->_this();
```

7. Связываем объектную ссылку на CORBA-объект с контекстом имен «RDO_context\RDO» и именем модели `ModelName`, используя которые, клиентская программа сможет получить объектную ссылку реализованного в серверной части приложения CORBA-объекта:

```
if( !bindObjectToName(orb, obj, ModelName) )
    return 1;
```

8. Реализация `bindObjectToName` представлена в приложении 2 и является стандартной с точки зрения механизмов CORBA. В нашем случае важным моментом является сам контекст имен, с которым будет связан наш CORBA-объект, и имя этого объекта.

9. Далее должен следовать запуск менеджера POA, который и будет работать с CORBA-объектами:

```
PortableServer::POAManager_var pman = poa->
    the_POAManager();
```

10. Далее требуется активизировать менеджер POA для запуска обработки клиентских запросов:
```
pman->activate();
```
11. Также после активизации менеджера POA необходимо сообщить ORB, что он готов принимать запросы клиентов:
```
g_orb->run();
```
12. Далее следует цикл обработки клиентских запросов, который является основной целью работы распределенного приложения.

    Вызовы клиентами удаленных методов серванта `myrdocorba`.
13. По окончанию работы распределенного приложения работа CORBA в качестве «промежуточного слоя» прекращается.
```
orb->destroy();
```

Разработка и отладка серверной части приложения осуществлялась с помощью справочных данных, приведенных в документах [14,15,16] и полученных с помощью [13].

4.4 Реализация клиентского фрагмента кода

Реализацию клиентской части приложения будем осуществлять в соответствии с разработанным ранее алгоритмом и последовательностью добавления распределенных ресурсов и их типов в модель. Листинг файла реализации клиентской части приложения с комментариями приведен в приложении 3. Укрупненно последовательность действий при разработке клиентской части описана ниже.

1) Был создан метод, использующий интерфейсы удаленных объектов на этапе компиляции модели и создающий локальные копии распределенных ресурсов и их типов в своей модели:

```
void RDOParserCorbaRTP::parse();
```

2) Действия, выполняемые клиентом, соответствуют алгоритму подготовки клиента вызову методов удаленных объектов:

1. Для начала работы необходимо инициализировать брокер объектных запросов, чтобы можно было обращаться к его функциональности в дальнейшем:

```
int argc = 0;
CORBA::ORB_var orb = CORBA::ORB_init(argc, NULL);
```

2. Далее необходимо получить ссылку на CORBA-объект необходимого удаленного объекта, для чего используется Сервис именования. Необходимая нам ссылка привязана к контексту «RDO_context\RDO» и имени `ObejctName`, указанном в объекте прогона модели с помощью ключевого слова `external_model`:

```
CORBA::Object_var obj=getObjectReference(orb,left);
```

Описание `getObjectReference` является стандартным (с точки зрения механизмов CORBA) и приведено в приложении 3.

3. Теперь для того чтобы пользоваться функциональностью удаленного объекта, необходимо сгенерировать сначала объектную ссылку по ссылке на CORBA-объект.

```
rdoParse::RDOCorba_var rdocorbaref =
    rdoParse::RDOCorba::_narrow(obj);
```

4. Следующим шагом идет вызов удаленного метода, который возвращает информацию обо всех распределенных типах ресурсов:

```
rdoParse::RDOCorba::GetRTP_var tmp_rtp =
    rdocorbaref->getRDORTPlist( rtp_count );
```

```
rdoParse::RDOCorba::GetRTP_var
   my_rtpList(tmp_rtp);
```

5. Далее выполняется создание локальных копии типов ресурсов в соответствии с полученной информацией о распределенных типах ресурсов (подробный код приведен в приложении 3).

6. Следующим шагом идет вызов удаленного метода, который возвращает информацию обо всех распределенных ресурсах:

```
rdoParse::RDOCorba::GetRSS_var tmp_rss =
   rdocorbaref->getRDORSSlist( rss_count );

rdoParse::RDOCorba::GetRSS_var
   my_rssList(tmp_rss);
```

7. Далее выполняется создание локальных копии ресурсов в соответствии с полученной информацией о распределенных ресурсах (подробный код приведен в приложении 3).

8. Наконец, брокер объектных запросов деактивируется:

```
orb->destroy();
```

Разработка и отладка клиентской части приложения осуществлялась с помощью справочных данных, приведенных в документах [14,15,16] и полученных с помощью [13].

4.5 Апробирование системы на примере

Необходимо осуществить проверку функциональности приложения в соответствии с целями и задачами, поставленными на разработку. В данном случае необходимо проверить возможность использования распределенных ресурсов и их типов при компиляции модели на РДО.

4.5.1 Разработка тестового примера распределенной модели

Целью разработки тестового примера является проверка следующей функциональности системы: использование состояния распределенных ресурсов и их типов при компиляции модели на РДО.

В соответствии с этой целью разработаны две модели (склада и цеха), которые используют однотипные ресурсы (накопители и детали). В процессе проверки при компиляции модели цеха будут использоваться ресурсы (накопители и детали), описанные в модели склада.

Схема тестового примера распределенной имитационной модели приведена на 12 листе дипломного проекта (см. Рисунок 19).

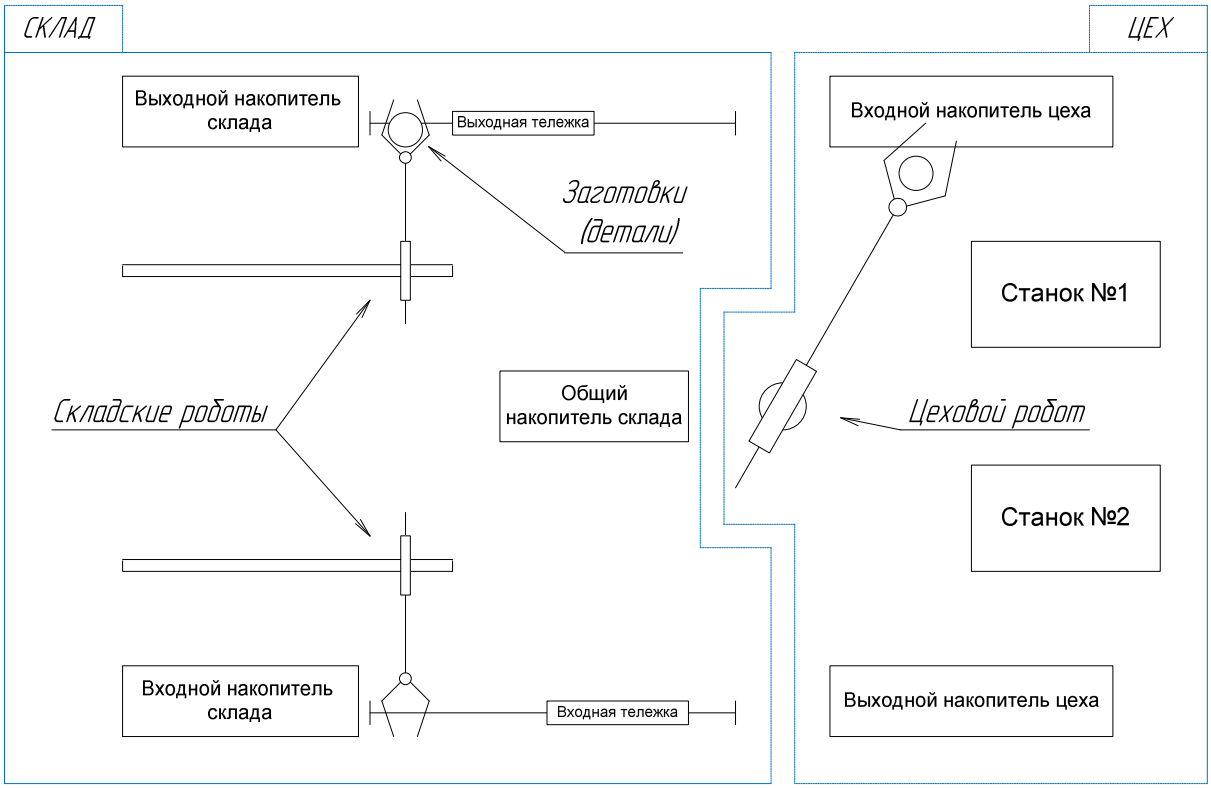

Рисунок 19 - Схема тестового примера распределенной ИМ.

Модель склада функционирует следующим образом: после запуска модели прибытие заготовок в выходной накопитель склада имитируется клавиатурной операцией (пробел), далее при наличии детали и свободной тележки робот выполняет загрузку и возвращается на исходную позицию, затем тележка перемещается к общему накопителю склада, там

самостоятельно разгружается и возвращается обратно. Входная тележка перемещает детали из общего накопителя во входной, а робот помогает ей при разгрузке.

Модель цеха функционирует следующим образом: после запуска модели прибытие заготовок во входной накопитель цеха имитируется клавиатурной операцией (пробел), далее при наличии детали в накопителе робот выполняет погрузку на свободный станок, который начинает обработку, робот возвращается в исходное положение. Когда станок заканчивает обработку, освободившийся робот перемещается к нему для разгрузки и помещает деталь в выходной накопитель цеха.

Обе модели останавливаются, когда все десять позиций их целевых накопителей оказываются заполненными.

Ресурсы моделей изображены выше. Для модели склада это: накопители, детали, тележки и складские роботы. Для модели цеха это: накопители, цеховой робот, станки и детали. С точки зрения межмодельного взаимодействия ресурсы «накопители» и «детали» (и их типы) являются распределенными. Текст моделей склада и цеха на языке РДО приведен соответственно в приложении 4 и приложении 5.

4.5.2 Проведение тестирования

Экранные формы РДО при выполнении тестирования системы приведены на 13 листе дипломного проекта.

Последовательность действий при тестировании следующая:

1) Выполняем настройки операционной системы в соответствии с инструкцией по настройке ПО omniorb-4.1.2 [14] (ранее рассмотрено на этапе рабочего проектирования).
2) Запускаем РДО и загружаем модель склада, переходим на вкладку «RTP» (см. Рисунок 20). Проверяем описание используемых типов ресурсов: Накопители, Детали, Роботы, Тележки.

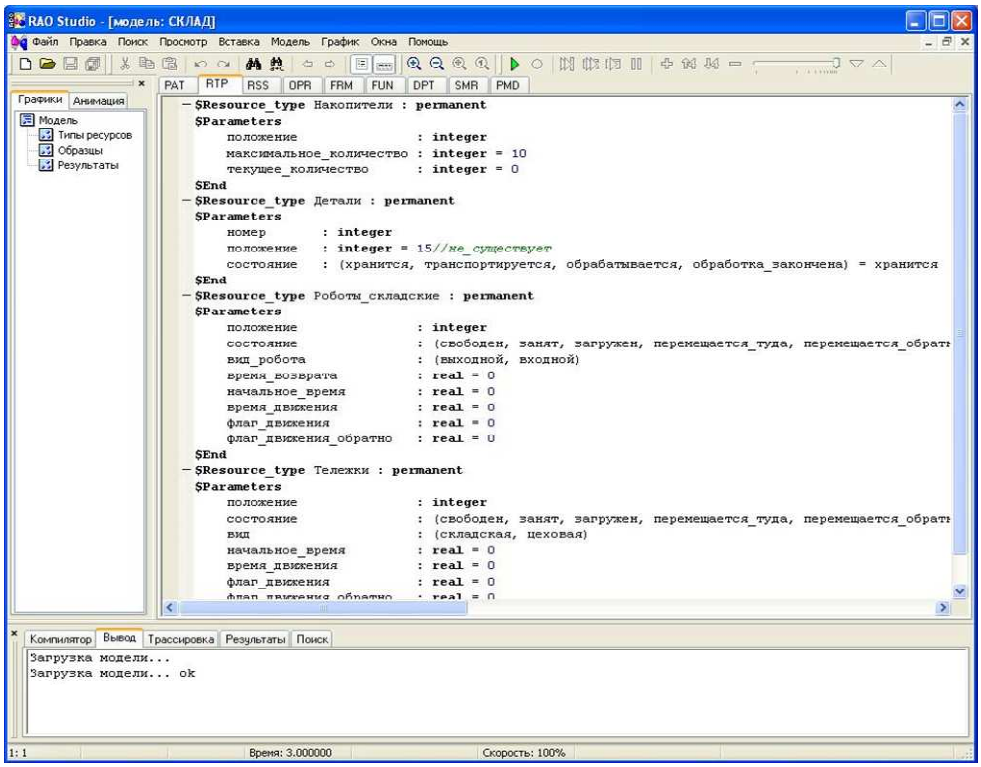

Рисунок 20 - Типы ресурсов модели склада.

3) Переходим на вкладку «RSS» (см. Рисунок 21). Проверяем описание используемых в модели ресурсов.

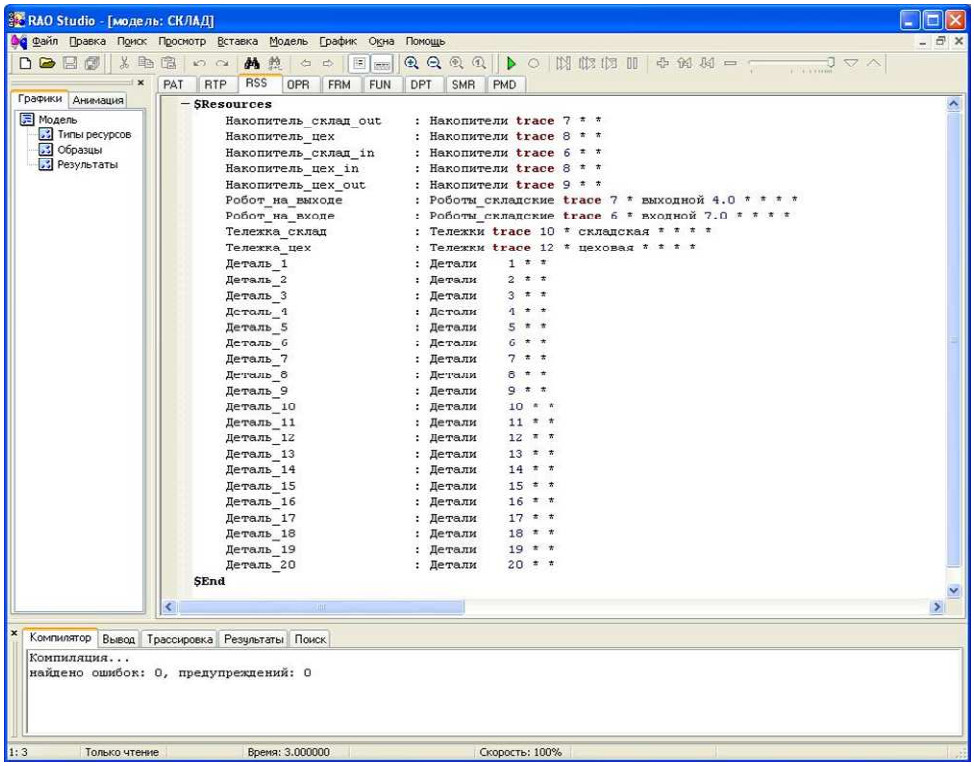

Рисунок 21 - Ресурсы модели склада.

4) Компилируем и запускаем модель (F5), проверяем ее работоспособность (см. Рисунок 22). Получаем необходимые результаты прогона.

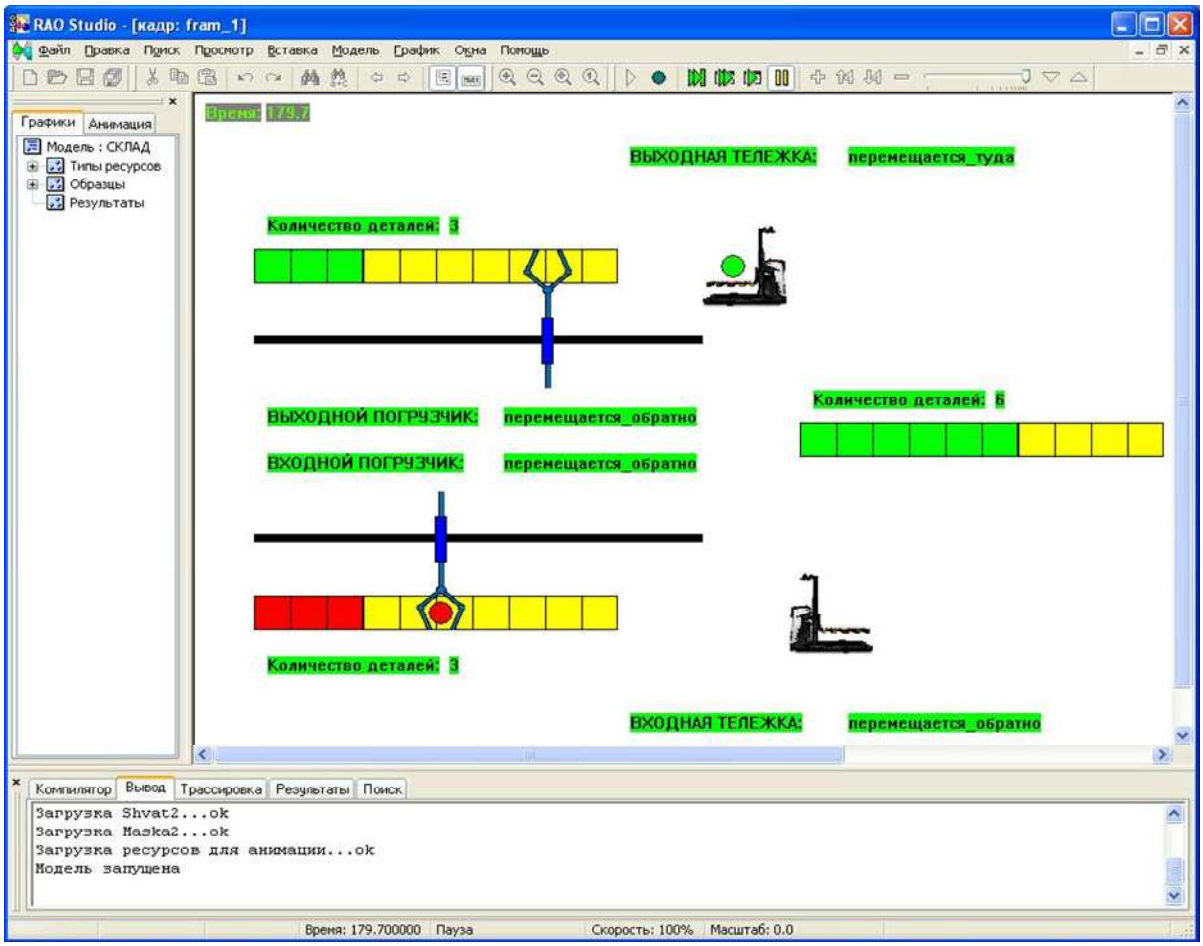

Рисунок 22 - Анимация при выполнении модели склада.

5) Выполняем настройки операционной системы на любом другом компьютере в пределах локальной сети, связанном с сервером (модель Склада), в соответствии с инструкцией по настройке ПО omniorb-4.1.2 [14132] (ранее рассмотрено на этапе рабочего проектирования).
6) Запускаем РДО и загружаем модель цеха, переходим на вкладку «RTP» (см. Рисунок 23). Проверяем описание используемых в модели типов ресурсов: Станки, Роботы_цеховые.

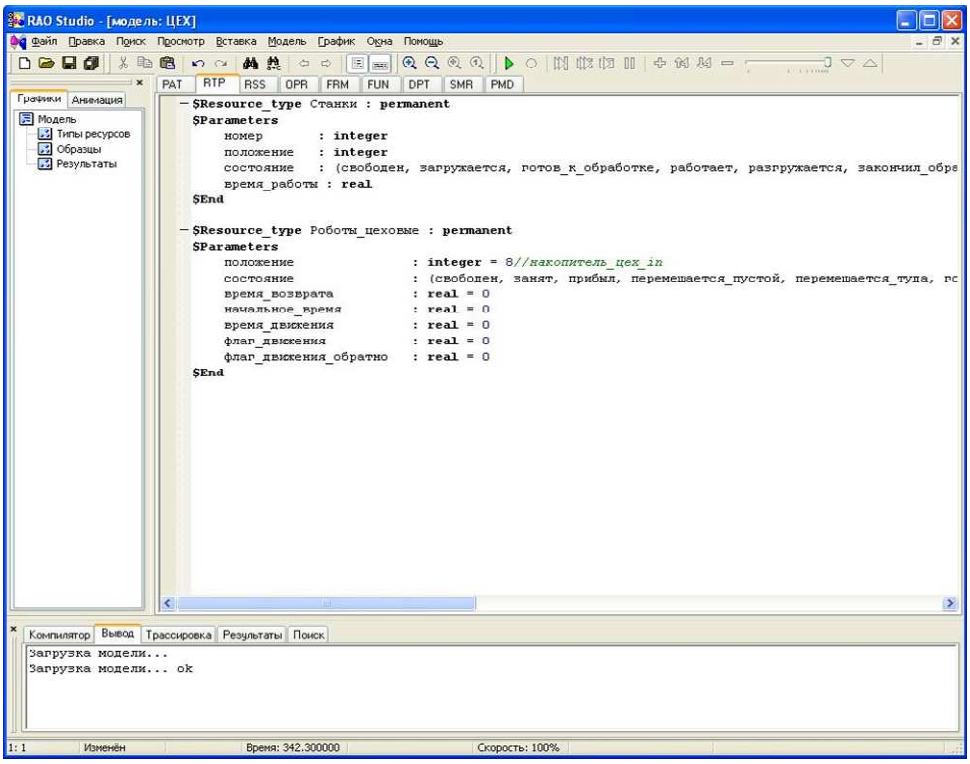

Рисунок 23 - Типы ресурсов модели цеха.

7) Переходим на вкладку «RSS» (см. Рисунок 24). Проверяем описание используемых в модели ресурсов.

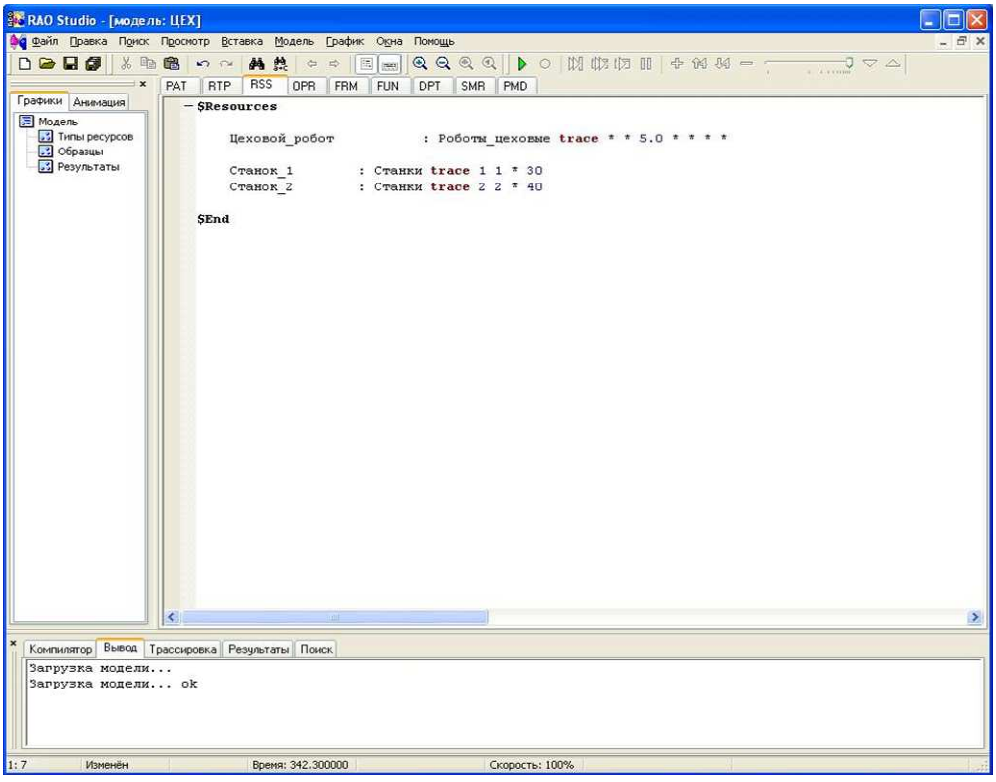

Рисунок 24 - Ресурсы модели цеха.

8) Компилируем и запускаем модель (F5), проверяем ее работоспособность (см. Рисунок 25).

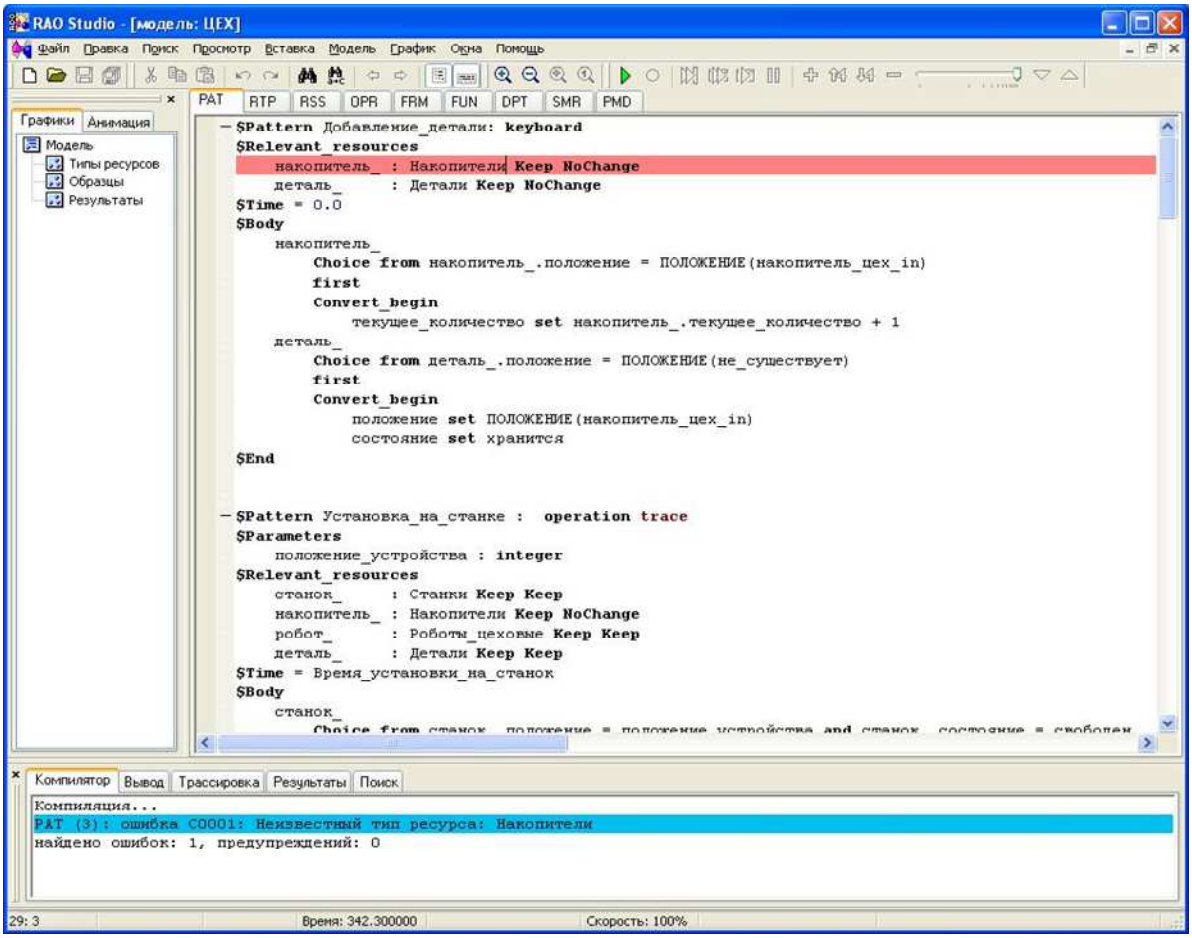

Рисунок 25 – Ошибка компиляции модели цеха.

В результате компиляции система показывает, что для нее неизвестен используемый в модели тип ресурса «Накопители» («Детали»). Однако нам известно, что необходимые нам типы ресурсов и ресурсы (накопители и детали) описаны в модели склада, находящейся в пределах локальной сети.

9) Переходим на вкладку «SMR» и указываем имя внешней модели «СКЛАД» (см. Рисунок 26).

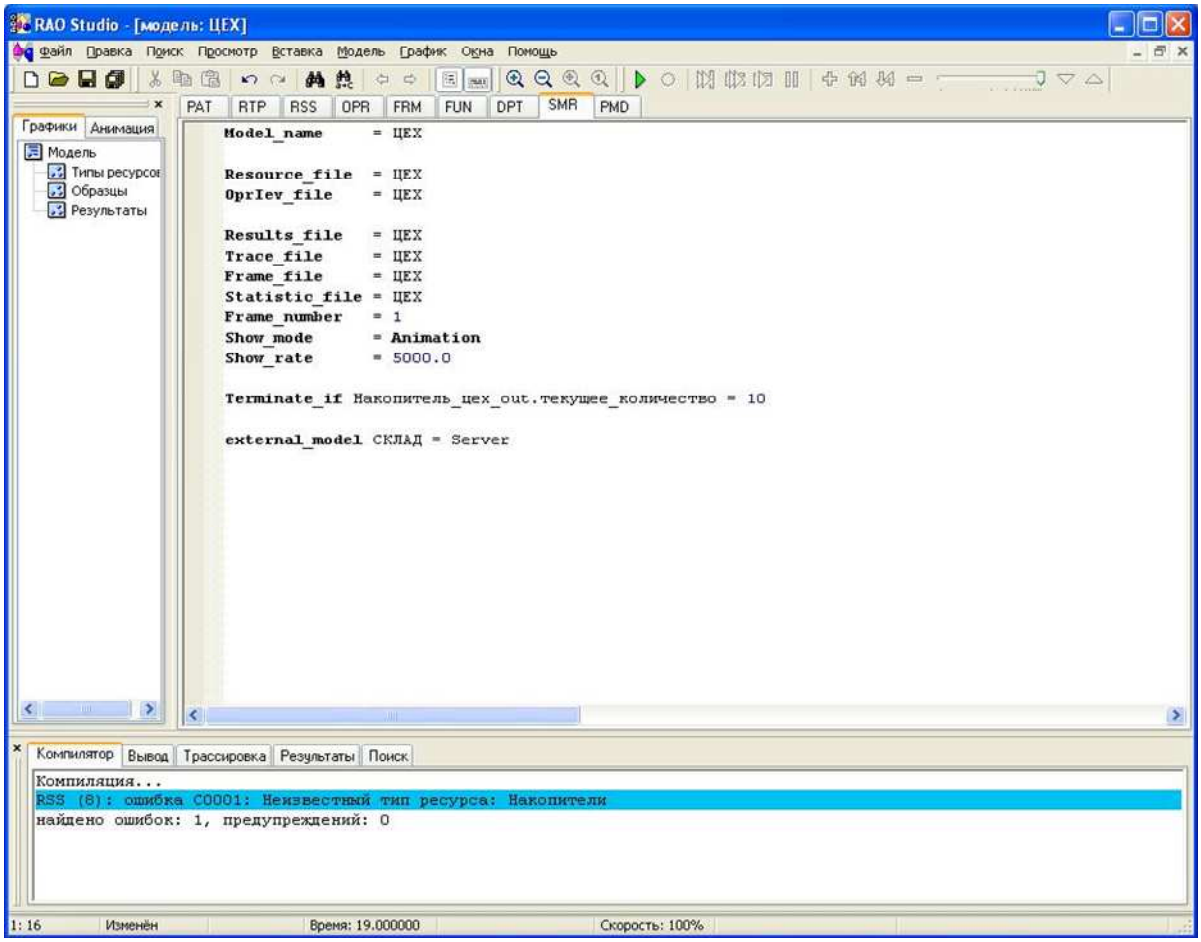

Рисунок 26 - Параметры прогона модели цеха.

10) Компилируем и запускаем модель (F5), проверяем ее работоспособность (см. Рисунок 27). Получаем необходимые результаты прогона.

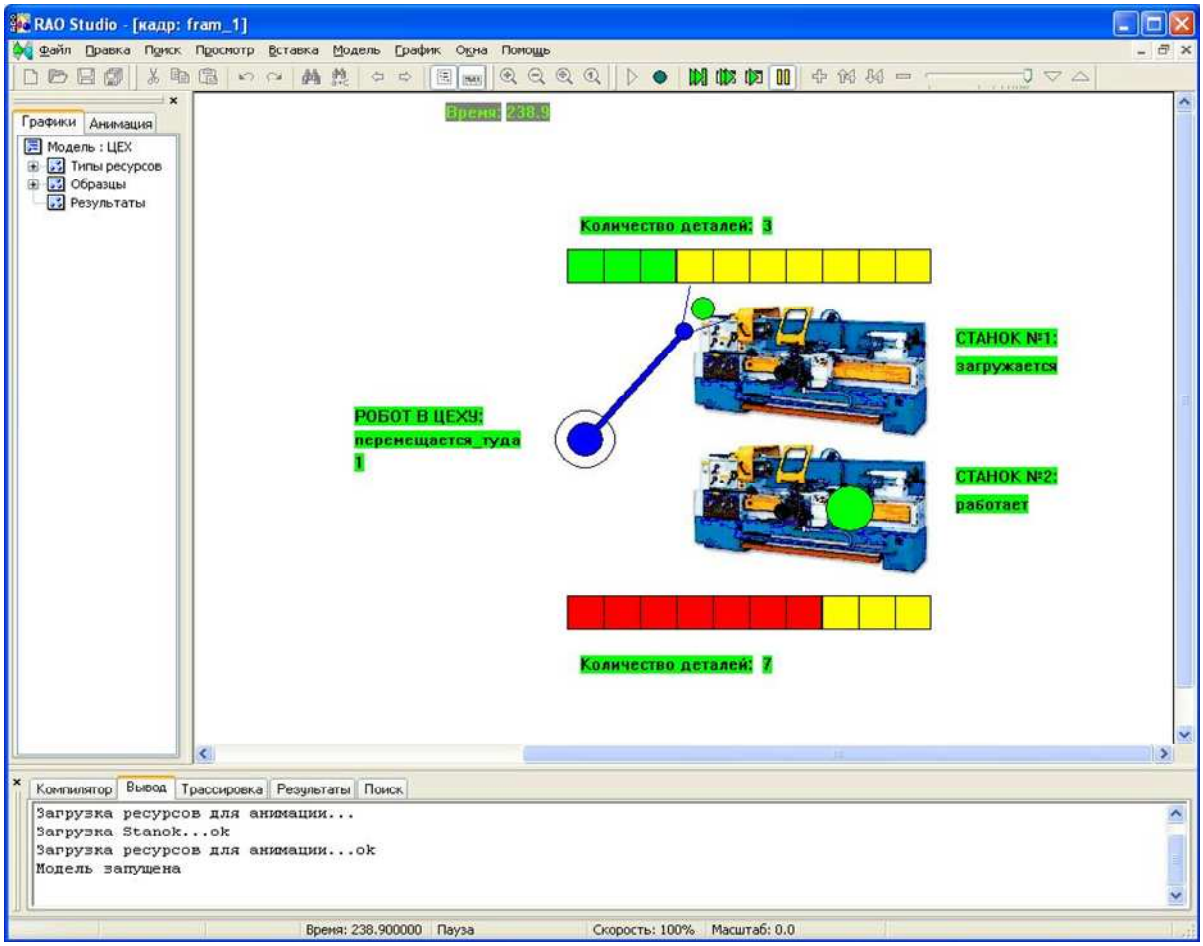

Рисунок 27 - Анимация при выполнении модели цеха.

Так как результаты прогона модели цеха успешно получены, следует считать, что требуемый функционал системы реализован. Система позволяет использовать состояние распределенных ресурсов и их типов при компиляции модели на РДО.

5 Исследовательская часть

5.1 Постановка задачи на исследование

Для ускорения работы распределенной многомодельной системы дискретного имитационного моделирования на основе РДО очень важно по возможности минимизировать загрузку сети. В первом приближении, это означает, что при рассмотрении альтернативных вариантов информационного взаимодействия элементов распределенной системы необходимо добиться уменьшения количества запросов между ними в течение времени выполнения распределенной модели.

Таким образом, необходимо исследовать альтернативные варианты обмена информацией в режиме выполнения моделирования с точки зрения поддержания в системе актуальной информации о состоянии ресурсов и уменьшения при этом суммарного количества запросов, передаваемых по сети, для ее поддержания.

5.2 Выбор метода решения задачи

Решить задачу можно аналитическими методами, так как задача поддается формальному описанию.

5.3 Формализация задачи

Примем для расчета следующие допущения:

1) будем рассматривать состояние единственного ресурса P в системе;
2) в системе присутствует один «сервер», где изначально описан ресурс P;
3) в системе присутствуют n «клиентов», использующих в своей модели состояние ресурса P.

Поддерживать актуальность состояния ресурсов в распределенной системе можно двумя способами.

Схема варианта А обновления параметров распределенных ресурсов в системе представлена на 12 листе дипломного проекта (см. Рисунок 28).

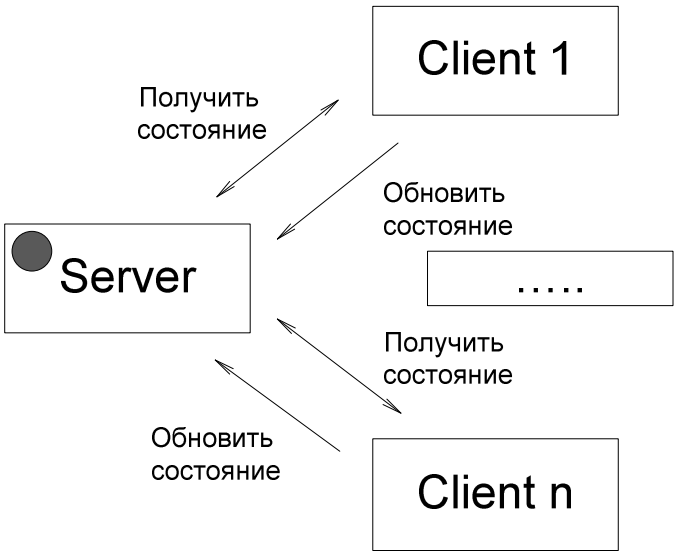

Рисунок 28 – Вариант А обновления состояния ресурсов в распределенной системе.

Этот вариант характеризуется тем, что актуальное состояние ресурса поддерживается на сервере ● (там, где он описан в модели), у которого клиенты запрашивают при необходимости данные (для краткости обозначим это запрос «get»). При обновлении состояния ресурса на клиенте, эти обновления фиксируются на сервере запросом «set».

Схема варианта Б обновления параметров распределенных ресурсов в системе представлена также на 12 листе дипломного проекта (см. Рисунок 29).

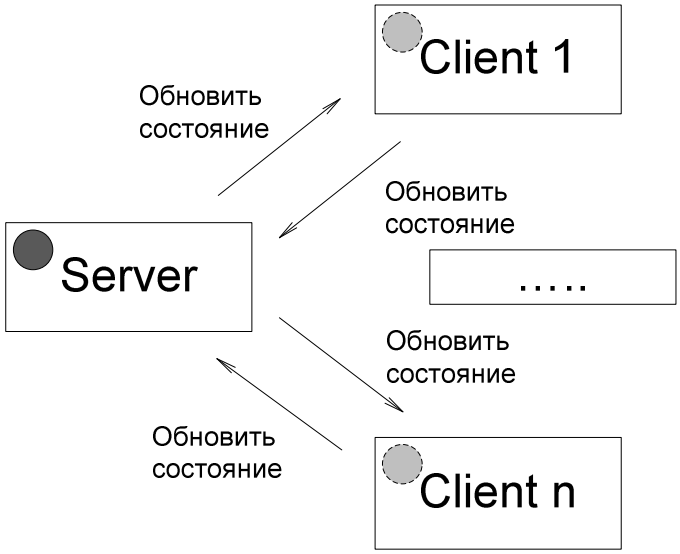

Рисунок 29 – Вариант Б обновления состояния ресурсов в распределенной системе.

Этот вариант характеризуется тем, что актуальное состояние ресурса поддерживается как на сервере ●, так и на клиентах ●, где он используется, с помощью создания локальной копии состояния ресурса. Изменение состояния ресурса в любом месте модели приводит к обновлению состояния на сервере («set»), который транслирует эти изменения на остальные локальные копии ресурсов.

Пусть m_S – количество изменений параметров ресурса, производимых на сервере за время работы модели, m_{CLi} – количество изменений параметров ресурса, производимых на *i*-ом клиенте за время работы модели, k_{CLi} – количество обращений к состоянию ресурса, производимых на *i*-ом клиенте за время работы модели.

5.4 Расчет количества сетевых сообщений

В соответствии с исходными данными и принятыми допущениями рассчитаем общее количество запросов в системе за все время работы модели, которое необходимо для поддержания актуальности состояния ресурса *P* в системе.

Для варианта А.

1) При изменении состояния ресурса на клиентах по сети за время работы модели происходят $\sum_{i=1}^{n} m_{CLi}$ обновлений состояния (set) ресурса на сервере.
2) При использовании каждым клиентом значений параметров ресурса, хранящегося на сервере, за время работы модели происходят $\sum_{i=1}^{n} k_{CLi}$ обращений (get) к серверу.
3) Соответственно всего в сети появляется $\sum_{i=1}^{n} (m_{CLi} + k_{CLi})$ запросов.

Для варианта Б.

1) При изменении состояния ресурса на клиентах по сети за время работы модели происходят $\sum_{i=1}^{n} m_{CLi}$ обновлений состояния (set) ресурса на сервере, каждое из которых инициирует *(n-1)* обновлений оставшихся локальных копий в системе.
2) При изменении состояния ресурса на сервере, он выдает $m_S \cdot n$ обновлений на все локальные копии состояния ресурса.
3) Соответственно всего в сети появляется $m_S \cdot n + \sum_{i=1}^{n} m_{CLi} \cdot (n-1)$ запросов.

Таким образом, решение задачи сводится к сравнению общего количества передаваемых по сети запросов за время работы модели для обоих вариантов: $\sum_{i=1}^{n} (m_{CLi} + k_{CLi})$ и $m_S \cdot n + \sum_{i=1}^{n} m_{CLi} \cdot (n-1)$. Докажем справедливость $\sum_{i=1}^{n} (m_{CLi} + k_{CLi}) > m_S \cdot n + \sum_{i=1}^{n} m_{CLi} \cdot (n-1)$.

Предположим, что количество изменений и обращений к состоянию ресурса на каждом клиенте приблизительно равны, т.е. $m_{CLi} \approx m_{CL}$ и $k_{CLi} \approx k_{CL}$, тогда неравенство имеет вид:

$n \cdot (m_{CL} + k_{CL}) > m_S \cdot n + m_{CLi} \cdot n \cdot (n-1)$, или

$m_{CL} + k_{CL} > m_S + m_{CLi} \cdot (n-1)$, с учетом $m_S \approx m_{CL}$ получаем:

$k_{CL} > m_{CLi} \cdot (n-1)$

Необходимо заметить, что для любой модели $k_{CL} >> m_{CLi}$ ввиду того, что анализ состояния ресурса производится постоянно для каждого момента времени модели, так как от состояния ресурсов зависит выполнение событий, действий и операций, тогда как обновление значений параметров – это результат событий, действий или операций. Для большинства моделей *n* может иметь значение порядка *10 (для очень сложной модели) или менее*. Проверка состояния ресурса производится более чем в *n* раз чаще обновления состояния ресурса.

Получаем, что для принятых допущений рассматриваемое неравенство является верным, т.е. нагрузка на сеть в случае А больше, чем в случае Б.

5.5 Выводы по проведенному исследованию

В результате проведенного исследования выяснилось, что для уменьшения загрузки сети (ускорения работы системы) необходимо хранить локальные копии состояний ресурсов на каждом модуле, который их использует, и соответственно транслировать по всей системе любые их изменения.

6 Организационно-экономическая часть

В этой части производится расчет затрат на разработку «Распределенной многомодельной системы дискретного имитационного моделирования на основе РДО»

6.1 Организация и планирование процесса разработки программного продукта

Разработка «Распределенной многомодельной системы дискретного имитационного моделирования на основе РДО» является сложным и длительным процессом, требующим выполнения большого числа разнообразных операций.

Организация и планирование процесса разработки программного продукта или программного комплекса при традиционном методе планирования предусматривает выполнение следующих работ:

- формирование состава выполняемых работ и группировка их по стадиям разработки;
- расчет трудоемкости выполнения работ;
- определение продолжительности выполнения отдельных этапов разработки;
- построение календарного графика выполнения разработки;
- контроль выполнения календарного графика.

Ниже приведен перечень стадий и состава работ (Таблица 2) при создании ПП в соответствии с таблицей 1 из [17].

Таблица 2 - Укрупненный состав работ по стадиям разработки программного продукта.

Стадия разработки ПП	Состав выполняемых работ
Техническое задание	Постановка задач, выбор критериев эффективности. Разработка технико-экономического обоснования разработки. Определение состава пакета прикладных программ, состава и структуры информационной базы. Выбор языков программирования. Предварительный выбор методов выполнения работы. Разработка календарного плана выполнения работ
Эскизный проект	Предварительная разработка структуры входных и выходных данных. Разработка общего описания алгоритмов реализации решения задач. Разработка пояснительной записки. Консультации разработчиков постановки задач. Согласование и утверждение эскизного проекта.
Технический проект	Разработка алгоритмов решения задач. Разработка пояснительной записки. Согласование и утверждение технического проекта. Разработка структуры программы. Разработка программной документации и передача ее для включения в технический проект. Уточнение структуры, анализ и определение формы представления входных и выходных данных. Выбор конфигурации технических средств.
Рабочий проект	Комплексная отладка задач и сдача в опытную эксплуатацию. Разработка проектной документации. Программирование и отладка программ. Описание контрольного примера. Разработка программной документации. Разработка, согласование программы и методики испытаний. Предварительное проведение всех видов испытаний.

Внедрение	Подготовка и передача программной документации для сопровождения с оформлением соответствующего Акта. Передача программной продукции в фонд алгоритмов и программ. Проверка алгоритмов и программ решения задач, корректировка документации после опытной эксплуатации программного продукта.

Трудоемкость разработки программной продукции зависит от ряда факторов, основными из которых являются следующие: степень новизны разрабатываемого программного комплекса, сложность алгоритма его функционирования, объем используемой информации, вид ее представления и способ обработки, а также уровень используемого алгоритмического языка программирования. Чем выше уровень языка, тем трудоемкость меньше.

По степени новизны разрабатываемый проект относится к группе новизны А – разработка программной продукции, требующих использования принципиально новых методов их создания, проведение НИР и т.п.

По степени сложности алгоритма функционирования проект относится к 1 группе сложности - программная продукция, реализующая оптимизационные и моделирующие алгоритмы.

Используемая информация представлена в виде переменной информации – описание имитационной модели на языке РДО.

По виду представления исходной информации программный продукт относится к группе 11 - исходная информация представлена в форме документов, имеющих различный формат и структур, а также к группе 22 – требуется вывод на печать одинаковых документов.

Планируемый срок разработки системы - 1 год.

Трудоемкость разработки системы $\tau_{ПП}$ может быть определена как сумма величин трудоемкости выполнения отдельных стадий разработки «Распределенной многомодельной системы дискретного имитационного моделирования на основе РДО» из выражения:

$$\tau_{ПП} = \tau_{ТЗ} + \tau_{ЭП} + \tau_{ТП} + \tau_{РП} + \tau_{В}, \quad \text{где} \qquad (1)$$

$\tau_{ТЗ}$ – трудоёмкость разработки технического задания на систему;

$\tau_{ЭП}$ – трудоёмкость разработки эскизного проекта системы;

$\tau_{ТП}$ – трудоёмкость разработки технического проекта системы;

$\tau_{РП}$ – трудоёмкость разработки рабочего проекта системы;

$\tau_{В}$ – трудоёмкость внедрения разработанной системы.

6.1.1 Расчет трудоемкости разработки технического задания

$$\tau_{ТЗ} = T^{З}_{РЗ} + T^{З}_{РП}, \text{ где} \qquad (2)$$

$T^{З}_{РЗ}$ – затраты времени разработчика постановки задач на разработку ТЗ, *чел. дни*;

$T^{З}_{РП}$ – затраты времени разработчика программного обеспечения на разработку ТЗ, *чел. дни*.

Значения $T^{З}_{РЗ}$ и $T^{З}_{РП}$ рассчитывают по формулам:

$$T^{З}_{РЗ} = t_{З} \cdot K^{З}_{РЗ};\ T^{З}_{РП} = t_{З} \cdot K^{З}_{РП}, \text{ где}$$

$t_{З}$ – норма времени на разработку ТЗ на программный продукт в зависимости от функционального назначения и степени новизны разрабатываемого продукта, *чел. дни* (таблица 2 из [17]);

$K^{З}_{РЗ}$ – коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задач на стадии ТЗ;

$K^3_{РП}$ – коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком программного обеспечения на стадии ТЗ.

Для разрабатываемой системы (в соответствие с [17]):

$t_З$ = *79 чел. дней*, т.к. система относится к группе новизны А и решает задачи перспективного планирования (моделирование систем);

$K^3_{РЗ}$ = *0,65*, а $K^3_{РП}$ = *0,35*, так как разработчик постановки задач и разработчик ПП работают над проектом совместно.

Получаем трудоемкость разработки технического задания в соответствии с формулой (2):

$\tau_{ТЗ} = T^3_{РЗ} + T^3_{РП}$ = *79 · (0,65 + 0,35) =79 (чел. дней).*

6.1.2 Расчет трудоемкости разработки эскизного проекта

$\tau_{ЭП} = T^3_{РЗ} + T^3_{РП}$, где (3)

$T^3_{РЗ}$ – затраты времени разработчика постановки задач на разработку ЭП, *чел. дни*;

$T^3_{РП}$ – затраты времени разработчика программного обеспечения на разработку ЭП, *чел. дни*.

Значения $T^3_{РЗ}$ и $T^3_{РП}$ рассчитывают по формулам:

$T^3_{РЗ} = t_Э \cdot K^3_{РЗ}$; $T^3_{РП} = t_Э \cdot K^3_{РП}$, где

$t_З$ – норма времени на разработку ЭП программного продукта в зависимости от функционального назначения и степени новизны разрабатываемого продукта, *чел. дни* (таблица 3 из [17]);

$K^3_{РЗ}$ – коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задач на стадии ЭП;

$K^3_{РП}$ – коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком программного обеспечения на стадии ЭП.

Для разрабатываемой системы (в соответствии с [17]):

$t_Э$ = *175 чел. дней,* т.к. система относится к группе новизны А и решает задачи перспективного планирования (моделирование систем);

$K^3_{РЗ}$ = *0,7,* а $K^3_{РП}$ = *0,3,* так как разработчик постановки задач и разработчик ПП работают над проектом совместно.

Получаем трудоемкость разработки эскизного проекта по формуле (3):

$$\tau_{ЭП} = T^3_{РЗ} + T^3_{РП} = 175 \cdot (0{,}7 + 0{,}3) = 175 \text{ (чел. дней).}$$

6.1.3 Расчет трудоемкости разработки технического проекта

$$\tau_{ТП} = (t^T_{РЗ} + t^T_{РП}) \cdot K_В \cdot K_Р \text{, где} \qquad (4)$$

$t^T_{РЗ}$ и $t^T_{РП}$ – нормы времени, затрачиваемого на разработку ТП разработчиком постановки задач и разработчиком программного обеспечения соответственно, *чел. дни* (таблицы 4-16 из [17]);

$K_В$ – коэффициент учёта вида используемой информации;

$K_Р$ – коэффициент учёта режима обработки информации (таблица 17 [1]).

Определим значение коэффициента $K_В$ из выражения:

$$K_В = \frac{K_П \cdot n_П + K_{НС} \cdot n_{НС} + K_Б \cdot n_Б}{n_П + n_{НС} + n_Б} \text{, где} \qquad (5)$$

$K_П, K_{НС}, K_Б$ – значения коэффициентов учёта вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно (таблица 18 из [17]);

$n_П, n_{НС}, n_Б$ – количество наборов данных переменной, нормативно-справочной информации и баз данных соответственно.

Для разрабатываемой системы (в соответствие с таблицами 4-18 из [17]):

$t^T_{РЗ}$ = *125 чел. дней* и $t^T_{РП}$ = *36 чел. дней*, так как система решает задачи перспективного планирования и имеет: 9 форм входной информации – текст имитационной модели на языке РДО в 9 текстовых файлах; 3 формы выходной информации - таблицы результатов моделирования, графики, анимация.

$K_Р$ = *1,67*, так как группа новизны - А, а режим обработки информации - в реальном времени.

Для группы новизны А:

$K_П$ = *1,7*; $K_{НС}$ = *1,45*; $K_Б$ = *4,37*.

$n_П$ = *9*; $n_{НС}$ = *0*; $n_Б$ = *0*.

Подставляя известные величины в формулу (5), получаем значение следующего коэффициента:

$K_В$ = *1,7*

Получаем трудоемкость разработки технического проекта в соответствии с формулой (4):

$$\tau_{ТП} = (t^T_{РЗ} + t^T_{РП}) \cdot K_В \cdot K_Р = (125 + 36) \cdot 1{,}7 \cdot 1{,}67 = 457 \text{ (чел. дней)}.$$

6.1.4 Расчет трудоемкости разработки рабочего проекта

$$\tau_{ТП} = K_К \cdot K_Р \cdot K_Я \cdot K_З \cdot K_{ИА} \cdot (t^P_{РЗ} + t^P_{РП}) \qquad (6)$$

$K_К$ – коэффициент учета сложности контроля информации (таблица 19 [1]);

$K_Р$ – коэффициент учёта режима обработки информации (таблица 17 [1]).

$K_Я$ – коэффициент учета уровня используемого алгоритмического языка программирования (таблица 20 из [17]);

$K_З$ – коэффициент учета степени использования готовых программных модулей (таблица 21 из [17]);

$K_{ИА}$ – коэффициент учета вида используемой информации и сложности алгоритма программного продукта;

$t_{РЗ}^{Р}$ и $t_{РП}^{Р}$ – норма времени, затраченного на разработку рабочего проекта на алгоритмическом языке высокого уровня разработчиком постановки задач и разработчиком программного обеспечения, *чел. дни* (таблицы 23-35 [1]);

$$K_{ИА} = \frac{K'_{П} \cdot n_{П} + K'_{НС} \cdot n_{НС} + K'_{Б} \cdot n_{Б}}{n_{П} + n_{НС} + n_{Б}}, \text{ где} \qquad (7)$$

$K'_{П}, K'_{НС}, K'_{Б}$ – значения коэффициентов учета сложности алгоритма программного продукта и вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно (таблица 22 из [17]);

$n_{П}, n_{НС}, n_{Б}$ – количество наборов данных переменной, нормативно-справочной информации и баз данных соответственно.

Для разрабатываемой системы (в соответствие с таблицами 19-35 из [17]):

$K_К$ = *1,07*, так как степень сложности контроля входной информации – 11, а степень сложности контроля выходной информации – 22;

$K_P = 1,67$, так как группа новизны - А, а режим обработки информации - в реальном времени.

$K_Я = 1,0$, так как среда объектно-ориентированного программирования является языком высокого уровня;

$K_З = 0,8$, так как степень использования готовых программных модулей не превышает 20-25%;

Для группы новизны А и степени сложности алгоритма 1:

$K'_П = 2,27;\ K'_{НС} = 1,36;\ K'_Б = 1,14.$

$n_П = 9;\ n_{НС} = 0;\ n_Б = 0.$

Подставляя известные величины в формулу (7), получаем:

$K_{ИА} = 2,27$

$t^P_{РЗ} = 41$ *чел. день* и $t^P_{РП} = 221$ *чел. день*, так как система решает задачи перспективного планирования и имеет: 9 форм входной информации – текст имитационной модели на языке РДО в 9 текстовых файлах; 3 формы выходной информации - таблицы результатов моделирования, графики, анимация.

Получаем трудоемкость разработки рабочего проекта в соответствии с формулой (6):

$\tau_{ТП} = K_К \cdot K_Р \cdot K_Я \cdot K_З \cdot K_{ИА} \cdot (t^P_{РЗ} + t^P_{РП}) = 1,07 \cdot 1,67 \cdot 1,0 \cdot 0,8 \cdot 2,27 *$

$*(41+221) = 850$ *(чел. дней).*

6.1.5 Расчет трудоемкости внедрения

$$\tau_В = (t^B_{РЗ} + t^B_{РП}) \cdot K_К \cdot K_Р \cdot K_З \qquad (8)$$

$t^B_{РЗ}$ и $t^B_{РП}$ – норма времени, затрачиваемого разработчиком постановки задач и разработчиком программного обеспечения соответственно

на выполнение процедур внедрения программного продукта, *чел. дни* (таблицы 36-48 [17]);

K_K – коэффициент учета сложности контроля информации (таблица 19 [17]);

K_P – коэффициент учёта режима обработки информации (таблица 17 [17]).

$K_З$ – коэффициент учета степени использования готовых программных модулей (таблица 21 из [17]);

Для разрабатываемой системы (в соответствие с таблицами 36-48 из [17]):

$K_K = 1,07$, так как степень сложности контроля входной информации – 11, а степень сложности контроля выходной информации – 22;

$K_P = 1,67$, так как группа новизны - А, а режим обработки информации - в реальном времени.

$K_З = 0,8$, так как степень использования готовых программных модулей не превышает 20-25%;

$t^B_{РЗ} = 41$ *чел. день* и $t^B_{РП} = 44$ *чел. дня*, так как система решает задачи перспективного планирования и имеет: 9 форм входной информации – текст имитационной модели на языке РДО в 9 текстовых файлах; 3 формы выходной информации - таблицы результатов моделирования, графики, анимация.

Получаем трудоемкость внедрения в соответствии с формулой (6) равна:

$$\tau_B = (t^B_{РЗ} + t^B_{РП}) \cdot K_K \cdot K_P \cdot K_З = (41+44) \cdot 1,07 \cdot 1,67 \cdot 0,8 = 121 \text{(чел. день)}$$

Общая трудоёмкость разработки «Распределенной многомодельной системы дискретного имитационного моделирования на основе РДО» в соответствие с формулой (1) составляет:

$\tau_{ПП} = \tau_{ТЗ} + \tau_{ЭП} + \tau_{ТП} + \tau_{РП} + \tau_{В} = 79 + 175 + 457 + 850 + 121 =$

$=1682$ *(чел. дня)*

6.1.6 Определение продолжительности выполнения работ по этапам разработки

Продолжительность выполнения всех работ по этапам разработки определяется из выражения

$$T_i = \frac{\tau_i + Q}{n_i}$$, где (9)

τ_i – трудоемкость *i*-ой работы, *чел. дни*;

Q – трудоемкость дополнительных работ, выполняемых исполнителем, *чел. дни*;

n_i – количество исполнителей, выполняющих *i*-ую работу, *чел.*

Расчёты производились для группы разработчиков. Группа состоит из восьми человек. Четыре человека занимаются разработкой технического задания и эскизного проекта. Это люди с наивысшей квалификацией, способные оценивать возникающие альтернативные варианты проектов и выбрать оптимальный. Другие четверо разрабатывают технический и рабочий проекты. Они должны быть специалистами по математическим и техническим методам решения задач, по кодированию алгоритмов, полученных на стадии технического проектирования. Внедрением занимаются четыре человека – по два из каждой группы.

Продолжительности этапов работ в соответствии с формулой (9):

$T_{ТЗ} = 20(дней)$; $T_{ЭП} = 44(дней)$; $T_{ТП} = 114(дней)$

$T_{РП} = 212\,(дней)$; $T_{В} = 30\,(дней)$

Общее время выполнения проекта:

$$T_{ПП} = T_{ТЗ} + T_{ЭП} + T_{ТП} + T_{РП} + T_{В} = 420\,(дней)$$

6.2 Определение затрат на создание системы

6.2.1 Затраты на оплату труда

В данную статью включаются заработная плата всех исполнителей, непосредственно занятых разработкой данного программного продукта с учетом их должностных окладов и времени участия. Расчет проводится по формуле:

$$C_{ЗО} = \sum_{i} \frac{З_i}{d} \cdot \tau_i \text{, где}$$

$З_i$ – среднемесячный оклад *i*-го исполнителя, *руб.*;

d – среднее количество рабочих дней в месяце, $d = 21$;

τ_i – трудоемкость работ, выполняемых *i*-ым исполнителем, *чел. дни.*

Расчет затраты на оплату труда каждого исполнителя:

$C_{ЗО1} = 25000 \cdot (T_{ТЗ} + T_{ЭП})/21 = 76190$ *(руб.)*

$C_{ЗО2} = 25000 \cdot (T_{ТЗ} + T_{ЭП})/21 = 76190$ *(руб.)*

$C_{ЗО3} = 25000 \cdot (T_{ТЗ} + T_{ЭП} + T_{В})/21 = 111905$ *(руб.)*

$C_{ЗО4} = 25000 \cdot (T_{ТЗ} + T_{ЭП} + T_{В})/21 = 111905$ *(руб.)*

$C_{ЗО5} = 20000 \cdot (T_{ТП} + T_{РП})/21 = 310476$ *(руб.)*

$C_{ЗО6} = 20000 \cdot (T_{ТП} + T_{РП})/21 = 310476$ *(руб.)*

$C_{ЗО7} = 20000 \cdot (T_{ТП} + T_{РП} + T_{В})/21 = 339048$ *(руб.)*

$C_{ЗО8} = 20000 \cdot (T_{ТП} + T_{РП} + T_{В})/21 = 339048$ *(руб.)*

Общая заработная плата равна:

$C_{ЗО} = 1675238$ *(руб.)*

В данной статье также учитываются выплаты непосредственным исполнителям за время, не проработанное на производстве, в том числе: оплата очередных отпусков, компенсация за недоиспользованный отпуск, оплата льготных часов подросткам и др. Дополнительная заработная плата рассчитывается по формуле:

$$C_{ЗД} = C_{ЗО} \cdot \alpha_{Д}, \text{ где} \qquad (10)$$

$\alpha_{Д}$ – коэффициент отчислений на дополнительную зарплату.

С учетом $\alpha_{Д} = 0{,}2$ получаем по формуле (10):

$C_{ЗД} = 335047$ *(руб.)*

6.2.2 Отчисления на социальное страхование

В статье учитываются отчисления в бюджет социального страхования по установленному законодательством тарифу от суммы основной и дополнительной заработной платы. Расчет производится следующим образом:

$$C_{СС} = \alpha_{СС} \cdot (C_{ЗО} + C_{ЗД}), \text{ где} \qquad (11)$$

$\alpha_{СС}$ – коэффициент отчислений на социальное страхование:

Подставляя ранее полученные значения величин и $\alpha_{СС} = 0{,}266$ (Единый социальный налог и страхование от несчастных случаев) в формулу (11), получаем:

$C_{СС} = \alpha_{СС} \cdot (C_{ЗО} + C_{ЗД}) = 0{,}266 \cdot (1675238 + 335047) = 534735$ *(руб.)*

6.2.3 Амортизационные отчисления

В элементе "Амортизация основных фондов" отражается сумма амортизационных отчислений на полное восстановление основных производственных фондов, исчисленная из балансовой стоимости и утвержденных в установленном порядке норм, включая и ускоренную амортизацию их активной части, производимую в соответствии с законодательством.

$$C_A = \frac{A}{F_Д} \cdot T \text{, где} \qquad (12)$$

A – годовые амортизационные отчисления, *руб./год*;

T – время работы оборудования, *час*;

$F_Д$ – действительный годовой фонд рабочего времени на ПЭВМ, *час/год* (Значения показателей представлены ниже (Таблица 3)).

Таблица 3 - Исходные данные для расчета амортизационных отчислений.

Наименование показателя	Значение
Цена ПЭВМ (на 1 января 2009), *руб.*	25000
Количество ПЭВМ, участвующих в разработке	4
Процент на амортизационные отчисления (табл. 50 из [17])	12%
Годовой фонд рабочего времени на ПЭВМ (5-ти дневная неделя, 8-и часовой рабочий день) , *час*	2080

$A = 0{,}12 \cdot 22500 \cdot 4 = 12000$ *руб.*

Время работы оборудования: T = *13 мес. = 273 дня = 2184 часа.*

Всего амортизационные отчисления при разработке системы составят:

$$C_A = 12000 \cdot 2184/2080 = 12600 \text{ (руб.)}$$

6.2.4 Накладные расходы

В данную статью входят другие затраты, входящие в состав себестоимости продукции (работ, услуг), но не относящиеся к ранее перечисленным элементам затрат: общехозяйственные расходы, непроизводительные расходы и расходы на управление. Накладные расходы определяют в процентном отношении к основной заработной плате, т.е.

$$C_H = \alpha_H \cdot C_{ЗО}, \text{ где} \qquad (13)$$

α_H – коэффициент накладных расходов; $\alpha_H = 1,8$

Подставляя в формулу (13) известные значения, вычисляем значение затрат, связанных с накладными расходами:

$$C_H = \alpha_H \cdot C_{ЗО} = 1,8 \cdot 1675238 = 3015428 \text{ (руб.)}$$

6.2.5 Результаты расчетов затрат

Результаты расчетов приведены ниже (Таблица 4).

Таблица 4 - Результаты расчетов затрат на разработку системы.

№п/п	Наименование статьи	Сметная стоимость, руб.	Примечание
1	Затраты на оплату труда	2010286	
2	Отчисления в ФСС	534735	
3	Амортизация оборудования	12600	
4	Накладные расходы	3015428	
	Итого:	5573049	

Вывод: затраты на разработку «Распределенной многомодельной системы дискретного имитационного моделирования» составляют 5573049 руб.

6.3 Определение стоимости разработки системы

Цена создания определяется следующим образом:

$$Ц = К \cdot С + П_Р \text{, где} \qquad (14)$$

$С$ – затраты на разработку системы (сметная себестоимость), *руб.*;

$К$ – коэффициент учета затрат на изготовление опытного образца системы как продукции производственно-технического назначения ($К$ *= 1,1*);

$П_Р$ – нормативная прибыль, рассчитываемая по формуле:

$$П_Р = С \cdot \rho_Н / 100 \text{, где} \qquad (15)$$

$\rho_Н$ – норматив рентабельности, $\rho_Н$ *= 30%*.

В соответствии с (15) нормативная прибыль равна:

$$П_Р = 5573049 \cdot 0{,}3 = 1671915 \text{ (руб.)}$$

Подставляя в (14) известные величины, определяем цену создания системы:

$$Ц = К \cdot С + П_Р = 1{,}1 \cdot 5573049 + 167915 = 7802269 \text{ (руб.)}$$

Вывод: цена создания «Распределенной многомодельной системы дискретного имитационного моделирования на основе РДО» равна 7802269 руб.

7 Техника безопасности и условия жизнеобеспечения

7.1 Требования безопасности при работе с «Распределенной многомодельной системой дискретного имитационного моделирования на основе РДО»

7.1.1 Введение

В дипломном проекте разрабатывается интерфейс системы имитационного моделирования, функционал которого сводится к возможности организации информационного взаимодействия между приложениями в пределах локальной сети, поддерживающими этот интерфейс.

На данный момент система не является коммерческим продуктом, однако в будущем возможен ее вывод на рынок после доработок функционала. Предполагается, что пользователи будут работать с системой в течение длительных промежутков времени, возможно, весь рабочий день (при использовании на предприятиях и других коммерческих организациях). Заметим, что основное назначение системы в настоящее время – использование в учебных целях при изучении студентами (аспирантами) дисциплин, связанных с имитационным моделированием дискретных систем, и при решении задач анализа и синтеза сложных систем для достижения различных учебных, научных и исследовательских целей.

В этой связи при работе с разработанной системой необходимо отслеживать выполнение всех требований, предъявляемых к помещению для работы с ПЭВМ, и соответствие условий работы санитарно-эпидемиологическим правилам и нормативам "Гигиенические требования к персональным электронно-вычислительным машинам и организации

работы. СанПиН 2.2.2/2.4.1340-03", ориентируясь на учащихся. Также необходимо осуществлять контроль соблюдения учащимися техники безопасности при работе с ПЭВМ.

В соответствии с СанПиН 2.2.2/2.4.1340-03 требования к ПЭВМ и организации работы делятся на несколько групп, которые рассматриваются далее отдельно. Приведены допустимые нормы и методы их обеспечения [18].

7.1.2 Требования к помещениям для работы с ПЭВМ

Помещения для эксплуатации ПЭВМ должны иметь естественное и искусственное освещение. Эксплуатация ПЭВМ в помещениях без естественного освещения допускается только при соответствующем обосновании и наличии положительного санитарно-эпидемиологического заключения, выданного в установленном порядке.

Естественное и искусственное освещение должно соответствовать требованиям действующей нормативной документации. Окна в помещениях, где эксплуатируется вычислительная техника, преимущественно должны быть ориентированы на север и северо-восток. Оконные проемы должны быть оборудованы регулируемыми устройствами типа: жалюзи, занавесей, внешних козырьков и др.

Площадь на одно рабочее место пользователей ПЭВМ с ВДТ на базе электроннолучевой трубки (ЭЛТ) должна составлять не менее 6 $м^2$, с ВДТ на базе плоских дискретных экранов (жидкокристаллические, плазменные) - 4,5 $м^2$.

Для внутренней отделки интерьера помещений, где расположены ПЭВМ, должны использоваться диффузно-отражающие материалы с коэффициентом отражения для потолка - 0,7 - 0,8; для стен - 0,5 - 0,6; для пола - 0,3 - 0,5.

Полимерные материалы используются для внутренней отделки интерьера помещений с ПЭВМ при наличии санитарно-эпидемиологического заключения.

Помещения, где размещаются рабочие места с ПЭВМ, должны быть оборудованы защитным заземлением (занулением) в соответствии с техническими требованиями по эксплуатации.

Не следует размещать рабочие места с ПЭВМ вблизи силовых кабелей и вводов, высоковольтных трансформаторов, технологического оборудования, создающего помехи в работе ПЭВМ.

7.1.3 Требования к микроклимату, содержанию аэроионов и вредных химических веществ в воздухе на рабочих местах

В помещениях, в которых работа с использованием ПЭВМ является основной и связана с нервно-эмоциональным напряжением, должны обеспечиваться оптимальные параметры микроклимата для категории работ 1а и 1б в соответствии с действующими санитарно-эпидемиологическими нормативами микроклимата производственных помещений. Вид работы с компьютером зависит от задач решаемых программным комплексом. В нашем случае, вид работы – основной, так как работа оператора с программным продуктом подразумевает непрерывную работу и может занимать более 50% времени от смены.

Характер труда пользователей разработанной системы относится к категории 1а, потому что работы производятся сидя и не требуют физического напряжения, расход энергии составляет до 120 ккал/ч, следовательно, необходимо проектировать помещение с учетом оптимальных норм микроклимата для категории работ 1а.

В рабочей зоне допустимые параметры микроклимата должны соответствовать требованиям СанПиН 2.2.4.548-96 [19]. Величины этих

допустимых параметров приведены ниже (Таблица 5), применительно к выполнению работ категории 1а в холодный и теплый периоды года.

Перепады температуры воздуха по высоте и по горизонтали, а также изменения температуры воздуха в течение смены при обеспечении оптимальных величин микроклимата на рабочих местах не должны превышать 2°С и выходить за пределы величин, указанных в таблице 1 для отдельных категорий работ.

Таблица 5 - Допустимые величины показателей микроклимата на рабочих местах производственных помещений.

Период года	Категория работ по уровню энергозатрат, Вт	Температура воздуха, 0С	Температура поверхностей, 0С	Относительная влажность воздуха, %	Скорость движения воздуха, м/с
Холодный	Ia (до 139)	22-24	21-25	60-40	0,1
Теплый	Ia (до 139)	23-25	22-26	60-40	0,1

В помещениях, оборудованных ПЭВМ, должна проводится ежедневная влажная уборка и систематическое проветривание после каждого часа работы на ПЭВМ.

Уровни положительных и отрицательных аэроионов в воздухе помещений, где расположены ПЭВМ, должны соответствовать действующим санитарно-эпидемиологическим нормативам (СанПиН 2.2.4.1294–03 [20] и Таблица 6). Ограничения накладываются на концентрацию аэроионов обеих полярностей и коэффициент униполярности У, определяемый, как отношение концентрации аэроионов положительной полярности к концентрации аэроионов отрицательной полярности.

В зонах дыхания персонала на рабочих местах, где имеются источники электростатических полей (видеодисплейные терминалы или другие виды оргтехники) допускается отсутствие аэроионов положительной полярности.

Для нормализации аэроионного состава воздуха следует применять соответствующие, прошедшие санитарно-эпидемиологическую оценку и имеющие действующее санитарно-эпидемиологическое заключение, аэроионизаторы или деионизаторы, предназначенные для использования в санитарно-гигиенических целях.

Таблица 6 - Уровни ионизации воздуха помещений при работе с ПЭВМ.

Нормируемые показатели	Концентрация аэронов, р0 (ионов/см3)		Коэффициент униполярности, У
	Положительной полярности	Отрицательной полярности	
Минимально допустимые	ро+ > 400	ро- >= 600	0.4 <= У < 1.0
Максимально допустимые	ро+ < 50 000	ро- < 50 000	

Содержание вредных химических веществ в воздухе производственных помещений, в которых работа с использованием ПЭВМ является вспомогательной, не должно превышать предельно допустимых концентраций вредных веществ в воздухе рабочей зоны в соответствии с действующими гигиеническими нормативами (ГН 2.2.5.1313–03 “Предельно допустимые концентрации (ПДК) вредных веществ в воздухе рабочей зоны” [21] и Таблица 7).

Таблица 7 - Предельно допустимые концентрации вредных веществ в воздухе рабочей зоны.

№ п/п	Наименование вещества	Формула	Предельно допустимая среднесуточная концентрация, мг/м3	Класс опасности
1	Азота диоксид	NO2	0,04	2
2	Азот (II) оксид	NO	0,06	3
3	Углерод оксид	CO	3	4

Для обеспечения соблюдения требований к микроклимату в помещениях, оборудованных ПЭВМ рекомендуется:

- применять увлажнители воздуха, заправляемые ежедневно дистиллированной или прокипяченной питьевой водой (для повышения влажности воздуха);
- проветривать помещения с ВДТ и ПЭВМ (обеспечивает улучшение качественного состава воздуха, в том числе и аэроионный режим);
- устраивать системы общеобменной приточно-вытяжной механической вентиляции;
- использовать фильтры очистки воздуха для удаления из воздуха вредных веществ перед его подачей на рабочие места (ветвь притока системы вентиляции).

7.1.4 Требования к уровням шума и вибрации на рабочих местах

В производственных помещениях при выполнении вспомогательных работ с использованием ПЭВМ уровни шума на рабочих местах не должны превышать предельно допустимых значений, установленных для данных видов работ в соответствии с действующими санитарно-эпидемиологическими нормативами (СН 2.2.4/2.1.8.562–96 [22]). Предельно допустимые уровни звукового давления, уровни звука и

эквивалентные уровни звука для первого вида трудовой деятельности представлены ниже (Таблица 8).

Таблица 8 - Допустимые значения уровней звукового давления.

Уровни звукового давления, дБ, в октавных полосах со среднегеометрическими частотами, Гц									Уровни звука и эквивалентные уровни звука (в дБА)
31,5	63	125	250	500	1000	2000	4000	8000	
86	71	61	54	49	45	42	40	38	50

При выполнении работ с использованием ПЭВМ в производственных помещениях уровень вибрации не должен превышать допустимых значений вибрации для рабочих мест (категория 3, тип "в") в соответствии с действующими санитарно-эпидемиологическими нормативами (СН 2.2.4/2.1.8.566-96 [23]). Предельно допустимые значения вибраций рабочих мест представлены ниже (Таблица 9).

Таблица 9 - Предельно допустимые значения вибрации рабочих мест.

Средне-геометрические частоты полос, Гц	Предельно допустимые значения вибрации рабочих мест категории 3 - технологической типа "в"							
	виброускорения				виброскорости			
	$м/с^2$		дБ		$м/с \cdot 10^{-2}$		дБ	
	1/3 окт	1/1 окт	1/3 окт	1/1 окт	1/3 окт	1/1 окт	1/3 окт	1/1 окт
1,6	0,0130		82		0,130		88	
2,0	0,0110	0,020	81	86	0,089	0,180	85	91
2,5	0,0100		80		0,063		82	
3,15	0,0089		79		0,045		79	
4,0	0,0079	0,014	78	83	0,032	0,063	76	82
5,0	0,0079		78		0,025		74	
6,3	0,0079		78		0,020		72	
8,0	0,0079	0,014	78	83	0,016	0,032	70	76

10,0	0,0100		80		0,016		70	
12,5	0,0130		82		0,016		70	
16,0	0,0160	0,028	84	89	0,016	0,028	70	75
20,0	0,0200		86		0,016		70	
25,0	0,0250		88		0,016		70	
31,5	0,0320	0,056	90	95	0,016	0,028	70	75
40,0	0,0400		92		0,016		70	
50,0	0,0500		94		0,016		70	
63,0	0,0630	0,110	96	101	0,016	0,028	70	75
80,0	0,0790		98		0,016		70	
Корректированные и эквивалентные корректированные значения и их уровни		0,014		83		0,028		75

Для снижения вибрации в помещениях ПЭВМ необходимо устанавливать на амортизирующие прокладки.

Снизить уровень шума в помещениях с ВДТ и ПЭВМ можно с помощью акустической обработки потолка звукопоглощающими материалами с максимальными коэффициентами звукопоглощения в области частот 63 - 8000 Гц.

Дополнительным звукопоглощением служат однотонные занавеси из плотной ткани, гармонирующие с окраской стен и подвешенные в складку на расстоянии 15 - 20 см от ограждения. Ширина занавеси должна быть в 2 раза больше ширины окна.

Шумящее оборудование (печатающие устройства, серверы и т.п.), уровни шума которого превышают нормативные, должно размещаться вне помещений с ПЭВМ.

7.1.5 Требования к освещению на рабочих местах

Рабочие столы следует размещать таким образом, чтобы видеодисплейные терминалы были ориентированы боковой стороной к световым проемам, чтобы естественный свет падал преимущественно слева.

Искусственное освещение в помещениях для эксплуатации ПЭВМ должно осуществляться системой общего равномерного освещения. В производственных и административно-общественных помещениях, в случаях преимущественной работы с документами, следует применять системы комбинированного освещения (к общему освещению дополнительно устанавливаются светильники местного освещения, предназначенные для освещения зоны расположения документов).

Освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300 - 500 лк. Освещение не должно создавать бликов на поверхности экрана. Освещенность поверхности экрана не должна быть более 300 лк.

Следует ограничивать прямую блесткость от источников освещения, при этом яркость светящихся поверхностей (окна, светильники и др.), находящихся в поле зрения, должна быть не более 200 кд/м^2.

Следует ограничивать отраженную блесткость на рабочих поверхностях (экран, стол, клавиатура и др.) за счет правильного выбора типов светильников и расположения рабочих мест по отношению к источникам естественного и искусственного освещения, при этом яркость бликов на экране ПЭВМ не должна превышать 40 кд/м^2 и яркость потолка не должна превышать 200 кд/м^2.

Показатель ослепленности для источников общего искусственного освещения в производственных помещениях должен быть не более 20. Показатель дискомфорта в административно-общественных помещениях - не более 40.

Яркость светильников общего освещения в зоне углов излучения от 50^0 до 90^0 с вертикалью в продольной и поперечной плоскостях должна составлять не более 200 кд/м^2, защитный угол светильников должен быть не менее 40 градусов.

Светильники местного освещения должны иметь не просвечивающий отражатель с защитным углом не менее 40 градусов.

Следует ограничивать неравномерность распределения яркости в поле зрения пользователя ПЭВМ, при этом соотношение яркости между рабочими поверхностями не должно превышать 3:1 - 5:1, а между рабочими поверхностями и поверхностями стен и оборудования - 10:1.

В качестве источников света при искусственном освещении следует применять преимущественно люминесцентные лампы типа ЛБ и компактные люминесцентные лампы (КЛЛ). При устройстве отраженного освещения в производственных и административно-общественных помещениях допускается применение металлогалогенных ламп. В светильниках местного освещения допускается применение ламп накаливания, в том числе галогенных.

Для освещения помещений с ПЭВМ следует применять светильники с зеркальными параболическими решетками, укомплектованными электронными пускорегулирующими аппаратами (ЭПРА). Допускается использование многоламповых светильников с электромагнитными пускорегулирующими аппаратами (ЭПРА), состоящими из равного числа опережающих и отстающих ветвей.

Применение светильников без рассеивателей и экранирующих решеток не допускается.

При отсутствии светильников с ЭПРА лампы многоламповых светильников или рядом расположенные светильники общего освещения следует включать на разные фазы трехфазной сети.

Общее освещение при использовании люминесцентных светильников следует выполнять в виде сплошных или прерывистых линий светильников, расположенных сбоку от рабочих мест, параллельно линии зрения пользователя при рядном расположении видеодисплейных терминалов. При периметральном расположении компьютеров линии светильников должны располагаться локализовано над рабочим столом ближе к его переднему краю, обращенному к оператору.

Коэффициент запаса (Кз) для осветительных установок общего освещения должен приниматься равным 1,4.

Коэффициент пульсации не должен превышать 5%.

Для обеспечения нормируемых значений освещенности в помещениях для использования ПЭВМ следует проводить чистку стекол оконных рам и светильников не реже двух раз в год и проводить своевременную замену перегоревших ламп.

7.1.6 Требования к уровням электромагнитных полей на рабочих местах

Временные допустимые уровни ЭМП, создаваемых ПЭВМ на рабочих местах пользователей представлены ниже (Таблица 10).

Таблица 10 - Временные допустимые уровни электромагнитных полей, создаваемых ПЭВМ на рабочих местах пользователей.

Наименование параметров		ВДУ
Напряженность электрического поля	в диапазоне частот 5 Гц - 2 кГц	25 В/м
	в диапазоне частот 2 кГц - 400 кГц	2,5 В/м
Плотность магнитного потока	в диапазоне частот 5 Гц - 2 кГц	250 нТл
	в диапазоне частот 2 кГц - 400 кГц	25 нТл
Напряженность электростатического поля		15 кВ/м

Для снижения уровня электромагнитных полей на рабочих местах можно предпринять ряд мер, таких как обеспечение мониторов на электроннолучевой трубке специальными защитными экранами, обеспечить заземление всех терминалов ПК и периферийных устройств. (Мониторы, основанные на жидких кристаллах, не нуждаются в защитных экранах) Так же целесообразно рационально разместить терминалы ПК, таким образом, что бы они не располагались в одном месте. Как дополнительное средство защиты можно обеспечить системные блоки компьютеров качественными экранирующими корпусами.

Контроль уровней ЭМП на рабочих местах пользователей ПЭВМ производится в соответствии с приложением 3 к СанПиН 2.2.2/2.4.1340-03 [18].

7.1.7 Требования к организации рабочих мест пользователей

При размещении рабочих мест с ПЭВМ расстояние между рабочими столами с видеомониторами (в направлении тыла поверхности одного видеомонитора и экрана другого видеомонитора) должно быть не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов - не менее 1,2 м.

Рабочие места с ПЭВМ в помещениях с источниками вредных производственных факторов должны размещаться в изолированных кабинах с организованным воздухообменом.

Рабочие места с ПЭВМ при выполнении творческой работы, требующей значительного умственного напряжения или высокой концентрации внимания, рекомендуется изолировать друг от друга перегородками высотой 1.5 - 2.0 м.

Экран видеомонитора должен находиться от глаз пользователя на расстоянии 600 - 700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики. Поверхность рабочего стола должна иметь коэффициент отражения 0,5 - 0,7.

Конструкция рабочего стула (кресла) должна обеспечивать поддержание рациональной рабочей позы при работе на ПЭВМ, позволять изменять позу с целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления. Тип рабочего стула (кресла) следует выбирать с учетом роста пользователя, характера и продолжительности работы с ПЭВМ.

Рабочий стул (кресло) должен быть подъемно-поворотным, регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию.

Поверхность сиденья, спинки и других элементов стула (кресла) должна быть полумягкой, с нескользящим, слабо электризующимся и воздухопроницаемым покрытием, обеспечивающим легкую очистку от загрязнений.

Помещения для занятий оборудуются одноместными столами, предназначенными для работы с ПЭВМ.

Конструкция одноместного стола для работы с ПЭВМ должна предусматривать:

- две раздельные поверхности: одна горизонтальная для размещения ПЭВМ с плавной регулировкой по высоте в пределах 520 – 760 мм и вторая – для клавиатуры с плавной регулировкой

по высоте и углу наклона от 0 до 15 градусов с надежной фиксацией в оптимальном положении (12 – 15 градусов);

- ширину поверхностей для ВДТ и клавиатуры не менее 750 мм (ширина обеих поверхностей должна быть одинаковой) и глубину не менее 550 мм;
- опору поверхностей для ПЭВМ или ВДТ и для клавиатуры на стояк, в котором должны находиться провода электропитания и кабель локальной сети. Основание стояка следует совмещать с подставкой для ног;
- отсутствие ящиков;
- увеличение ширины поверхностей до 1200 мм при оснащении рабочего места принтером.

Высота края стола, обращенного к работающему с ПЭВМ и высота пространства для ног должны соответствовать росту обучающихся в обуви (Таблица 11).

Таблица 11 - Высота одноместного стола для занятий с ВТ.

Рост учащихся или студентов в обуви, см	Высота над полом	
	поверхность стола	пространство для ног, не менее
116-130	520	400
131-145	580	520
146-160	640	580
161-175	700	640
выше 175	760	700

При наличии высокого стола и стула, несоответствующего росту обучающихся, следует использовать регулируемую по высоте подставку для ног.

Линия взора должна быть перпендикулярна центру экрана и оптимальное ее отклонение от перпендикуляра, проходящего через центр экрана в вертикальной плоскости не должна превышать +-5 градусов, допустимое +- 10 градусов.

Рабочее место с ПЭВМ оборудуют стулом, основные размеры которого должны соответствовать росту обучающихся в обуви (Таблица 12).

Таблица 12 - Основные размеры стула для учащихся и студентов.

Параметры стула	Рост учащихся и студентов в обуви, см				
	116-130	131-145	146-160	161-175	>175
Высота сиденья над полом, мм	300	340	380	420	460
Ширина сиденья, не менее, мм	270	290	320	340	360
Глубина сиденья, мм	290	330	360	380	400
Высота нижнего края спинки над сиденьем, мм	130	150	160	170	190
Высота верхнего края над сиденьем, мм	280	310	330	360	400
Высота линии прогиба спинки, не менее, мм	170	190	200	210	220
Радиус изгиба переднего края сиденья, мм	20-50				
Угол наклона сиденья, ^{0}	0-4				
Угол наклона спинки, ^{0}	95-108				
Радиус спинки в плане, не менее, мм	300				

7.1.8 Требования к электробезопасности на рабочих местах

Электробезопасность – система организационных и технических мероприятий и средств, обеспечивающих защиту людей от вредного и

опасного воздействия электрического тока, электрической дуги, электромагнитного поля и статического электричества.

Основными мерами защиты от поражения током являются: обеспечение недоступности токоведущих частей, находящихся под напряжением, для случайного прикосновения, защитным заземлением, занулением, защитным отключением и др.; организация безопасной эксплуатации электроустановок.

В помещении без повышенной электроопасности корпус любой электроустановки напряжением свыше 50 В необходимо заземлять или занулять. В помещениях с повышенной электроопасностью необходимо заземлять корпуса установок напряжением свыше 25 В.

По способу защиты человека от поражения электрическим током согласно ГОСТ12.2.007.0-75 [24] ПЭВМ можно отнести к 01 классу.

Согласно ГОСТ 12.1.038-82 [25], предельно-допустимые напряжения прикосновения и токи через человека при нормальном (неаварийном) режиме работы изделия приведены ниже (Таблица 13).

Таблица 13 - Предельно допустимые напряжения прикосновения и токи.

Вид тока	Напряжение, В, не более	Ток, мА, не более
Переменный 50 Гц	2	0.3
Постоянный	8	1.0

Длительность действия напряжений, указанных в таблице, не более 10 минут на сутки и установлено из реакции чувствительности.

Весь персонал, работающий на терминалах, должен периодически проходить инструктаж об опасности электрического тока и способах оказания первой помощи.

7.1.9 Требования к пожаробезопасности на рабочих местах

Помещение, оборудованное терминалами ПК, относится к категории В пожарной опасности. К ней относятся помещения, в которых есть горючие и трудногорючие жидкости, твердые горючие и трудногорючие вещества и материалы, вещества и материалы способные при взаимодействии с водой, кислородом воздуха или друг с другом только гореть.

При рассмотрении пожарной безопасности помещения необходимо руководствоваться нормами пожарной безопасности (НПБ 105-03) [26].

Пожарная безопасность может быть обеспечена мерами пожарной профилактики и активной пожарной защиты. Понятие пожарной профилактики включает комплекс мероприятий, необходимых для предупреждения возникновения пожара или уменьшения его последствий. Под активной пожарной защитой понимаются меры, обеспечивающие успешную борьбу с возникающими пожарами или взрывоопасной ситуацией.

Для обеспечения пожарной безопасности необходимо определить категорию помещения, в котором работает персонал, и, исходя из этого, подобрать соответствующие системы и средства противопожарной защиты.

При определении видов и количества первичных средств пожаротушения следует учитывать физико-химические и пожароопасные свойства горючих веществ, их отношение к огнетушащим веществам, а также площадь производственных помещений. Выбор типа огнетушителя (передвижной или ручной) обусловлен размерами возможных очагов пожара. При их значительных размерах необходимо использовать передвижные огнетушители.

В нашем случае рекомендуется использовать углекислотные или порошковые огнетушители.

Для снижения вероятности возникновения пожара необходимо проводить различные профилактические мероприятия:

- Организационные – правильная эксплуатация электрооборудования, правильное содержание зданий и помещений;
- Технические — соблюдение противопожарных правил и норм, норм при проектировании зданий, при устройстве отопления, вентиляции освещения, правильное размещение оборудования;
- Мероприятия режимного характера – запрещение курения в неустановленных местах и т.д.;
- Эксплуатационные — своевременные профилактические осмотры и ремонт неисправного электрооборудования.

Для снижения вероятности возникновения и распространения пожара на ранней стадии необходимо:

- установить пожарную сигнализацию с системой оповещения работников, дежурного по объекту и, желательно, автоматическое оповещение противопожарных служб;
- иметь в наличии несколько ручных углекислотных огнетушителей (например, огнетушитель марки ОУ-3);

7.2 Типовой расчет зануления ПЭВМ

7.2.1 Общие сведения

Зануление применяется в трехфазных четырехпроводных электрических сетях напряжением до 1000В с глухозаземленной нейтралью переменного тока, в трехпроводных сетях постоянного тока с глухозаземленной средней точкой обмотки источника энергии, а также в однофазных двухпроводных сетях переменного тока с глухозаземленным выводом обмотки источника питания. Зануление обязательно при

напряжении не менее 380В переменного тока и 440В постоянного, а в помещениях с повышенной опасностью и особо опасных, а также вне помещений – при напряжении выше 42В переменного и вы 110В постоянного тока. Занулению подлежат металлические нетоковедущие части электроприемников, в том числе металлические корпуса электрических приборов, контрольных и наладочных стендов, трансформаторов, пусковых и регулировочных реостатов, выключателей, светильников, стационарных, переносных электроприемников, металлические оболочки и соединительные муфты контрольных и силовых кабелей и т.п. Принципиальная схема зануления в трехфазной сети переменного тока приведена ниже (см. Рисунок 30).

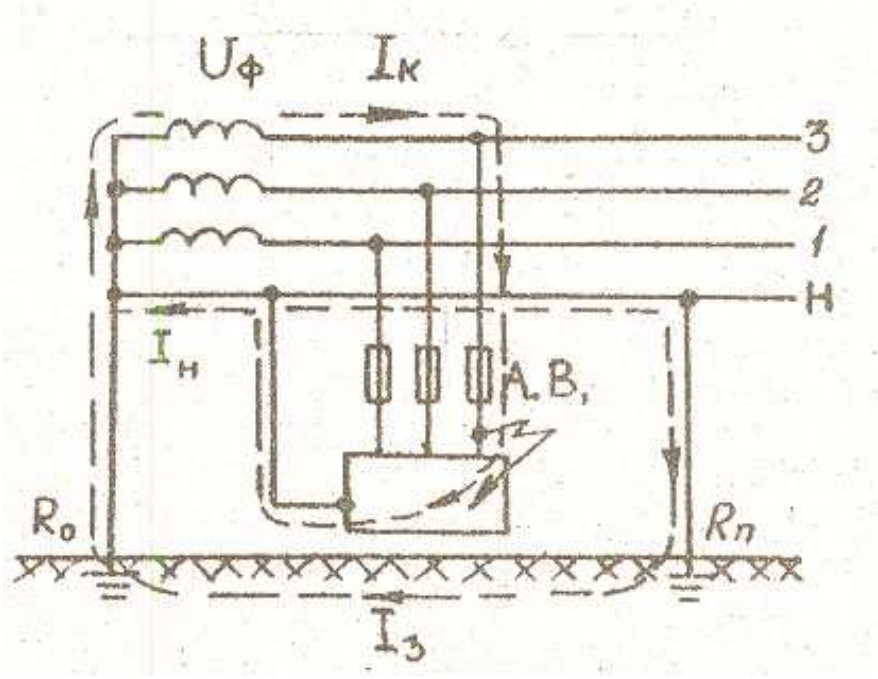

Рисунок 30 - Принципиальная схема зануления.

Обозначениями показаны: $U_Ф$ – фазное напряжение сети, *АВ* – автоматический выключатель; *1*, *2*, *3* – фазные проводники; Н – нулевой защитный проводник, $I_К$ – ток короткого замыкания, $I_Н$ – ток, протекающий через нулевой защитный проводник, $I_З$ – ток, протекающий через между заземлителями, R_0 – сопротивление заземления нейтрали, $R_П$ – сопротивление повторных заземлителей.

Цель зануления – обеспечить быстрое отключение установки от сети при замыкании фазы (или фаз) на ее корпус, а также снизить напряжение на корпусе в аварийный период. В соответствии с этим зануление рассчитывается прежде всего на отключающую способность.

7.2.2 Исходные данные для расчета

Для питания учебной аудитории с пятнадцатью ПЭВМ используется сеть переменного тока напряжением 220В, частотой 50 Гц с глухозаземлённой нейтралью.

Для обеспечения электробезопасности при монтаже, наладке и работе с сетью необходимо обратить особое внимание на создание защитных мер от попадания пользователей и обслуживающего персонала под напряжение, для предотвращения электротравматизма при работе с сетью.

На рабочем месте необходимо наличие зануления. Все электронные устройства необходимо занулить.

Электропитание рабочего места должно быть подключено через рубильник, установленный в месте, удобном для быстрого отключения питания рабочего места, должны быть предприняты меры для обесточивания рабочего места в аварийных режимах. Обычно ставится автоматический выключатель с защитой от короткого замыкания.

При работе необходимо соблюдать ряд предохранительных мер по предотвращению электротравматизма, для отключения поврежденного электрооборудования используется автоматический выключатель, отключающий рабочее место от сети.

Все работы, связанные с наладкой и эксплуатацией сети ведутся в помещении, относящемся к категории "без повышенной опасности" поражения электрическим током.

В сети 380/220 с глухозаземленной нейтралью (сопротивление заземления нейтрали не более 4 Ом) при однофазном замыкании на корпус необходимо обеспечить автоматическое отключение поврежденного электрооборудования. При кратковременном аварийном режиме создается безопасность обслуживания и сохранность электрооборудования. Однако кратковременность может быть обеспечена только созданием определенной кратности тока короткого замыкания на корпус по отношению к номинальному току защитного аппарата. Этого можно добиться только прокладкой специального провода достаточной проводимости - нулевого провода, к которому присоединяются корпуса электрооборудования.

Для того чтобы снизить опасные потенциалы при замыкании на корпус, используются повторные заземлители с сопротивлением не более 10 Ом.

В помещении, где производится монтаж сети, питание ПЭВМ осуществляется от подстанции с трансформатором 630 кВт (схема соединения обмоток «звезда-звезда», напряжением 10/0,4 кВ), удаленной от рабочего места на 200 м. Питание к распределительному щиту подводится одним кабелем, в котором находятся 3 медных провода площадью сечения 25 мм^2, а роль нулевого проводника выполняет четвертый кабель диаметром сечения 6 мм. ПЭВМ защищены автоматическим выключателем, имеющим электромагнитный расцепитель.

7.2.3 Расчет защитного зануления

Для надежного срабатывания автоматического выключателя с электромагнитным расцепителем требуется выполнение условия:

$I_K \geq k \cdot I_{НОМ}$, где (16)

I_K – ток короткого замыкания фазы на корпус электроустановки, *А*;

k – коэффициент кратности номинального тока ($k = 1,4$);

$I_{НОМ}$ – номинальный ток аппарата защиты, *А*.

В соответствии с данными, приведенными ниже (Таблица 14), выберем $I_{НОМ} = 40\ А$ как ближайший меньший из условия несрабатывания автоматического выключателя при протекании через него рабочих токов вычислительного комплекса.

Таблица 14 - Номинальные токи вычислительного комплекса.

Наименование устройства	Ток, А	Кол-во, шт.	Итого, А
Системный блок ATX MidiTower	1,5	15	22,5
Монитор Iiyama Vision Master Pro 411 19”	1,0	15	15,0
Принтер Epson LX-300	0,3	1	0,3
Итого:			37,8

Номинальным током плавкой вставки $I_{НОМ}$ называется ток, значение которого указано непосредственно заводом-изготовителем.

Ток, протекающий через нулевой защитный проводник, $I_Н\ (А)$ вычисляется по следующей приближенной формуле:

$$I_Н = \frac{U_Ф}{Z_Т / 3 + Z_П}\text{, где} \qquad (17)$$

$U_Ф$ – фазное напряжение сети, *В*;

$Z_Т$ – сопротивление обмотки трансформатора, *Ом*;

$Z_П$ – сопротивление проводников петли «фаза-нуль», *Ом*.

Так как обычно сопротивление проводников петли «фаза-нуль» не превышает 2 Ом, ток $I_З$, протекающий через землю, много меньше тока $I_Н$, проходящего по нулевому защитному проводнику, и можно считать $I_К \approx I_Н$.

По таблице 7.1 из [27] находим значение $Z_Т = 0{,}129\ Ом$, соответствующее исходным данным задачи.

Сопротивление проводников петли «фаза-нуль» $Z_П$ *(Ом)* рассчитывается по следующей формуле:

$$Z_П = \sqrt{(R_Ф + R_Н)^2 + (X_Ф + X_Н + X_П)^2}\text{, где} \qquad (18)$$

$R_Ф$ и $R_Н$ – активные сопротивления фазного и нулевого защитного проводников, *Ом*;

$X_Ф$ и $X_Н$ – внутренне индуктивное сопротивление фазного и нулевого защитного проводников, *Ом*;

$X_П$ – внешнее индуктивное сопротивление проводников петли «фаза-нуль», Ом.

Вычислим величины этих сопротивлений.

Активные сопротивления для медных и алюминиевых проводников *R* (*Ом)* вычисляются по формуле:

$$R = \rho\frac{l}{S}\text{, где} \qquad (19)$$

ρ – удельное сопротивление материала, *Ом·мм²/м*;

$\rho_{медь}$ *= 0,0175 Ом·мм²/м;*

l — длина участка, *м;*

l = 200 м;

S — сечение провода, *мм²;*

S = 25 мм².

Подставляя известные величины в (19), получаем значение активного сопротивления фазного проводника:

$R_Ф = 0{,}0175 \cdot 200/25 = 0{,}14$ *(Ом)*

Так как нулевой защитный проводник выполнен из стали круглого сечения, то:

1) его активное сопротивление $R_Н$ *(Ом)* вычисляется по формуле:

$$R_Н = R_1 \cdot l\text{, где} \qquad (20)$$

R_1 – активное сопротивление 1 км проводника, *Ом*;

l — длина участка, *км*.

2) его внутренне индуктивное сопротивление X_H *(Ом)* вычисляется по формуле:

$$X_H = X_1 \cdot l \text{, где} \quad (21)$$

X_1 – внутреннее индуктивное сопротивление 1 км проводника, *Ом*;

l — длина участка, *км*.

Для нахождения табличных значений R_1 и X_1 необходимо вычислить значение ожидаемой плотности тока в проводнике i_H *(А/мм^2)* по формуле:

$$i_H = \frac{I_K}{S_H} \text{, где} \quad (22)$$

I_K – ожидаемый ток короткого замыкания, А;

$I_K = k \cdot I_{НОМ} = 56$ *А;*

S_H — площадь сечения, *мм^2;*

$S_H = 28{,}27$ *мм^2;*

Подставляя в (7) известные величины, получаем значение ожидаемой плотности тока в проводнике:

$i_H = 56/28{,}27 \approx 2$ *(А/мм^2)*

Считая $i_H = 2$ *А/мм^2* по таблице 7.2 из [27] находим:

1) значение активного сопротивления 1 км проводника $R_1 = 8$ *Ом/км*.
2) значение внутреннего индуктивного сопротивления 1 км проводника $X_1 = 4{,}8$ *Ом/км*.

Подставляя найденные значения в формулы (20) и (21) соответственно, находим значения:

1) активное сопротивление нулевого защитного проводника:

$R_H = 8 \cdot 0{,}2 = 1{,}6$ *(Ом)*

2) внутренне индуктивное сопротивление нулевого защитного проводника:

$X_H = 4{,}8 \cdot 0{,}2 = 0{,}96$ (Ом)

Так как нулевой защитный проводник является четвертой жилой кабеля, в котором проложены фазные проводники, то значением сопротивления $X_П$ можно пренебречь.

Так как значение X_H для медных проводников мало, им можно пренебречь, и при подстановке известных значений в формулу (18), получаем значение сопротивление проводников петли «фаза-нуль»:

$$Z_П = \sqrt{(0{,}14 + 1{,}60)^2 + (0{,}96)^2} = \sqrt{3{,}03 + 0{,}92} = 1{,}99(Ом)$$

Подставляя известные величины в формулу (17), находим значение тока, протекающего через нулевой защитный проводник:

$$I_H = \frac{220}{0{,}129/3 + 1{,}99} = \frac{220}{2{,}03} \approx 108\,(A)$$

Подставляя известные значение величин в неравенство (16), получаем верное неравенство:

$I_К \geq k \cdot I_{НОМ}$

$108\,A \geq 1{,}4 \cdot 40\,A$

$108\,A \geq 56\,A$

Верное неравенство означает, что условие надежного срабатывания автоматического выключателя в аварийной ситуации выполняется.

ЗАКЛЮЧЕНИЕ

В рамках данной работы были получены следующие результаты:

1) Проведено предпроектное исследование системы имитационного моделирования РДО, выявлены актуальные проблемы ее развития, а также сформулированы предпосылки создания на ее основе распределенной системы дискретного имитационного моделирования. Сформулированы положительные выводы относительно целесообразности получения успешного результата разработки.

2) Разработана концепция распределенной многомодельной системы имитационного моделирования на основе РДО, обозначены цели разработки и новые функции системы. Сделан выбор технологии реализации системы в пользу архитектуры распределенного программирования CORBA, изучены ее основные понятия, определения и алгоритмы разработки. Приведена схема структуры и уровней взаимодействия распределенной системы имитационного моделирования на основе РДО. Разработана функциональная модель работы системы. Кратко рассмотрены вопросы синхронизации модельного времени в распределенной системе, вопросы подготовки и проведения имитационных экспериментов с распределенными моделями. Разработано техническое задание на систему.

3) На этапе технического проектирования с помощью диаграмм нотации UML показаны варианты использования системы, диаграмма классов, функционирование системы в процессе работы с распределенной моделью (диаграмма действий), а также последовательность добавления распределенных ресурсов и их типов в модель на РДО. Приведена структура программных средств с помощью диаграмм компонентов и развертывания системы. Разработаны алгоритмы подготовки распределенного приложения к вызову удаленных методов. На основе

информационной модели ресурсов РДО разработана информационная модель данных, передаваемых по сети.

4) На этапе рабочего проектирования подготовлено и настроено программное обеспечение, реализующее идеи CORBA, описаны интерфейсы распределенных объектов, по которым сгенерированы необходимые файлы с исходными кодами. В соответствии с поставленными практическими целями дипломного проекта была реализована законченная часть функционала системы: использование состояния распределенных ресурсов и их типов при компиляции модели на РДО. Проведено апробирование функционала системы на разработанном примере имитационной модели.

5) Проведено исследование вариантов обновления состояния ресурсов в распределенной системе с точки зрения уменьшения загрузки сети, проведены необходимые расчеты и получены результаты.

6) В организационно-экономической части дипломного проекта проведен расчет затрат на разработку распределенной многомодельной системы дискретного имитационного моделирования на основе РДО.

7) Приведены требования безопасности при работе пользователей с разработанной системой, а также проведен типовой расчет зануления ПЭВМ.

Все поставленные цели дипломного проекта достигнуты, требуемый функционал системы реализован.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Окольнишников В.В. Представление времени в имитационном моделировании // Вычисл. технологии. – 2005. – Т. 10, № 5. – С. 57 – 80, [http://www.gpss.ru].

2. RAO-Studio – Руководство пользователя, 2007 [http://rdo.rk9.bmstu.ru/forum/viewtopic.php?t=900].

3. Справка по языку РДО (в составе программы) [http://rdo.rk9.bmstu.ru/forum/viewforum.php?f=15].

4. Емельянов В.В., Ясиновский С.И. Имитационное моделирование систем: Учеб. пособие. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2009. – 584 с.: ил. (Информатика в техническом университете).

5. Современные технологии построения распределенных программных систем, Афанасьев А.П., Ваньков А.И., Волошинов В.В., Кривцов В.Е., Попков Е.Ю., Шляев П.Г. [http://www.isa.ru].

6. Common Object Request Broker Architecture (CORBA/IIOP) [http://www.omg.org].

7. Технология CORBA [http://kunegin.narod.ru].

8. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78.

9. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-90. Условные обозначения и правила выполнения.

10. Язык UML. Руководство пользователя [http://alice.pnzgu.ru/~dvn/uproc/books/uml_user_guide/index.htm].

11. «Использование CORBA», кафедра МО ЭВМ, Нижегородский государственный университет им Н.И. Лобачевского [http://ru.sun.com].
12. Ciaran McHale, CORBA Explained Simply [http://www.orbacus.com].
13. Microsoft Developer Network [http://msdn2.microsoft.com].
14. Документация на ПО omniORB-4.1.2, входящая в состав исходных кодов [http://omniorb.sourceforge.net/index.html].
15. Advanced CORBA® Programming with C++. Michi Henning, Steve Vinoski; Publisher: Addison Wesley; First Edition February 12, 1999; ISBN: 0-201-37927-9, 1120 pages.
16. Бьерн Страуструп. Язык программирования C++. Специальное издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2007 г. – 1104 с.: ил.
17. В.В. Арсеньев, Ю.Б. Сажин, Методические указания к выполнению организационно-экономической части дипломных проектов по созданию программной продукции. – М.: Изд-во МГТУ, 1994. – 52 с.
18. Гигиенические требования к персональным электронно-вычислительным машинам и организации работы . СанПиН 2.2.2/2.4.1340–03 [http://www.mhts.ru/biblio/].
19. Санитарные правила и нормы. Гигиенические требования к микроклимату производственных помещений. СанПиН 2.2.4.548-96 [http://www.mhts.ru/biblio/].
20. СанПиН 2.2.4.1294-03. Санитарно-эпидемиологические правила и нормативы. Гигиенические требования к аэроионному составу воздуха производственных и общественных помещений [http://www.mhts.ru/biblio/].
21. Предельно допустимые концентрации (ПДК) вредных веществ в воздухе рабочей зоны. ГН 2.2.5.1313–03[http://www.mhts.ru/biblio/].

22. Шум на рабочих местах, в помещениях жилых, общественных зданий и на территории жилой застройки. СанПиН 2.2.4/2.1.8.562–96 [http://www.mhts.ru/biblio/].
23. Производственные вибрации в помещениях жилых и общественных зданий. СН 2.2.4/2.1.8.566–96 [http://www.mhts.ru/biblio/].
24. Изделия электротехнические. ГОСТ 12.2.007.0-75 [http://www.mhts.ru/biblio/].
25. Система стандартов безопасности труда. Электробезопасность. Предельно допустимые значения напряжений прикосновения и токов. ГОСТ 12.1.038-82 [http://www.mhts.ru/biblio/].
26. Нормы пожарной безопасности «Определение категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности». НПБ 105-03 [http://www.mhts.ru/biblio/].
27. Сборник типовых расчетов по курсу «Охрана труда» для студентов факультета П. Под редакцией С.В. Белова. МВТУ, Москва, 1984г.

ПРИЛОЖЕНИЕ 1

Листинг файла описания интерфейсов распределенных объектов

RDOCorba.idl:

```
module rdoParse{
	interface RDOCorba{

		////////////////////////////////////Типы ресурсов
		//Постоянный/временный тип ресурса
		enum TypeRTP { rt_permanent, rt_temporary };
		//Тип параметра int, double, enum
		enum TypeParam { int_type, double_type,  enum_type };

		typedef sequence<string> my_enum;

		struct ParamRTP{
			//Имя параметра
			string m_name;
			//Тип параметра
			TypeParam m_type;

			//Минимальные и максимальные значения типа int
			boolean m_diap_int;
			long m_min_int;
			long m_max_int;
			//Значение по умолчанию типа int
			boolean m_default_int_ch;
			long m_default_int;

			//Минимальные и максимальные значения типа double
			boolean m_diap_double;
			double m_min_double;
			double m_max_double;
			//Значение по умолчанию типа double
			boolean m_default_double_ch;
			double m_default_double;

			//Количество значений перечислимого типа
			long m_var_enum_count;
			//Последовательность  значений перечислимого типа
			boolean m_var_enum_ch;
			my_enum m_var_enum;

			//Значение по умолчанию перечислимого типа
			boolean m_default_enum_ch;
			string m_default_enum;
		};

		typedef sequence<ParamRTP> ParamsRTP;

		struct MY_RTP{
			//Имя типа ресурса
			string m_name;
			//Тип ресурса
			TypeRTP m_type;
			//Количество параметров типа ресурсов
			long m_param_count;
			//Последовательность параметров
			ParamsRTP m_param;
		};

		typedef sequence<MY_RTP> GetRTP;

		//Количество типов ресурсов rtp_count;
		//Метод, возвращающий структуру с описанием типов ресурсов
		GetRTP getRDORTPlist(out long rtp_count);


		////////////////////////////////////Ресурсы
		struct ParamRSS{
			//Имя параметра
			string m_name;
			//Тип параметра
			TypeParam m_type;
```

```
                //Значение типа int
                long m_int;
                //Значение типа double
                double m_double;
                //Значение по умолчанию перечислимого типа
                string m_enum;
        };

        typedef sequence<ParamRSS> ParamsRSS;

        struct MY_RSS{
                //Имя ресурса
                string m_name;
                //Имя типа ресурса
                string m_type;
                //Количество параметров ресурса
                long m_param_count;
                //Последовательность значений параметров
                ParamsRSS m_param;
        };

        typedef sequence<MY_RSS> GetRSS;

        //Количество ресурсов rss_count;
        //Метод, возвращающий структуру со значениями параметров всех ресурсов
        GetRSS getRDORSSPlist(out long rss_count);

    };
};
```

ПРИЛОЖЕНИЕ 2

Листинг файла реализации серверной части приложения

rdosimwin.cpp:

```
#include <omniORB4/CORBA.h>

namespace rdoCorba
{
CORBA::ORB_var g_orb;

class RDOCorba_i: public POA_rdoParse::RDOCorba
{
public:
	inline RDOCorba_i() {}
	virtual ~RDOCorba_i() {}

	virtual rdoParse::RDOCorba::GetRTP* getRDORTPlist(::CORBA::Long& rtp_count);
	virtual rdoParse::RDOCorba::GetRSS* getRDORSSPlist(::CORBA::Long& rss_count);

	static CORBA::Boolean bindObjectToName(CORBA::ORB_ptr orb, CORBA::Object_ptr objref);
};

rdoParse::RDOCorba::GetRTP* RDOCorba_i::getRDORTPlist(::CORBA::Long& rtp_count)
{
	//Создаем список структур для хранения информации об искомых типах ресурсов
	rdoParse::RDOCorba::GetRTP_var my_rtpList = new rdoParse::RDOCorba::GetRTP;
	//Получаем необходимые нам данные о типах ресурсов РДО
	kernel->sendMessage(kernel->simulator(),RDOThread::RT_CORBA_PARSER_GET_RTP,&my_rtpList);

	return my_rtpList._retn();
}

rdoParse::RDOCorba::GetRSS* RDOCorba_i::getRDORSSPlist(::CORBA::Long& rss_count)
{
	//Создаем список структур для хранения информации об искомых ресурсах
	rdoParse::RDOCorba::GetRSS_var my_rssList = new rdoParse::RDOCorba::GetRSS;
	//Получаем необходимые нам данные о ресурсах РДО
	kernel->sendMessage(kernel->simulator(),RDOThread::RT_CORBA_PARSER_GET_RSS,&my_rssList);

	return my_rssList._retn();
}

CORBA::Boolean bindObjectToName(CORBA::ORB_ptr orb, CORBA::Object_ptr
	objref, const char* ModelName)
{
	CosNaming::NamingContext_var rootContext;

	// Obtain a reference to the root context of the Name service:
	CORBA::Object_var obj;
	obj = orb->resolve_initial_references("NameService");

	// Narrow the reference returned.
	rootContext = CosNaming::NamingContext::_narrow(obj);
	if( CORBA::is_nil(rootContext) ) {
		TRACE( "Failed to narrow the root naming context." );
			return 0;
	}

	// Bind a context called "RDO" to the root context:
	CosNaming::Name contextName;
	contextName.length(1);
	contextName[0].id = (const char*) "RDO";
	contextName[0].kind = (const char*) "RDO_context";

	CosNaming::NamingContext_var testContext;

	// Bind the context to root.
	testContext = rootContext->bind_new_context(contextName);

	// Bind objref with name Echo to the testContext:
	CosNaming::Name objectName;
	objectName.length(1);

	objectName[0].id = ModelName;
```

```
	objectName[0].kind = (const char*) "Object";

	testContext->bind(objectName, objref);
	return 1;
}

unsigned int RDOThreadCorba::corbaRunThreadFun( void* param )
{
	int argc = 0;
	g_orb = CORBA::ORB_init(argc, NULL);

	CORBA::Object_var obj = g_orb->resolve_initial_references("RootPOA");
	PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);

	RDOCorba_i* myrdocorba = new RDOCorba_i();

	PortableServer::ObjectId_var myrdocorbaid = poa->activate_object(myrdocorba);

	obj = myrdocorba->_this();

	//const char* ModelName = "ЦЕХ";
	//const char* ModelName = "СКЛАД";

	if( !bindObjectToName(g_orb, obj, ModelName) ) return 1;

	myrdocorba->_remove_ref();
	PortableServer::POAManager_var pman = poa->the_POAManager();

	pman->activate();
	g_orb->run();
	return 0;
}
void RDOThreadCorba::stop()
{
	// Место для остановки корбы
	if ( g_orb != CORBA::ORB::_nil() )
	{
		g_orb->shutdown( true );
		g_orb->destroy();
	}

} // namespace rdoCorba

namespace rdoSimulator
{

void RDOThreadSimulator::proc( RDOMessageInfo& msg )
{
	switch ( msg.message ) {
		case RT_CORBA_PARSER_GET_RTP: {
			msg.lock();
			corbaGetRTP(*static_cast<rdoParse::
				RDOCorba::GetRTP_var*>(msg.param) );
			msg.unlock();
			break;
		}
		case RT_CORBA_PARSER_GET_RSS: {
			msg.lock();
			corbaGetRSS(*static_cast<rdoParse::
				RDOCorba::GetRSS_var*>(msg.param) );
			msg.unlock();
			break;
		}
	}
}

void RDOThreadSimulator::corbaGetRTP( rdoParse::RDOCorba::GetRTP_var& my_rtpList )
{
	// Пропарсели типы и ресурсы текста модели (текущие, а не записанные)
	rdoParse::RDOParserCorba parser;
	try {
		parser.parse();
	}
	catch ( rdoParse::RDOSyntaxException& ) {
	}
	catch ( rdoRuntime::RDORuntimeException& ) {
	}
```

```
::CORBA::Long i = 0, j = 0, rtp_count = 0;

// Пробежались по всем типам и переписали в RTPList
rdoMBuilder::RDOResTypeList rtpList( &parser );

//Считаем количество типов ресурсов
rdoMBuilder::RDOResTypeList::List::const_iterator rtp_it = rtpList.begin();

while ( rtp_it != rtpList.end() )
{
      rtp_count++; rtp_it++;
}

//Выделяем пямять под последовательность
my_rtpList->length( rtp_count );
//Снова возвращаемся в начало списка типов ресурсов
rtp_it = rtpList.begin();

while ( rtp_it != rtpList.end() )
{
      // Создаем текстовую структуру
      my_rtpList[i].m_name = CORBA::string_dup(
            rtp_it->name().c_str());

      if ((rtp_it->getType()) == rdoMBuilder::RDOResType::rt_permanent )
            my_rtpList[i].m_type=rdoParse::RDOCorba::rt_permanent;
      else
            my_rtpList[i].m_type=rdoParse::RDOCorba::rt_temporary;

      //Считаем количество параметров i-го типа ресурса
      rdoMBuilder::RDOResType::ParamList::List::const_iterator param_it
            = rtp_it->m_params.begin();

      my_rtpList[i].m_param_count = 0;

      while ( param_it != rtp_it->m_params.end() )
      {
            my_rtpList[i].m_param_count++;
            param_it++;
      }

      //Выделяем память под последовательность параметров i-го типаресурсов
      my_rtpList[i].m_param.length(my_rtpList[i].m_param_count);

      //Снова возвращаемся в начало списка параметров i-го типа ресурсов
      param_it = rtp_it->m_params.begin();

      while ( param_it != rtp_it->m_params.end() )
      {
            // Добавляем в структуру параметр!
            my_rtpList[i].m_param[j].m_name=CORBA::string_dup(param_it->name().c_str());

            my_rtpList[i].m_param[j].m_diap_int = 0;
            my_rtpList[i].m_param[j].m_default_int_ch = 0;
            my_rtpList[i].m_param[j].m_diap_double = 0;
            my_rtpList[i].m_param[j].m_default_double_ch = 0;
            my_rtpList[i].m_param[j].m_var_enum_ch = 0;
            my_rtpList[i].m_param[j].m_default_enum_ch = 0;
            my_rtpList[i].m_param[j].m_var_enum_count = 0;

            switch (param_it->typeID())
            {
                  case rdoRuntime::RDOType::t_int:{
                        my_rtpList[i].m_param[j].m_type = rdoParse::RDOCorba::int_type;

                        if ( param_it->hasDiap() )
                        {
                              my_rtpList[i].m_param[j].m_min_int = param_it->
                                    getMin().getInt();
                              my_rtpList[i].m_param[j].m_max_int = param_it->
                                    getMax().getInt();
                              my_rtpList[i].m_param[j].m_diap_int = 1;

                        }
                        if ( param_it->hasDefault() )
                        {
```

```
                                    my_rtpList[i].m_param[j].m_default_int =
                                          param_it->getDefault().getInt();
                                    my_rtpList[i].m_param[j].m_default_int_ch = 1;
                              }
                              break;
                        }
                        case rdoRuntime::RDOType::t_real:{my_rtpList[i].m_param[j].m_type =
                                    rdoParse::RDOCorba::double_type;

                              if ( param_it->hasDiap() )
                              {
                                    my_rtpList[i].m_param[j].m_min_double =
                                          param_it->getMin().getDouble();
                                    my_rtpList[i].m_param[j].m_max_double =
                                          param_it->getMax().getDouble();
                                    my_rtpList[i].m_param[j].m_diap_double = 1;
                              }
                              if ( param_it->hasDefault() )
                              {
                                    my_rtpList[i].m_param[j].m_default_double =
                                          param_it->getDefault().getDouble();
                                    my_rtpList[i].m_param[j].m_default_double_ch=1;
                              }
                              break;
                        }
                        case rdoRuntime::RDOType::t_enum:{my_rtpList[i].m_param[j].m_type =
                                    rdoParse::RDOCorba::enum_type;

                              //Считаем количество значений перечислимого типа
                              rdoRuntime::RDOEnumType::CIterator enum_it =
                                    param_it->getEnum().begin();

                              CORBA::Long k = 0;

                              while ( enum_it != param_it->getEnum().end() )
                              {
                                    k++; enum_it++;
                              }

                              //Выделяем память под последовательность значений
                                    j-го параметра перечислимого типа i-го типа ресурсов
                              my_rtpList[i].m_param[j].m_var_enum.length(k);

                              enum_it = param_it->getEnum().begin();
                              k = 0;

                              while ( enum_it != param_it->getEnum().end() )
                              {
                                    my_rtpList[i].m_param[j].m_var_enum[k] =
                                          CORBA::string_dup( enum_it->c_str() );
                                    enum_it++;
                                    k++;
                              }

                              if ( param_it->hasDefault() )
                              {
                                    my_rtpList[i].m_param[j].m_default_enum =
                                          CORBA::string_dup( param_it->
                                                getDefault().getAsString().c_str() );
                                    my_rtpList[i].m_param[j].m_default_enum_ch = 1;
                              }

                              my_rtpList[i].m_param[j].m_var_enum_ch = 1;
                              my_rtpList[i].m_param[j].m_var_enum_count = k;
                              break;
                        }
                        default: break;
                  }
                  j++; param_it++;
            }
            j = 0; i++; rtp_it++;
      }
}

void RDOThreadSimulator::corbaGetRSS( rdoParse::RDOCorba::GetRSS_var& my_rssList )
{
      // Пропарсели типы и ресурсы текста модели (текущие, а не записанные)
```

```
	rdoParse::RDOParserCorba parser;
	parser.parse();

	// Пробежались по всем ресурсам и переписали в RSSList
	rdoMBuilder::RDOResourceList rssList( &parser );
	rdoMBuilder::RDOResourceList::List::const_iterator rss_it = rssList.begin();

	::CORBA::Long i = 0, j = 0, rss_count = 0;

	//Считаем количество ресурсов
	while ( rss_it != rssList.end() )
	{
		rss_count++;		rss_it++;
	}

	//Выделяем пямять под последовательность
	my_rssList->length( rss_count );
	//Снова возвращаемся в начало списка типов ресурсов
	rss_it = rssList.begin();

	while ( rss_it != rssList.end() )
	{
		// Заполняем значения структуры
		my_rssList[i].m_name = CORBA::string_dup( rss_it->name().c_str() );
		my_rssList[i].m_type = CORBA::string_dup( rss_it->getType().name().c_str() );

		//Считаем количество параметров i-го типа ресурса
		rdoMBuilder::RDOResource::Params::const_iterator param_it = rss_it->begin();

		my_rssList[i].m_param_count = 0;

		while ( param_it != rss_it->end() )
		{
			my_rssList[i].m_param_count++;
			param_it++;
		}

		//Выделяем память под последовательность параметров i-го ресурса
		my_rssList[i].m_param.length(my_rssList[i].m_param_count);
		//Снова возвращаемся в начало списка параметров i-го ресурса
		param_it = rss_it->begin();

		while ( param_it != rss_it->end() )
		{
			my_rssList[i].m_param[j].m_name = CORBA::string_dup(
				param_it->first.c_str() );;

			switch (param_it->second.typeID())
			{
				case rdoRuntime::RDOType::t_int:{
					my_rssList[i].m_param[j].m_int = param_it->second.getInt();
					my_rssList[i].m_param[j].m_type = rdoParse::RDOCorba::int_type;
					break;
				}
				case rdoRuntime::RDOType::t_real:{
					my_rssList[i].m_param[j].m_double=param_it->second.getDouble();
					my_rssList[i].m_param[j].m_type=rdoParse::RDOCorba::double_type;
					break;
				}
				case rdoRuntime::RDOType::t_enum:{
					my_rssList[i].m_param[j].m_enum = param_it->
						second.getAsString().c_str();
					my_rssList[i].m_param[j].m_type = rdoParse::RDOCorba::enum_type;
					break;
				}
				default: break;
			}
			param_it++; j++;
		}
		j = 0; rss_it++; i++;
	}
}
} // namespace rdoSimulator
```

ПРИЛОЖЕНИЕ 3

Листинг файла реализации клиентской части приложения

rdoparser_corba.cpp:

```
namespace rdoParse
{
static CORBA::Object_ptr getObjectReference(CORBA::ORB_ptr orb, const char* ObjectName)
{
	CosNaming::NamingContext_var rootContext;

	CORBA::Object_var obj;
	obj = orb->resolve_initial_references("NameService");

	// Narrow the reference returned.
	rootContext = CosNaming::NamingContext::_narrow(obj);

	if( CORBA::is_nil(rootContext) ) {
		TRACE( "Failed to narrow the root naming context.\n" );
		return CORBA::Object::_nil();
	}

	// Create a name object, containing the name test/context:
	CosNaming::Name name;
	name.length(2);
	name[0].id = (const char*) "RDO";
	name[0].kind = (const char*) "RDO_context";
	name[1].id = (const char*) ObjectName;
	name[1].kind = (const char*) "Object";

	// Resolve the name to an object reference.
	return rootContext->resolve(name);
	//return CORBA::Object::_nil();
}

void RDOParserCorbaRTP::parse()
{
	// Тут надо запросить все типы ресурсов у парного РДО, вызвав с помощью корбы некий
		метод, который вернёт кучу структур с описанием RTP и насоздавать этих типов
	rdoParse::RDOParserSMRInfo parser;
	parser.parse();

	rdoParse::RDOSMR::StringTable tmp = parser.getSMR()->getExternalModelList();
	rdoParse::RDOSMR::StringTable::const_iterator it = tmp.begin();

	const char* left, right;

	while ( it != tmp.end() )
	{
		left = it->first.c_str();
		right = it->second.c_str();

		int argc = 0;

		CORBA::ORB_var orb = CORBA::ORB_init(argc, NULL);
		CORBA::Object_var obj = getObjectReference(orb, left);
		rdoParse::RDOCorba_var rdocorbaref = rdoParse::RDOCorba::_narrow(obj);

		CORBA::Long rtp_count = 0;

		rdoParse::RDOCorba::GetRTP_var tmp_rtp = rdocorbaref->getRDORTPlist( rtp_count );
		rdoParse::RDOCorba::GetRTP_var my_rtpList ( tmp_rtp );

		//Добавляем к существующим типам ресурсов и выводим в трассировке
		// Получили список всех описанных типов ресурсов
		rdoMBuilder::RDOResTypeList rtpList( m_parser );

		for (unsigned int i = 0; i < my_rtpList->length(); i++)
		{
			rdoMBuilder::RDOResType rtp( my_rtpList[i].m_name.in());
			// Наполняем его параметрами

			for (unsigned int j = 0; j !=
					my_rtpList[i].m_param_count; j++)
			{
```

```
                switch ( my_rtpList[i].m_param[j].m_type ){
                      case rdoParse::RDOCorba::int_type:{

                            rdoMBuilder::RDOResType::Param par_int(
                            my_rtpList[i].m_param[j].m_name.in() , rdoRuntime::g_int );

                            if ( my_rtpList[i].m_param[j].m_diap_int == 1 )
                                  par_int.setDiap (my_rtpList[i].m_param[j].m_min_int ,
                                   my_rtpList[i].m_param[j].m_max_int);

                            if (my_rtpList[i].m_param[j].m_default_int_ch
                                  == 1 )
                                  par_int.setDefault(
                                  my_rtpList[i].m_param[j].m_default_int );

                            rtp.m_params.append( par_int );
                            break;
                      }
                      case rdoParse::RDOCorba::double_type:{

                            rdoMBuilder::RDOResType::Param par_double(
                            my_rtpList[i].m_param[j].m_name.in(), rdoRuntime::g_real);

                            if (my_rtpList[i].m_param[j].m_diap_double==1)
                                  par_double.setDiap
                                  (my_rtpList[i].m_param[j].m_min_double ,
                                  my_rtpList[i].m_param[j].m_max_double);

                            if (my_rtpList[i].m_param[j].m_default_double_ch == 1 )
                                  par_double.setDefault( my_rtpList[i].m_param[j].
                                  m_default_double);

                            rtp.m_params.append( par_double );
                            break;
                      }
                      case rdoParse::RDOCorba::enum_type:{

                            //Создадим список значений параметра перечислимого типа
                            rdoRuntime::RDOEnumType::Enums enumList;

                            for (CORBA::Long k = 0; k <
                                  my_rtpList[i].m_param[j].m_var_enum_count; k++)
                            {
                                  enumList.push_back(my_rtpList[i].
                                  m_param[j].m_var_enum[k].pd_data );
                            }

                            // Создадим параметр перечислимого типа
                            rdoMBuilder::RDOResType::Param par_enum(
                            my_rtpList[i].m_param[j].m_name.in() ,
                            new rdoRuntime::RDOEnumType(m_parser->runtime(),enumList));

                            //Добавляем, если есть значение по умолчанию
                            if(my_rtpList[i].m_param[j].m_default_enum_ch==1)
                            {
                                  std::string temp_string;
                                  temp_string = my_rtpList[i].
                                        m_param[j].m_default_enum.in();
                                  par_enum.setDefault( temp_string );
                            }
                                  rtp.m_params.append( par_enum );
                            break;
                      }
                      default: break;
                }
          }

          //Добавляем к существующим типам ресурсов
          if ( rtpList.append( rtp ) )
                // Добавили успешно
                TRACE( "Еще один тип ресурсов добавился!!!\n" );
          else
                // Неудача, возможно, тип с таким именем уже есть
                TRACE( "Где-то есть ошибка или тип ресурса уже существует!!!\n" );
    }

    CORBA::Long rss_count = 0;
```

```
            rdoParse::RDOCorba::GetRSS_var tmp_rss = rdocorbaref->getRDORSSPlist( rss_count );
            rdoParse::RDOCorba::GetRSS_var my_rssList ( tmp_rss );

            //Добавляем к существующим ресурсам и выводим в трассировке
            // Переписали имеющиеся ресурсы в rssList
            rdoMBuilder::RDOResourceList rssList( m_parser );

            for (unsigned int i = 0; i < my_rssList->length(); i++)
            {
                  // Нашли тип ресурса по известному имени и создали ресурс
                        указанного типа
                  rdoMBuilder::RDOResType _rtp = rtpList[my_rssList[i].m_type.in()];
                  rdoMBuilder::RDOResource rss( _rtp , my_rssList[i].m_name.in() );

                  for (unsigned int j=0; j!= my_rssList[i].m_param_count; j++)
                  {
                        //Записываем начальные значения параметров ресурса
                        switch ( my_rssList[i].m_param[j].m_type ){
                              case rdoParse::RDOCorba::int_type:{
                                    rss[ my_rssList[i].m_param[j].m_name.in()] =
                                          my_rssList[i].m_param[j].m_int;
                                    break;
                              }
                              case rdoParse::RDOCorba::double_type:{
                                    rss[ my_rssList[i].m_param[j].m_name.in()] =
                                          my_rssList[i].m_param[j].m_double;
                                    break;
                              }
                              case rdoParse::RDOCorba::enum_type:{
                                    std::string temp_string;
                                    temp_string = my_rssList[i].m_param[j].m_enum.in();
                                    rss[ my_rssList[i].m_param[j].m_name.in()] = temp_string;
                                    break;
                              }
                              default: break;
                        }
                  }
                  //Добавляем к существующим ресурсам
                  if ( rssList.append<rdoParse::RDORSSResource>( rss ) )
                        TRACE( "Еще один ресурс добавился!!!\n" );
                  else
                        TRACE( "Где-то есть ошибка или ресурс уже существует!!!\n" );

            }
            orb->destroy();
            it++;
      }
}
} // namespace rdoParse
```

ПРИЛОЖЕНИЕ 4

Код имитационной модели склада на языке РДО

СКЛАД.rtp (Типы ресурсов):

```
$Resource_type Накопители : permanent
$Parameters
      положение                     : integer
      максимальное_количество       : integer = 10
      текущее_количество            : integer = 0
$End
$Resource_type Детали : permanent
$Parameters
      номер             : integer
      положение         : integer = 15//не_существует
      состояние :(хранится, транспортируется, обрабатывается, обработка_закончена) = хранится
$End

$Resource_type Роботы_складские : permanent
$Parameters
      положение                     : integer
      состояние:(свободен, занят, загружен, перемещается_туда, перемещается_обратно)= свободен
      вид_робота                    : (выходной, входной)
      время_возврата                : real = 0
      начальное_время               : real = 0
      время_движения                : real = 0
      флаг_движения                 : real = 0
      флаг_движения_обратно         : real = 0
$End

$Resource_type Тележки : permanent
$Parameters
      положение                     : integer
      состояние: (свободен, занят, загружен, перемещается_туда, перемещается_обратно, прибыл,
                        ожидает) = свободен
      вид                           : (складская, цеховая)
      начальное_время               : real = 0
      время_движения                : real = 0
      флаг_движения                 : real = 0
      флаг_движения_обратно         : real = 0
$End
```

СКЛАД.rss (Ресурсы):

```
$Resources
      Накопитель_склад_out    : Накопители trace 7 * *
      Накопитель_цех          : Накопители trace 8 * *
      Накопитель_склад_in     : Накопители trace 6 * *
      Накопитель_цех_in       : Накопители trace 8 * *
      Накопитель_цех_out      : Накопители trace 9 * *

      Робот_на_выходе         : Роботы_складские trace 7 * выходной 4.0 * * * *
      Робот_на_входе          : Роботы_складские trace 6 * входной 7.0 * * * *

      Тележка_склад           : Тележки trace 10 * складская * * * *
      Тележка_цех             : Тележки trace 12 * цеховая * * * *

      Деталь_1                : Детали    1 * *
      Деталь_2                : Детали    2 * *
      Деталь_3                : Детали    3 * *
      Деталь_4                : Детали    4 * *
      Деталь_5                : Детали    5 * *
      Деталь_6                : Детали    6 * *
      Деталь_7                : Детали    7 * *
      Деталь_8                : Детали    8 * *
      Деталь_9                : Детали    9 * *
      Деталь_10               : Детали    10 * *
      Деталь_11               : Детали    11 * *
      Деталь_12               : Детали    12 * *
      Деталь_13               : Детали    13 * *
      Деталь_14               : Детали    14 * *
      Деталь_15               : Детали    15 * *
      Деталь_16               : Детали    16 * *
      Деталь_17               : Детали    17 * *
      Деталь_18               : Детали    18 * *
```

```
      Деталь_19               : Детали    19 * *
      Деталь_20               : Детали    20 * *
$End
```

СКЛАД.fun (Константы, последовательности и функции):

```
$Function Длительность_операции_тележки : real = 0
$Type = table
$Parameters
      вид_тележки  : such_as Тележки.вид
      тип_операции : ( погрузка, доставка, разгрузка, возврат)
$Body
      { вид_тележки =   складская   цеховая }
      { тип_операции }

      { погрузка  }        7.0           2.0
      { доставка  }        5.0           15.0
      { разгрузка }        2.0           2.0
      { возврат }          3.0           8.0
$End

$Function Длительность_операции_робота : real = 0
$Type = table
$Parameters
      вид_робота  : such_as Роботы_складские.вид_робота
      тип_операции : (погрузка, разгрузка)
$Body
      { вид_робота       =    выходной    входной }
      { тип_операции }
      { погрузка  }            7.0         17.0
      { разгрузка }            7.0         17.0
$End

$Function где_тележка : integer = 10
$Type = list
$Parameters
      вид_тележки  : such_as Тележки.вид
      тип_операции : (погрузка, разгрузка)
$Body
      цеховая   погрузка  = 12
      складская погрузка  = 10
      цеховая   разгрузка = 13
      складская разгрузка = 11
$End

$Function Время_возврата : real = 0
$Type = algorithmic
$Parameters
      вид_робота  : such_as Роботы_складские.вид_робота
$Body
      Calculate_if вид_робота = выходной
            Время_возврата = Робот_на_выходе.время_возврата

      Calculate_if вид_робота = входной
            Время_возврата = Робот_на_входе.время_возврата
$End

$Function ПОЛОЖЕНИЕ : integer = 0
$Type = list
$Parameters
      положение_ : (станок_1, станок_2, робот_1, робот_2, цеховой_робот,
                  накопитель_склад_in, накопитель_склад_out, накопитель_цех_in,
                  накопитель_цех_out, тележка_склад_н, тележка_склад_к, тележка_цех_н,
                  тележка_цех_к, нигде, не_существует, в_пути)
$Body
      станок_1                = 1
      станок_2                = 2
      робот_1                 = 3
      робот_2                 = 4
      цеховой_робот           = 5
      накопитель_склад_in     = 6
      накопитель_склад_out    = 7
      накопитель_цех_in       = 8
      накопитель_цех_out      = 9
      тележка_склад_н         = 10
      тележка_склад_к         = 11
      тележка_цех_н           = 12
```

```
      тележка_цех_к           = 13
      нигде                   = 14
      не_существует           = 15
      в_пути                  = 16
$End
```

СКЛАД.pat (Образцы операций):

```
$Pattern Добавление_детали: keyboard
$Relevant_resources
      накопитель_       : Накопители Keep NoChange
      деталь_           : Детали Keep NoChange
$Time = 0.0
$Body
      накопитель_
            Choice from накопитель_.положение = ПОЛОЖЕНИЕ(накопитель_склад_out)
            first
            Convert_begin
                  текущее_количество set накопитель_.текущее_количество + 1
      деталь_
            Choice from деталь_.положение = ПОЛОЖЕНИЕ(не_существует)
            first
            Convert_begin
                  положение set ПОЛОЖЕНИЕ(накопитель_склад_out)
                  состояние set хранится
$End

$Pattern Погрузка_складским_роботом :  operation trace
$Parameters
      положение_накопителя    : integer
      вид_тележки             : such_as Тележки.вид
      вид_робота              : such_as Роботы_складские.вид_робота
$Relevant_resources
      накопитель_       : Накопители Keep NoChange
      деталь_           : Детали Keep Keep
      робот_            : Роботы_складские Keep Keep
      тележка_          : Тележки Keep Keep
$Time = Длительность_операции_робота (вид_робота, погрузка)
$Body
      накопитель_
            Choice from накопитель_.положение = положение_накопителя and
                  накопитель_.текущее_количество > 0
            first
            Convert_begin
                  текущее_количество set накопитель_.текущее_количество - 1
      деталь_
            Choice from деталь_.положение = положение_накопителя
            first
            Convert_begin
                  положение set ПОЛОЖЕНИЕ(нигде)
                  состояние set транспортируется
            Convert_end
                  положение set где_тележка(вид_тележки, погрузка)
      робот_
            Choice from робот_.положение=положение_накопителя and робот_.состояние = свободен
            first
            Convert_begin
                  состояние set перемещается_туда
                  положение set ПОЛОЖЕНИЕ(в_пути)
                  начальное_время set Time_now
                  флаг_движения set 1
            Convert_end
                  состояние set свободен
                  положение set где_тележка(вид_тележки, погрузка)
                  начальное_время set 0
                  флаг_движения set 0
      тележка_
            Choice from тележка_.положение = где_тележка(вид_тележки,погрузка)
                        and тележка_.состояние = свободен
            first
            Convert_begin
                  состояние set занят
            Convert_end
                  состояние set загружен

$End
```

```
$Pattern Погрузка_детали :  operation trace
$Parameters
      положение_накопителя    : integer
      вид_тележки              : such_as Тележки.вид
$Relevant_resources
      накопитель_       : Накопители Keep NoChange
      деталь_           : Детали Keep Keep
      тележка_          : Тележки Keep Keep
$Time = Длительность_операции_тележки(вид_тележки, погрузка)
$Body
      накопитель_
            Choice from накопитель_.положение = положение_накопителя and
                        накопитель_.текущее_количество > 0
            first
            Convert_begin
                  текущее_количество set накопитель_.текущее_количество - 1
      деталь_
            Choice from деталь_.положение = положение_накопителя
            first
            Convert_begin
                  положение set ПОЛОЖЕНИЕ(нигде)
                  состояние set транспортируется
            Convert_end
                  положение set где_тележка(вид_тележки, погрузка)
      тележка_
            Choice from тележка_.положение = где_тележка(вид_тележки,погрузка)
                                    and тележка_.состояние = свободен
            first
            Convert_begin
                  состояние set занят
            Convert_end
                  состояние set загружен
$End

$Pattern Возврат_складского_робота :  operation trace
$Parameters
      вид_робота                          : such_as Роботы_складские.вид_робота
      начальное_положение_устройства      : integer
      конечное_положение_устройства       : integer
$Relevant_resources
      робот_                              : Роботы_складские Keep Keep
$Time = Время_возврата (вид_робота)
$Body
      робот_
            Choice from робот_.положение = начальное_положение_устройства
                        and робот_.состояние = свободен
            first
            Convert_begin
                  состояние set перемещается_обратно
                  начальное_время set Time_now
                  флаг_движения set 1
                  положение set ПОЛОЖЕНИЕ(в_пути)
            Convert_end
                  состояние set свободен
                  положение set конечное_положение_устройства
                  начальное_время set 0
                  флаг_движения set 0
$End

$Pattern Доставка_детали :  operation trace
$Parameters
      вид_тележки                         : such_as Тележки.вид
      начальное_положение_устройства      : integer
      конечное_положение_устройства       : integer
$Relevant_resources
      тележка_          : Тележки Keep Keep
      деталь_           : Детали NoChange Keep
$Time = Длительность_операции_тележки(вид_тележки, доставка)
$Body
      тележка_
            Choice from тележка_.положение = начальное_положение_устройства
                        and тележка_.состояние = загружен
            first
            Convert_begin
                  состояние set перемещается_туда
                  начальное_время set Time_now
                  флаг_движения set 1
```

```
                  положение set ПОЛОЖЕНИЕ(в_пути)
            Convert_end
                  положение set конечное_положение_устройства
                  состояние set прибыл
                  начальное_время set 0
                  флаг_движения set 0
      деталь_
            Choice from деталь_.положение = начальное_положение_устройства
            first
            Convert_end
                  положение set конечное_положение_устройства

$End

$Pattern Разгрузка_складским_роботом :  operation trace
$Parameters
      положение_накопителя    : integer
      вид_тележки             : such_as Тележки.вид
      вид_робота              : such_as Роботы_складские.вид_робота
$Relevant_resources
      деталь_           : Детали Keep NoChange
      робот_            : Роботы_складские Keep Keep
      тележка_          : Тележки Keep Keep
$Time = Длительность_операции_робота(вид_робота, разгрузка)
$Body
      деталь_
            Choice from деталь_.положение = где_тележка(вид_тележки,разгрузка)
            first
            Convert_begin
                  положение set ПОЛОЖЕНИЕ(робот_2)
                  состояние set транспортируется
      робот_
            Choice from робот_.положение=положение_накопителя and робот_.состояние = свободен
            first
            Convert_begin
                  состояние set перемещается_туда
                  положение set ПОЛОЖЕНИЕ(в_пути)
                  начальное_время set Time_now
                  флаг_движения set 1
            Convert_end
                  состояние set свободен
                  положение set где_тележка(вид_тележки, разгрузка)
                  начальное_время set 0
                  флаг_движения set 0
      тележка_
            Choice from тележка_.положение = где_тележка(вид_тележки,разгрузка)
                                             and тележка_.состояние = прибыл
            first
            Convert_begin
                  состояние set ожидает
            Convert_end
                  состояние set свободен
$End

$Pattern Разгрузка_тележки :  operation trace
$Parameters
      положение_накопителя    : integer
      вид_тележки             : such_as Тележки.вид

$Relevant_resources
      накопитель_       : Накопители NoChange Keep
      деталь_           : Детали Keep Keep
      тележка_          : Тележки Keep Keep
$Time = Длительность_операции_тележки(вид_тележки, разгрузка)
$Body
      накопитель_
            Choice from накопитель_.положение = положение_накопителя and
                  накопитель_.текущее_количество < накопитель_.максимальное_количество
            first
            Convert_end
                  текущее_количество set накопитель_.текущее_количество + 1
      деталь_
            Choice from деталь_.положение = где_тележка(вид_тележки, разгрузка)
            first
            Convert_begin
                  положение set ПОЛОЖЕНИЕ(нигде)
                  состояние set транспортируется
```

```
            Convert_end
                  положение set положение_накопителя
      тележка_
            Choice from тележка_.положение = где_тележка(вид_тележки,разгрузка) and
                                                тележка_.состояние = прибыл
            first
            Convert_begin
                  состояние set ожидает
            Convert_end
                  состояние set свободен
$End

$Pattern Возврат_тележки :  operation trace
$Parameters
      вид_тележки                         : such_as Тележки.вид
      начальное_положение_устройства      : integer
      конечное_положение_устройства       : integer
$Relevant_resources
      тележка_          : Тележки Keep Keep
$Time = Длительность_операции_тележки(вид_тележки, возврат)
$Body
      тележка_
            Choice from тележка_.положение = начальное_положение_устройства
                        and тележка_.состояние = свободен
            first
            Convert_begin
                  состояние set перемещается_обратно
                  начальное_время set Time_now
                  флаг_движения set 1
                  положение set ПОЛОЖЕНИЕ(в_пути)
            Convert_end
                  состояние set свободен
                  положение set конечное_положение_устройства
                  начальное_время set 0
                  флаг_движения set 0
$End

$Pattern Продвижение_времени : irregular_event trace
$Parameters
      значение                : integer
$Relevant_resources
      тележка_1               : Тележка_склад Keep
      тележка_2               : Тележка_цех Keep
      робот_1                 : Робот_на_входе Keep
      робот_2                 : Робот_на_выходе Keep
$Time = 0.1
$Body
      тележка_1
            Convert_event
                  время_движения set тележка_1.время_движения + 0.1
      тележка_2
            Convert_event
                  время_движения set тележка_2.время_движения + 0.1
      робот_1
            Convert_event
                  время_движения set робот_1.время_движения + 0.1
      робот_2
            Convert_event
                  время_движения set робот_2.время_движения + 0.1
$End

$Pattern Добавление_детали_: rule
$Relevant_resources
      накопитель_       : Накопители Keep
      робот_            : Роботы_складские Keep
      деталь_           : Детали Keep
$Body
      накопитель_
            Choice from накопитель_.положение = ПОЛОЖЕНИЕ(накопитель_склад_in) and
                        накопитель_.текущее_количество < накопитель_.максимальное_количество
            first
            Convert_rule
                  текущее_количество set накопитель_.текущее_количество + 1
      робот_
            Choice from робот_.положение = ПОЛОЖЕНИЕ(накопитель_склад_in) and
                                          робот_.состояние = свободен
            first
```

```
            Convert_rule
                  состояние set свободен
      деталь_
            Choice from деталь_.положение = ПОЛОЖЕНИЕ(робот_2) and
                                          деталь_.состояние = транспортируется
            first
            Convert_rule
                  положение set ПОЛОЖЕНИЕ(накопитель_склад_in)
                  состояние set хранится
$End
```

СКЛАД.opr (Операции):

```
$Operations

      Таймер                        : Продвижение_времени 0
      Добавление_на_склад_out       : Добавление_детали 'SPACE'
      Возврат_выходного_робота      : Возврат_складского_робота выходной 10 7
      Возврат_входного_робота       : Возврат_складского_робота  входной 13 6

      Погрузка_на_складе            : Погрузка_складским_роботом 7 складская выходной
      Погрузка_в_цеху               : Погрузка_детали 8 цеховая
      Доставка_детали_в_цех         : Доставка_детали складская 10 11
      Доставка_детали_на_склад      : Доставка_детали цеховая 12 13

      Разгрузка_в_цеху              : Разгрузка_тележки 8 складская
      Разгрузка_на_складе           : Разгрузка_складским_роботом 6 цеховая входной

      Возврат_тележки_на_склад      : Возврат_тележки складская 11 10
      Возврат_тележки_в_цех         : Возврат_тележки цеховая 13 12
      Добавление_на_склад_in        : Добавление_детали_

$End
```

СКЛАД.pmd (Показатели):

```
$Results
      Занятость_выходного_погрузчика      : watch_state Робот_на_выходе.состояние =
            перемещается_туда or Робот_на_выходе.состояние = перемещается_обратно
      Занятость_входного_погрузчика : watch_state Робот_на_входе.состояние = перемещается_туда
            or Робот_на_входе.состояние = перемещается_обратно
      Простой_выходной_тележки            : watch_state Тележка_склад.состояние = свободен
      Простой_входной_тележки             : watch_state Тележка_цех.состояние = свободен
$End
```

СКЛАД.smr (Прогон):

```
Model_name     = СКЛАД
Resource_file  = СКЛАД
OprIev_file    = СКЛАД

Results_file   = СКЛАД
Trace_file     = СКЛАД
Frame_file     = СКЛАД
Statistic_file = СКЛАД
Frame_number   = 1
Show_mode      = Animation
Show_rate      = 5000.0

Terminate_if Накопитель_склад_in.текущее_количество = 10
```

ПРИЛОЖЕНИЕ 5

Код имитационной модели цеха на языке РДО

ЦЕХ.rtp (Типы ресурсов):

```
$Resource_type Станки : permanent
$Parameters
      номер             : integer
      положение         : integer
      состояние         : (свободен, загружается, готов_к_обработке, работает, разгружается,
                                                          закончил_обработку) = свободен
      время_работы : real
$End

$Resource_type Роботы_цеховые : permanent
$Parameters
      положение                     : integer = 8//накопитель_цех_in
      состояние   : (свободен, занят, прибыл, перемещается_пустой, перемещается_туда,
                        готов_к_разгрузке, разгружает, перемещается_обратно) = свободен
      время_возврата                : real = 0
      начальное_время               : real = 0
      время_движения                : real = 0
      флаг_движения                 : real = 0
      флаг_движения_обратно         : real = 0
$End
```

ЦЕХ.rss (Ресурсы):

```
$Resources
      Цеховой_робот     : Роботы_цеховые trace * * 5.0 * * * *
      Станок_1          : Станки trace 1 1 * 30
      Станок_2          : Станки trace 2 2 * 40
$End
```

СКЛАД.fun (Константы, последовательности и функции):

```
$Function ПОЛОЖЕНИЕ : integer = 0
(аналогична одноименной функции склада)
$End
```

СКЛАД.pat (Образцы операций):

```
$Pattern Добавление_детали: keyboard
$Relevant_resources
      накопитель_       : Накопители Keep NoChange
      деталь_           : Детали Keep NoChange
$Time = 0.0
$Body
      накопитель_
            Choice from накопитель_.положение = ПОЛОЖЕНИЕ(накопитель_цех_in)
            first
            Convert_begin
                  текущее_количество set накопитель_.текущее_количество + 1
      деталь_
            Choice from деталь_.положение = ПОЛОЖЕНИЕ(не_существует)
            first
            Convert_begin
                  положение set ПОЛОЖЕНИЕ(накопитель_цех_in)
                  состояние set хранится
$End

$Pattern Установка_на_станке :  operation trace
$Parameters
      положение_устройства : integer
$Relevant_resources
      станок_           : Станки Keep Keep
      накопитель_       : Накопители Keep NoChange
      робот_            : Роботы_цеховые Keep Keep
      деталь_           : Детали Keep Keep
$Time = Время_установки_на_станок
$Body
      станок_
            Choice from станок_.положение=положение_устройства and станок_.состояние=свободен
            first
            Convert_begin
```

```
                  состояние set загружается
            Convert_end
                  состояние set готов_к_обработке

      накопитель_
            Choice from накопитель_.положение = ПОЛОЖЕНИЕ(накопитель_цех_in)
                        and накопитель_.текущее_количество > 0
            first
            Convert_begin
                  текущее_количество set накопитель_.текущее_количество - 1
      робот_
            Choice from робот_.положение = ПОЛОЖЕНИЕ(накопитель_цех_in) and
                                                            робот_.состояние = свободен
            first
            Convert_begin
                  состояние set перемещается_туда
                  положение set положение_устройства
                  начальное_время set Time_now
                  флаг_движения set 1
            Convert_end
                  состояние set свободен
                  начальное_время set 0
                  флаг_движения set 0
      деталь_
            Choice from деталь_.положение = ПОЛОЖЕНИЕ(накопитель_цех_in)
            first
            Convert_begin
                  состояние set транспортируется
                  положение set ПОЛОЖЕНИЕ(цеховой_робот)
            Convert_end
                  положение set положение_устройства
$End

$Pattern Обработка_на_станке :  operation trace
$Parameters
      положение_устройства : integer
$Relevant_resources
      станок_            :    Станки Keep Keep
      деталь_            : Детали Keep Keep
$Time = станок_.время_работы
$Body
      станок_
            Choice from станок_.положение = положение_устройства and
                                          станок_.состояние = готов_к_обработке
            first
            Convert_begin
                  состояние set работает
            Convert_end
                  состояние set закончил_обработку

      деталь_
            Choice from деталь_.положение = положение_устройства
            first
            Convert_begin
                  состояние set обрабатывается
            Convert_end
                  состояние set обработка_закончена
$End

$Pattern Перемещение_для_разгрузки :  operation trace
$Parameters
      положение_устройства : integer
$Relevant_resources
      станок_           :     Станки NoChange NoChange
      накопитель_       : Накопители NoChange NoChange
      робот_            : Роботы_цеховые Keep Keep
      деталь_           : Детали NoChange NoChange
$Time = Время_перемещения_к_станку
$Body
      станок_
            Choice from станок_.положение = положение_устройства and
                                          станок_.состояние = закончил_обработку
            first
      накопитель_
            Choice from накопитель_.положение = ПОЛОЖЕНИЕ(накопитель_цех_out)
                        and накопитель_.текущее_количество<накопитель_.максимальное_количество
            first
```

```
      робот_
            Choice from робот_.положение = ПОЛОЖЕНИЕ(накопитель_цех_in) and
                                                робот_.состояние = свободен
            first
            Convert_begin
                  состояние set перемещается_пустой
                  положение set положение_устройства
                  начальное_время set Time_now
                  флаг_движения set 1
            Convert_end
                  состояние set готов_к_разгрузке
                  начальное_время set 0
                  флаг_движения set 0
      деталь_
            Choice from деталь_.положение = положение_устройства
            first
$End

$Pattern Разгрузка_станков :  operation trace
$Parameters
      положение_устройства : integer
$Relevant_resources
      станок_           :     Станки Keep Keep
      накопитель_       : Накопители NoChange Keep
      робот_            : Роботы_цеховые Keep Keep
      деталь_           : Детали Keep Keep
$Time = Время_разгрузки_станка
$Body
      станок_
            Choice from станок_.положение = положение_устройства and
                                                станок_.состояние = закончил_обработку
            first
            Convert_begin
                  состояние set разгружается
            Convert_end
                  состояние set свободен
      накопитель_
            Choice from накопитель_.положение = ПОЛОЖЕНИЕ(накопитель_цех_out)
                        and накопитель_.текущее_количество<накопитель_.максимальное_количество
            first
            Convert_end
                  текущее_количество set накопитель_.текущее_количество + 1
      робот_
            Choice from робот_.положение = положение_устройства and
                                          робот_.состояние = готов_к_разгрузке
            first
            Convert_begin
                  состояние set разгружает
                  положение set положение_устройства
                  начальное_время set Time_now
                  флаг_движения set 1
            Convert_end
                  состояние set свободен
                  положение set ПОЛОЖЕНИЕ(накопитель_цех_out)
                  начальное_время set 0
                  флаг_движения set 0
      деталь_
            Choice from деталь_.положение = положение_устройства
            first
            Convert_begin
                  состояние set транспортируется
                  положение set ПОЛОЖЕНИЕ(цеховой_робот)
            Convert_end
                  состояние set хранится
                  положение set ПОЛОЖЕНИЕ(накопитель_цех_out)
$End

$Pattern Возврат_цехового_робота :  operation trace
$Parameters
      начальное_положение_устройства : integer
      конечное_положение_устройства  : integer
$Relevant_resources
      робот_            : Роботы_цеховые Keep Keep
$Time = Цеховой_робот.время_возврата
$Body
      робот_
            Choice from робот_.положение = начальное_положение_устройства
```

```
                  and робот_.состояние = свободен
            first
            Convert_begin
                  состояние set перемещается_обратно
                  начальное_время set Time_now
                  флаг_движения set 1
                  положение set начальное_положение_устройства
            Convert_end
                  состояние set свободен
                  положение set конечное_положение_устройства
                  начальное_время set 0
                  флаг_движения set 0
$End

$Pattern Продвижение_времени : irregular_event trace
$Parameters
      значение                : integer
$Relevant_resources
      робот_цех_              : Цеховой_робот Keep
$Time = 0.1
$Body
      робот_цех_
            Convert_event
                  время_движения set робот_цех_.время_движения + 0.1
$End
```

СКЛАД.opr (Операции):

```
$Operations
      Таймер                        : Продвижение_времени 0
      Добавление_в_цех_in           : Добавление_детали 'SPACE'

      Установка_на_станок_1         : Установка_на_станке 1
      Установка_на_станок_2         : Установка_на_станке 2

      Возврат_робота_со_станка_1    : Возврат_цехового_робота 1 8
      Возврат_робота_со_станка_2    : Возврат_цехового_робота 2 8
      Возврат_робота_от_нак_out     : Возврат_цехового_робота 9 8

      Обработка_на_станке_1         : Обработка_на_станке 1
      Обработка_на_станке_2         : Обработка_на_станке 2

      Перемещение_к_станку_1        : Перемещение_для_разгрузки 1
      Перемещение_к_станку_2        : Перемещение_для_разгрузки 2

      Разгрузка_станка_1            : Разгрузка_станков 1
      Разгрузка_станка_2            : Разгрузка_станков 2
$End
```

СКЛАД.pmd (Показатели):

```
$Results
      Простой_цехового_робота             : watch_state Цеховой_робот.состояние = свободен
      Производительность_станка_1         : watch_state Станок_1.состояние = работает
      Производительность_станка_2         : watch_state Станок_2.состояние = работает
$End
```

СКЛАД.smr (Прогон):

```
Model_name     = ЦЕХ

Resource_file  = ЦЕХ
OprIev_file    = ЦЕХ

Results_file   = ЦЕХ
Trace_file     = ЦЕХ
Frame_file     = ЦЕХ
Statistic_file = ЦЕХ
Frame_number   = 1
Show_mode      = Animation
Show_rate      = 5000.0

Terminate_if Накопитель_цех_out.текущее_количество = 10

external_model СКЛАД = Server
```