Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э.Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Компьютерные системы автоматизации производства

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ДИПЛОМНОМУ ПРОЕКТУ

НА ТЕМУ:

Система имитационного моделирования Rao X

Студент РК9-Д2 (Группа)	_____ (Подпись, дата)	П.А. Богачев (И.О.Фамилия)
Руководитель дипломного проекта	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по научно-исследовательской части	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по конструкторской части	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по технологической части	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по организационно-экономической части	_____ (Подпись, дата)	Л.А. Силаева (И.О.Фамилия)
Консультант по охране труда и экологии	_____ (Подпись, дата)	А.С. Козодаев (И.О.Фамилия)
Нормоконтролер	_____ (Подпись, дата)	М.Н. Святкина (И.О.Фамилия)

2016 г.

АННОТАЦИЯ

Отчет …

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ, ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ, РЕСУРС-ДЕЙСТВИЕ-ОПЕРАЦИЯ, МОДЕЛЬ, СИНТАКСИС, ГРАММАТИКА.

Объект разработки – программный продукт, система имитационного моделирования Rao X.

Цель работы – разработка системы имитационного моделирования Rao X. Система создается на основе системы RAO-XT, разработанной на кафедре РК-9 МГТУ им. Н.Э. Баумана.

При создании системы были произведены исследования производительности алгоритмов, используемых в системе при решении задач поиска на графе, предложен новый алгоритм работы с состояниями модели и доказана его эффективность по сравнению с существующим.

Результат разработки – система имитационного моделирования Rao X, включающая более мощный язык моделирования, расширенную библиотеку моделирования и современные средства среды разработки.

Эффективность системы заключается в предоставлении пользователям принципиально новых возможностей при разработке имитационных моделей.

Данная система может быть использована в учебном процессе при обучении имитационному моделированию.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, СИМВОЛОВ И СПЕЦИАЛЬНЫХ ТЕРМИНОВ С ИХ ОПРЕДЕЛЕНИЕМ

ИМ – Имитационное Моделирование

СМО – Система Массового Обслуживания

СДС – Сложная Дискретная Система

ПО – Программное Обеспечение

ПП – Программный Продукт

ПЭВМ – Персональная Электронно-Вычислительная Машина

ТЗ– Техническое Задание

ЭП – Эскизный Проект

ТП – Технический Проект

РП – Рабочий Проект

IDE – Integrated Development Environment (Интегрированная Среда Разработки)

ECL – Eclipse Command Language (Язык Команд Eclipse)

GUI – Graphical User Interface (Графический Интерфейс Пользователя)

Плагин — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей.

Фреймворк — программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Юнит-тестирование — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Интеграционное тестирование — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе.

Регрессионное тестирование — собирательное название для всех видов тестирования программного обеспечения, направленных на обнаружение ошибок в уже протестированных участках исходного кода.

Дисплейный менеджер — графический пользовательский интефейс, который отображается при окончании процесса загрузки вместо стандартного терминала.

Валидация — процесс приведения доказательств того, что требования конкретного пользователя продукта, услуги или системы удовлетворены.

Визард — это инструмент, помогающий пользователю быстро и наглядно осуществить те или иные настройки в программе.

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов.

ВВЕДЕНИЕ

Математическое моделирование является неотъемлемой частью современного мира информационных технологий. Эксперты все чаще прибегают к имитационному моделированию. Моделирование вобрало в себя весь арсенал новейших информационных технологий, включая развитые графические оболочки для целей конструирования моделей и интерпретации выходных результатов, мультимедийные средства и видео, поддерживающие анимацию в реальном масштабе времени, объектно-ориентированное программирование и др. В силу своей привлекательности и доступности эти технологии с легкостью окинули академические стены и сегодня осваиваются IT - специалистами в бизнесе.

Широкое использование ИМ в задачах анализа и синтеза систем объясняется сложностью (а иногда и невозможностью) применения строгих методов оптимизации, которая обусловлена размерностью решаемых задач и невозможностью формализации сложных систем. Так как современные технологии стремительно развиваются, одной из основных задач разработчиков программного обеспечения является поддержка продукта «на волне» новых технологий, постоянное обновление и развитие функциональных возможностей продукта.

РДО – не исключение. Система активно развивается, расширяются ее функциональные возможности. На определенном этапе система достигает состояния, в котором дальнейший развитие сопровождается возрастающими трудностями из-за устаревающих технологий, лежащих в основе продукта. Появляется необходимость перехода на новые, современные технологии, которые позволили бы и дальше развивать систему и поддерживать ее в актуальном состоянии.

Таким образом, разработка новой версии системы имитационного моделирования РДО на основе современных технологиий обеспечит

дальнейшее развитие функционала системы, расширение областей ее использования, позволит вывести ее на качественно более высокий уровень.

1. Предпроектное исследование

1.1. Основные положения языка РДО

Основные положения языка РДО[3] могут быть сформулированы следующим образом:

- Все элементы СДС представлены как ресурсы, описываемые некоторыми параметрами. Ресурсы могут быть разбиты на несколько типов; каждый ресурс определенного типа описывается одними и теми же параметрами.
- Состояние ресурса определяется вектором значений всех его параметров; состояние СДС - значением всех параметров всех ресурсов.
- Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояние ресурсов; действия ограничены во времени двумя событиями: событиями начала и событиями конца.
- Нерегулярные события описывают изменения состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отоношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.
- Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения состояния ресурсов в начале и в конце соответствующего действия.
- Множество ресурсов R и множество операций O образуют модель СДС.

1.2. Синтаксический анализ

Анализ исходной программы разбивает её на составные части и накладывает на них грамматическую структуру. Анализ исходный программы разбивается на две фазы: лексический и синтаксический анализ. Лексический анализатор читает поток символов, составляющих исходную программу, и группирует эти символы в значащие последовательности, называемые лексемами. Для каждой лексемы анализатор строит выходной токен, состоящий из имени токена и значения его атрибута. Этот токен передается на следующую фазу – синтаксический анализ, которая также называется разбором (parsing). Синтаксический анализатор использует первые компоненты токенов, полученные при лексическом анализе для создания древовидного промежуточного представления, которое описывает грамматическую структуру потока токенов.

Синтаксис языка программирования описывает корректный вид его программы, а семантика определяет смысл написанной на нем программы. Иерархическая структура множества конструкций языка программирования, т.е. его синтаксис естественным образом описывается грамматикой. Синтаксис большинства языков программирования описывается контекстно-свободной грамматикой или BNF (Backus-Naur Form – Форма Бэкуса-Наура). Контекстно-свободная грамматика имеет 4 компонента:

- множество терминальных символов - элементарных символов языка, определяемых грамматикой, т.е. токенов;
- множество нетерминальных символов – синтаксических переменных;
- множество продукций, каждая из которых состоит из нетерминала, называемого заголовком или левой частью продукции, стрелки и последовательности терминалов или нетерминалов, называемых телом или правой частью продукции. Продукции определяют один из возможных видов конструкции языка;

- стартовый (начальный) символ – специально указанный нетерминальный символ.

Грамматика выводит (порождает) строки, начиная со стартового символа и неоднократно замещая нетерминалы телом продукции этих нетерминалов. Строки токенов, порождаемые из стартового символа, образуют язык, определяемый грамматикой.

Таким образом, синтаксический анализ представляет собой выяснение для полученной строки терминалов способа её вывода из стартового символа грамматики. [2]

1.3. Система имитационного моделирования RAO-XT

Система имитационного моделирования RAO-XT представляет собой плагин для интегрированной среды разработки Eclipse, позволяющий вести разработку имитационных моделей на языке РДО. Система написана на языке Java[6] и состоит из трех основных компонентов:

- rao – компонент, производящий преобразование кода на языке РДО в код на языке Java.
- rao.lib – библиотека системы. Этот компонент реализует ядро системы имитационного моделирования.
- rao.tests – компонент, реализующий функционал, необходимый для проведения юнит-тестирования системы.
- rao.ui – компонент, реализующий графический интерфейс системы с помощью библиотеки SWT [4].

Для описания грамматики языка в системе используется фреймворк Xtext[5]. Xtext предоставляет язык описания грамматик, а также синтаксический анализатор, способный работать с контекстно-свободными грамматиками. Помимо этого Xtext предоставляет стек технологий для генерации кода на языке Java из кода разрабатываемого языка.

В системе RAO-XT генерация кода на Java происходит непосредственно перед запуском модели или при ее пересборке. Код генерируется как текст по правилам, описанным в генераторе кода системы, т.е. по окончанию процесса генерации кода никаких связей между кодом модели на языке РДО и сгенерированным кодом на языке Java не остается. При запуске модели исполняется сгенерированный код на языке Java.

Для того, чтобы различные функции среды разработки, помогающие пользователям эффективно разрабатывать код (навигация по коду, автоматическое переименование элементов, подсказки при наборе кода), работали для элементов грамматики РДО необходимо специальным образом описывать правила, в которых данные элементы грамматики могут использоваться. В случае сложных грамматических конструкций, в частности грамматики языка выражений, учет всех возможных элементов модели, которые могут в нем использоваться, является крайне сложной и трудоемкой задачей. Данная задача решена в системе RAO-XT лишь частично, и, как результат, при использовании языка выражений многие функциональные возможности среды разработки оказываются недоступными, либо работают некорректно.

Описанные выше недостатки системы могут быть решены только принципиальной переработкой архитектуры компонентов генерации кода.

2. Разработка технического задания на систему

2.1. Общие сведения

2.1.1. Полное наименование системы

Система имитационного моделирования Rao X.

2.1.2. Краткие наименования системы

Rao X, система Rao X.

2.1.3. Наименование предприятий заказчика и разработчика системы

Заказчик: кафедра «Компьютерные системы автоматизации производства» МГТУ им. Н.Э. Баумана.

Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Богачев П.А.

2.1.4. Основание для выполнения работ

Задание на выполнение дипломного проекта.

2.2. Назначение и цели создания системы

2.2.1. Назначение системы

Предоставить пользователю интегрированную среду разработки имитационных моделей на языке РДО.

2.2.2. Цель системы

Целью создания системы является обеспечение удобства разработки имитационных моделей на языке РДО и повышения функциональных возможностей языка, что позволяет создавать более сложные и более гибкие модели.

2.3. Требования к системе

2.3.1. Требования к функциональным характеристикам

Грамматика новой системы должна разрешить следующие проблемы, присущие старой грамматике:

- Несоответствие стилей разных компонентов языка.
- Перегруженность ключевыми словами.
- Слишком большой объем текста моделей.
- Несоответствие синтаксиса некоторых конструкций их смысловому содержанию.
- Большое количество искусственных ограничений.
- Слабый и невалидируемый язык выражений.
- Возможность использовать язык выражений только в ограниченном числе специально отведенных мест в коде модели.

Компоненты интегрированной среды разработки системы должны обладать следующими функциональными возможностями:

- Автодополнение кода. Автодополнение должно работать не только для ключевых слов, но и для любых других конструкций языка. Автодополнение должно выводить пользователю список доступных методов, переменных и операторов.
- Автоматическое форматирование выделенного фрагмента кода или всей модели целиком.
- Навигация по коду модели.

- Подсветка синтаксиса в коде модели.
- Рефакторинг кода: переименование элементов модели с обновление всех упоминаний данного элемента в модели.

2.3.2. Требования к надежности

Основное требование к надежности направлено на поддержание в исправном и работоспособном состоянии ЭВМ, на которой происходит использование системы.

2.3.3. Условия эксплуатации

- 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.

2.3.4. Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 2 Гб;
- объем жесткого диска не менее 100 Гб;
- микропроцессор с тактовой частотой не менее 2ГГц;
- монитор с разрешением от 1368 * 768 и выше.

2.3.5. Требования к информационной и программной совместимости

Система должна работать под управлением следующих ОС: Windows 7, Windows 8, Ubuntu 15.10.

2.3.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

2.3.7. Требования к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

2.3.8. Требования к программной документации

Требования к программной документации не предъявляются.

2.4. Технико-экономические показатели

2.4.1. Стадии и этапы разработки

Состав, содержание и сроки выполнения работ по созданию системы в соответствии с календарным планом на выполнение дипломного проекта.

2.4.2. Порядок контроля и приемки

Контроль и приемка программного продукта должны состоять из следующих этапов:

1. Запуск системы Rao X в операционной системе Ubuntu 15.10, проверка основного функционала системы на моделях из визарда проектов Rao X.
2. Запуск системы Rao X в операционной системе Windows 7, проверка основного функционала системы на моделях из визарда проектов Rao X.
3. Запуск интеграционного тестирования на удаленном сервере, проверка успешного прохождения системой всех тестов.

3. Концептуальное проектирование

3.1. Обеспечение глубокой интеграции с Java

Для принципиального решения проблем системы RAO-XT, описанных в предпроектном исследовании, представляется возможным использовать новые технологии, которые предоставляет фреймворк Xbase[13]. Технологии, предоставляемые данным фреймворком, являются развитием технологий, заложенных в фреймворк Xtext.

Xbase позволяет достичь глубокой интеграции разрабатываемого языка с языком Java засчет следующих принципов его работы:

- Xbase позволяет использовать произвольные классы языка Java в коде моделей на языке, который используют данную технологию.
- Код на языке Java не просто генерируется в виде текста, а дополнительно связывается с элементами разрабатываемого языка. Для связанных таким образом элементов языка возможно использование стандартных средств разработки платформы Eclipse, таких как автодополнение, автоматическое переименование, навигация по коду и т.п. Фреймворк Xbase обеспечивает автоматическое переключение между элементами Java и элементами разрабатываемого языка в зависимости от контекста работы с ними.
- Xbase позволяет генерировать код на языке Java непосредственно в процессе написания модели и не требует ручного запуска сборки модели.
- Xbase предоставляет мощный язык выражений, для которого уже реализованы правила генерации и связывания кода, а также функции валидации, правила форматирования, автодополнения и других возможностей среды разработки. Язык выражений Xbase

может быть включен в грамматику разрабатываемого языка с полным сохранением возможностей, описанных выше.

Таким образом в основу архитектуру новой системы необходимо заложить технологии, предоставляемые фреймворком Xbase.

3.2. Разработка нового синтаксиса языка РДО

Стилистически новый синтаксис должен соответствовать языкам Java и Xtend, на которых основана система.

Синтаксис был обновлен для следующих компонентов языка РДО:

- Объявления типов ресурсов.
- Создания ресурсов.
- Описания образцов.
- Описания событий.
- Описания точек принятия решений.
- Описания последовательностей.
- Описания функций.
- Описания констант.
- Инициализации прогона.

В приложении А приведен код модели многоканальной СМО в старом синтаксисе, а в приложении Б - в новом синтаксисе.

3.2.1. Синтаксис объявления типов ресурсов

На момент начала работы синтаксис объявления типов ресурсов имела следующий вид:

```
$Resource_type Клиенты : temporary
$Parameters
```

```
      тип       : ( Тип1, Тип2 )
      состояние: ( Пришел, Начал_стрижку )
$End

$Resource_type Парикмахеры: permanent
$Parameters
      состояние_парикмахера : ( Свободен, Занят )
      количество_обслуженных: integer
      длительность_min      : integer
      длительность_max      : integer
      тип_клиента           : such_as Клиенты.тип
$End
```

Данный синтаксис обладает следующими недостатками:

- Объявление перегружено неудобными ключевыми словами. Было принято решение использовать ключевое слово type вместо *$Resource_type*, а список параметров заключать в фигурные скобки, в результате чего ключевые слова *$Parameters* и *$End* могут быть удалены.
- Объявление типа ресурса постоянным или временным неудобно, так как разные ресурсы одного и того же типа могут фигурировать в модели как постоянно, так и на коротком промежутке модельного времени. Наиболее удобным решение представляется объявление всех типов ресурсов как временных и снятие связанных с этим ограничений.
- Типы данных параметров типа ресурса не соотносятся с типами данных языка Java, на основе которого построена система. Следует избавиться от специальных ключевых слов для типов данных и использовать стандартные типы.
- Стиль объявление параметров, при которым тип следует после имени сейчас является устаревшим. Кроме того, данный стиль не согласуется со стилем языка Java, в котором имя параметра следует после типа параметра. Следует перейти на стиль языка Java.
- Объявление перечислимого типа производится непосредственно при объявлении параметра, при этом перечислимые типы не имеют

собственных имен. В результате, для объявления двух параметров, имеющих одинаковый перечислимый тип, необходимо из одного параметра ссылаться на другой с помощью конструкции *such_as*. Эту схему следует изменить следующим образом:

- Объявлять перечислимые типы отдельно от параметров ресурса.
- Назначать перечислимым типам имена.
- Ссылаться на имя перечислимого типа, а не на имя параметра ресурса.
- Удалить конструкцию *such_as*.

В результате объявление типа ресурса из примера выше будет иметь следующий вид:

```
enum Тип_клиента {Тип1, Тип2}
enum Состояние_клиента {Пришел, Начал_стрижку}
enum Состояние_парикмахера {Свободен, Занят}

type Клиенты {
      Тип_клиента тип;
      Состояние_клиента состояние;
}

type Парикмахеры {
      Состояние_парикмахера состояние_парикмахера
      int количество_обслуженных;
      int длительность_min;
      int длительность_max;
      Тип_клиента тип_клиента;
}
```

3.2.1. Синтаксис создания ресурсов

На момент начала работы синтаксис создания ресурсов имел следующий вид:

```
$Resources
      Парикмахерская = Парикмахерские(0);
      Парикмахер_1 = Парикмахеры(Состояние_парикмахера.Свободен, 0,
                  20, 40, Тип1);
      Парикмахер_2 = Парикмахеры(Состояние_парикмахера.Свободен, 0,
```

```
                25, 70, Тип2);
      Парикмахер_3 = Парикмахеры(Состояние_парикмахера.Свободен, 0,
                30, 60, Тип2);
$End
```

Данный синтаксис уже был частично переработан ранее, однако все еще имеет некоторые недостатки:

- Создание ресурсов таким образом может производиться только до начала прогона. Конструкция создания ресурсов в событиях и образцах координально отличается. Следует унифицировать конструкцию создания ресурсов для любого места модели. Стиль объявления ресурсов должен соответствовать стилю языка выражений, основанному на языке Java. Для этого было принято решение использовать метод *create()* у типа ресурса. Тип ресурса в данной конструкции играет роль фабрики. Объединять создание ресурсов в группы не нужно.
- Для того, чтобы избежать коллизий имен перечислимых типов следует использовать полные имена для задания значений параметров этих типов.

В результате описание создания ресурсов из примера выше будет иметь следующий вид:

```
resource парикмахерская = Парикмахерские.create(0);
resource парикмахер_1 =
           Парикмахеры.create(Состояние_парикмахера.Свободен, 0,
                20, 40, Тип_клиента.Тип1);
resource парикмахер_2 =
           Парикмахеры.create(Состояние_парикмахера.Свободен, 0,
                25, 70, Тип_клиента.Тип2);
resource парикмахер_3 =
           Парикмахеры.create(Состояние_парикмахера.Свободен, 0,
                30, 60, Тип_клиента.Тип2);
```

3.2.2. Синтаксис описания образцов

На момент начала работы синтаксис описания образцов имел следующий вид:

```
$Pattern Образец_обслуживания_клиента : operation
$Relevant_resources
      _Парикмахерская: Парикмахерская Keep NoChange
      _Клиент        : Клиенты        Keep Erase
      _Парикмахер    : Парикмахеры    Keep Keep
$Time = Длительность_обслуживания( _Парикмахер.длительность_min,
            _Парикмахер.длительность_max )
$Body
_Парикмахерская:
      Choice from _Парикмахерская.количество_в_очереди > 0
      Convert_begin
            количество_в_очереди--;

_Клиент:
      Choice from _Клиент.состояние == Пришел
      Convert_begin
            состояние = Начал_стрижку;

_Парикмахер:
      Choice from _Парикмахер.состояние_парикмахера == Свободен and
            _Парикмахер.тип_клиента == _Клиент.тип
      with_min( _Парикмахер.количество_обслуженных )
      Convert_begin
            состояние_парикмахера  = Занят;
      Convert_end
            состояние_парикмахера  = Свободен;
            количество_обслуженных++;
$End
```

Данный синтаксис обладает следующими недостатками:

- Конструкция подбора релевантных ресурсов отделена от конструкции их объявления. Следует описать правила подбора в том же месте, где релевантным ресурсам задаются имена. Конструкцию подбора релеватных ресурсов следует изменить таким образом, чтобы вместо ключевых слов использовались методы, предоставляемые библиотекой Xbase или библиотекой Rao X.

- Для каждого релевантного ресурса задается отдельное тело (конвертер) внутри образца с описаниями действий над этим релевантным ресурсов в начале и в конце операции или при выполении правила. Следует определить для операции методы *begin()* и *end()*, а для продукционных правил метод *execute()*, в теле которых будут описываться действия над всеми релевантными ресурсами образца сразу. Методы будут переопределяться для каждого образца с помощью ключевого слова set.
- Задание длительности операции следует так же вынести в отдельный метод *duration()*, аналогичный методам, описанным выше. При этом порядок переопределения методов внутри образца неважен, однако методы не могут быть переопределены более одного раза.
- В результате проведения описанных выше изменений ограничения, задаваемые статусами конверторов (*Keep*, *NoChange*, и т.д.) теряют смысл, и потому могут быть удалены. Конверторы *Create* и *Erase*, используемые для создания и удаления ресурсов в образцах должны быть заменены конструкциями внутри переопределенных методов образцов. Конструкция удаления ресурса должна иметь формат, аналогичный конструкции создания ресурса. Для этого следует использовать вызов метода *erase()*.
- Механизм релевантных ресурсов используется даже для обращения к параметрам ресурсов, которые не надо подбирать. Следует разрешить обращение к таким параметрам по глобальному имени ресурса без объявления его как релевантного.
- Ключевые слова образцов, как и для других компонентов языка, следует упростить и свести к единому стилю.

- Так как образец может не иметь параметров, то в случае их отсутствия следует так же ставить скобки после имени образца, но пустые.

В результате описание операции из примера выше будет иметь следующий вид:

```
operation Образец_обслуживания_клиента() {
	relevant _Парикмахерская =
		парикмахерская.onlyif[количество_в_очереди > 0]
	relevant _Клиент =
		Клиенты.all.filter[состояние.equals(
			Состояние_клиента.Пришел)].any;
	relevant _Парикмахер =
		Парикмахеры.all.filter[состояние_парикмахера.equals(
			Состояние_парикмахера.Свободен) &&
			тип_клиента.equals(_Клиент.тип)]
		.minBySafe[количество_обслуженных];

	set duration() {
		return длительность_обслуживания.next(
			_Парикмахер.длительность_min,
			_Парикмахер.длительность_max);
	}

	set begin() {
		_Парикмахерская.количество_в_очереди =
			_Парикмахерская.количество_в_очереди - 1;
		_Клиент.состояние = Состояние_клиента.Начал_стрижку;
		_Парикмахер.состояние_парикмахера =
			Состояние_парикмахера.Занят;
	}

	set end() {
		_Парикмахер.состояние_парикмахера =
			Состояние_парикмахера.Свободен;
		_Парикмахер.количество_обслуженных =
			_Парикмахер.количество_обслуженных + 1;
		_Клиент.erase();
	}
}
```

3.2.1. Синтаксис описания событий

На момент начала работы синтаксис описания событий имел следующий вид:

```
$Pattern Образец_прихода_клиента : event
```

```
$Relevant_resources
      _Парикмахерская: Парикмахерская Keep
      _Клиент        : Клиенты        Create
$Body
_Парикмахерская:
      Convert_event
            Образец_прихода_клиента.planning( time_now
                  + Интервал_прихода( 30 ) );
            количество_в_очереди++;

_Клиент:
      Convert_event
            тип       = Тип_клиента;
            состояние = Пришел;
$End
```

В текущей грамматике языка РДО события являются особым типом образцов. Однако концептуально это не так: события – это элементы событийного подхода к имитационному моделированию, а образцы – элементы подхода сканирования активностей. Их мнимая похожесть в языке РДО объяснялась необходимостью использования механизма релевантных ресурсов для создания ресурсов в процессе прогона или обращения к параметрам ресурсов, созданных до начала прогона и имеющих имена. В результате изменений, проведенных в предыдущих пунктах модели, появилась возможность полностью разделить события и образцы.

В результате описание события из примера выше будет иметь следующий вид:

```
event Событие_прихода_клиента() {
      Клиенты.create(случайный_тип_клиента.next(),
            Состояние_клиента.Пришел);
      Событие_прихода_клиента.plan(currentTime
            + интервал_прихода.next());
      парикмахерская.количество_в_очереди =
            парикмахерская.количество_в_очереди + 1;
}
```

3.2.1. Синтаксис описания точек принятия решений

На момент начала работы синтаксис описания точек принятия решений типа *some* имел следующий вид:

```
$Decision_point model: some
$Activities
      Обслуживание_клиента: Образец_обслуживания_клиента;
$End
```

Синтаксис описания точек принятия решений типа *search* имел следующий вид:

```
$Decision_point Расстановка_фишек : search
$Condition Exist(Фишка: Фишка.Номер <> Фишка.Местоположение)
$Term_condition
      For_All(Фишка: Фишка.Номер == Фишка.Местоположение)
$Evaluate_by 0
$Compare_tops = YES
$Activities
      Перемещение_вправо: Перемещение_фишки(справа,  1)
            value after 1;
      Перемещение_влево : Перемещение_фишки (слева, -1)
            value after 1;
      Перемещение_вверх : Перемещение_фишки (сверху,-3)
            value after 1;
      Перемещение_вниз  : Перемещение_фишки (снизу,  3)
            value after 1;
$End
```

Помимо этого, имелись так же точки принятия решений типа *prior*, во всем аналогичные точкам типа *some*, но обладающие дополнительными возможностями задания приоритета.

Данный синтаксис обладает следующими недостатками:

- Точки принятия решений *some* и *prior* описываются различными конструкциями. При этом точки принятия решений типа some являются частным случае точек принятия решения типа *prior* (в случае одинаковых приоритетов всех активностей). Следует использовать для них единую конструкцию с опциональным заданием приоритета. Если приоритет не задан, точки принятия решений будут работать так же, как some.
- Точки принятия решений типа *search* имеют большое количество характеристик, которые необходимо сконфигурировать при инициализации точек. При этом задание этих характеристик производится в строгом, но не имеющим определенного

логического обоснования, порядке. Следует организовать задание параметров точки в виде вызова соответствующих методов в произвольном порядке. Сами вызовы будут производиться в теле метода *init()* точки принятия решений.

- Точки принятия решений типа *search* имеют множество принципиальных отличий, и потому грамматику их заданий следует принципиально разделить с грамматикой создания точек принятия решений остальных типов. Более простые точки принятия решений следует назвать *logic*. Активности точек принятия решений типа *search* следует переименовать в *edge*, что отражает их соответствие вершинам графа при поиске.
- Грамматику списка активностей/вершин необходимо привести к единому стилю с уже преобразованной грамматикой. Вместо конструкций *value before* и *value after* следует задавать соответствующие параметры активностям.
- Каждая активность/вершина работает с собственной копией образца, поэтому в конструктор активности должна передаваться фабрика образцов.

В результате описание точек принятия из примера выше будет иметь следующий вид:

```
logic Model {
      activity обслуживание_клиента =
            new Activity(Образец_обслуживания_клиента.create());
}

search Расстановка_фишек {
      edge перемещение_вправо = new Edge(Перемещение_фишки.create(
                  Место_дырки.СПРАВА, 1), 1)
      edge перемещение_влево = new Edge(Перемещение_фишки.create(
                  Место_дырки.СЛЕВА, -1), 1)
      edge перемещение_вверх = new Edge(Перемещение_фишки.create(
                  Место_дырки.СВЕРХУ, -3), 1)
      edge перемещение_вниз = new Edge(Перемещение_фишки.create(
                  Место_дырки.СНИЗУ, 3), 1)

      set init() {
```

```
            startCondition = [Фишка.all.exists[номер != место]]
            terminateCondition = [Фишка.all.forall[
                        номер == место]]
            compareTops = true
            heuristic = [0.0]
        }
}
```

3.2.1. Синтаксис описания последовательностей

На момент начала работы синтаксис описания точек принятия решений типа some имел следующий вид:

```
$Sequence Интервал_прихода : real
$Type = exponential 123456789
$End

$Sequence Длительность_обслуживания : real
$Type = uniform 123456789
$End
```

Данный синтаксис обладает следующими недостатками:

- Излишние ключевые слова.
- Доступные виды последовательностей определяются ключевыми словами, что накладывает значительное ограничение на возможности расширения набора доступных последовательностей.
- Невозможность задания параметров распределения на этапе объявления последовательности. Следует обеспечить возможность задать параметры как в момент объявления последовательности, так и в момент вызова метода получения следующего значения последовательности. Получение следующего значения последовательности следует реализовать через вызов метода next().
- Для экспоненциального распределения в качестве параметра задается среднее арифметическое, а не интенсивность. Задание

распределения через интенсивность является более общепринятым вариантом, и реализовать следует именно его.

В результате описание последовательности из примера выше будет иметь следующий вид:

```
sequence интервал_прихода = new Exponential(123456789, 1/30.0);
sequence длительность_обслуживания = new Uniform(123456789);
```

3.2.2. Синтаксис описания функций

На момент начала работы синтаксис описания функций имел следующий вид:

```
$Function Фишка_на_месте : integer
$Type = algorithmic
$Parameters
      _Номер: such_as Фишка.Номер
      _Место: such_as Фишка.Местоположение
$Body
      if (_Номер == _Место)
return 1;
      else
return 0;
$End
```

Внутри тела функция использует Java-подобный язык выражений, однако сама структура объявления функции имеет совершенно другой стиль. Этот стиль перегружен ключевыми словами и в целом неудобен. Следует свести стиль объявления функций к Java-стилю.

В результате описание функции из примера выше будет иметь следующий вид:

```
int Фишка_на_месте(int номер, int место) {
      if (номер == место)
            return 1
      else
            return 0
}
```

3.2.3. Синтаксис описания констант

На момент начала работы синтаксис объявления констант имел следующий вид:

```
$Constant
      Длина_поля : integer = 3
$End
```

Синтаксис объявления констант следует свести к стилю остальных компонентов языка, уже описанных выше. Тип констант, как и тип всех других элементов модели следует выводить автоматически исходя из ее значения. В результате объявление констант из примера выше будет иметь следующий вид:

```
constant длина_поля = 3;
```

3.2.4. Синтаксис инициализации модели

На момент начала работы синтаксис инициализации модели имел следующий вид:

```
$Simulation_run
      Образец_прихода_клиента.planning( time_now
            + Интервал_прихода( 30 ) );
      Terminate_if Time_now >= 360;
$End
```

В данной конструкции смешаны конструкции из языка выражений и специфические конструкции РДО. Необходимо сделать следующее:

- Разделить высказывания, связанные с инициализацией модели, и задание условия завершения модели.
- Высказывания, связанные с инициализацией модели определять в глобальном методе *init()*.
- Вычисление условия окончания моделирования осуществлять в глобальном методе *terminateCondition()*.

В результате описание инициализации модели из примера выше будет иметь следующий вид:

```
set init() {
      Событие_прихода_клиента.plan(currentTime
            + интервал_прихода.next());
}

set terminateCondition() {
      return currentTime >= 7 * 12 * 60;
}
```

В Приложении 1 приведен полный код модели многоканальной СМО в старом синтаксисе. В Приложении 2 приведен полный код той же модели в новом синтаксисе.

3.3. Дополнительные возможности языка

3.3.1. Генераторы

При моделировании сложных систем нередко возникает необходимость задавать последовательность чисел по некоторому сложному алгоритму, а не простым перечислением или с помощью случайной последовательности.

Старая грамматика языка не предоставляла никаких средств для решения данной задачи. Поэтому было принято решение добавить такие грамматические конструкции, которые позволили бы пользователю самостоятельно описать алгоритм генерации чисел.

Для обеспечения большей гибкости, описание генераторов чисел отделено от объявления последовательностей. Это позволяет объявлять генераторы с параметрами, и переиспользовать один и тот же генератор в разных последовательностях, передавая туда различные параметры.

Возврат значения из тела генератора осуществляется вызовом метода *yield()*.

Разработанный синтаксис описания и использования генераторов имеет следующий вид:

```
generator gen<Double>(int x) {
	while (true) {
		for (var i = 0; i <= x; i++) {
			yield(Math.pow(10, i));
		}
	}
}

sequence upToHundred = new gen(2)
sequence upToMillion = new gen(6)
```

3.3.2. Секция включения пакетов

Использовании технологий, предоставляемых фреймворком Xbase, подразумевает возможность работы в модели с классами языка Java. Для того, чтобы исключить необходимость обращаться к используемым классами по их полному имени (включающему имена всех пакетов, в которых расположен данный класс), было принято решение добавить в модели на языке РДО секцию включения пакетов.

Синтаксис секции включения пакетов полностью повторяет синтаксис данных конструкций в языке Java:

```
import java.util.List
import java.util.ArrayList
```

4. Техническое проектирование

4.1. Разработка новой грамматики языка

Для того, чтобы использовать язык выражений, предоставляемый грамматикой Xbase, в языке РДО необходимо, чтобы грамматика РДО наследовала грамматику Xbase.

Для этого начала файла с грамматикой должно иметь следующий вид:

```
grammar ru.bmstu.rk9.rao.Rao with org.eclipse.xtext.xbase.Xbase
```

В результате появляется возможность использовать грамматические конструкции, предоставляемые Xbase, в любых грамматических конструкциях РДО.

4.1.1. Глобальные компоненты языка

Тело продукции стартового символа грамматики представляет собой множество возможных глобальных компонентов языка. Для описания новых компонентов языка к телу продукции стартового символа были добавлены следующие нетерминалы: *DefaultMethod*, *EnumDeclaration*, *Generator*, а также грамматическое правило для секции импортов, которая всегда предшествует остальным элементам грамматики. Помимо этого, события стали включаться в модель непосредственно, а не через грамматику образцов, а точки принятия решений были разделены на *Logic* и *Search*. В результате грамматические правила для описания глобальных компонентов модели выглядят следующим образом:

```
RaoModel:
	{RaoModel}
	importSection = XImportSection?
	objects += RaoEntity*
;
```

```
RaoEntity
      : ResourceType
      | Generator
      | FunctionDeclaration
      | Event
      | Pattern
      | Frame
      | Result
      | DefaultMethod
      | EnumDeclaration
      | Logic
      | Search
      | EntityCreation
      | ResourceDeclaration
;

EntityCreation
      : Sequence
      | Constant
;
```

Наличие секции включения пакетов позволяет пользователю импортировать необходимые классы и пакеты, предоставляемые как библиотеками *Java* и *Xbase*, так и любыми библиотеками, которые пользователь подключит в *Rao*-проект.

4.1.2. Грамматика переопределения стандартных методов

Грамматическое правило переопределения стандартных методов модели имеет следующий вид:

```
DefaultMethod:
      'set' name = ID '(' ')'
            body = XBlockExpression
;
```

Конструкция *XBlockExpression* представляется грамматикой Xbase и описывает блок выражений, окруженный фигурными скобками. Таким образом тело стандартных методов описывается произвольными выражениями на языке Xbase.

4.1.3. Грамматика объявления перечислимых типов

Грамматическое правило описания перечислимых типов модели имеет следующий вид:

```
EnumDeclaration:
	'enum' name = ID '{' values += ID (',' values += ID)* '}'
;
```

4.1.4. Грамматика описания ресурсов

Грамматические правила для объявления типов ресурсов выглядят следующим образом:

```
ResourceType:
	'type' name = ID '{'
		(parameters += FieldDeclaration)*
	'}'
;

FieldDeclaration
	: declaration = FullJvmFormalParameter ';'?
;
```

Данная граматика позволяет создавать параметры ресурсов произвольного типа, и не ограничиваться набором стандартных, как это было ранее.

Грамматическое правило глобального объявления ресурса имеет следующий вид:

```
ResourceDeclaration:
	'resource' name = ID '=' constructor = XOrExpression ';'?
;
```

При этом конструктор ресурса представляет из себя произвольное выражение на языке *Xbase*. Данное выражение затем валидируется, чтобы убедиться, что возвращаемое им значение действительно имеет тип ресурса.

Следует отметить, что параметры ресурсов теперь объявляются и создаются конструкциями языка Xbase, из-за чего становится невозможным

сохранить в языке правила для задания им значений по умолчанию. Это объясняется тем, что в языке Xbase нет специального символа или ключевого слова для описания того, что переданное в конструктор значение является значением по умолчанию. Данная задача может быть решена расширением языка выражений Xbase и переработкой структуры классов, отвечающих за объявление и создание ресурсов, однако в данном дипломном проекте эта задача не решается.

4.1.5. Грамматика описания событий

Описание событий было грамматически отдельно от описания остальных образцов. Из грамматики событий была удалена возможность объявления релевантных ресурсов, что позволило максимально упростить их синтаксис:

```
Event:
	'event' name = ID '('
		(parameters += FullJvmFormalParameter (
			',' parameters += FullJvmFormalParameter)* )?
	')' body = XBlockExpression
;
```

В новой грамматике события имеют только список параметров и тело, описываемое набором произвольного количества высказываний на языке выражений *Xbase*.

4.1.6. Грамматика описания образцов

Вынесение событий в отдельный компонент позволило свести все виды оставшихся образцов к единому синтаксису. В теле образцов поочередно следуют конструкции простого подбора релевантных ресурсов, конструкции комбинаторного подбора релевантных ресурсов и переопределения стандартных методов образцов.

```
Pattern:
	type = PatternType name = ID '('
```

```
            (parameters += FullJvmFormalParameter (
                ',' parameters += FullJvmFormalParameter)* )? ')'
        '{'

            relevantResources += RelevantResource*
            relevantTuples += RelevantResourceTuple*
            defaultMethods += DefaultMethod*
        '}'
;

enum PatternType
        : OPERATION = 'operation'
        | RULE = 'rule'
;
```

В новой грамматике могут одновременно присутствовать произвольное число конструкций простого подбора релевантных ресурсов и произвольное число конструкций их комбинаторного подбора. Таким образом снимается ограничение, из-за которого приходилось комбинаторно подбирать все релевантные ресурсы, если это было нужно хотя бы двум из них.

Грамматические правила подбора релевантных ресурсов имеют следующий вид:

```
RelevantResource:
        'relevant' name = ID '=' value = XOrExpression ';'?
;

RelevantResourceTuple:
        'relevants' names += ID (',' names += ID)*
            '=' value = XOrExpression ';'?
;
```

Правила выбора релевантных ресурсов таким образом описываются языком выражений. Методы, позволяющие пользователю простым и удобным образом описать параметры, по которым ему необходимо произвести подбор ресурсов, частично предоставляются библиотекой Rao и частично - библиотекой Xbase.

В случае комбинаторного подбора имена подбираемых релевантных ресурсов перечисляются через запятую, а выражение в правой части возвращает их комбинацию (пару, тройку и т.п.).

4.1.7. Грамматика описания точек принятия решений

Точки принятия решений типа *prior* и *some* были объединены в общую грамматическую конструкцию, и названы единым термином *logic*. Точки принятия решений типа *search* вынесены в отдельную грамматическую конструкцию.

Таким образом грамматика точек принятия решений типа *logic* имеет следующий вид:

```
Logic:
	'logic' name = ID '{'
		activities += Activity*
		defaultMethods += DefaultMethod*
	'}'
;

Activity:
	'activity' name = ID '=' constructor = XOrExpression ';'?
;
```

У *logic* возможен один стандартный метод *init*(), в котором при необходимости производится определение тех параметров точек, которые раньше задавались ключевыми словами:

- Приоритет.
- Условие выполнения.
- Родительская точка.

Активности точек принятия решений создаются с помощью конструктора, возможные виды которого предоставляются библиотекой *Rao*. Обязательным параметром любого из конструкторов активности является фабрика паттернов, которые данная активность будет выполнять.

Грамматика точек принятия решений типа *search* имеет аналогичный вид, за исключением того, что активности точек принятия решения данного типа теперь объявляются как *edge*.

```
Search:
	'search' name = ID '{'
```

```
            edges += Edge*
            defaultMethods += DefaultMethod*
      '}'
;

Edge:
      'edge' name = ID '=' constructor = XOrExpression ';'?
;
```

Как и в случае с *logic*, все параметры, которые раньше задавались ключевыми словами, теперь задаются в стандартном методе *init*().

4.1.8. Грамматика описания функций и последовательностей

Грамматика описания функций имеет следующий вид:

```
FunctionDeclaration:
      type = JvmTypeReference name = ID '('
            (parameters += FullJvmFormalParameter (
                  ',' parameters += FullJvmFormalParameter)*)?
      ')' body = XBlockExpression
;
```

Тело функции представляет из себя набор произвольных выражений на языке *Xbase*.

Грамматика описания констант имеет следующий вид:

```
Constant:
      'constant' name = ID '=' constructor = XOrExpression ';'?
;
```

Константы в новой грамматике могут иметь произвольный тип. При этом тип константы выводится из правой части выражения и пользователю не нужно указывать его в явном виде.

4.1.9. Грамматика описания последовательностей

Описание всех типов последовательностей сведено к единой грамматической конструкции:

```
Sequence:
```

```
	'sequence' name = ID '=' constructor = XOrExpression ';'?
;
```

Возможные виды последовательностей предоставляются библиотекой системы Rao X.

4.1.10. Грамматика описания генераторов

Грамматика описания генераторов имеет следующий вид:

```
Generator:
	'generator' name = ID '<' type = JvmTypeReference '>' '('
		(parameters += FullJvmFormalParameter (
			',' parameters += FullJvmFormalParameter)*)?
	')' body = XBlockExpression
;
```

Генераторы, описанные таким образом, могут быть в дальнейшем использоваться для инициализации последовательностей вместо одного из стандартных типов, предоставляемых библиотекой *Rao*.

4.1.11. Грамматика описания анимации

Грамматика описания кадров анимации имеет следующий вид:

```
Frame:
	'frame' name = ID '{'
		defaultMethods += DefaultMethod*
	'}'
;
```

Кадры анимации могут содержать два стандартных метода: *init()* и *draw()*.

В методе *init()* описываются действия, которые необходимо выполнять единожды в начале прогона модели (например, отрисовка фона).

В методе *draw()* описывается алгоритм отрисовки кадра. Функции отрисовки предоставляются библиотекой системы Rao X.

4.2. Использование технологий Xbase в подсистеме генерации кода

Для перехода на технологии, предоставляемые фреймворком Xbase, необходимо определить следующие классы:

- *RaoJvmModelInferrer*, наследующий класс *AbstractModelInferrer*. В данном классе описываются правила генерации и связывания кода.
- *RaoValidator*, наследующий класс *AbstractRaoValidator*. В данном классе описываются правила валидации кода на языке РДО.
- *RaoFormatter*, наследующий класс *XbaseFormatter*. В данном классе описываются правила форматирования кода на языке РДО.
- *RaoProposalProvider*, наследующий класс *AbstractRaoProposalProvider*. В данном классе описываются правила, определяющие какие подсказки при написании кода на языке РДО должны выводиться.
- *RaoHighlightningCalculator*, наследующий класс. *XbaseHighlightingCalculator*. В данном классе описываются правила подсветки кода на языке РДО.
- *RaoImplicitlyImportedFeatures*, наследующий класс *ImplicitlyImportedFeatures*. В данном классе описываются функции библиотек (в т.ч. библиотеки Rao X), которые могут использоваться в коде модели на языке РДО без необходимости их импорта в явном виде.
- *RaoXImportSectionNamespaceScopeProvider*, наследующий класс *XImportSectionNamespaceScopeProvider*. В данном классе описываются пакеты, содержимое которых может использоваться к коде модели на языке РДО без необходимости явных импортов.

4.3. Иерархия классов последовательностей

Вследствие того, что типы последовательностей в новой грамматике определяются не ключевыми словами, а классами, предоставляемыми библиотекой моделирования, необходимо разработать удобную иерархию классов последовательностей.

Т.к. ограничений на возможные типы возвращаемых значений последовательностей в новой грамматике нет, то в качестве базовых интерфейсов для всех последовательностей будут служить два класса: *NumericSequence* и *ArbitraryTypeSequence*.

Класс *NumericSequence* является базовым для простых последовательностей, генерирующих числа с плавающей точкой. Данные последовательности должны определять метод *next()* в следующем виде:

```
public Double next();
```

Класс *NumericSequence* будет являться родительским для следующими классов последовательностей:

- *Uniform*. Последовательность, генерирующая числа по равномерному распределению.
- *Exponential*. Последовательность, генерирующая числа по экспоненциальному распределению.
- *Normal*. Последовательность, генерирующая числа по нормальному распределению.
- *Triangular*. Последовательность, генерирующая числа по треугольному распределению.
- *ContinuousHistogram*. Последовательность, генерирующая числа с вероятностями, определенными гистограммой. При этом гистограмма задается непрерывными интервалами.

Классы, наследующие класс *ArbitraryTypeSequence<T>* параметризуется произвольным типом *T*. В результате метод *next()* будет определяться последовательностями данной категории в следующем виде:

```
public T next();
```

Класс *ArbitraryTypeSequence<T>* будет являться родительским для следующих классов последовательностей:

- *Values<T>*. В качестве параметра в конструктор последовательностей даннного типа передается список объектов типа *T*. При вызове метода *next()* данные последовательности возвращают значения в том порядке, в которым они были заданы в конструкторе.
- *DiscreteHistogram<T>*. Последовательность, генерирующая объекты типа *T* с вероятностями, определенными гистограммой.
- *Generator<T>*. Абстрактный класс, использующийся в качестве родителей для генераторов.

Таким образом существующий в системе RAO-XT набор доступных последовательностей значительно расширен засчет возможности создавать последовательности произвольных типов, а также создавать последовательности с произвольным алгоритмом генерации чисел.

5. Рабочее проектирование

5.1. Реализация генерации и связывания кода

При использовании фреймворка Xbase для генерации и связывания кода необходимо определить класс, который будет наследовать класс *AbstractModelInferrer* и описывать, каким именно образом элементы языка РДО будут связываться с элементами языка Java.

Для этого в данном классе необходимо переопределить метод *infer()*. В данный метод помимо информации, необходимой Xbase для внутреннее работы, передается корневой элемент грамматики языка, т.е. *RaoModel*.

```
class RaoJvmModelInferrer extends AbstractModelInferrer {

	def dispatch void infer(RaoModel element,
			IJvmDeclaredTypeAcceptor acceptor,
			boolean isPreIndexingPhase) {
		acceptor.accept(element.toClass(
				QualifiedName.create(
					element.eResource.URI.projectName,
					element.nameGeneric))) [
			for (entity : element.objects) {
				entity.compileRaoEntity(it, isPreIndexingPhase)
			}

			element.compileInitialization(it, isPreIndexingPhase)
		]
	}

	// ...
}
```

В теле данного метода производится создание нового класса с именем, совпадающим с именем файла модели. Корневой элемент – модель – связывается с этим классом. В коде инициализатора, размещенном в квадратных скобках, производится итерация по всем элементам модели (потомкам *RaoModel*) и для каждого из них вызывается метод, отвечающий за генерацию и связывание кода для этого элемента.

В качестве примера рассмотрим генерацию кода для событий. При вызове метода *compileRaoEntity()* у события вызывается следующая конструкция:

```
def dispatch compileRaoEntity(Event event, JvmDeclaredType it,
          boolean isPreIndexingPhase) {
     members += event.asClass(jvmTypesBuilder,
          _typeReferenceBuilder, it, isPreIndexingPhase)
}
```

В теле данного метода к вложенным элементам класса, который был сгенерирован для модели, добавляется класс, который будет сгенерирован для события конструкцией *event.asClass()*.

Метод *asClass()* для событий определяется в отдельном классе *EventCompiler*:

```
def static asClass(Event event, JvmTypesBuilder jvmTypesBuilder,
          JvmTypeReferenceBuilder typeReferenceBuilder,
          JvmDeclaredType it, boolean isPreIndexingPhase) {
     // ...
}
```

В данном методе с помощью кострукции *event.toClass()* производится связывания события с новым вложенным классом, наследующим класс *Event*, предоставляемый библиотекой моделирования:

```
return event.toClass(eventQualifiedName) [
     static = true
     superTypes += typeRef(ru.bmstu.rk9.rao.lib.event.Event)

     // ...
}
```

Далее в коде инициализатор производится описание внутреннее структуры класса.

Для данного события создается публичный конструктор, первым параметров которого является время его запланированного выполнения, а оставшимися – параметры, описанные при объявлении события в коде на языке РДО. В теле конструктора данные параметры сохраняются в приватные поля класса события, чтобы впоследствии их можно было использовать в теле самого события, т.е. в коде, который выполняется при совершении события. Код генерации конструктора для события приведен ниже:

```
members += event.toConstructor [
     visibility = JvmVisibility.PUBLIC
```

```
        parameters += event.toParameter("time", typeRef(double))
        for (param : event.parameters)
            parameters += param.toParameter(param.name,
                    param.parameterType)
        body = '''
            «FOR param : parameters»this.«param.name» = «param.name»;
            «ENDFOR»
        '''
]

for (param : event.parameters)
        members += param.toField(param.name, param.parameterType)
```

Тело метода *execute()* события напрямую формируется из тела события, описанного на языке РДО:

```
members += event.toMethod("execute", typeRef(void)) [
        visibility = JvmVisibility.PROTECTED
        final = true
        annotations += overrideAnnotation()
        body = event.body
]
```

Статический метод *plan()* события генерируется таким образом, чтобы в качестве параметров принимать запланированное время выполнения события, а также все параметры, описанные при объявлении события. В теле метода происходит создание новой копии события и помещение его в планировщик событий текущего симулятора.

```
members += event.toMethod("plan", typeRef(void)) [
        visibility = JvmVisibility.PUBLIC
        static = true
        final = true
        parameters += event.toParameter("time", typeRef(double))
        for (param : event.parameters)
            parameters += event.toParameter(param.name,
                    param.parameterType)
        body = '''
            «event.name» event = new «event.name»(
            «FOR param : parameters»«param.name»«
                IF parameters.indexOf(param) != parameters.size - 1», 
                «ENDIF»
            «ENDFOR»);
            ru.bmstu.rk9.rao.lib.simulator.
                    CurrentSimulator.pushEvent(event);
        '''
]
```

Таким образом событие было сгенерировано как класс, имеющий метод *execute()*, соответствующий телу события и статический метод *plan()*, позволяющий создавать новые сущности события и помещать их в планировщик. Такая архитектура класса позволяет из любого места модели планировать новые события через вызов статического метода, а сам метод появляется в списке доступных методов после обращения к имени события через точку.

Пример меню автодополнения при создании конструкции планирования события в модели представлен на рисунке 5.1.

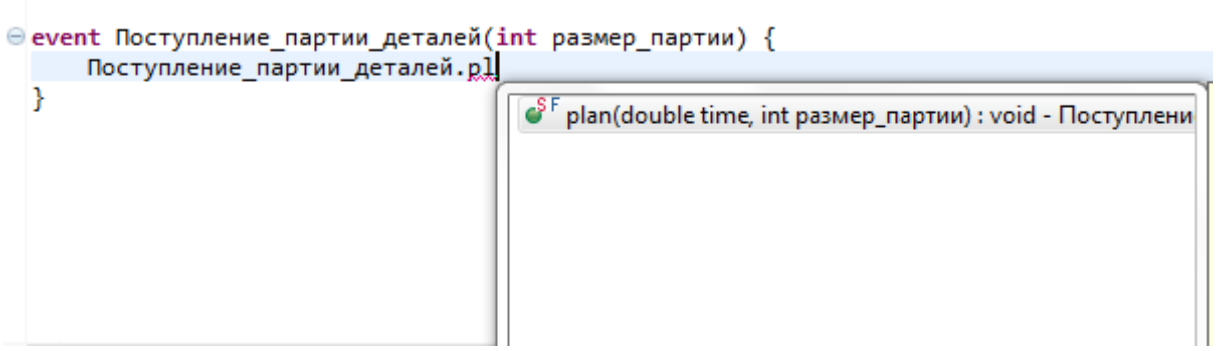

Рисунок 5.1 – Автодополнение при планировании события в модели

Для всех остальных элементов модели методы, генерирующие и связывающие код, разработаны, следуя аналогичным принципам.

5.2. Валидация грамматических правил

Eclipse предоставляет возможность валидировать корректность программ на ходу, непосредственно в процессе их написания. Данный процесс называется валидацией. При обнаружении ошибок в ходе валидации программы в окно Problems View выставляются маркеры ошибок. Для грамматических правил, которые не покрываются грамматикой Xbase, а являются уникальными для грамматики языка РДО, необходимо описать правила, по которым они будут валидироваться.

5.2.1. Валидация типов создаваемых элементов

В новом синтаксисе языке во многих случаях тип объявляемого элемента языка (ресурса, последовательности, активностей и др.). Вследствие этого при валидации модели необходимо проверить, что тип значения в правой части является валидным для данной конструкции. Например, тип выражения, которым инициализируется ресурс, должен быть подклассом класса *Resource*.

Метод валидации, проверяющий тип ресурса приведен ниже:

```
@Check
def checkResourceDeclaration(ResourceDeclaration resource) {
     if (!resource.constructor.actualType.isSubtypeOf(
               typeof(ru.bmstu.rk9.rao.lib.resource.Resource)))
          error("Error in declaration of \"" + resource.name
               + "\": only Rao resources are allowed.",
               RaoPackage.eINSTANCE.entityCreation_Constructor)
}
```

Аналогичные проверки также проделываются для релевантных ресурсов, последовательностей, активностей и вершин.

5.2.2. Проверки на совпадения имен

Для того, чтобы избежать коллизий имен элементов модели необходимо еще на этапе валидации выявить потенциальные совпадения имен, а также совпадения имен элементов модели с именем самой модели.

Данная валидация проводится в методе *checkDuplicateNamesForEntities()*. Для этого на первом шаге выполняется итерация по всем валидируемым элементам модели. В данном цикле проверяется имя каждого из элементов на совпадение с именем модели, а также заполняется список имен-дубликатов:

```
for (eObject : checklist) {
     if (eObject.nameGeneric.equals(model.nameGeneric))
          error("Error - object cannot have the same name as model ('"
               + eObject.nameGeneric + "').",
               eObject, eObject.getNameStructuralFeature)

     val name = eObject.fullyQualifiedName
```

```
	if (entities.contains(name)) {
		if (!duplicates.contains(name))
			duplicates.add(name)
	} else
		entities.add(name)
}
```

Когда список дубликатов был получен, выполняется второй цикл - по этому списку. В данном цикле выявляются все элементы, имена которых повторяются в модели, и для всех них выводятся маркеры ошибок.

```
if (!duplicates.empty) {
	for (eObject : checklist) {
		val name = eObject.fullyQualifiedName
		val hasNoName = (name.contains(".")
			&& name.substring(name.lastIndexOf(".") + 1) == "null")
		if (!hasNoName && duplicates.contains(name))
			error("Error - multiple declarations of object '"
				+ name + "'.", eObject,
				eObject.getNameStructuralFeature)
	}
}
```

5.2.3. Валидация стандартных методов модели

Грамматика конструкций для переопределения стандартных методов модели была описана в максимально обобщенном виде. Чтобы пользователь получал информацию об ошибках в задании стандартных методов в процессе редактирования модели, был использован механизм валидации.

Был создан класс *DefaultMethodsHelper*, в котором для каждой грамматической конструкции, внутри которой могут переопределяться методы, были описаны допустимые имена переопределяемых методов, а также тип действия (вывод ошибки, вывод предупреждения или игнорирование) в случае, если метод не был задан.

```
	public static enum ValidatorAction {
		ERROR, WARNING, NOTHING
	};

	public static class MethodInfo {
		MethodInfo(ValidatorAction action) {
			this.action = action;
		}
```

```
            int count = 0;
            ValidatorAction action;
      }
```

В классе *RDOValidator* для каждого образца, а также для корня модели выполняются проверки на корректность переопределения в них методов. Для этого предварительно формируется *HashMap*, содержаний описание всех корректных методов и счетчик числа каждого из них.

```
var Map<String, DefaultMethodsHelper.MethodInfo> counts =
            new HashMap<String, DefaultMethodsHelper.MethodInfo>()
for (v : DefaultMethodsHelper.GlobalMethodInfo.values)
      counts.put(v.name,
                  new DefaultMethodsHelper.MethodInfo(v.validatorAction)
)
```

После этого проводится итерирование по всем найденным внутри данной конструкции переопределениям метода и выполняются проверки на корректность его имени и отсутствие множественного переопределения. В случае обнаружения проблемы, пользователю выводится соответствующее сообщение об ошибке.

```
for (m : methods) {
      if (!counts.containsKey(m.name))
            error("Error - incorrect default method name", m,
                  m.getNameStructuralFeature
            )
      else if (counts.get(m.name).count > 0)
            error("Error - default method cannot be set more than
                        once", m, m.getNameStructuralFeature
            )
      else {
            var count = counts.get(m.name)
            count.count++
            counts.put(m.name, count)
      }
}
```

После проведения данных проверок производятся дополнительные проверки для методов, которые не были переопределены ни разу. Пользователю выводится ошибка или предупреждение в зависимости от заданного в классе *DefaultMethodsHelper* действия.

5.3. Реализация функций работы с кодом модели

5.3.1. Форматирование кода

Реализация правил форматирования кода модели производится в классе *RaoFormatter*. Для этого переопределяется метод *format()* данного класса для каждого из возможных методов модели.

Для многих элементов модели требуется форматирование, аналогичное форматированию блоков кода в языке Java. Для описания правил подобного форматирования был написан вспомогательный класс метод *formatAsBlock()*:

```
def formatAsBlock(EObject obj, extension IFormattableDocument document) {
	val open = obj.regionFor.keyword("{")
	val close = obj.regionFor.keyword("}")

	if (obj.eContainer == null)
		obj.surround[noSpace]

	interior(open, close)[indent]
	open.prepend(XbaseFormatterPreferenceKeys.bracesInNewLine)
	open.append(
		XbaseFormatterPreferenceKeys.blankLinesAroundExpression)

	close.append(
		XbaseFormatterPreferenceKeys.blankLinesAroundExpression)
}
```

В качестве примера ниже рассмотрено описание правил форматирования кода для генераторов чисел:

```
def dispatch void format(Generator generator,
		extension IFormattableDocument document) {
	formatAsBlock(generator, document)

	for (parameter : generator.parameters)
		format(parameter, document)

	format(generator.body, document)

	generator.body.append(
		XbaseFormatterPreferenceKeys.blankLinesAroundExpression)
}
```

Помимо форматирования тела самого генератора как блока кода, вызываются также методы форматирования для всех внутренних элементов

генератора (его параметров и его тела). Так как данные элементы описываются грамматическими правилами, предоставляемыми грамматикой Xbase, то дополнительно описывать правила их форматирования не требуется - в фреймворке Xbase данные правила уже реализованы.

Соответствующим образом были реализованы правила форматирования для всех элементов модели. По умолчанию автоматическое форматирование вызывается в системе Rao X выделением участка кода, который необходимо форматировать, и последующим нажатием сочетания клавиш Ctrl + Shift + F.

5.3.2. Автодопополнение

Большинство грамматических правил языка РДО, требующих использования автодополнения при работе, описываются грамматикой, предоставляемой Xbase, и потому не требуют описания дополнительных правил по составлению списка подсказок, который выводится при обращении к функциям автодополнения.

Тем не менее подобные правила требуется в явном виде описать для имен стандартных методов. Реализация правил автодополнения производится в классе *RaoProposalProvider*.

Выполняется переопределение метода, отвечающего за автодополнение при написании стандартных методов:

```
override completeDefaultMethod_Name(EObject model,
          Assignment assignment, ContentAssistContext context,
          ICompletionProposalAcceptor acceptor) {
     var EObject toComplete

     if (model instanceof DefaultMethod) {
          toComplete = model.eContainer
     } else {
          toComplete = model
     }

     if (!(toComplete instanceof Pattern || toComplete instanceof Frame
               || toComplete instanceof Logic
               || toComplete instanceof Search
               || toComplete instanceof RaoModel))
```

```
            return;

      internalCompleteDefaultMethod_Name(
                  toComplete, context, acceptor)
}
```

В данном метода выполняются необходимые проверки на то, является ли родительских элемент грамматики метода по умолчанию валидным, и вызывается внутренний метод *internalCompleteDefaultMethod_Name()*, который, в свою очередь, определен только для тех элементов грамматики, которые могут содержать внутри себя методы по умолчанию.

Для реализации подсказок имен методов по умолчанию используется вспомогательный класс *DefaultMethodsHelper*, который также использовался для реализации валидации стандартных методов. Пример реализации внутренненего метода для образцов представлен ниже:

```
def dispatch internalCompleteDefaultMethod_Name(
      Pattern pattern, ContentAssistContext context,
ICompletionProposalAcceptor acceptor) {

      switch (pattern.type) {
      case OPERATION: {
            for (value: DefaultMethodsHelper.OperationMethodInfo.values) {
                  acceptor.accept(
                        createCompletionProposal(value.name, context))
            }
      }
      case RULE: {
            for (value: DefaultMethodsHelper.RuleMethodInfo.values) {
                  acceptor.accept(
                        createCompletionProposal(value.name, context))
            }
      }
      }
}
```

В данном методе производится проверка на тип образца и в качестве подсказываемых имен выбираются валидные имена стандартных методов для данного типа образца.

Аналогичным образом реализованы методы по автодополнению кода стандартных методов и для остальных элементов модели.

6. Исследовательская часть

В данной части дипломного проекта было проведено исследование эффективности использования механизма копирования при записи в работе с состояниями модели при проведении поиска по графу.

Каждая вершина графа в контексте имитационного моделирования представляет из себя одно из возможных состояний модели. При порождении каждой последующей вершины перед выполнение продукционных правил необходимо произвести копирование состояния модели, чтобы действия, выполняемые в правиле, не изменили состояние модели в родительской вершине.

Работа механизма копирования при записи в данном случае будет заключаться в следующем: при порождении новой вершины не создается глубокой копии состояния модели, а создается лишь пустой каркас (псевдо-копия), в котором все ресурсы являются ресурсами из родительского состояния. При этом все ресурсы, попавшие в псевдо-копию, помечаются как требующие копирования. В момент, когда в продукционном правиле один из параметров ресурса должен измениться, производится глубокое копирование данного ресурса (остальные же ресурсы продолжают быть псевдо-копиями, и потерь времени на их копирование не происходит).

Для исследования скорости работы алгоритмов полного копирования и копирования при записи была использована модель *game5*, производящая поиск решения в игре "Пятнашки". Модель была модифицирована таким образом, чтобы возможно было варьировать размер поля пятнашек. Кроме того, начальная конфигурация создавалась таким образом, чтобы она оказывалась нерешаемой. Это гарантирует порождение максимально возможного числа вершин для каждого из размеров полей.

Для удобства варьирования размером поля пятнашек в модель было введено две константы:

```
constant длина_поля = 64
constant высота_поля = 64
```

Для гарантии получения нерешаемой комбинации были произведены следующие модификации модели:

- Дырка всегда создается в правом нижнем углу поля

```
resource дырка = Дырка.create(длина_поля * высота_поля)
```

- Создание фишек производится в момент инициализации модели в цикле. При этом все фишки, кроме двух последних, помещаются на места, соответствующие их номерам. Две последние фишки "переставляются местами".

```
set init() {
      val размер_поля = длина_поля * высота_поля
      for (var i = 1; i < размер_поля - 2; i++) {
            Фишка.create(i, i)
      }
      Фишка.create(размер_поля - 2, размер_поля - 1)
      Фишка.create(размер_поля - 1, размер_поля - 2)
}
```

Производилось измерение числа порожденных вершин при вариации следующих параметров:

- Размер поля.
- Размер параметров фишки. Чем больше байт занимает объект, представляющий фишку, тем более дорогостоящим является его копирование.
- Время работы модели.

6.1. Измерение числа порожденных вершин в зависимости от размера поля

При проведении данного измерения время моделирования и размер параметров фишки фиксируются и принимают следующие значения:

- Время моделирования: 300 секунд.
- Размер параметров фишки: 8 байт (оригинальная задача).

Для каждого из размеров поля произведено 3 измерения и подсчитано среднее арифметическое. Полученные значения были округлены до целых и занесены в таблицу 6.1.

Таблица 6.1 – Число порожденных вершин в зависимости от размера поля

Размер поля	Число порожденных вершин	
	Полное копирование	Копирование при записи
4 x 4	34220	35691
8 x 8	27454	28998
16 x 16	9033	9450
32 x 32	1419	1866
64 x 64	313	422

Из приведенной таблицы видно, что алгоритм копирования при записи обеспечивает порождение большего числа вершин при любом размере поля. При этом чем больше размер поля, тем заметнее выигрыш при использовании алгоритма копирования при записи по сравнению с алгоритмом полного копирования.

В случае размера поля 4x4 алгоритм копирования при записи позволил за то же время породить на 4.3% больше вершин.

В случае размера поля 64x64 алгоритм копирования при записи позволил за то же время породить на 34.8% больше вершин.

6.2. Измерение числа порожденных вершин в зависимости от размера параметров фишки

При проведении данного измерения время моделирования и размер поля фиксируются и принимают следующие значения:

- Время моделирования: 300 секунд.
- Размер поля: 3 x 3.

Для каждого из размеров поля произведено 3 измерения и подсчитано среднее арифметическое. Полученные значения были округлены до целых и занесены в таблицу 6.2.

Таблица 6.2 – Число порожденных вершин в зависимости от размера параметров фишки

Размер параметров фишки, байт	Число порожденных вершин	
	Полное копирование	Копирование при записи
8 (оригинальная задача)	42776	45904
16	39209	43011
32	35175	41410
64	33240	38800
128	32075	35562
256	28837	31980
512	27002	30065

Из приведенной таблицы видно, что алгоритм копирования при записи обеспечивает порождение большего числа вершин при любом размере параметров фишки. При этом чем больше размер поля, тем заметнее выигрыш при использовании алгоритма копирования при записи по сравнению с алгоритмом полного копирования.

В случае размера параметров фишки 8 байт алгоритм копирования при записи позволил за то же время породить на 7.3% больше вершин.

В случае размера параметров фишки 64 байта алгоритм копирования при записи позволил за то же время породить на 11.3% больше вершин.

6.3. Измерение числа порожденных вершин в зависимости от времени работы модели

Так как в двух предыдущих эксперимента было выявлено, что алгоритм копирования при записи при малых значениях размера поля и размера параметров фишки имеет незначительный выигрыш по производительности, а при больших значениях данных параметров выигрыш становится куда более существенным, то было принято решение провести два дополнительных эксперимента, а не один.

При проведении первого эксперимента были выбраны небольшие значения размера поля и размера параметров фишки:

- Размер поля: 4x4
- Размер параметров фишки: 8 байт (оригинальная задача)

Для каждого из размеров поля произведено 3 измерения и подсчитано среднее арифметическое.

При проведении второго эксперимента были выбраны большие значения размера поля и размера параметров фишки:

- Размер поля: 32x32
- Размер параметров фишки: 512 байт

6.4. Выводы

Алгоритм копирования при записи показал лучшие результаты по сравнению с алгоритмом полного копирования на всех проведенных тестах. Таким образом, данный алгоритм целесообразно использовать при поиске по графу в системе Rao X.

7. Апробирование системы

7.1. Анализ существующей системы автоматического тестирования

Система автоматического тестирования, которую имела среда разработки имитационных моделей RAO-XT, позволяла проводить сборку системы и юнит-тестирование как локально, так и автоматически на удаленном сервере.

Существовавшая архитектура тестирования позволяла проводить тестирование отдельных компонентов библиотеки RAO-XT, но не позволяла производить тестирование графических компонентов системы и тестирование подсистемы кодогенерации.

Чтобы получить возможность проводить всестороннее тестирование системы Rao X необходимо было реализовать подсистему интеграционного тестирования. Данная подсистема была реализована с помощью фреймворка RCPTT[7].

7.2. Фреймворк RCPTT

RCPTT (Rich Client Platform Testing Tool) – это фреймворк, предназначенный для автоматизированного тестирования GUI приложений, разработанных на базе Eclipse. RCPTT скрывает внутреннее устройство Eclipse от конечного пользователя, предоставляя вместо этого возможность описать сценарий теста с помощью скриптового языка ECL.

Помимо ручного запуска созданных тестов непосредственно из среды Eclipse, RCPTT позволяет запускать все существующие тесты с помощью консольного интерфейса. Данный интерфейс предоставляется плагином для Eclipse - *Test Runner*. Помимо предоставления консольного интерфейса, *Test Runner* также позволяет логировать ошибки, генерировать отчеты в формате HTML, корректно обрабатывать зависание тестируемого приложения.

7.3. Реализация интеграционных тестов

В рамках реализации подсистемы интеграционного тестирования были созданы тесты 3-х категорий:

- Базовые тесты.
- Тесты моделей, созданных по шаблонам визарда проектов.
- Тесты моделей, создаваемых из текстового шаблона.

В общей сложности было создано более 30 интеграционных тестов.

Код функций, которые использовались для реализации тестов всех категорий, приведен в приложении В.

7.3.1. Базовые тесты

В тестах данной категории не производится запуска моделей на языке РДО, а проводится лишь тестирование наиболее базовых компонентов среды: создание проекта, открытие перспективы Rao и т.п.

7.3.2. Тесты моделей, созданных по шаблонам визарда проектов

Тесты данной категории производят проверку корректности работы стандартных моделей, которые позволяет создать визард проектов Rao X. Все тесты в данной категории представлены в 2-х вариациях:

- Тестирование без трассировки. Простой запуск модели и проверка успешности ее завершения.
- Тестирование с трассировкой. Запуск модели с включенной сериализацией и посимвольное сравнение вывода трассировки модели с заранее сохраненным в тестовой системе эталоном.

Для тестирования с трассировкой моделей, созданных по шаблонам визарда, осуществляются следующие действия:

- Открыть перспективу Rao.
- Перейти в визард проектов Rao.
- Выбрать шаблон и имя модели исходя из начальных условий теста, создать проект.
- Включить трассировку с помощью окна *Monitoring Configuration*.
- Запустить прогон модели.
- Произвести экспорт трассировочной информации из окна *Trace View*.
- Сравнить экспортированную трассировочную информацию с эталонной.

В данной категории были реализованы тесты следующих моделей:

- Пустая модель.
- Модель простейшей СМО.
- Модель простейшей СМО на событиях.
- Модель простейшей СМО с клиентами.

7.3.3. Тесты моделей, создаваемых из текстового шаблона

Тесты данной категории производят проверку корректности работы произвольных моделей, исходный текст которых был заранее сохранен в тестовой системе.

Как и в случае с моделями, созданным по шаблонам визарда проектов, тестирование проводится в двух вариациях: без трассировки и с трассировкой.

Как правило, тесты в данной категории стремятся проверять какую-то одну функциональную возможность языка и, насколько это возможно, избежать использования остальных возможностей языка или минимизировать его.

Для тестирования с трассировкой моделей, создаваемых из текстового шаблона, осуществляются следующие действия:

- Открыть перспективу Rao.
- Перейти в визард проектов Rao.
- Выбрать имя модели исходя из начальных условий теста, выбрать шаблон "Пустая модель", создать проект.
- Скопировать содержимое текстового шаблона в пустой файл модели, который был создан визардом.
- Включить трассировку с помощью окна *Monitoring Configuration*.
- Запустить прогон модели.
- Произвести экспорт трассировочной информации из окна *Trace View*.
- Сравнить экспортированную трассировочную информацию с эталонной.

В данной категории были реализованы модели, тестирующий следующий функционал:

- Генераторы.
- Подбор релевантных ресурсов по условию доступности.
- Последовательности типа Uniform.
- Последовательности типа Normal.
- Последовательности типа Exponential.
- Последовательности типа Triangular.
- Последовательности типа Values.
- Последовательности типа ContinuousHistogram.
- Последовательности типа DiscreteHistogram.
- Точки принятия решений типа search (поиск по графу).

7.4. Автоматическое интеграционное тестирование на удаленном сервере

Для своевременного выявления и локализации ошибок в системе необходимо проводить тестирование после любого изменения, которое было внесено в систему. Т.к. полный прогон всех тестов - это достаточно длительный процесс, то такое тестирование следует выполнять на удаленном сервере в автоматическом режиме.

Кафедральные сервера, на которых производится автоматическая сборка и тестирование системы Rao X, не обладают десктопным окружением. В таких условиях для проведения тестов, требующих использования графического пользовательского интерфейса, требуется запускать на сервере дисплейный менеджер и открывать сессию с ним. По завершению тестирования сессию с дисплейным менеджером необходимо прекращать.

Для реализации данной возможности был использован плагин *Xvnc Plugin* для платформы *Jenkins*[12]. Данный плагин позволяет автоматически открывать сессию с сервером *XVNC* (VNC-сервер для дисплейного менеджера X) на время проведения тестирования.

Таким образом, при наличии запущенного дисплейного менеджера для проведения автоматического тестирования требуется выполнить лишь следующие действия:

- Загрузить на сервер последние исходники Rao X из репозитория системы контроля версий git с указанием ветки, на которой ведется разработка.
- Собрать архив с плагинами Rao X и установить плагин в заранее сохраненную на сервере среду разработки Eclipse.
- Запустить тестирование, используя плагин *Test Runner*, предоставляемый фреймворком RCPTT.

Скрипт для платформы Jenkins, позволяющий выполнить описанные действия приведен ниже:

```
export MAVEN_OPTS="-Xmx512M"
mvn initialize -N
mvn -U clean package

cp assembly/target/plugins/*
     /home/rdo/eclipses/to_test/eclipse/dropins/plugins/

./ru.bmstu.rk9.rao.rcptt/run_tests.sh \
    /home/rdo/rcptt/eclipse \
    /home/rdo/eclipses/to_test/eclipse
```

Для удоства запуска полного интеграционного тетстирования как на удаленном сервере был написан скрипт *run_tests.sh*, принимающий в качестве параметров два пути:

- Путь к Eclipse, тестирование которого будет проводиться (target).
- Путь к Eclipse, содержащему плагин *Test Runner*, который будет инициировать и контролировать тестирование.

По итогам работы было запущено регрессионное тестирование на ветке репозитория raox, на которой велась разработка дипломного проекта.

8. Организационно-экономическая часть

8.1. Введение

Выполнен расчет затрат на разработку компонентов, составляющих ядро системы имитационного моделирования Rao X.

Для этого были произведены следующие расчеты:

- Расчет трудоемкости разработки программного продукта.
- Расчет суммарных затрат на разработку программного продукта и расчет его цены.

Расчет действителен на 2-ой квартал 2016 года (цены на программное обеспечение, оборудование, расходные материалы, уровень заработной платы исполнителей).

8.2. Организация и планирование процесса разработки ПП

Организация и планирование процесса разработки ПП предусматривает выполнение следующих работ:

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

Трудоемкость разработки ПП зависит от ряда факторов:

По степени новизны разрабатываемый программный продукт относится к группе новизны «В» (разработка ПП, имеющего аналоги).

По степени сложности алгоритма функционирования – к 1 группе сложности (реализующие оптимизационные и моделирующие алгоритмы).

По виду представления исходной информации ПП относится к группе 11 – исходная информация представлена в виде документов, имеющих различный формат и структуру.

По структуре выходной информации ПП относится к группе 22 – требуется вывод на печать одинаковых документов, вывод информационных носителей на машинные носители.

Укрупненный состав работ по стадиям разработки ПП по ГОСТ 19.004-80 и ГОСТ 19.102-77 приведен в таблице 8.1.

Таблица 8.1 – Укрупненный состав работ по стадиям разработки ПП

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. 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 приема-сдачи работ. Проверка алгоритмов и программ решения задач, корректировка документации после опытной эксплуатации программного продукта.

8.2.1. Расчет трудоемкости разработки технического задания

Трудоемкость разработки технического задания рассчитывается по формуле:

$$\tau_{\text{ТЗ}} = T^{3}_{\text{РЗ}} + T^{3}_{\text{РП}} \quad (1)$$

$T^{3}_{\text{РЗ}}$ - затраты времени разработчика постановки задачи на разработку ТЗ, чел.-дни;

$T^{3}_{\text{РП}}$ – затраты времени разработчика ПП на разработку ТЗ, чел.-дни.

$$T^{3}_{\text{РЗ}} = t_{\text{З}} \cdot K^{3}_{\text{РЗ}} \quad (2)$$

$$T^{3}_{\text{РП}} = t_{\text{З}} \cdot K^{3}_{\text{РП}} \quad (3)$$

$t_{\text{З}}$ - норма времени на разработку ТЗ на ПП; чел.-дни;

$K^{3}_{\text{РЗ}}$ - коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задачи на стадии ТЗ;

$K^{3}_{\text{РП}}$ - коэффициент, учитывающий удельный вес трудоемкости работы, выполняемых разработчиком ПП на стадии ТЗ.

Исходя из рекомендаций [8] для группы новизны В и 1 группы сложности:

$t_З = 37$ чел.-дн

Т.к. разработка ТЗ выполняется разработчиком постановки задачи и разработчиком ПО совместно:

$K^З_{РЗ} = 0.65$

$K^З_{РП} = 0.35$

Таким образом:

$\tau_{ТЗ} = T^З_{РЗ} + T^З_{РП} = t_З \cdot K^З_{РЗ} + t_З \cdot K^З_{РП} = 37 \cdot 0.65 + 37 \cdot 0.35 = 37$ чел.-дн

8.2.2. Расчет трудоемкости разработки эскизного проекта

Трудоемкость разработки эскизного проекта рассчитывается по формуле:

$$\tau_{ЭП} = T^Э_{РЗ} + T^Э_{РП} \quad (4)$$

$T^Э_{РЗ}$ - затраты времени разработчика постановки задачи на разработку ЭП, чел.-дни;

$T^Э_{РП}$ – затраты времени разработчика ПП на разработку ЭП, чел.-дни.

$$T^Э_{РЗ} = t_Э \cdot K^Э_{РЗ} \quad (5)$$

$$T^Э_{РП} = t_Э \cdot K^Э_{РП} \quad (6)$$

$t_Э$ - норма времени на разработку ЭП ПП; чел.-дни;

$K^Э_{РЗ}$ - коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задачи на стадии ЭП;

$K^Э_{РП}$ - коэффициент, учитывающий удельный вес трудоемкости работы, выполняемых разработчиком ПП на стадии ЭП.

Исходя из рекомендаций [8] для группы новизны В и 1 группы сложности:

$t_З = 77$ чел.-дн

Т.к. разработка ЭП выполняется разработчиком постановки задачи и разработчиком ПО совместно:

$K_{\text{РЗ}}^{\text{Э}} = 0.7$

$K_{\text{РП}}^{\text{Э}} = 0.3$

Таким образом:

$\tau_{\text{ЭП}} = T_{\text{РЗ}}^{\text{Э}} + T_{\text{РП}}^{\text{Э}} = t_{\text{Э}} \cdot K_{\text{РЗ}}^{\text{Э}} + t_{\text{Э}} \cdot K_{\text{РП}}^{\text{Э}} = 77 \cdot 0.7 + 77 \cdot 0.3 = 77$ чел.-дн

8.2.3. Расчет трудоемкости выполнение технического проекта

Трудоемкость выполнения технического проекта $\tau_{\text{ТП}}$ зависит от функционального назначения ПП, количества разновидностей форм входной и выходной информации и определяется как сумма времени, затраченного разработчиком постановки задач и разработчиком программного обеспечения.

$$\tau_{\text{ТП}} = \left(t_{\text{РЗ}}^{\text{Т}} + t_{\text{РП}}^{\text{Т}}\right) \cdot K_{\text{В}} \cdot K_{\text{Р}} \qquad (7)$$

$t_{\text{РЗ}}^{\text{Т}}$, $t_{\text{РП}}^{\text{Т}}$ - норма времени, затрачиваемого на разработку ТП разработчиком постановки задач и разработчиком программного обеспечения соответственно, чел-дни;

$K_{\text{В}}$ - коэффициент учета вида используемой информации;

$K_{\text{Р}}$ - коэффициент учета режима обработки информации.

Нормы времени $t_{\text{РЗ}}^{\text{Т}}$, $t_{\text{РП}}^{\text{Т}}$ определяются в зависимости в количества разновидностей форм входной и выходной информации и функционального назначения.

Количество форм входной информации - 1.

Количество форм выходной информации - 4.

Тогда исходя из рекомендаций [8, табл. 4]:

$t_{\text{РЗ}}^{\text{Т}} = 46$ чел-дн;

$t_{\text{РП}}^{\text{Т}} = 10$ чел-дн;

Коэффициент учета вида используемой информации вычисляется по следующей формуле:

$$K_{\text{В}} = \frac{K_{\text{П}} \cdot n_{\text{П}} + K_{\text{НС}} \cdot n_{\text{НС}} + K_{\text{Б}} \cdot n_{\text{Б}}}{n_{\text{П}} + n_{\text{НС}} + n_{\text{Б}}} \quad (8)$$

$K_{\text{П}}$, $K_{\text{НС}}$, $K_{\text{Б}}$ - значения коэффициентов учета вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно;

$n_{\text{П}}$, $n_{\text{НС}}$, $n_{\text{Б}}$ - количество наборов данных для переменной, нормативно-справочной информации и баз данных соответственно;

В данном случае:

$n_{\text{П}} = 1$

$n_{\text{НС}} = 0$

$n_{\text{Б}} = 0$

Для группы новизны В:

$K_{\text{П}} = 1.00$

$K_{\text{НС}} = 0.72$

$K_{\text{Б}} = 2.08$

Таким образом:

$$K_{\text{В}} = \frac{K_{\text{П}} \cdot n_{\text{П}} + K_{\text{НС}} \cdot n_{\text{НС}} + K_{\text{Б}} \cdot n_{\text{Б}}}{n_{\text{П}} + n_{\text{НС}} + n_{\text{Б}}} = \frac{1.00 \cdot 1 + 0.72 \cdot 0 + 2.08 \cdot 0}{1 + 0 + 0} = 1$$

Коэффициент учета режима обработки информации:

$K_{\text{Р}} = 1.26$

Таким образом:

$\tau_{\text{ТП}} = \left(t_{\text{РЗ}}^{\text{Т}} + t_{\text{РП}}^{\text{Т}}\right) \cdot K_{\text{В}} \cdot K_{\text{Р}} = (46 + 10) \cdot 1 \cdot 1.26 = 70.56$ чел-дн;

8.2.4. Расчет трудоемкости выполнения рабочего проекта

Трудоёмкость разработки рабочего проекта зависит от функционального назначения ПП, количества разновидностей форм входной и выходной

информации, сложности алгоритма функционирования, сложности контроля информации, степени использования готовых программных модулей, уровня алгоритмического языка программирования.

$$\tau_{\text{РП}} = \left(t_{\text{РЗ}}^{\text{Р}} + t_{\text{РП}}^{\text{Р}}\right) \cdot K_{\text{К}} \cdot K_{\text{Р}} \cdot K_{\text{Я}} \cdot K_{\text{З}} \cdot K_{\text{ИА}} \qquad (9)$$

$t_{\text{РЗ}}^{\text{Р}}$, $t_{\text{РП}}^{\text{Р}}$ - норма времени, затраченного на разработку РП на алгоритмическом языке высокого уровня разработчиком постановки задач и разработчиком программного обеспечения соответственно, чел-дни;

$K_{\text{К}}$ - коэффициент учета сложности контроля информации;

$K_{\text{Я}}$ - коэффициент учета уровня используемого алгоритмического языка программирования;

$K_{\text{З}}$ - коэффициент учета степени использования готовых программных модулей;

$K_{\text{ИА}}$ - коэффициент учета вида используемой информации и сложности алгоритма ПП;

Исходя из [8, табл. 35]:

$t_{\text{РЗ}}^{\text{Р}} = 14$ чел-дн;

$t_{\text{РП}}^{\text{Р}} = 86$ чел-дн;

Исходя из степени сложности контроля входной и выходной информации (для типов входной и выходной информации 11 и 22 соответственно):

$K_{\text{К}} = 1.07$

Для языков программирования Java и Xtend:

$K_{\text{Я}} = 1.00$

Коэффициент использования готовых программных модулей:

$K_{\text{З}} = 0.7$

Значение коэффициента вида используемой информации и сложности алгоритма ПП определяется по следующей формуле:

$$K_{\text{ИА}} = \frac{K'_{\text{П}} \cdot n_{\text{П}} + K'_{\text{НС}} \cdot n_{\text{НС}} + K'_{\text{Б}} \cdot n_{\text{Б}}}{n_{\text{П}} + n_{\text{НС}} + n_{\text{Б}}} \qquad (10)$$

Для сложности алгоритма 1 и группы новизны В:

$K'_{\text{П}} = 1.20$

$K'_{\text{НС}} = 0.65$

$K'_{\text{П}} = 0.54$

Таким образом:

$$K_{\text{ИА}} = \frac{K'_{\text{П}} \cdot n_{\text{П}} + K'_{\text{НС}} \cdot n_{\text{НС}} + K'_{\text{Б}} \cdot n_{\text{Б}}}{n_{\text{П}} + n_{\text{НС}} + n_{\text{Б}}} = \frac{1.20 \cdot 1 + 0.65 \cdot 0 + 0.54 \cdot 0}{1 + 0 + 0} = 1.2$$

$$\tau_{\text{РП}} = \left(t^{\text{Р}}_{\text{РЗ}} + t^{\text{Р}}_{\text{РП}}\right) \cdot K_{\text{К}} \cdot K_{\text{Р}} \cdot K_{\text{Я}} \cdot K_{\text{З}} \cdot K_{\text{ИА}} = (14 + 86) \cdot 1.07 \cdot 1.26 \cdot 1 \cdot 0.7 \cdot 1.2 = 113.249$$

8.2.5. Расчет трудоемкости выполнения внедрения

Трудоемкость выполнения стадии "Внедрение" рассчитывается по формуле:

$$\tau_{\text{В}} = \left(t^{\text{В}}_{\text{РЗ}} + t^{\text{В}}_{\text{РП}}\right) \cdot K_{\text{К}} \cdot K_{\text{Р}} \cdot K_{\text{З}} \qquad (11)$$

$t^{\text{В}}_{\text{РЗ}}$, $t^{\text{В}}_{\text{РП}}$ - норма времени, затрачиваемая разработчиком постановки задачи и расзработчиком программного обеспечения соответственно на выполнение процедур внедрения ПП, чел-дни;

Исходя из [8, табл. 36]:

$t^{\text{В}}_{\text{РЗ}} = 16$ чел-дн;

$t^{\text{В}}_{\text{РП}} = 19$ чел-дн;

Таким образом:

$\tau_{\text{В}} = \left(t^{\text{В}}_{\text{РЗ}} + t^{\text{В}}_{\text{РП}}\right) \cdot K_{\text{К}} \cdot K_{\text{Р}} \cdot K_{\text{З}} = (33 + 96) \cdot 1.07 \cdot 1.26 \cdot 0.7 = 33.031$ чел-дн;

8.2.6. Расчет суммарной трудоемкости

Общая трудоемкость разработки ПП:

$\tau_{ПП} = \tau_{ТЗ} + \tau_{ЭП} + \tau_{ТП} + \tau_{РП} + \tau_{В} = 37 + 77 + 70.56 + 113.249 + 33.031 = 330.84$ чел.-дн.

Трудоемкость разработки программного продукта по этапам приведена в таблице 8.2.

Таблица 8.2 – Трудоемкость этапов разработки ПП

№	Стадия разработки	Трудоемкость, чел.-дни;
1	Техническое задание	37
2	Эскизный проект	77
3	Технический проект	70.56
4	Рабочий проект	113.249
5	Внедрение	33.031
Итого:		330.84

Принимаем:

$\tau_{ПП} = 331$ чел.-дн;

8.3. Определение цены программной продукции

Для определения стоимости работ необходимо на основании плановых сроков выполнения работ и численности исполнителей рассчитать общую сумму затрат на разработку программного продукта.

Если ПП рассматривается и создается как продукция производственно-технического назначения, допускающая многократное тиражирование и отчуждение от непосредственных разработчиков, то ее цена определяется по формуле:

$$Ц = К \cdot С + Пр \quad (12)$$

С - затраты на разработку ПП (сметная себестомость), руб.;

К - коэффициент учета затрат на изготовления опытного образца ПП как продукции производственно-технического назначения;

Пр - нормативная прибыль, руб.

Затраты, образующие себестоимость ПП, группируются в соответствии с иъ экономическим содержанием по следующим элементам:

- амортизация оборудования и нематериальных активов;
- основная и дополнительная заработные платы;
- отчисления на социальные нужны;
- накладные расходы.

8.3.1. Расчет амортизации оборудования и нематериальных активов

В данной статье учитываются суммарные затраты на приобретение/амортизацию оборудования и нематериальных активов, требуемых для разработки данного ПП.

$$\mathrm{С_{со}} = \sum_{i} \frac{\mathrm{Ц_{б}}i \cdot \alpha_i}{100 \cdot F_{\mathrm{Д}}} \cdot t_i \qquad (13)$$

$\mathrm{Ц_{б}}i$ - балансовая цена i-ого вида оборудования, руб.;

α_i - норма годовых амортизационных отчисления для оборудования i-ого вида, %;

$F_{\mathrm{Д}}$ - действительный годовой фонд времени на ПК;

t_i - время использования i-ого вида оборудования при выполнении данной разработки, ч;

Исходя из 5-и дневной рабочей недели и 8-и часового рабочего дня, действительный годовой фонд времени принимается равным:

$F_{\mathrm{Д}} = 2080$ ч

Согласно [8, табл. 50]:

$\alpha_i = 20.0$

Перечень необходимого оборудования и его стоимость приведены в таблице 8.3.

Таблица 8.3. – Стоимость оборудования

№	Наименования	Единица измерения	Количество	Цена за единицу, руб	Сумма, руб
1	ПЭВМ	шт	1	45000	45000

Суммарные затраты на оборудование:

$$С_{со} = \sum_i \frac{Ц_{бi} \cdot \alpha_i}{100 \cdot F_Д} \cdot t_i = \frac{45000 \cdot 20.0}{100 \cdot 2080} \cdot 331 \cdot 8 = 11454 \text{ руб.}$$

8.3.2. Расчет основной заработной платы

В статью включается основная заработная плата всех исполнителей, непосредственно занятых разработкой данного ПП, с учетом их должностного оклада и времени участия в разработке.

$$С_{ЗО} = \sum_i \frac{З_i}{d} \cdot \tau_i \qquad (14)$$

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

d - среднее количество рабочих дней в месяце;

τ_i - трудоемкость работ, выполняемых *i*-м исполнителем, чел.-дн.

$$С_{ЗО} = \frac{42000}{21} \cdot 331 = 708900 \text{ руб.}$$

8.3.3. Расчет дополнительной заработной платы

В статье учитываются выплаты непосредственным исполнителям за время (установленное законодательством), непроработанное на производстве, в то числе: оплата очередных отпусков, компенсации за недоиспользованный отпуск, оплата льготных часов подросткам и др.

$$С_{ЗД} = С_{ЗО} \cdot \alpha_{Д} \quad (15)$$

$\alpha_{Д}$ - коэффициент отчислений на дополнительную заработную плату

$\alpha_{Д} = 0.2$

$С_{ЗД} = С_{ЗО} \cdot \alpha_{Д} = 708900 \cdot 0.2 = 141800$ руб.

8.3.4. Расчет отчислений на социальное страхование

В статье учитываютя отчисления в бюджет социального страхования по установленному законодательством тарифу от суммы основной и дополнительной заработной платы.

$$С_{СС} = \alpha_{СС} \cdot (С_{ЗО} + С_{ЗД}) \quad (16)$$

$\alpha_{СС}$ - коэффициент отчислений на социальное страхование

$\alpha_{СС} = 0.302$

$С_{СС} = 0.302 \cdot (708900 + 141800) = 256920$ руб.

8.3.5. Расчет накладных расходов

В статье учитываются затраты на общехозяйственных расходы, непроизводственные расходы и расходы на управление. Накладные расходы определяют в процентном отношении к основной заработной плате.

$$С_{Н} = С_{ЗО} \cdot \alpha_{Н} \quad (17)$$

$\alpha_{Н}$ - коэффициент накладных расходов

$\alpha_{Н} = 1.8$

$С_{Н} = С_{ЗО} \cdot \alpha_{Н} = 708900 \cdot 1.8 = 1276000$ руб.

8.3.6. Расчет суммарных расходов

Результаты расчетов затрат на разработку ПП приведены в таблице 8.4.

Таблица 8.4 – Результаты расчет затрат на разработку ПП

Наименование статьи	Сметная стоимость, руб.
Затраты на нематериальные активы и амортизация оборудования	6900
Основная заработная плата	708900
Дополнительная заработная плата	141800
Отчисления в социальные фонды	256920
Накладные расходы	1276000
Итого	2390600

$С = С_{СО} + С_{ЗО} + С_{ЗД} + С_{СС} + С_{Н} = 11454 + 708900 + 141800 + 256920 + 1276000 = 2395074$ руб.

Цена ПП:

$$Ц = К \cdot С + Пр \quad (18)$$

$К = 1.1$

Т.к. разработка ПП ведется с целью использования в учебном учреждении, принимаем $Пр = 0$.

$Ц = 1.1 \cdot 2395074 + 0 = 2634581$ руб.

8.4. Вывод

Проведенный расчет показал следующее:

- Суммарная трудоемкость разработки программного продукта составляет 331 чел.-дн.
- Цена программного продукта составляет 2 634 581 руб.

9. Мероприятия по охране труда и технике безопасности

9.1. Анализ опасных и вредных факторов

Рассмотрены опасные и вредные факторы, действующие на пользователя системы Rao X в процессе ее эксплуатации. Рассмотренные факторы представлены в таблице 9.1.

Таблица 9.1. – Опасные и вредные факторы согласно ГОСТ 12.0.003-74

№	Фактор	Опасный	Вредный
	Физические		
1	Повышенный или пониженный уровень освещенности		+
2	Неравномерность распределения яркости в поле зрения		+
3	Повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека	+	
4	Повышенные уровни электромагнитного излучения		+
5	Повышенные уровни запыленности воздуха рабочей зоны		+
6	Повышенный уровень шума		+
7	Повышенный уровень вибрации		+
	Психофизиологические		
8	Монотонность труда		+
9	Интеллектуальные нагрузки		+

9.1.1. Повышенный или пониженный уровень освещенности

Причины возникновения:

- Наличие на рабочем месте как искусственного, так и естественного освещения.

Влияние на человека:

Недостаточно освещение вызывает преждевременное утомление, притупляет внимание рабочего, снижает производительнось труда, ухудшает показатели и может оказаться причиной несчастного случая. [10]

Регламентирующие документы:

СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы»

Предлагаемые меры по обеспечению безопасности:

- Проведение расчета освещения в помещении, в котором располагается рабочее место, и соответствующая коррекция осветительного оборудования.
- Проведение инструктажа персонала.

9.1.2. Неравномерность распределения яркости в поле зрения

Причины возникновения:

- Создание местным освещением бликов на поверхностях экранов ПЭВМ

Влияние на человека:

Повышенная отражательная способность экранов мониторов, а также ослепление слишком ярким источником света и частая переадаптация утомляют глаза.

Неравномерность распределения яркости в поле зрения значительно увеличивает нагрузку на мышцы глаз. Это вызывает повышенную усталость органа зрения, а в последующем — развитие близорукости. [11]

Регламентирующие документы:

СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы»

Предлагаемые меры по обеспечению безопасности:

- Проведение расчета освещения в помещении, в котором располагается рабочее место и замена освещения при необходимости.
- Проведение инструктажа персонала.

9.1.3. Повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека

Помещение, в котором располагаются ПЭВМ относится к I классу (помещения без повышенной опасности) по критерию опасности поражения электрическим током.

Причины возникновения:

- Наличие в помещении устройств, требующих электропитания от сети переменного тока с напряжением 220В.

Влияние на человека:

Проходя через организм человека, электрический ток производит термическое, электролитическое, механическое и биологическое воздействие.

Термическое действие тока проявляется ожогами отдельных участков тела, нагревом до высокой температуры органов, расположенных на пути тока, вызывая в них значительные функциональные расстройства. Электролитическое действие тока выражается в разложении органической жидкости, в том числе крови, в нарушении ее физико-химического состава. Механическое действие тока приводит к расслоению, разрыву тканей организма в результате электродинамического эффекта, а также мгновенного взрывоподобного образования пара из тканевой жидкости и крови. Биологическое действие тока проявляется раздражением и возбуждением живых тканей организма, а также нарушением внутренних биологических процессов.

Исход поражения человека электротоком зависит от многих факторов: силы тока и времени его прохождения через организм, характеристики тока

(переменный или постоянный), пути тока в теле человека, при переменном токе — от частоты колебаний. [9]

Регламентирующие документы:

ГОСТ 12.1.038-82 ССБТ. «Электробезопасность. Предельно допустимые значения напряжений прикосновения и токов».

Предлагаемые меры по обеспечению безопасности:

- Установка устройств защитного отключения.
- Оборудование защитного заземления/зануления.
- Обеспечение недоступности токоведущих частей для случайного прикосновения.
- Проведение инструктажа персонала.

9.1.4. Повышенные уровни электромагнитного излучения

Причины возникновения:

- Большое количество источников ЭМ излучения (ПЭВМ).

Влияние на человека:

Систематическое и длительное воздействие электромагнитных полей различных частот с интенсивностью, превышающей предельно допустимые уровни, может привести к некоторым функциональным изменениям в организме, в первую очередь в центральной нервной системе.

Эти изменения в организме могут проявляться в головной боли, нарушении сна, повышенной утомляемости, раздражительности и ряде других симптомов. Кроме функциональных возможны также необратимые изменения в организме: торможение рефлексов, понижение кровяного давления, замедление сокращений сердца, изменение состава крови, помутнение хрусталика глаза.

Степень воздействия на человека ЭМ полей зависит от интенсивности облучения, его длительности, расстояния от источника образования поля и от индивидуальной чувствительности организма человека. [10]

Регламентирующие документы:

СанПиН 2.2.4.1191-03 «Электромагнитные поля в производственных условиях»

Предлагаемые меры обеспечения безопасности:

- Проведение инструктажей и обучение персонала.
- Рациональное расположение источников излучения.

9.1.5. Повышенные уровни запыленности воздуха рабочей зоны

Причины возникновения:

- Расположение рабочих мест в офисном помещении.

Влияние на человека:

Пыль оказывает вредное действие главным образом на дыхательные пути и легкие. При длительном воздействии пыли на человека возможны серьезные поражения всего организма. Пыль, проникая глубоко в легкие, может привести к развитию в них заболеваний. Пыль может также оказывать неблагоприятное воздействие на кожу и глаза.

Регламентирующие документы:

ГОСТ 12.1.005-88 ССБТ. «Общие санитарно-гигиенические требования к воздуху рабочей зоны».

Предлагаемые меры обеспечения безопасности:

- Организация общей вентиляции.
- Регулярная влажная уборка помещений.

- Проведение инструктажей персонала и проверок, требование поддержания рабочего места в чистоте.

9.1.6. Повышенный уровень шума

Причины возникновения:

- Работа систем охлаждения помещения.
- Работа систем охлаждения ПЭВМ.
- Наличие в офисном помещении оборудования, являющегося источником повышенного уровня шума (принтеров, плоттеров, серверных ЭВМ).

Влияние на человека:

Шум оказывает влияние на весь организм человека: угнетает ЦНС, вызывает изменение скорости дыхания и пульса, способствует нарушению обмена веществ, возникновению сердечно-сосудистых заболеваний, гипертонической болезни, может приводить к профессиональным заболеваниям.

Шум с уровнем звукового давления до 30...35 дБ привычен для человека и не беспокоит его. Повышение этого уровня до 40...70 дБ в условиях среды обитания создает значительную нагрузку на нервную систему, вызывая ухудшение самочувствия и при длительном действии может быть причиной неврозов. Воздействие шума уровнем свыше 75 дБ может привести к потере слуха — профессиональной тугоухости. При действии шума высоких уровней (более 140 дБ) возможен разрыв барабанных перепонок, контузия, а при еще более высоких (более 160 дБ) и смерть.

Регламентирующие документы:

СН 2.2.4/2.1.8.562-96. «Шум на рабочих местах, в помещениях жилых, общественных зданий и на территории жилой застройки».

Предлагаемые меры по обеспечению безопасности:

- Проверка систем охлаждения на соответствие допустимым уровням шума и своевременная замена оборудования, не удовлетворяющего заданным требованиям.
- Размещение оборудования, являющегося источником повышенного шума, в помещениях, отделенных от рабочих мест звукоизоляцией.

9.1.7. Повышенный уровень вибрации

Причины возникновения:

- Мощные вентиляторы в системе кондиционирования здания.

Влияние на человека:

Внутренние органы и отдельные части тела человека (сердце, желудок, голову и т.д.) можно рассматривать как колебательные системы, имеющие различные сосредоточенные массы и соединенные между собой упругими элементами. Большинство внутренних органов имеют частоту колебаний в районе 6-9 Гц. Воздействие на организм человека внешних колебаний с такими же частотами может вызвать резонансные колебания внутренних органов, что представляет опасность их смещения и механических повреждений.

Вибрация может быть причиной нарушения у работающего нормальной деятельности центральной нервной системы, сердечно-сосудистой системы, дыхательных органов, причиной повышения кровяного давления, заболевания сосудов, мышц, зрения и слуха. При длительном и интенсивном воздействии вибраций может возникнуть тяжелое и трудно излечимое заболевание - вибрационная болезнь. [10]

Регламентирующие документы:

СН 2.2.4/2.1.8.566-96. Производственная вибрация, вибрация в помещениях жилых и общественных зданий.

Предлагаемые меры по обеспечению безопасности:

- Установка виброизоляторов на опоры кондиционеров.

9.1.8. Монотонность труда

Причины возникновения:

- Работа производится на ПЭВМ на одном рабочем месте в течение всего рабочего дня.

Влияние на человека:

Однообразие выполняемых операций приводит к определенному техническому состоянию человека, называемому монотомией. Признаком монотомии является либо перегрузка одинаковой информацией, либо недостаток новой. Это накладывает отпечаток на функциональное состояние человека: он теряет интерес к выполняемой работе. Для него рабочее время как бы остановилось, и он с нетерпением ждет окончания смены, его клонит ко сну. Монотонная работа снижает эффективность труда, увеличивает текучесть кадров, аварийность и, как следствие, травматизм на производстве. [9]

Регламентирующие документы:

Р 2.2.2006-05 «Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда»

Предлагаемые меры по обеспечению безопасности:

- Введение регламентированных перерывов в работе.
- Создание специализированных комнат для отдыха.
- Проведение регулярных спортивных мероприятий.

9.1.9. Интеллектуальные нагрузки

Причины возникновения:

- Большой объем информации, который необходимо учитывать при разработке крупных имитационных моделей.

Влияние на человека:

Длительная работа, требующая высоких интеллектуальных нагрузок может привести к утомлению или переутомлению. При умственном утомлении отмечается расстройство внимания, ухудшение памяти и мышления, ослабляется точность и координированность движения.

Регламентирующие документы:

Р 2.2.2006-05 «Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда»

Предлагаемые меры по обеспечению безопасности:

- Введение регламентированных перерывов в работе.
- Проведение курсов по повышению квалификации работников.

9.2. Расчет виброизоляции для системы кондиционирования

Для кондиционирования помещения используются кондиционеры с номинальной частотой вращения вентиляторов n = 730 об/мин. Масса кондиционера 50 кг.

В силу низких частот вращения вентиляторов, для подавления вибраций, производимых кондиционерами, будут использованы пружинные виброизоляторы.

Требования к подбираемым виброизоляторам:

- Рабочая нагрузка виброизолятора не менее расчетной
- Число рабочих витков пружины виброизолятора не менее расчетного
- Вертикальная жесткость виброизолятора обеспечивает оптимальное соотношение $\frac{f}{f_0}$

Согласно ГОСТ 12.4.093-80 оптимательное соотношение частоты вращения вентилятора в кондиционере к его собственной частоте определяется формулой

$$\frac{f}{f_0} = 3 \div 4 \qquad (19)$$

Частота колебаний возмущающей силы:

$$f = \frac{n}{60} = \frac{730}{60} = 12.167 \text{ Гц}$$

Необходимое значение собственной частоты системы:

$$f_0 = \frac{f}{3} = \frac{12.167}{3} = 4.056 \text{ Гц}$$

Необходимая общая жесткость системы виброизоляции:

$$q_{z_\Sigma} = m \cdot (2 \cdot \pi \cdot f_0)^2 = 50 \cdot (2 \cdot 3.14 \cdot 4.056)^2 = 3.247 \cdot 10^4 \text{ Н/м}$$

Выбираем число виброизоляторов N = 4.

Тогда требуемая жесткость одного виброизолятора:

$$q_{1_\Sigma} = \frac{q_{z_\Sigma}}{N} = \frac{3.247 \cdot 10^4}{4} = 8.117 \cdot 10^3 \text{ Н/м}$$

Для определения нагрузки на одну пружину необходимо определить статическую и динамическую составляющие нагрузки.

Статическая нагрузка на одну пружину определяется по формуле:

$$P_{\text{ст}} = \frac{P}{N} = \frac{m \cdot g}{N} = \frac{50 \cdot 9.81}{4} = 122.625 \text{ Н}$$

Для расчета динамической нагрузки определим амплитуду вертикальных колебаний объекта.

Круговая частота колебаний:

$$\omega = 2 \cdot \pi \cdot f = 2 \cdot 3.14 \cdot 12.167 = 76.445 \text{ Гц}$$

Амплитуда вертикальных колебаний:

$$z_m = \frac{m \cdot g}{m \cdot \omega^2 - q_{z_\Sigma}} = \frac{50 \cdot 9.81}{50 \cdot 76.445^2 - 3.247 \cdot 10^4} = 1.889 \cdot 10^{-3} \text{ м}$$

Тогда динамическая нагрузка на одну пружину:

$$P_{\text{дин}} = z_m \cdot q_{1_\Sigma} = 1.889 \cdot 10^{-3} \cdot 8.117 \cdot 10^3 \ = 15.328 \text{ Н}$$

Расчетная нагрузка на одну пружину:

$$P_1 = P_{\text{ст}} + 1.5 \cdot P_{\text{дин}} = 122.625 + 1.5 \cdot 15.328 = 145.617 \text{ Н}$$

Диаметр проволоки пружины:

$$d = 1.6 \sqrt{\frac{K \cdot P_1 \cdot C}{[\tau_{\text{кр}}]}} \qquad (20)$$

Принимаем индекс пружины С = 8.

Коэффициент сжимаемости пружины К определяем исходя из значения С. К = 1.2.

Допустимое напряжение сдвига при кручении $[\tau_{\text{кр}}] = 3.92 \cdot 10^{\,8}$ Н/м^2

Отсюда

$$d = 1.6 \sqrt{\frac{K \cdot P_1 \cdot C}{[\tau_{\text{кр}}]}} = 1.6 \sqrt{\frac{1.2 \cdot 145.617 \cdot 8}{3.92 \cdot 10^{\,8}}} = 5.545 \cdot 10^{-3} \text{ м}$$

Принимаем

$$d = 0.01 \text{ м}$$

Модуль упругости пружинной стали при сдвиге $G = 7.85 \cdot 10^{10}$ Н/м^2

Число рабочих витков пружины:

$$i_{\text{р}} = \frac{G \cdot d}{8 \cdot C^3 \cdot q_{z_\Sigma}} = \frac{7.85 \cdot 10^{10} \cdot 0.01}{8 \cdot 8^3 \cdot 3.247 \cdot 10^4} = 5.903$$

Принимаем

$$i_{\text{р}} = 6$$

Исходя из расчета получены следующие требования к выбираемым виброизоляторам:

- $P_{\text{рабочее}} > 146$ Н
- $i_{\text{р}} > 6$
- Вертикальная жесткость обеспечивает соотношение

$$\frac{f}{f_0} = 3 \div 4$$

Исходя из заданных требований выбран виброизолятор пружинный ДО40. Параметры выбранного виброизолятора представлены в таблице 9.2.

Таблица 9.2 – Параметры виброизолятора ДО40

Обозначение виброизолятора	Нагрузка, Н		Вертикальная жесткость, Н/см	Число рабочих витков пружины
	Рабочая, $P_{раб}$	Предельная, $P_{пред}$		
ДО40	339	424	81	6.5

Для данного виброизолятора:

$$f_0 = \frac{1}{2 \cdot \pi}\sqrt{\frac{q_{1_\Sigma} \cdot N}{m}} = \frac{1}{2 \cdot 3.14}\sqrt{\frac{8.1 \cdot 10^3 \cdot 4}{50}} = 4.051 \text{ Гц}$$

$$\frac{f}{f_0} = \frac{12.167}{4.051} = 3.003$$

Выбранные виброизоляторы ДО40 обеспечивают оптимальное соотношение $\frac{f}{f_0}$.

Таким образом, для устранения вибраций, создаваемых системой кондиционирования, на каждый кондиционер должны быть установлены 4 виброизолятора ДО40.

9.3. Утилизация ПЭВМ

В процессе утилизации ПЭВМ производится извлечение драгоценных металлов для их вторичного использования. Проблема использования вторичного сырья, содержащего драгоценные материалы из компьютеров, периферийного оборудования и иных средств вычислительной техники (СВТ) актуальна в связи с техническим перевооружением отраслей промышленности. К драгоценным металлам относятся: золото, серебро, платина, палладий, родий,

иридий, рутений, осмий, а также любые химические соединения и сплавы каждого из этих металлов.

Последовательность разборки определяется типом изделия СВТ, его конструкционными особенностями и комплектацией. Как правило, процесс разборки должен выполняется в последовательности, обратной процессу сборки изделия.

9.3.1. Порядок разборки ПЭВМ, рабочих станций и серверов.

Технологии разборки ПЭВМ, рабочих станций, серверов и информационно-вычислительных систем едины поскольку состав их модулей стандартный. Он содержит системный блок и комплект периферийных устройств. Разборку ПЭВМ и составных модулей целесообразно осуществлять по технологической схеме, представленной на рисунке 9.1.

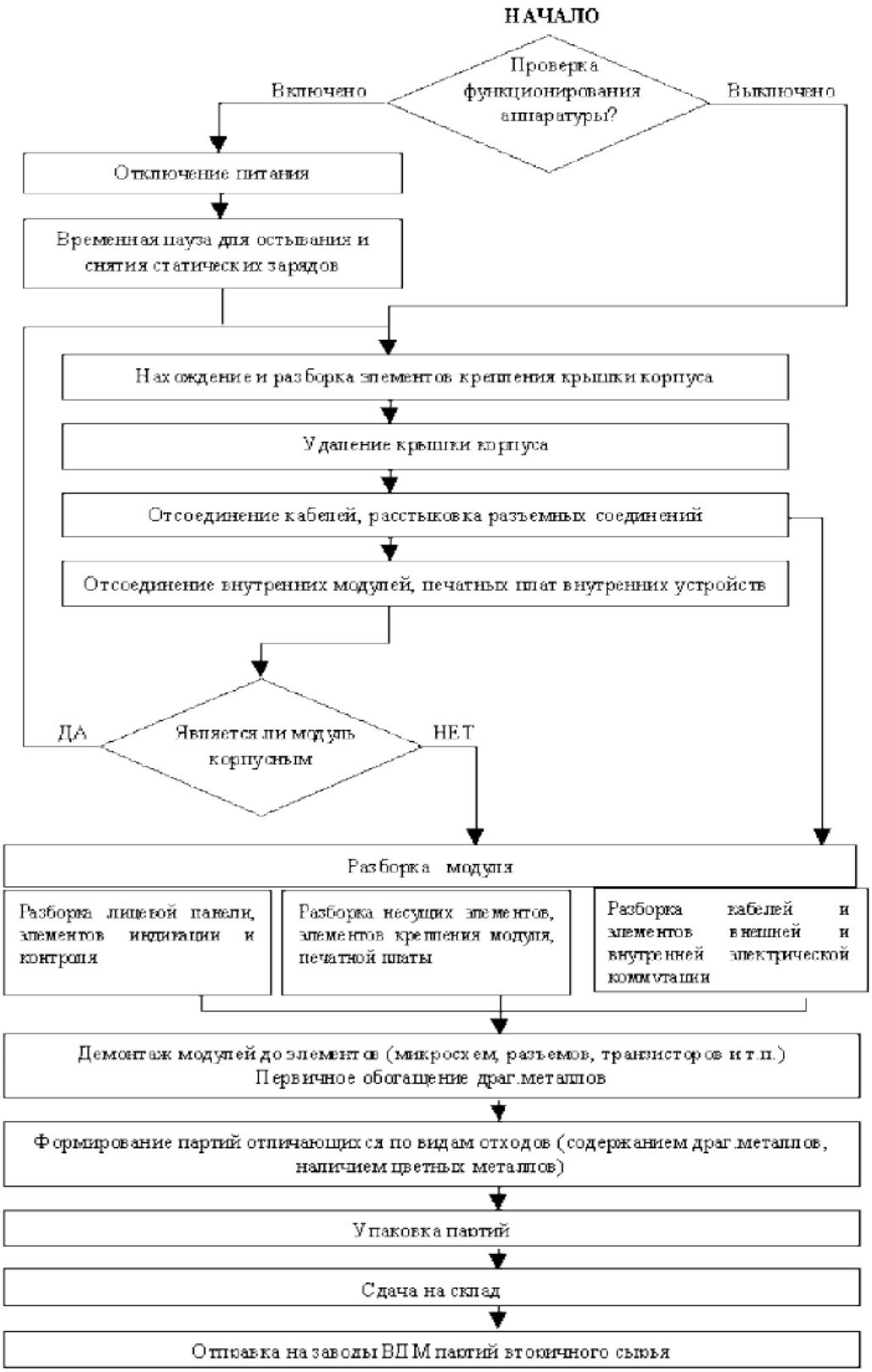

Рисунок 9.1 – Порядок разбора ПЭВМ

Порядок разборки системного блока:

Выключить компьютер и отсоединить шнур питания от розетки и системного блока. Отсоединить переходной шнур питания от системного блока к монитору.

Отсоединить от компьютера клавиатуру, монитор, манипулятор "мышь", принтер, сканер и иные внешние устройства. Найти элементы крепления крышки корпуса (винты, шурупы, пружинные защелки и т.д.). Освободить крышку от элемента крепления.

Снять крышку.

Отсоединить внутренние кабели и плоские шлейфы.

Найти элементы крепления дисководов (НМД, НГМД) в отсеке для дисководов (винты, шурупы, саморезные винты, пружинные защелки и др.). Освободить дисководы и извлечь их из дискового отсека. Освободить от крепёжных элементов периферийные платы.

Извлечь из разъёмов непосредственного контактирования все периферийные платы. Найти элементы крепления системной платы к корпусу (винты, шурупы). Освободить элементы крепления и извлечь системную плату из корпуса.

Извлечь модули памяти из разъёмов системной платы.

Найти элементы крепления блока питания к корпусу (винты, шурупы, саморезные винты, пружинные защелки и пр.). Освободить элементы крепления и извлечь блок питания.

Разобрать блок питания и извлечь высоковольтные конденсаторы, содержащие тантал.

Разобрать ПП и модули памяти до компонентов (микросхем, транзисторов, разъёмов и т.п.).

Произвести сортировку компонентов и сформировать партии электронного лома.

Упаковать партии, составить опись, произвести расчёт (анализ) драгметаллов и передать их на склад. Провести сортировку цветных и чёрных

металлов, пластмасс, сформировать партии и передать их на склад или на переработку.

При оценке содержания драгоценных металлов в партии электронного лома отечественных ПЭВМ необходимо руководствоваться паспортными данными. При оценке ПЭВМ импортного производства необходимо провести ориентировочные расчёты по отечественным аналогам.

9.3.2. Обеспечение комплектности

При разборке изделий СВТ образуются материалы и изделия, которые имеют материальную ценность и подлежат реализации. Их перечень представлен в таблице 9.3.

Таблица 9.3 – Перечень материалов, подлежащих утилизации

Вид материалов и изделий	Характеристика
печатные платы, разъемы, соединители, микросхемы	вторичные драгоценные металлы
электрические провода, кабели, соединители	вторичная медь и ее сплавы
свинец и олово из печатных плат	вторичные припойные пасты
танталовые конденсаторы	вторичный танталл
корпуса ПЭВМ	аллюминиевые сплавы
корпуса стоек, ячеек, шкафов, компьютеров	сталь
крепежные изделия	болты, гайки, винты
мониторы	стеклофаза, редкоземельные металлы

Таким образом, можно сделать вывод о целесообразности извлечения вторичных чёрных и цветных металлов, пластмасс, стекла, крепёжных изделий, вентиляторов и электромоторов.

9.3.3. Извлечение вторичных черных металлов

Отечественная практика показывает, что на 1 г извлекаемого золота приходится около 1 кг лома чёрных металлов. В связи с высокой стоимостью транспортно-погрузочных работ рекомендуется производить отгрузку предприятиям-покупателям партий лома чёрных металлов весом не менее 10 тонн. Блоки, панели, съёмные кожухи, рамы, каркасы шкафов и стоек стационарных ЭВМ, изготовленные из стального нормализованного профиля или листа, подвергаются сортировке, набираются в партии и реализуются.

Предпочтительно заключение договоров при условии, когда предприятие-покупатель своим транспортом вывозит вторичные металлы с территории предприятия-продавца.

Крепёжные изделия, заготовки стального профиля, листов, вентиляторы, электропускатели, кнопки, электрический кабель направляются на реализацию непосредственно в торговую сеть.

Опыт показывает, что денежные средства от реализации этих изделий не превышают 0,6 % от общей суммы.

9.3.4. Извлечение вторичных цветных металлов

В процессе разборки изделий СВТ образуется лом (содержащий медь) классификация которого должна проводиться по ГОСТ 1639-2009.

В соответствии с ГОСТ 1639-2009 медные шины целесообразно относить к классу А, группам I и II; латунь - к группам IV-VIII; бронзу - к группам XI-XII; отходы кабеля и проводов ПП следует относить к классу Г, группа XIII.

В процессе разборки изделий СВТ алюминий и его сплавы обычно содержатся в типовых конструкциях изделий. Их следует относить к классам А3 и Б5.

9.3.5. Завершающий этап утилизации

После полного разбора ПЭВМ все виды отходов сортируются, формируются в партии и реализовываются.

Этапы реализации партий:

- Классификация и сортировка отходов
- Сертификация партий
- Сдача партий на склад
- Заключение договоров на реализацию
- Транспортировка
- Сдача на переработку
- Получение денежных средств

На всем протяжении выполнения работ по утилизации списанных СВТ требуется соблюдение следующих правил техники безопасности по работе с вторичными драгоценными металлами:

- Не допускается сбор, заготовка и переработка радиоактивного лома и отходов драгоценных металлов.
- Требуется проведение контроля за содержанием вредных веществ в воздухе рабочей зоны в соответствии с ГОСТ 12.1.005 и ГОСТ 12.1.007.
- Производственные помещения в местах образования вредных веществ и пыли должны быть оборудованы вентиляцией согласно ГОСТ 12.4.021 с обеспечением санитарно-гигиенических требований к воздуху рабочей зоны.
- Для снятия статического электричества пылеприёмники, воздуховоды вентиляционных установок должны иметь заземление, выполненное и обозначенное в соответствии с ГОСТ 12.2.007.0, ГОСТ 12.2.007.14 и ГОСТ 21130.

- Для предотвращения попадания пыли, твёрдых веществ на слизистую оболочку глаз необходимо пользоваться защитными очками типа ПО-2, ПО-3 согласно ГОСТ 12.4.013.
- При работе с пылящими отходами необходимо пользоваться фильтрующими респираторами РУ-60 и РУ-60му по ГОСТ 17269 и респираторами "Лепесток" по ГОСТ 12.4.028.
- Средства индивидуальной защиты работающих с ломом и отходами драгоценных металлов и сплавов должны соответствовать типовым отраслевым нормам бесплатной выдачи рабочим и служащим металлургических производств. Спецодежда должна соответствовать ГОСТ 29057 и ГОСТ 29058.
- Помещения в местах выгрузки и загрузки лома и отходов, оказывающих вредное воздействие на организм человека, должны быть оборудованы местными отсосами согласно ГОСТ 12.4.021.
- Производственные помещения должны соответствовать требованиям "Санитарных норм проектирования промышленных предприятий СН 245-71".
- Метеорологические условия производственных помещений должны соответствовать санитарным нормам проектирования промышленных предприятий по ГОСТ 12.1.005.
- Требования безопасности при погрузочно-разгрузочных работах лома и отходов драгоценных металлов и сплавов должны соответствовать ГОСТ 12.3.009.
- Предприятия и организации, заготавливающие и перерабатывающие лом и отходы драгоценных металлов сплавов, должны проверять весь лом и отходы драгоценных металлов на взрывобезопасность. Из лома необходимо отобрать и удалить взрывоопасные предметы, материалы, в том числе электронно-вакуумные трубки дисплеев. Замкнутые сосуды, резервуары и другие полые предметы (баллоны, цилиндры, сосуды,

электровакуумные изделия и т.д.) разгерметизируются и освобождаются от содержимого (газов или жидкостей).

Таким образом, был рассмотрен процесс утилизации ПЭВМ и требования техники безопасности при работе с цветными металлами в процессе утилизации.

ЗАКЛЮЧЕНИЕ

В рамках данного дипломного проекта получены следующие результаты:

- Проведено предпроектное исследование, в ходе которого были выявлены основные недостатки системы RAO-XT.
- Сформулированы требования к новой системе имитационного моделирования Rao X, разработано техническое задание на создание системы.
- На концептуальном этапе проектирования был выбран стек технологий (Xbase), на котором должна базироваться система, сформулирована общая концепция функционирования системы. Разработан новый синтаксис языка имитационного моделирования РДО с учетом недостатков старого синтаксиса.
- На техническом этапе проектирования была разработана новая грамматика языка РДО. Выявлены ключевые классы, позволяющие реализовать новый функционал системы.
- На этапе рабочего проектирования был выполнен перевод системы на новый стек технологий, реализованы функции генерации и связывания кода, реализованы новые возможности библиотеки моделирования, а также новый функционал среды разработки, предоставляемой системой.
- В исследовательской части проведен сравнительный анализ производительности алгоритмов копирования состояний модели, которые могут использоваться при поиске по графу. Выявлен наиболее эффективный алгоритм.
- Спроектирована и внедрена система автоматизированного интеграционного тестирования. Написан ряд тестовых моделей.

Таким образом цель дипломного проекта достигнута в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Емельянов В.В., Ясиновский С.И. Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО. - М.: "Анвик", 1998. – 427 с., ил. 136.
2. Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. – М. : ООО "И.Д.Вильямс", 2008. - 1184 с. : ил. – Парал. тит. англ.
3. Документация по языку РДО [Электронный ресурс] – http://rdostudio.raox.ru/help/index.html
4. SWT Documentation [Электронный ресурс] – https://www.eclipse.org/swt/docs.php
5. Xtext Documentation [Электронный ресурс] – http://eclipse.org/Xtext/documentation.html
6. Java™ Platform, Standard Edition 8 [Электронный ресурс] – https://docs.oracle.com/javase/8/docs/api/
7. RCPTT Project Documentation [Электронный ресурс] – http://www.eclipse.org/rcptt/documentation/
8. Арсеньев В.В., Сажин Ю.Б. Методические указания к выполнению организационно-экономической части дипломных проектов по созданию программной продукции. — М.: Изд-во МГТУ, 1994. — 52 с.
9. Безопасность жизнедеятельности: Учебник для вузов/С.В. Белов, А.В. Ильницкая, А.Ф. Козьяков и др.; Под общ. ред. С.В. Белова. 7-е изд., стер. — М.: Высш.шк., 2007. — 616 с.: ил.
10. Полтев М.К. Охрана труда в машиностроении: Учебник. — М.: Высш. школа, 1980. — 294 с., ил.
11. Влияние освещенности на физиологическое состояние организма человека [Электронный ресурс] – http://www.12sanepid.ru/press/publications/187.html
12. Xvnc Plugin [Электронный ресурс] – https://wiki.jenkins-ci.org/display/JENKINS/Xvnc+Plugin
13. Xbase Documentation. [Электронный ресурс] – https://www.eclipse.org/Xtext/documentation/305_xbase.html

СПИСОК ИСПОЛЬЗОВАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Eclipse IDE for Java and DSL Developers Mars.2 Release (4.5.2)
2. Oracle JDK 8u91
3. RCPTT Test Runner 2.0.2
4. UMLet 14.2
5. Inkscape 0.48.4
6. Microsoft® Office Word 2007
7. Microsoft® Visio 2010
8. Railroad Diagram Generator v1.39.959

ПРИЛОЖЕНИЕ А

Код модели многоканальной СМО в старой грамматике

```
$Resource_type Парикмахерские: permanent
$Parameters
      количество_в_очереди: integer
$End

$Resource_type Клиенты : temporary
$Parameters
      тип       : ( Тип1, Тип2 )
      состояние: ( Пришел, Начал_стрижку )
$End

$Resource_type Парикмахеры: permanent
$Parameters
      состояние_парикмахера : ( Свободен, Занят )
      количество_обслуженных: integer
      длительность_min      : integer
      длительность_max      : integer
      тип_клиента           : such_as Клиенты.тип
$End

$Resources
      Парикмахерская = Парикмахерские(0);
      Парикмахер_1 = Парикмахеры(Свободен, 0, 20, 40, Тип1);
      Парикмахер_2 = Парикмахеры(Свободен, 0, 25, 70, Тип2);
      Парикмахер_3 = Парикмахеры(Свободен, 0, 30, 60, Тип2);
$End

$Pattern Образец_прихода_клиента : event
$Relevant_resources
      _Парикмахерская: Парикмахерская Keep
      _Клиент        : Клиенты        Create
$Body
_Парикмахерская:
      Convert_event
            Образец_прихода_клиента.planning( time_now +
Интервал_прихода( 30 ) );
            количество_в_очереди++;

_Клиент:
      Convert_event
            тип       = Тип_клиента;
            состояние = Пришел;
$End

$Pattern Образец_обслуживания_клиента : operation
$Relevant_resources
      _Парикмахерская: Парикмахерская Keep NoChange
      _Клиент        : Клиенты        Keep Erase
      _Парикмахер    : Парикмахеры    Keep Keep
$Time = Длительность_обслуживания( _Парикмахер.длительность_min,
_Парикмахер.длительность_max )
```

```
$Body
_Парикмахерская:
      Choice from _Парикмахерская.количество_в_очереди > 0
      Convert_begin
            количество_в_очереди--;

_Клиент:
      Choice from _Клиент.состояние == Пришел
      Convert_begin
            состояние = Начал_стрижку;

_Парикмахер:
      Choice from _Парикмахер.состояние_парикмахера == Свободен and
_Парикмахер.тип_клиента == _Клиент.тип
      with_min( _Парикмахер.количество_обслуженных )
      Convert_begin
            состояние_парикмахера  = Занят;
      Convert_end
            состояние_парикмахера  = Свободен;
            количество_обслуженных++;
$End

$Decision_point model: some
$Condition NoCheck
$Activities
      Обслуживание_клиента: Образец_обслуживания_клиента;
$End

$Sequence Интервал_прихода : real
$Type = exponential 123456789 legacy
$End

$Sequence Длительность_обслуживания : real
$Type = uniform 123456789 legacy
$End

$Sequence Тип_клиента : such_as Клиенты.тип
$Type = by_hist 123456789 legacy
$Body
      Тип1 1.0
      Тип2 5.0
$End

$Simulation_run
      Образец_прихода_клиента.planning( time_now +
Интервал_прихода( 30 ) );
      Terminate_if Time_now >= 360;
$End
```

ПРИЛОЖЕНИЕ Б

Код модели многоканальной СМО в новой грамматике

```
enum Тип_клиента {Тип1, Тип2}
enum Состояние_клиента {Пришел, Начал_стрижку}
enum Состояние_парикмахера {Свободен, Занят}

type Парикмахерские {
      int количество_в_очереди
}

type Клиенты {
      Тип_клиента тип
      Состояние_клиента состояние
}

type Парикмахеры {
      Состояние_парикмахера состояние_парикмахера
      int количество_обслуженных
      int длительность_min
      int длительность_max
      Тип_клиента тип_клиента
}

resource парикмахерская = Парикмахерские.create(0);
resource парикмахер_1 = Парикмахеры.create(
      Состояние_парикмахера.Свободен, 0, 20, 40, Тип_клиента.Тип1)
resource парикмахер_2 = Парикмахеры.create(
      Состояние_парикмахера.Свободен, 0, 25, 70, Тип_клиента.Тип2)
resource парикмахер_3 = Парикмахеры.create(
      Состояние_парикмахера.Свободен, 0, 30, 60, Тип_клиента.Тип2)

event Событие_прихода_клиента() {
      Клиенты.create(случайный_тип_клиента.next(),
            Состояние_клиента.Пришел);
      Событие_прихода_клиента.plan(currentTime
            + интервал_прихода.next());
      парикмахерская.количество_в_очереди =
            парикмахерская.количество_в_очереди + 1;
}

operation Образец_обслуживания_клиента() {
      relevant _Парикмахерская = парикмахерская.onlyif[
            количество_в_очереди > 0]
      relevant _Клиент = Клиенты.all.filter[
            состояние.equals(Состояние_клиента.Пришел)].any
      relevant _Парикмахер = Парикмахеры.all.filter[
            состояние_парикмахера.equals(
                  Состояние_парикмахера.Свободен)
            && тип_клиента.equals(_Клиент.тип)]
      .minBySafe[количество_обслуженных]

      set duration() {
            return длительность_обслуживания.next(
```

```
                    _Парикмахер.длительность_min,
                _Парикмахер.длительность_max)
        }

        set begin() {
                _Парикмахерская.количество_в_очереди =
                    _Парикмахерская.количество_в_очереди - 1
                _Клиент.состояние = Состояние_клиента.Начал_стрижку
                _Парикмахер.состояние_парикмахера =
                    Состояние_парикмахера.Занят
        }

        set end() {
                _Парикмахер.состояние_парикмахера =
                    Состояние_парикмахера.Свободен
                _Парикмахер.количество_обслуженных =
                    _Парикмахер.количество_обслуженных + 1
                _Клиент.erase()
        }
}

logic Model {
        activity обслуживание_клиента =
                new Activity(Образец_обслуживания_клиента.create())
}

sequence интервал_прихода = new Exponential(123456789, 1/30.0)
sequence длительность_обслуживания = new Uniform(123456789)
sequence случайный_тип_клиента =
        new DiscreteHistogram<Тип_клиента>(123456789,
        #[
                Тип_клиента.Тип1 -> 1.0,
                Тип_клиента.Тип2 -> 5.0
        ]
)

set init() {
        Событие_прихода_клиента.plan(
                currentTime + интервал_прихода.next())
}

set terminateCondition() {
        return currentTime >= 7 * 12 * 60
}
```

ПРИЛОЖЕНИЕ В

Функции, используемые при интеграционном тестировании

```
proc open_perspective [val perspective_name] {
      get-menu "Window/Perspective/Open Perspective/Other..." | click
      with [get-window "Open Perspective"] {
            get-table | select $perspective_name
            get-button OK | click
      }
}

proc create_rao_project [val template_name] [val project_name] {
      get-menu "File/New/Project..." | click
      with [get-window "New Project"] {
            with [get-editbox -after [get-label "Wizards:"]] {
                  key-type Enter
                  set-text rao
            }
            get-button "Next >" | click
      }

      with [get-window "New Rao Project"] {
            get-group Templates | get-button $template_name | click
            with [get-editbox -after [get-label "Project name:"]] {
                  set-text $project_name
            }
            get-button Finish | click
      }
}

proc open_rao_perspective {
      open_perspective Rao
}

proc open_java_perspective {
      open_perspective "Java (default)"
}

proc copy_model [val model_name] {
      to-clipboard [read-file -uri [concat "workspace:/etalon/"
$model_name ".rao.template"]]

      with [get-editor [concat $model_name ".rao"]] {
            get-editbox | get-menu Paste | click
            get-text-viewer | key-type "M1+s"
      }
}

proc enable_monitoring [val model_name] {
      with [get-view "Monitoring Configuration" | get-tree] {
            get-item $model_name | check
      }
}
```

```
proc compare_trace [val model_name] {
      get-menu "Model/Export Trace Output/Regular" | click

      read-file -uri [concat "workspace:/" $model_name "/" $model_name
".trc"] | equals [read-file -uri [concat "worksp" +
            "ace:/etalon/" $model_name ".trc.etalon"]] | verify-true
}

proc execute_model [val model_name] {
      get-button [concat "Execute model " $model_name] | click
}

proc test_model_template_notrace [val template_name] [val model_name] {
      open_rao_perspective

      create_rao_project $template_name $model_name

      execute_model $model_name

      get-view Trace | get-table | get-property itemCount | equals 3 |
verify-true
}

proc test_model_template_trace [val template_name] [val model_name] {
      open_rao_perspective

      create_rao_project $template_name $model_name

      enable_monitoring $model_name

      execute_model $model_name

      compare_trace $model_name
}

proc test_model_source_notrace [val model_name] {
      open_rao_perspective

      create_rao_project "Пустая модель" $model_name

      copy_model $model_name

      execute_model $model_name

      get-view Trace | get-table | get-property itemCount | equals 3 |
verify-true
}

proc test_model_source_trace [val model_name] {
      open_rao_perspective

      create_rao_project "Пустая модель" $model_name

      copy_model $model_name

      enable_monitoring $model_name
```

```
        execute_model $model_name

        compare_trace $model_name
}
```