Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« ___ » _____ 20 __ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине Технология компьютерно-интегрированных производств

Создание интегрированной среды разработки языка РДО
(Тема курсового проекта)

Студент Александровский К.Д., РК9-102
(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

1. Техническое задание

Создание многофункциональной среды разработки, способной редактировать и запускать имитационные модели на языке РДО.

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на 28 листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.)

1 лист А1 – Постановка задачи;
2 лист А2 – Диаграмма пакетов «Компоненты Xtext»;
3 лист А2 – Диаграмма активности «Процесс разработки имитационной модели»;
4 лист А1 – Синтаксическая диаграмма языка РДО;
5 лист А1 – Синтаксическая диаграмма языка РДО;
6 лист А1 – Диаграмма классов библиотеки симулятора РДО;
7 лист А1 – Результаты.

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ А.В. Урусов
(Подпись, дата) (И.О.Фамилия)

Студент _____ К.Д. Александровский
(Подпись, дата) (И.О.Фамилия)

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ РК

КАФЕДРА РК-9

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовому проекту на тему:

Создание интегрированной среды разработки языка РДО

Студент группы РК9-102 _____ К.Д. Александровский
(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____ А.В. Урусов
(Подпись, дата) (И.О.Фамилия)

Москва, 2014

Оглавление

1. Перечень сокращений 2
2. Введение 3
3. Предпроектное исследование 5
3.1. Основные положения языка РДО 5
3.2. Понятие интегрированной среды разработки 7
3.3. RAO-Studio 7
3.4. Постановка задачи 9
4. Концептуальный этап проектирования системы 10
4.1. Eclipse IDE 10
4.2. Xtext framework 10
5. Формирование ТЗ 12
5.1. Основания для разработки 12
5.2. Общие сведения. 12
5.3. Назначение и цели развития системы 12
5.4. Характеристики объекта автоматизации 12
5.5. Требования к системе 12
5.5.1. Требования к функциональным характеристикам 12
5.5.6. Требования к маркировке и упаковке 13
5.5.7. Требования к транспортированию и хранению 13
5.5.8. Порядок контроля и приемки 13
6. Технический этап проектирования системы 14
6.1. Разработка грамматики языка РДО 14
6.2. Формирование компонентов Xtext 15
7. Рабочий этап проектирования системы 17
7.1. Иерархическая структура модели 17
7.2. Механизм обнаружения ошибок 17
7.3. Компилятор языка 18
7.3.1. Java-библиотека языка РДО 18
7.3.2. Кодогенерация 19
8. Заключение 21
9. Список используемых источников 22
10. Список использованного программного обеспечения 23
Приложение А 24

1. Перечень сокращений

РП – Рабочий Проект

ТЗ – Техническое Задание

ТП – Технический Проект

ИМ – Имитационное Моделирование, Имитационная Модель

СДС – Сложная Дискретная Система

ЭВМ – Электронная Вычислительная Машина

ОЗУ – Оперативное Запоминающее Устройство

ИСР – Интегрированная Среда Разработки

2. Введение

Имитационное моделирование (ИМ) на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, морские порты, аэропорты, комплексы перекачки нефти и газа, ирригационные системы, программное обеспечение сложных систем управления, вычислительные сети и многие другие.

Интеллектуальное имитационное моделирование, характеризующиеся возможностью использования методов искусственного интеллекта и, прежде всего, знаний при принятии решений в процессе имитации, при управлении имитационным экспериментом, при реализации интерфейса пользователя, создании информационных банков ИМ, использовании нечетких данных, снимает часть проблем использования ИМ.

ИМ является эффективным, но и не лишенным недостатков, методом. Трудности использования ИМ, связаны с обеспечением адекватности описания системы, интерпретацией результатов, обеспечением стохастической сходимости процесса моделирования, решением проблемы размерности и т.п. К проблемам применения ИМ следует отнести также и большую трудоемкость данного метода.

Разработка интеллектуальной среды имитационного моделирования РДО – «Ресурсы, Действия, Операции», выполнена в Московском Государственном Техническом Университете им. Н.Э. Баумана на кафедре "Компьютерные системы автоматизации производства". Причинами создания РДО явились требования к универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей, а также моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в

реальном масштабе времени). Таким образом, среда РДО стала решением указанных выше проблем ИМ и обеспечила исследователя и проектировщика новыми возможностями.

Программный комплекс RAO-studio предназначен для разработки и отладки имитационных моделей на языке РДО. Основные цели данного комплекса - обеспечение пользователя легким в обращении, но достаточно мощным средством разработки текстов моделей на языке РДО, обладающим большинством функций по работе с текстами программ, характерных для сред программирования, а также средствами проведения и обработки результатов имитационных экспериментов

3. Предпроектное исследование

3.1. Основные положения языка РДО

В основе системы РДО лежат следующие положения:

- Все элементы сложной дискретной системы (СДС) представлены как ресурсы, описываемые некоторыми 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 состояния ресурсов; действия ограничены во времени двумя событиями: событиями начала и конца.
- Нерегулярные события описывают изменение состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению 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 ресурсов в начале и конце соответствующего действия.

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

Модель – совокупность объектов РДО-языка, описывающих какой-то реальный объект, собираемые в процессе имитации показатели, кадры

анимации и графические элементы, используемые при анимации, результаты трассировки.

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

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

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

Объектами исходных данных являются:

- типы ресурсов (с расширением .rtp);
- ресурсы (с расширением .rss);
- события (с расширением .evn);
- образцы активностей (с расширением .pat);
- точки принятия решений и процессы обслуживания(с расширением .dpt);
- константы, функции и последовательности (с расширением .fun);
- кадры анимации (с расширением .frm);
- требуемая статистика (с расширением .pmd);
- прогон (с расширением .smr);
- процессы обслуживания (с расширением .prc).

3.2. Понятие интегрированной среды разработки

Интегрированные среды разработки были созданы для того, чтобы максимизировать производительность программиста благодаря тесно связанным компонентам с простыми пользовательскими интерфейсами. Это позволяет разработчику сделать меньше действий для переключения различных режимов, в отличие от дискретных программ разработки. Однако, так как ИСР является сложным программным комплексом, то лишь после долгого процесса обучения среда разработки сможет качественно ускорить процесс разработки ПО. Для уменьшения барьера вхождения многие достаточно интерактивны, а для облегчения перехода с одной на другую интерфейс у одного производителя максимально близок, вплоть до использования одной ИСР.

ИСР, обычно, представляет собой единственную программу, в которой проводилась вся разработка. Она, обычно, содержит много функций для создания, изменения, компилирования, развертывания и отладки программного обеспечения. Цель среды разработки заключается в том, чтобы абстрагировать конфигурацию, необходимую, чтобы объединить утилиты командной строки в одном модуле, который позволит уменьшить время, чтобы изучить язык, и повысить производительность разработчика. Также считается, что трудная интеграция задач разработки может далее повысить производительность. Например, ИСР позволяет проанализировать код и тем самым обеспечить мгновенную обратную связь и уведомить о синтаксических ошибках. [3]

3.3. RAO-Studio

Программный комплекс RAO-Studio предназначен для разработки и отладки имитационных моделей на языке РДО. Основные цели данного комплекса - обеспечение пользователя легким в обращении, но достаточно мощным средством разработки текстов моделей на языке

РДО, обладающим большинством функций по работе с текстами программ, характерных для сред программирования, а также средствами проведения и обработки результатов имитационных экспериментов.

В соответствии с основной целью программный комплекс решает следующие задачи:

- синтаксический разбор текста модели и настраиваемая подсветка синтаксических конструкций языка РДО;
- открытие и сохранение моделей;
- расширенные возможности для редактирования текстов моделей;
- автоматическое завершение ключевых слов языка;
- поиск и замена фрагментов текста внутри одного модуля модели;
- поиск интересующего фрагмента текста по всей модели;
- навигация по тексту моделей с помощью закладок;
- наличие нескольких буферов обмена для хранения фрагментов текста;
- вставка синтаксических конструкций языка и заготовок (шаблонов) для написания элементов модели;
- настройка отображения текста моделей, в т. ч. скрытие фрагментов текста и масштабирование;
- запуск и остановка процесса моделирования;
- изменение режима моделирования;
- изменение скорости работающей модели;
- переключение между кадрами анимации в процессе моделирования;
- отображение хода работы модели в режиме реального времени;
- построение графиков изменения интересующих разработчика характеристик в режиме реального времени;
- обработка синтаксических ошибок при запуске процесса моделирования;
- обработка ошибок во время выполнения модели. [2]

3.4. Постановка задачи

Таким образом, несмотря на то, что текущая версия среды разработки языка РДО RAO-Studio является ИСР, некоторые её возможности, в полном объёме предоставляемые современными ИСР крупнейших фирм-разработчиков программного обеспечения (такие, как статический анализ кода, автодополнение, навигация по исходному коду с помощью гиперссылок и т.д.), довольно сложны в разработке, а на обеспечение их корректной работы разработчику необходимо потратить большое количество времени.

В связи с этим, более оптимальным решением данной проблемы становится разработка ИСР на основе уже существующего программного продукта с открытым исходным кодом, продукта, в котором уже реализованы все требуемые функции по ускорению и облегчению процесса разработки.

Описание требуемых от современной ИСР функций представлено на части листа А1 курсового проекта «Постановка задачи».

4. Концептуальный этап проектирования системы

4.1. Eclipse IDE

Программный комплекс Eclipse — это свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается Eclipse Foundation.

Наиболее известные приложения, основанные на Eclipse Platform — различные «Eclipse IDE» для разработки ПО на множестве языков.

Eclipse служит в первую очередь платформой для разработки расширений, чем он и завоевал популярность: любой разработчик может расширить Eclipse своими модулями. Уже существуют Java Development Tools (JDT), C/C++ Development Tools (CDT), разрабатываемые инженерами QNX совместно с IBM, и пр. от различных разработчиков. Множество расширений дополняет среду Eclipse менеджерами для работы с базами данных, серверами приложений и др.

Гибкость Eclipse обеспечивается за счёт подключаемых модулей, благодаря чему возможна разработка не только на Java, но и на других языках. [6]

Основной каркас среды Eclipse, называемый Eclipse Platform, предоставляет все необходимые инструменты для быстрой и качественной разработки из (**3.4**).

4.2. Xtext framework

Библиотека Xtext служит для создания языков программирования и предметно-ориентированных языков (domain specific languages).

Xtext покрывает все аспекты создания инфраструктуры разрабатываемого языка и предоставляет тесную интеграцию с Eclipse, а именно:

- Подсветка синтаксиса как на основе лексической, так и на основе семантической структуры языка;
- Автодополнение кода, как для ключевых слов, так для контекстно-зависимых конструкций языка;

- Проверка кода на наличие ошибок, производящаяся в фоновом режиме;
- Другие возможности по интеграции с компонентами среды Eclipse.

Все перечисленные выше пункты могут быть настроены под нужды разрабатываемого языка благодаря модульности реализующих их компонентов.

Исходный код на разрабатываемом языке преобразуется в код на языке Java с помощью правил генерации, определяемых разработчиком. Сгенерированный код компилируется и запускается на виртуальной машине Java.

Полученный проект Xtext экспортируется в виде расширения (plug-in) для Eclipse Platform, и работает независимо от модулей Xtext.

Диаграмма пакетов проекта Xtext представлена на листе А2 курсового проекта «Компоненты Xtext».

Процесс создания и запуска имитационной модели в разрабатываемой среде представлен на диаграмме активности, находящейся на листе А2 курсового проекта Диаграмма активности «Процесс разработки имитационной модели».

5. Формирование ТЗ

5.1. Основания для разработки

Задание на курсовой проект.

5.2. Общие сведения.

В рамках курсового проекта создаётся интегрированная среда разработки языка РДО.

5.3. Назначение и цели развития системы

Основная цель работы – создание многофункциональной среды разработки, способной редактировать и запускать имитационные модели на языке РДО.

5.4. Характеристики объекта автоматизации

РДО – язык имитационного моделирования, включающий все три основных подхода описания дискретных систем: процессный, событийный и сканирования активностей.

5.5. Требования к системе

5.5.1. Требования к функциональным характеристикам

В системе должны быть представлены следующие компоненты:

- Редактор кода имитационных моделей с подсветкой синтаксиса, автодополнением и проверкой ошибок;
- Компилятор языка РДО, преобразующий исходный код имитационных моделей в код на языке Java;
- Логика и алгоритмы языка РДО, объединённые в библиотеку симулятора.

5.5.2. Требования к надёжности

Основное требование к надёжности направлено на поддержание в исправном и работоспособном состоянии ЭВМ, на которой происходит использование программного комплекса.

5.5.3. Условия эксплуатации

Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В±10%, 50Гц с защитным заземлением.

5.5.4. Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 1 Гб;
- объем жёсткого диска не менее 20 Гб;
- микропроцессор с тактовой частотой не менее 1ГГц;
- монитор с разрешением от 800*600 и выше;

5.5.5. Требования к информационной и программной совместимости

Данная система должна работать под управлением операционных систем Windows XP, Windows 7, Windows 8, а также MacOS или Linux.

5.5.6. Требования к маркировке и упаковке

Не предъявляются.

5.5.7. Требования к транспортированию и хранению

Не предъявляются.

5.5.8. Порядок контроля и приемки

Контроль работоспособности программного комплекса осуществляется с помощью имитационных моделей, написанных на языка РДО.

6. Технический этап проектирования системы

6.1. Разработка грамматики языка РДО

В программировании для преобразования какого-либо языка в набор команд используются синтаксические и лексические анализаторы, т.е. программы, позволяющие превратить какой либо язык в удобный для машинной обработки вид с помощью формального описания этого языка – грамматики. [5]

В RAO-Studio для этой цели использовался GNU bison – LR-анализатор, который выполняет разбор входного потока данных снизу вверх, преобразуя входные данные с код на языке C++.

В библиотеке Xtext имеется собственный синтаксический анализатор, основанный на анализаторе ANTLR. Его алгоритм, LL(1) – это нисходящий алгоритм синтаксического разбора с просмотром на один символ вперед. При этом вместо исходного кода на каком-либо языке программирования анализатор Xtext создает синтаксическое дерево, с которым можно работать в дальнейшем. Вершины этого дерева – структуры, полями которых являются элементарные типы данных, ссылки на другие вершины, и даже массивы таких ссылок в случаях, когда требуется описание неопределенного количества однотипных элементов.

Таким образом, на основании описания языка РДО, представленного в документации [1], была составлена его грамматика в нотации Xtext. Синтаксические диаграммы, соответствующие грамматике, представлены на двух листах А1 курсового проекта «Синтаксическая диаграмма языка РДО».

Помимо этого, был написан язык выражений (expressions) и команд (statement), то есть объектов, присущих всем процедурным

языкам программирования. Процедурный язык также является неотъемлемой частью РДО на текущем этапе его развития.

Пример нетерминального символа, отвечающего за точку принятия решений типа “some”:

```
DecisionPointSome:
    '$Decision_point' name = ID ':' 'some' trace ?= 'trace'?
   ('$Parent' parent = [DecisionPoint|FQN])?
   ('$Condition' (condition = ExpressionOr | 'NoCheck'))?
    '$Activities'
         activities += DecisionPointActivity+
    '$End'
;
```

Язык выражений и команд, не вошедший в листы курсового проекта, представлен в Приложении А.

6.2. Формирование компонентов Xtext

После написания грамматики Xtext генерирует начальные версии всех своих компонентов на ее основе. В их число входят следующие пакеты:

- ru.bmstu.rk9.rdo. formatting

  Отвечает за автоматическое выравнивание кода при редактировании, а также позволяет определить свои правила форматирования.

- ru.bmstu.rk9.rdo. scoping

  Отвечает за область видимости для терминальных символов, являющихся ссылками на описанные в модели нетерминальные символы.

- ru.bmstu.rk9.rdo.validation

  Предоставляет механизм проверки кода модели на ошибки согласно логике, описанной в файле RDOValidator.xtend

- ru.bmstu.rk9.rdo. generator

  Содержит код, в результате выполнения которого в папке проекта создаются сгенерированные файлы с исходным кодом на языке Java. Программный код для такого преобразования пишется разработчиком, и подразумевает использование синтаксического дерева, получаемого на этапе синтаксического разбора.

- ru.bmstu.rk9.rdo.ui.contentassist

  В данном пакете находятся правила, изменяющие поведение стандартного меню автодополнения для каких либо объектов грамматики языка.

- ru.bmstu.rk9.rdo.ui.labeling

  Отвечает за текстовое представление имен объектов грамматики.

- ru.bmstu.rk9.rdo.ui.outline

  Содержит правила, позволяющие изменить стандартный механизм построения дерева иерархической структуры.

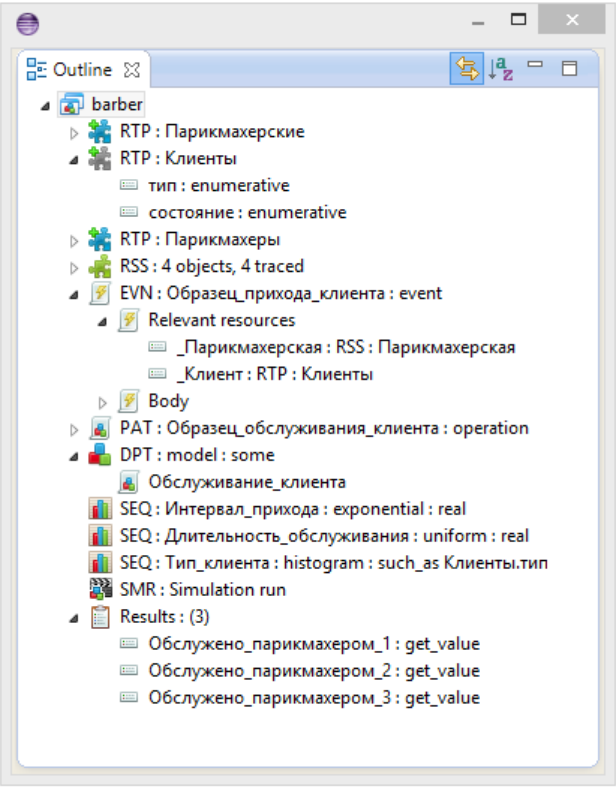
- ru.bmstu.rk9.rdo.ui.quickfix

  Правила, изменяющие и расширяющие меню подсказок при возникновении каких-либо ошибок в коде модели.

7. Рабочий этап проектирования системы

7.1. Иерархическая структура модели

Внешний вид окна структуры модели определяется грамматикой языка, а также изменениями, описываемыми в файлах `RDOOutlineTreeProvider.xtend` и `RDODescriptionLabelProvider.xtend` пакетов `ru.bmstu.rk9.rdo.outline` и `ru.bmstu.rk9.rdo.labeling` соответственно. Данные изменения позволяют запретить элементам иерархии порождать дочерние вершины, а также определяют имена и значки для каждого типа элемента. На рисунке представлен внешний вид иерархии для модели парикмахерской:



7.2. Механизм обнаружения ошибок

В файле `RDOValidator.xtend` пакета `ru.bmstu.rk9.rdo.validaton` описываются правила анализа исходного кода модели, которые могут выводить те или иные сообщения об ошибках, а также предупреждения, связанные с исходным кодом имитационной модели. Пример кода,

выявляющий совпадения в именах релевантных ресурсов и именах параметров образца:

```
@Check
def checkNamesInPatterns (Pattern pat)
{
	val List<EObject> paramlist  = pat.eAllContents.filter[e |
		e instanceof PatternParameter
	].toList
	val List<EObject> relreslist = pat.eAllContents.filter[e |
		e instanceof OperationRelevantResource ||
		e instanceof RuleRelevantResource      ||
		e instanceof EventRelevantResource
	].toList

	for (e : paramlist)
		if (relreslist.map[r | r.nameGeneric].contains(e.nameGeneric))
			error("Error - parameter name shouldn't match
				relevant resource name '" + e.nameGeneric + "'.",
					e, e.getNameStructuralFeature)

	for (e : relreslist)
		if (paramlist.map[r | r.nameGeneric].contains(e.nameGeneric))
			error("Error - relevant resource name shouldn't
				match parameter name '" + e.nameGeneric + "'.", e,
					e.getNameStructuralFeature)
}
```

7.3. Компилятор языка

Правила генерации Java-кода из объектов синтаксического дерева необходимо описать в файле `RDOGenerator.xtend`. В этом файле находится наибольший объем логики и алгоритмов разрабатываемой системы.

7.3.1. Java-библиотека языка РДО

Для неизменной части языка РДО, т.е. алгоритмов работы с базой данных и базой знаний, алгоритмов работы с событиями и продукционными правилами, была разработана общая библиотека, содержащая в себе всю логику работы с моделями на языке РДО - **rdo_lib**.

Также в этой библиотеке представлены интерфейсы, стандартизующие разработку изменяемой части Java-кода имитационной модели.

Библиотека включает в себя работу с событиями, точками принятия решений, алгоритмы подбора релевантных ресурсов, алгоритм поиска на графе состояний и т. д. Ее диаграмма классов представлена на листе А1 курсового проекта «Диаграмма классов библиотеки симулятора РДО».

7.3.2. Кодогенерация

Кодогенерация представляет собой процесс записи в файловую систему текстовых файлов, созданных на основе объектов синтаксического дерева. В ее процессе для каждого объекта языка РДО в папке проекта создается свой Java-класс, описывающий его поведение, и так или иначе использующий функции из библиотеки rdo_lib. Далее полученные классы компилируются Java-модулем среды Eclipse, после чего происходит запуск имитационной модели.

Пример процедуры, создающей класс константы языка:

Вызов:

```
fsa.generateFile(filename+"/"+e.name+".java", e.compileConstant(filename))
```

Тело функции:

```
def compileConstant(ConstantDeclaration con, String filename)
{
	'''
	package «filename»;

	public class «con.name»
	{
		public static final «con.type.compileType» value = «
			IF con.type.compileType.endsWith("_enum")»«
			con.value.compileExpressionContext(
			(new LocalContext).populateWithEnums(
				con.type.resolveAllSuchAs as RDOEnum)).value»«
			ELSE»«con.value.compileExpression.value»«ENDIF»;
		«IF con.type instanceof RDOEnum»
```

```
            public enum «(con.type as RDOEnum)
                    .getEnumParentName(false)»_enum
            {
                «(con.type as RDOEnum).makeEnumBody»
            }«
        ENDIF»
    }
    '''
}
```

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

1. Проведено предпроектное исследование системы имитационного моделирования РДО и определены основные функции и требования, предъявляемые к интегрированной среде разработки.
2. На этапе концептуального проектирования системы и выбрана платформа, на которой будет основываться создаваемая интегрированная среда разработки. Составлено техническое задание.
3. На этапе технического проектирования описана грамматика языка и на ее основе созданы начальные версии компонентов среды.
4. На этапе рабочего проектирования были расширены и дополнены компоненты создаваемой среды разработки, а также написана библиотека симулятора языка РДО и правила генерации Java-кода из исходного текста имитационных моделей.

Данные изменения в будущем позволят реализовывать более сложные архитектурные концепции, связанные как с самим синтаксисом языка РДО, так и с введением новых подходов в имитационном моделировании. В ближайшие планы входит создание подсистемы анимации, трассировки и сбора статистической информации с последующем построением графиков на ее основе.

9. Список используемых источников

1. Справка по языку РДО [http://rdo.rk9.bmstu.ru/help/];
2. Справка по RAO-studio [http://rdo.rk9.bmstu.ru/help/];
3. Емельянов В. В., Ясиновский С. И. Имитационное моделирование систем: Учеб. Пособие – М.: Издательство МГТУ им. Н. Э. Баумана, 2009. – 584 с.: ил. (Информатика в техническом университете);
4. [http://ru.wikipedia.org/wiki/Интегрированная_среда_разработки]
5. [http://ru.wikipedia.org/wiki/Синтаксический_анализ]
6. [http://ru.wikipedia.org/wiki/Eclipse_(среда_разработки)]
7. Martin Fowler – Domain-Specific Languages: Addison-Wesley, 2010. - 640 с.
8. Lorenzo Bettini – Implementing Domain-Specific Languages with Xtext and Xtend: Packt Publishing, 2013. – 343 с.
9. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78;
10. Леоненков. Самоучитель по UML [http://khpi-iip.mipk.kharkiv.edu/library/case/leon/index.html];
11. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-90. Условные обозначения и правила выполнения.

10. Список использованного программного обеспечения

1. Eclipse 4.2 Kepler + Xtext + Xtend
2. Eclipse Papyrus UML plugin
3. LibreOffice Draw 4.3
4. LibreOffice Writer 4.3
5. Inkscape SVG Editor 0.52
6. Dassault DraftSight 1.5b
7. Git

Приложение А

```
// ============================================================
//                       Statement language
// ============================================================
StatementList:
        (statements += Statement)+
;

Statement
        : empty ?= ';'
        | ExpressionStatement ';'
        | NestedStatement
        | LocalVariableDeclaration ';'
        | IfStatement
        | ForStatement
        | BreakStatement ';'
        | ReturnStatement ';'
//      | process_input_statement
        | NoChangeStatement ';'
        | StoppingStatement ';'
        | PlanningStatement ';'
        | LegacySetStatement ';'
//      | watch_start
//      | watch_stop
;

ExpressionStatement:
        expr = ExpressionAssignment
;

NestedStatement:
        invoke = '{' statements = StatementList?  '}'
;

LocalVariableDeclaration:
        type = RDOType list = VariableDeclarationList
;

VariableDeclarationList:
        declarations += VariableDeclarationInit (',' declarations +=
VariableDeclarationInit)*
;

VariableDeclarationInit:
        name = ID ('=' value = Expression)?
;

IfStatement:
        'if' '(' condition = Expression ')'
                then = Statement
        ( => 'else'
                else = Statement
        )?
;
```

```
ForStatement:
        'for' '('
        ( declaration = LocalVariableDeclaration | init = Expression )? ';'
          condition = Expression? ';'
          update = Expression? ')'
        body = Statement
;

BreakStatement:
        break ?= 'break'
;

ReturnStatement:
        invoke = 'return' return = Expression?
;

NoChangeStatement:
        name = ID 'NoChange'
;

StoppingStatement:
        event = [Event|FQN] '.' 'stopping' '(' ')'
;

PlanningStatement:
        event = [Event|FQN] '.' 'planning' '(' value = Expression ')'
        ('(' parameters = ResourceExpressionList ')')?
;

LegacySetStatement:
        call = FQN 'set' value = Expression
;

// ============================================================
//                       Expression language
// ============================================================
ExpressionList:
        values += Expression  (',' values += Expression)*
;

Expression
        : ExpressionAssignment
;

// Assignment: left associative, higher priority (-1)
ExpressionAssignment returns Expression:
        ExpressionOr ( => op = ('='|'+='|'-='|'*='|'/='|'%=')
                {ExpressionAssignment.left = current} next = ExpressionOr
        )*
;
```

```
// Or: left associative, priority 0
ExpressionOr returns Expression:
	ExpressionAnd (
		('or' | '||') {ExpressionOr.left = current}
right = ExpressionAnd
	)*
;

// And: left associative, priority 1
ExpressionAnd returns Expression:
	ExpressionCompare (
		('and' | '&&') {ExpressionAnd.left = current}
			right = ExpressionCompare
	)*
;

// Comparisons: left associative, priority 2
ExpressionCompare returns Expression:
	ExpressionPlusMinus (
		('>'  {ExpressionLarger.left=current}
			right=ExpressionPlusMinus) |
		('>=' {ExpressionLarger_Equal.left=current}
			right=ExpressionPlusMinus) |
		('<'  {ExpressionSmaller.left=current}
			right=ExpressionPlusMinus) |
		('<=' {ExpressionSmaller_Equal.left=current}
			right=ExpressionPlusMinus) |
		('==' {ExpressionEqual.left=current}
			right=ExpressionPlusMinus) |
		(('!=' | '<>') {ExpressionNot_Equal.left=current}
			right=ExpressionPlusMinus)
	)*
;

// addition/subtraction: left associative, priority 3
ExpressionPlusMinus returns Expression:
	ExpressionMultiplicationDivisionModulo (
		('+' {ExpressionPlus.left=current}
			right=ExpressionMultiplicationDivisionModulo) |
		('-' {ExpressionMinus.left=current}
			right=ExpressionMultiplicationDivisionModulo)
	)*
;

// multiplication/division, left associative, priority 4
ExpressionMultiplicationDivisionModulo returns Expression:
	ExpressionUnary (
		('*'  {ExpressionMultiplication.left=current}
			right=ExpressionUnary) |
		('/'  {ExpressionDivision.left=current}
			right=ExpressionUnary) |
		(('%' | 'mod') {ExpressionModulo.left=current}
			right=ExpressionUnary)
	)*
;
```

```
// Unary operators: right associative, priority 5
ExpressionUnary returns Expression
        : ExpressionExponentiation
        | (('!' | 'not') {ExpressionNegate} exp=ExpressionUnary)
        | ('-'   {ExpressionInvert} exp=ExpressionUnary)
;

// exponentiation: right associative, priority 6
ExpressionExponentiation returns Expression:
        Primary
        ( ('^' | '**') {ExpressionExponentiation.left=current}
                right=ExpressionExponentiation )?
;

Primary returns Expression
        : {Primary}'(' exp = Expression ')'
        | Atomic
;

Atomic returns Expression
        : {IntConstant}    value = INT
        | {DoubleConstant} value = DOUBLE
        | {TimeNow} invoke = ('Time_now' | 'time_now')
        | {StringConstant} value = STRING
        | {BoolConstant}   value = BOOL
        | {GroupExpression} type = BuiltInLogical '(' arg = GroupBy ')'
        | {SelectExpression} 'Select' '(' arg = GroupBy ')' '.'
                method = SelectMethod '(' (arg2 = ExpressionOr | 'NoCheck'?) ')'
        | ArrayValues
        | VariableIncDecExpression
;

ArrayValues:
        '[' {ArrayValues} values = ExpressionList? ']'
;

enum BuiltInLogical
        : EXIST = 'Exist'
        | NOTEXIST = 'Not_Exist'
        | FORALL = 'For_All'
        | NOTFORALL = 'Not_For_All'
;

enum SelectMethod
        : EXIST = 'Exist'
        | NOTEXIST = 'Not_Exist'
        | FORALL = 'For_All'
        | NOTFORALL = 'Not_For_All'
        | EMPTY = 'Empty'
        | SIZE = 'Size'
;
```

```
GroupBy:
	type = [ResourceType|FQN] ':'
		(condition = ExpressionOr | 'NoCheck')
;

VariableIncDecExpression
	: (pre  = '++' | pre  = '--' ) => var = VariableMethodCallExpression
	| var = VariableMethodCallExpression ( => post = '++' | => post = '--'
	)?
;

VariableMethodCallExpression
	: calls += VariableExpression ('.' calls += VariableExpression)*
;

VariableExpression
	: call = ID
	( => functionfirst ?= '(' (args = ExpressionList)? ')'
		( => arraylast ?= '[' iterator = Expression ']' )?
	| => arrayfirst ?= '[' iterator = Expression ']'
		( => functionlast ?= '(' (args = ExpressionList)? ')' )?
	)?
;
```