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:

Разработка процедурного языка в системе дискретного имитационного моделирования РДО

Студент группы РК9-101 _____ Е.В. Поподьянец

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ А.В.Урусов

(Подпись, дата) (И.О.Фамилия)

Москва, 2012

1. Оглавление

2. ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

БД – База Данных

В – Внедрение

НИР – Научно-Исследовательская Работа

ПИ – Переменная Информация

ПО – Программное Обеспечение

ПП – Программный Продукт

ППП – Пакет Прикладных Программ

РВ – Реальный масштаб Времени

РП – Рабочий Проект

ТЗ – Техническое Задание

ТПР – Типовое Проектное Решение

ТП – Технический Проект

ЭП – Эскизный Проект

ИМ – Имитационная Модель

СДС – Сложная Дискретная Система

СМО – Система Массового Обслуживания

БЗ – База Знаний

ЭВМ - Электронная Вычислительная Машина

ОЗУ - Оперативное Запоминающее Устройство

3. ВВЕДЕНИЕ

Имитационное моделирование предназначено для прогноза поведения сложных систем, а так же результатов их взаимодействия. Дискретно – событийное моделирование подразумевает такой подход к моделированию, при котором абстрагируются от непрерывности происходящих событий. При дискретно – событийном моделировании считается, что состояние системы меняется мгновенно через сколь угодно короткие промежутки времени. Но в промежутке между двумя ближайшими событиями состояние системы остаётся неизменным.

Язык имитационного моделирования РДО поддерживает дискретно-событийную парадигму имитационного моделирования и реализует три основных подхода моделирования: процессно-ориентированный подход, событийный подход, подход сканирования активностей. Он является проблемно-ориентированным языком, направленным на решение задач моделирования.

Проблемная ориентация обычно производится в контексте некоторого универсального языка программирования, по отношению к которому проблемно-ориентированный язык является либо надъязыком, либо предъязыком, либо подъязыком. Надъязык получается обогащением универсального языка дополнительными конструкциями, особенно удобными для формулирования задачи из класса. В предъязыке дополнительные конструкции полностью "загораживают" универсальный язык и переводятся на него специальным препроцессором. Подъязык получается из универсального языка отказом от конструкций, неупотребительных в данном классе задач, либо предварительным составлением библиотеки "стандартных программ", в совокупности достаточных для выражения любой задачи из класса. Во всех случаях выгода от употребления П.-о. я. состоит в том, что вместо программирования заново каждой задачи из класса достаточно лишь указать средствами П.-о. я. параметры, отличающие одну задачу от другой. РДО является предъязыком в контексте процедурного и объектно-ориентированного языка программирования.

Методика имитационного моделирования часто используется для описания реальных систем. В контексте решения задач моделирования реальных систем часто приходится решать проблемы описания сложных алгоритмических зависимостей. Для описания алгоритмов наиболее удобно использование процедурного (императивного) программирования. Про реальные системы можно сказать, что они локально императивны. То есть, если взять достаточно узкую задачу, то ее можно вполне легко описать методами процедурного программирования. Процедурное программирование наиболее пригодно для реализации небольших подзадач, где очень важна скорость исполнения.

4. ПРЕДПРОЕКТНОЕ ИССЛЕДОВАНИЕ

4.1. Основные положения языка РДО

В основе системы РДО (1)– «Ресурсы, Действия, Операции» – 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);
- образцы операций (с расширением .pat);
- точки принятия решений (с расширением .dpt);
- константы, функции и последовательности (с расширением .fun);
- кадры анимации (с расширением .frm);
- прогон (с расширением .smr).

Объекты, создаваемые РДО-имитатором при выполнении прогона:

- результаты (с расширением .pmv);
- трассировка (с расширением .trc).

4.2. Постановка задачи

Основная цель данного курсового проекта – приведение операторов процедурного программирования языка РДО к валидному и пригодному для использования состоянию.

Как в любом процедурном языке программирования, в языке РДО реализован механизм вызова функций. Функция в РДО – подпрограмма, описанная на закладке FUN, с помощью языка процедурного программирования, реализующая определенный алгоритм и вызываемая в других местах с помощью оператора вызова. На данный момент функция являлась основным местом использования операторов процедурного программирования. Однако язык позволяет использовать их и в описании образцов и событий. К сожалению, эта возможность, ввиду различных причин, развивалась отдельно от использования операторов в функциях, что привело к сильному расхождению исходного кода и появлению большого количества ошибок.

Данную проблему можно рассмотреть на примере нескольких моделей, текст которых приведен в приложении (Приложение 1).

В ходе работы с этими моделями были выявлены следующие ошибки: неправильная работа оператора break, невозможность его использования в образцах и событиях, неинформативность сообщений об ошибках и предупреждений, падение модели, при использовании оператора break вне цикла.

Так же в языке присутствует такой тип данных, как массив. Массив является поименованной группой значений одного типа, в том числе и других массивов. Основные отличия от перечислимого типа – все элементы проиндексированы и имеют тип (одинаковый для всех).

Однако в РДО была возможность обращаться лишь ко всему массиву целиком, что резко снижало возможности его использования, т.к. главный признак массива – индекс элементов - фактически не был доступен пользователю. Соответственно возникает задача обеспечить доступ к элементу массива , т.е. ввести операторы установки и получения значения элемента массива. В такой задаче важным является и контроль за границами массива – нельзя обратиться к элементу массива, которого не существует.

Дополняет неудобство использования операторов полнейшее отсутствие справочного материала.

4.3. Концептуальное Проектирование

Система имитационного моделирования РДО безусловно является сложной и статически, и динамически. На это указывает сложная иерархическая структура системы со множеством различных связей между компонентами и ее сложное поведение во времени.

Ярко выраженная иерархическая структура и модульность системы определяют направление изучения системы сверху вниз. Т.е. мне необходимо применять принцип декомпозиции нужных модулей до тех пор, пока не будет достигнут уровень абстракции, представление на котором нужных объектов не нуждается в дальнейшей детализации для решения данной задачи.

4.4. Диаграмма компонентов

Для отображения зависимости между компонентами системы РДО и выделения среди них модернизируемых служит соответствующая диаграмма в нотации UML (3).

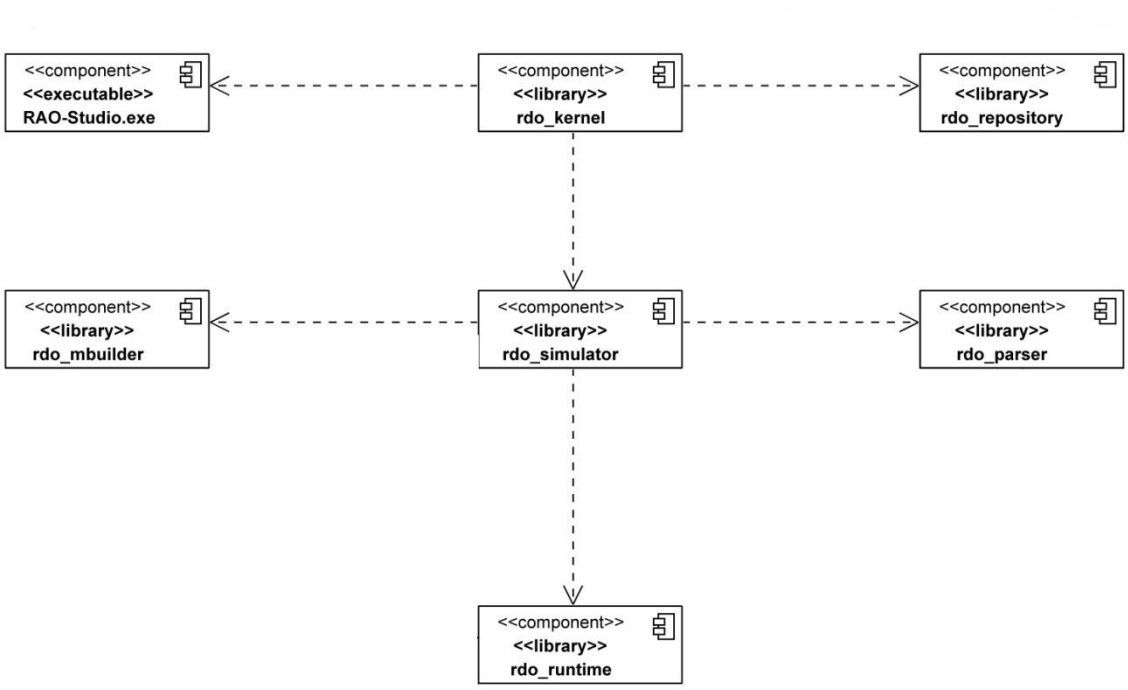

Рис. 1. Упрощенная диаграмма компонентов.

Базовый функционал представленных на диаграмме компонентов (Рис. 1.):

rdo_kernel реализует ядровые функции системы. Не изменяется при разработке системы.

RAO-studio.exe реализует графический интерфейс пользователя. Не изменяется при разработки системы.

rdo_repository реализует управление потоками данных внутри системы и отвечает за хранение и получение информации о модели. Не изменяется при разработке системы.

rdo_mbuilder реализует функционал, используемый для программного управления типами ресурсов и ресурсами модели. Не изменяется при разработке системы.

rdo_simulator управляет процессом моделирования на всех его этапах. Он осуществляет координацию и управление компонентами rdo_runtime и rdo_parser. Не изменяется при разработке системы.

rdo_parser производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО. Модернизируется при разработке системы.

rdo_runtime отвечает за непосредственное выполнение модели, управление базой данных и базой знаний. Модернизируется при разработке системы.

Объекты компонента rdo_runtime инициализуруются при разборе исходного текста модели компонентом rdo_parser. Например, конструктор rdoParse:: RDORTPEnumParamType::constructorSuchAs содержит следующее выражение:

В дальнейшем компоненты rdo_parser и rdo_runtime описываются более детально.

5. Техническое задание

5.1. Основания для разработки

Задание на курсовой проект.

5.2. Общие сведения

Основной разработчик РДО – кафедра РК-9, МГТУ им. Н.Э. Баумана.

5.3. Назначение и цели развития системы

Исправить существующие ошибки в операторах языка процедурного программирования и добавить возможность работы с элементами массива.

5.4. Характеристики объекта автоматизации

РДО – язык имитационного моделирования, включающий на данный момент подход сканирования активностей, процессный и событийный подход.

5.5. Требования к системе

5.5.1. Требования к функциональным характеристикам

- обеспечение в РДО возможности корректно прерывать циклы с помощью оператора break;
- унификация операторов процедурного языка для закладок событий, образцов и функций;
- обеспечение возможности присваивать значение элементу массива;
- обеспечение корректных предупреждений и сообщений об ошибках для исправляемых и внедряемых операторов процедурного программирования;
- создать модульные и системные тесты для исправляемых и внедряемых операторов процедурного программирования;
- написание справочной документации для операторов процедурного программирования.

5.5.2. Требования к надежности

Основное требование к надежности направлено на поддержание в исправном и работоспособном ЭВМ на которой происходит использование программного комплекса RAO-Studio.

5.5.3. Условия эксплуатации

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

5.5.4. Требования к составу и параметрам технических средств

Совпадают с таковыми у программного комплекса RAO-Studio.

5.5.5. Требования к информационной и программной совместимости

Данная система должна работать под управлением операционных систем Windows 2000, Windows XP, Windows Vista и Windows 7.

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 осуществляется выполнением набора модульных и системных тестов в системе непрерывной интеграции(CIS) Jenkins.

6. ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

6.1. Унификация синтаксиса процедурного языка программирования

Для создания лексического анализатора в системе РДО используется генератор лексических анализаторов общего назначения *Bison,* который преобразует описание контекстно-свободной LALR(1) грамматики в программу на языке C++ для разбора этой грамматики. Для того, чтобы Bison мог разобрать программу на каком-то языке, этот язык должен быть описан *контекстно-свободной грамматикой*. Это означает, необходимо определить одну или более *синтаксических групп* и задать правила их сборки из составных частей. Например, в языке C одна из групп называется `выражение'. Правило для составления выражения может выглядеть так: "Выражение может состоять из знака `минус' и другого выражения". Другое правило: "Выражением может быть целое число". Правила часто бывают рекурсивными, но должно быть по крайней мере одно правило, выводящее из рекурсии.

Наиболее распространённой формальной системой для представления таких правил в удобном для человека виде является *форма Бэкуса-Наура* (БНФ, Backus-Naur Form, BNF), которая была разработана для описания языка Algol 60. Любая грамматика, выраженная в форме Бэкуса-Наура является контекстно-свободной грамматикой. Bison принимает на вход, в сущности, особый вид БНФ, адаптированный для машинной обработки.

В правилах формальной грамматики языка каждый вид синтаксических единиц или групп называется *символом*. Те из них, которые формируются группировкой меньших конструкций в соответствии с правилами грамматики, называются *нетерминальными символами*, а те, что не могут разбиты -- *терминальными символами* или *типами лексем*. Мы называем часть входного текста, соответствующую одному терминальному символу *лексемой*, а соответствующую нетерминальному символу -- *группой*.

Каждая группа, так же как и её нетерминальный символ, может иметь семантическое значение. В компиляторе языка программирования выражение обычно имеет семантическое значение в виде дерева, описывающего смысл выражения.

Синтаксическая диаграмма алгоритмической функции представлена на листе 2 курсового проекта.

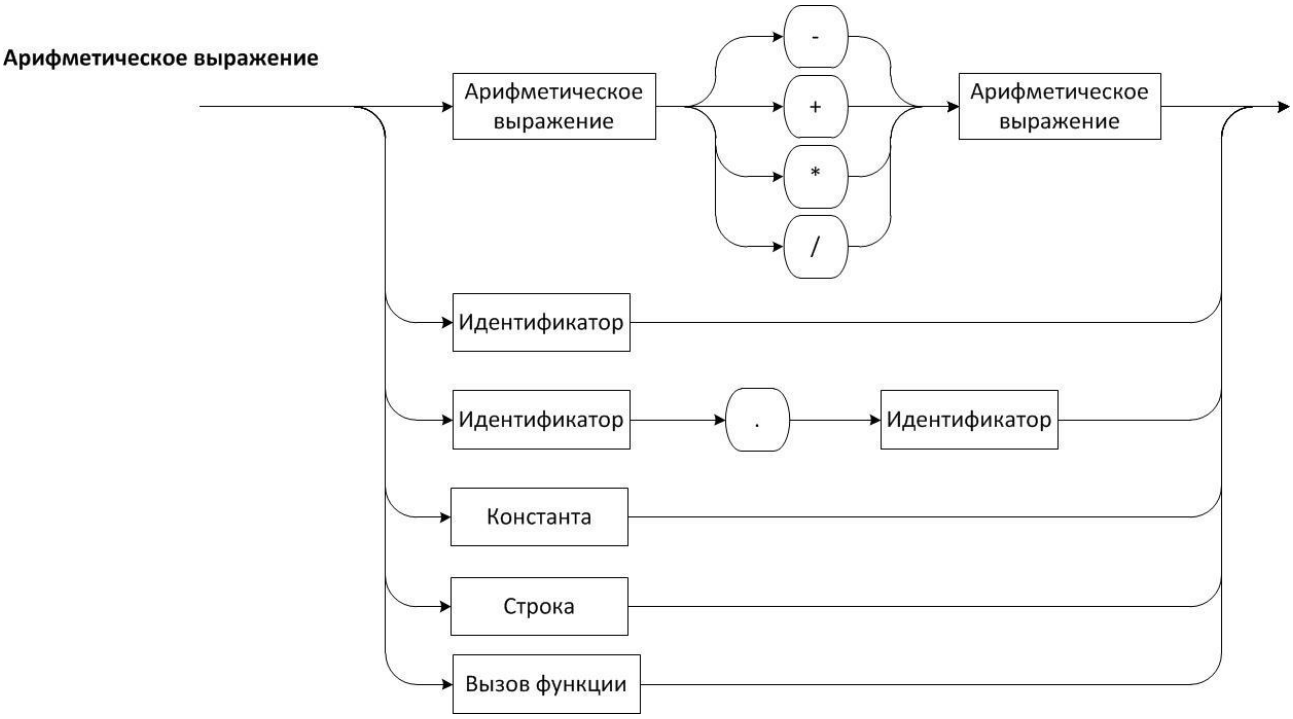

Рис. 2.Синтаксическая диаграмма группы «арифметическое выражение».

Описание алгоритмической функции следует за описанием констант в объекте символьных констант, функций и последовательностей (с расширением .fun).

Как можно было убедиться на этапе предпроектного исследования, тело функции практически идентично телу конвертора образца (с расширением .pat) и события (с расширением .evn). Основные отличия: отсутствие оператора return в образцах и событиях, наличие в них специфических инструкций, таких как планирование события и т.п., а также невозможность функций менять свои параметры.

Таким образом, можно свести синтаксис и грамматику 3х объектов (тела функции, конвертора образца и конвертора события) к таковым у тела функции с небольшими изменениями.

6.2. Разработка архитектуры компонента rdo_parser

Для возможности обработки новой конструкции в коде модели и модификации старых требуют изменений синтаксический анализатор РДО, необходимо создать новый контекст для оператора break, а также разработать алгоритм проверки функции на возможность вернуть значение.

6.3. Разработка архитектуры компонента rdo_runtime

В пространстве имен rdoRuntime необходимо добавить классы и методы, реализующие логику выполнения инструкций, показанную на диаграмме активностей. Должны быть созданы два типа списков – базовый и прерываемый, а так же ловушки для инструкций прерывания break и return. Так же необходимо внести проверку выхода за пределы массива, ее мы добавим в классы RDOCalcSetArrayItem и RDOCalcSetArrayItem.

7. РАБОЧЕЕ ПРОЕКТИРОВАНИЕ

7.1. Изменения в файлах синтаксического анализатора

Для реализации в среде имитационного моделирования нового инструмента разработанного на концептуальном и техническом этапах проектирования, в первую очередь необходимо добавить новые термальные символы в лексический анализатор РДО.

7.1.1. Синтаксический анализ установки элемента массива

Правило для установки элемента массива в системе РДО имеет вид:

```
set_array_item_statement
  : RDO_IDENTIF '[' fun_arithm ']' '=' fun_arithm
```

Код на языке C++, выполняемый при свертке данной группы:

```
LPRDOValue    pParamName   = PARSER->stack().pop<RDOValue>($1);
ASSERT(pParamName);
LPRDOFUNArithm pArrayArithm = RDOFUNArithm::generateByIdentificator(pParamName);
LPRDOFUNArithm pArithmInd   = PARSER->stack().pop<RDOFUNArithm>($3);
LPRDOFUNArithm pRightArithm = PARSER->stack().pop<RDOFUNArithm>($6);
ASSERT(pArrayArithm);
ASSERT(pArithmInd  );
ASSERT(pRightArithm);

if (!pArrayArithm->typeInfo()->type().object_dynamic_cast<RDOArrayType>())
{
        PARSER->error().error(@1, rdo::format(_T("'%s' не является массивом."),
                pParamName->value().getIdentificator().c_str()));
}

LPRDOType pType = pArrayArithm->typeInfo()->type();
ASSERT(pType);
LPRDOArrayType pArrayType = pType.object_dynamic_cast<RDOArrayType>();
ASSERT(pArrayType);

LPTypeInfo pItemType = pArrayType->getItemType()->type_cast(pRightArithm->typeInfo(),
RDOParserSrcInfo(@1));
ASSERT(pItemType);
```

```
rdo::runtime::LPRDOCalc pArrayItemCalc =
rdo::Factory<rdo::runtime::RDOCalcSetArrayItem>::create(pArrayArithm->calc(), pArithmInd-
>calc(), pRightArithm->calc());
ASSERT(pArrayItemCalc);

tstring   paramName = pParamName->value().getIdentificator();
LPContext pContext  = PARSER->context();
ASSERT(pContext);
LPContextMemory pContextMemory = pContext->cast<ContextMemory>();
ASSERT(pContextMemory);
LPLocalVariableListStack pLocalVariableListStack = pContextMemory->getLocalMemory();
ASSERT(pLocalVariableListStack);
LPLocalVariable pLocalVariable = pLocalVariableListStack->findLocalVariable(paramName);
rdo::runtime::LPRDOCalc pCalc;
if (pLocalVariable)
{
        pCalc=rdo::Factory<rdo::runtime::RDOCalcSetLocalVariable<rdo::runtime::
                ET_EQUAL> >::create(paramName, pArrayItemCalc);
}
else
{
        RDOFUNArithm::wrongVarInit(pParamName, paramName);
}

LPExpression pExpression = rdo::Factory<Expression>::create(pArrayArithm->typeInfo(), pCalc,
RDOParserSrcInfo(@1));
ASSERT(pExpression);

$$ = PARSER->stack().push(pExpression);
```

В данном правиле присутствует ограничение, появляющееся из-за того, что массив может быть параметром функции и его менять мы не имеем права. В паттерне данное ограничение отсутсвует.

7.1.2. Синтаксический анализ оператора For

В файлах грамматик синтаксического анализатора были изменены правила сверток для оператора for. Теперь они происходят в несколько этапов и выглядят так:

```
for_statement
: for_header statement

for_header
: RDO_for '(' local_variable_declaration ';' fun_logic ';' equal_statement ')'
| RDO_for '(' equal_statement ';' fun_logic ';' equal_statement ')
```

7.1.3. Синтаксический анализ оператора If

В файлах грамматик синтаксического анализатора были изменены правила сверток для оператора if. Теперь они выглядят так:

```
else_statement
: RDO_else statement

then_statement
: statement

if_statement
: if_condition then_statement
| if_condition then_statement else_statement

if_condition
  : RDO_if '(' fun_logic ')'
```

7.2. Изменения в пространстве имен rdoParser

7.2.1. Объявление класса RDOContextReturnable

Класс определяет возвращаемое пространство одной из веток функции. Необходим для проверки функции на возможность вернуть значение.

```
CLASS(ContextStatementBase):
        INSTANCE_OF (Context)
{
DECLARE_FACTORY(ContextStatementBase);
protected:
  ContextStatementBase();
};
DECLARE_POINTER(ContextStatementBase);
```

```
CLASS(ContextReturnable):
        INSTANCE_OF (ContextStatementBase)
{
DECLARE_FACTORY(ContextReturnable);
public:

  typedef std::vector<LPContextReturnable> ContextReturnableList;
  void addContext(REF(LPContextReturnable) pContext);

  void setReturnFlag();
  bool returnFlag();

private:
  bool checkChildFlags();

protected:
  ContextReturnable();

  ContextReturnableList m_contextReturnableList;
  bool                  m_returnFlag;

};
```

7.2.2. Объявление класса RDOContextReturnable

Класс определяет контекст в котором можно применять оператор break. Необходим для проверки на правильность использования оператора break.

```
CLASS(ContextBreakable):
        INSTANCE_OF (ContextStatementBase)
{
DECLARE_FACTORY(ContextBreakable);
protected:
  ContextBreakable();
};
DECLARE_POINTER(ContextBreakable);
```

7.3. Изменения в пространстве имен rdoRuntime

7.3.1. Объявление класса RDOCalcArrayItem

```
//! Возвращает элемент массива
CALC(RDOCalcArrayItem)
{
DECLARE_FACTORY(RDOCalcArrayItem)
private:
  RDOCalcArrayItem(CREF(LPRDOCalc) pArray, CREF(LPRDOCalc) pArrayInd);

  LPRDOCalc m_pArray;
  LPRDOCalc m_pArrayInd;
```

```
    DECLARE_ICalc;
};
```

7.3.2. Объявление класса RDOCalcSetArrayItem

```
//! Устанавливает элемент массива
CALC(RDOCalcSetArrayItem)
{
    DECLARE_FACTORY(RDOCalcSetArrayItem)
private:
    RDOCalcSetArrayItem(CREF(LPRDOCalc) pArray, CREF(LPRDOCalc) pArrayInd,
CREF(LPRDOCalc) pSetItem);

    LPRDOCalc m_pArray;
    LPRDOCalc m_pArrayInd;
    LPRDOCalc m_pSetItem;

    DECLARE_ICalc;
};
```

7.3.3. Объявление класса RDOCalcIf

```
//! Условный оператор if () then {}
CALC(RDOCalcIf)
{
DECLARE_FACTORY(RDOCalcIf)
public:
    void  setThenStatement(CREF(LPRDOCalc) pStatement);
    void  setElseStatement(CREF(LPRDOCalc) pStatement);

    rbool hasElse() const;

private:
    typedef  std::pair<LPRDOCalc, LPRDOCalc>  Statements;

    RDOCalcIf(CREF(LPRDOCalc) pCondition);

    LPRDOCalc   m_pCondition;
    Statements  m_statements;

    DECLARE_ICalc;
};
```

7.3.4. Объявление класса RDOCalcFor

```
//! Оператор цикла for
CALC(RDOCalcFor)
{
DECLARE_FACTORY(RDOCalcFor)
public:
    void setStatement(CREF(LPRDOCalc) pStatement);

private:
    RDOCalcFor(CREF(LPRDOCalc) pDeclaration, CREF(LPRDOCalc) pCondition,
CREF(LPRDOCalc) pExpression);
```

```
        LPRDOCalc m_pDeclaration;
        LPRDOCalc m_pCondition;
        LPRDOCalc m_pExpression;
        LPRDOCalc m_pStatement;

        DECLARE_ICalc;
};
```

7.3.5. Объявление класса RDOCalcBaseStatementList

```
//! Простой список операторов
CALC(RDOCalcBaseStatementList)
{
DECLARE_FACTORY(RDOCalcBaseStatementList)
public:
        void      addCalcStatement(CREF(LPRDOCalc) pStatement);
        RDOCalcList statementList();

protected:
        RDOCalcBaseStatementList();

        RDOCalcList m_calcStatementList;

        DECLARE_ICalc;
};
```

7.3.6. Объявление класса RDOCalcStatementList

```
//! Останавливаемый список операторов
CALC_SUB(RDOCalcStatementList, RDOCalcBaseStatementList)
{
DECLARE_FACTORY(RDOCalcStatementList)
private:
        RDOCalcStatementList();

        DECLARE_ICalc;
};
```

7.3.7. Объявление класса RDOCalcBreakCatch

Класс предназначен для остановки работы цикла, если был использован оператор break. Меняет флаг остановки для продолжения выполнения модели.

```
//! Ловушка для break
CALC(RDOCalcBreakCatch)
{
DECLARE_FACTORY(RDOCalcBreakCatch)
public:
        void addStatementList(CREF(LPRDOCalc) pStatementList);

private:
        RDOCalcBreakCatch();

        LPRDOCalc m_pStatementList;
```

```
    DECLARE_ICalc;
};
```

7.3.8. Объявление класса RDOCalcReturnCatch

Класс предназначен для остановки работы функции, если был использован оператор return и проброса наружу исполняемого калка значения функции. Меняет флаг остановки для продолжения выполнения модели.

```
//! Ловушка для return
CALC(RDOCalcReturnCatch)
{
DECLARE_FACTORY(RDOCalcReturnCatch)
public:
    void setTryCalc(CREF(LPRDOCalc) pTryCalc);

private:
    RDOCalcReturnCatch();

    LPRDOCalc m_pTryCalc;

    DECLARE_ICalc;
};
```

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

1) Проведено предпроектное исследование системы имитационного моделирования РДО.

2) На этапе концептуального проектирования системы с помощью диаграммы компонентов нотации UML укрупненно показано внутреннее устройство РДО и выделены те компоненты, которые потребуют внесения изменений в ходе этой работы.

3) На этапе технического проектирования доработан синтаксис событий, функции, образцов, который представлен на синтаксической диаграмме. С помощью диаграммы классов разработана архитектура новой системы. С помощью диаграммы активностей смоделирован механизм выполнения последовательностей инструкций в функции, паттерне и образце. С помощью блок-схемы показан алгоритм проверки функции на возможность вернуть значение.

4) На этапе рабочего проектирования написан программный код для реализации спроектированных ранее алгоритмов работы и архитектуры компонента rdo_parser и rdo_runtime системы РДО. Проведены отладка и тестирование нового функционала системы, в ходе которых исправлялись найденные ошибки.

5) Для демонстрации новых возможностей системы модели, представленные на этапе постановки задачи, были реализованы в системе РДО. На их основе созданы системыне тесты для системы непрерывной интеграции. Результаты тестов позволяют сделать вывод об адекватной работе функций системы, которые были изменены и добавлены.

6) Все внесенные в систему изменения отражены в справочной информации по системе РДО, что позволяет пользователям оперативно получать справку по новым функциям системы.

9. Список использованных источников

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

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

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

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

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

6. Бьерн Страуструп. Язык моделирования C++. Специальное издание. Пер. с англ. – М.: ООО «Бином-пресс», 2007 г. – 1104 с.: ил.

10. ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ ФУНКЦИЙ, ОБРАЗЦОВ И СОБЫТИЙ В ТЕСТОВЫХ МОДЕЛЯХ

Модель для оператора break

EVN:

```
$Pattern Тестирование_события : event
$Relevant_resources
PEC : Ресурс_для_события Keep
$Body
  PEC
  Convert_event
  integer i = 0;
  for( integer k = 0; k < 10; k++)
  {
       i++;
       if  ( i >= 5)
       {
       break;
       }
   }
  Параметр = i;
$End
```

PAT:

```
$Pattern Тестирование_паттерна : rule
$Relevant_resources
PEC : Ресурс_для_паттерна Keep
$Body
  PEC
  Choice from PEC.Параметр == 0
  Convert_rule
  integer i = 0;
  for( integer k = 0; k < 10; k++)
  {
       i++;
            if  ( i >= 5)
       {
```

```
      break;
      }
   }
  Параметр = i;
$End

$Pattern Тестирование_функции : rule
$Relevant_resources
PEC : Ресурс_для_функции Keep
$Body
  PEC
  Choice from PEC.Параметр == 0
  Convert_rule
  Параметр  = Функция();
$End
```

FUN:

```
$Function Функция : integer
$Type = algorithmic
$Body
integer i = 0;
for( integer k = 0; k < 10; k++)
{
  i++;
    if  ( i >= 5)
  {
            break;
    }
}
return i;
$End
```

Модель для оператора установки элемента массива

EVN:

```
$Pattern testing : event
$Relevant_resources
REL_TEST : TEST Keep
$Body
```

```
  REL_TEST
  Convert_event
        testing_of_array(Param_1);
$End
```

FUN:

```
$Function testing_of_array :integer
$Type = algorithmic
$Parameters
неизменный_параметр: such_as array_container.Param_1
$Body
  неизменный_параметр[1] = 1;
  return неизменный_параметр.size;
$End
```