*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:**

Реализация передачи ресурсов в качестве параметров функции в системе имитационного моделирования РДО

Студент группы РК9-91 _____ Е.В. Поподьянец

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ А.В.Урусов

(Подпись, дата) (И.О.Фамилия)

Москва, 2011

*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 робототехнических комплексов

Реализация передачи ресурсов в качестве параметров функции в системе имитационного моделирования РДО

(Тема курсового проекта)

Студент Поподьянец Е.В., РК9-91

(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к 4 нед., 50% к 8 нед., 75% к 10 нед., 100% к 16 нед.

***1. Техническое задание***

Реализовать передачу ресурсов в качестве параметров функции в системе имитационного моделирования РДО

***2. Оформление курсового проекта***

2.1. Расчетно-пояснительная записка на 34 листах формата А4.

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

1 лист А1 – Постановка задачи

2 лист А1 – Синтаксическая диаграмма

3 лист А1 - Диаграмма классов

4 лист А2 - Диаграмма активностей

5 лист А2 - Диаграмма активностей

6 лист А1 - Лист результатов

Дата выдачи задания « 10 » сентября 2011 г.

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

Студент _____ Поподьянец Е.В.
(Подпись, дата) (И.О.Фамилия)

# 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 – дополнение механизма передачи данных в функцию возможностью передавать и использовать ресурс как параметр функции.

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. Функция имеет свои параметры, используемые во время вызова для передачи внешних, по отношению к пространству памяти функции, данных. Однако в существующей версии РДО нет возможности передать напрямую в функцию ресурс для дальнейшего использования его параметров. Поэтому приходится передавать каждый параметр ресурса по-отдельности, что достаточно неудобно и противоречит парадигме моделирования сложившейся в РДО.

Сложившаяся ситуация приводит к резкому увеличению объема кода и возможности возникновения ошибок из-за лишнего кода при большом количестве функций, вызывающих друг друга либо большого количества различных ресурсов, к которым применяются одни и те же функции.

Данную проблему можно рассмотреть на примере модели игры «Пять» (урезанные пятнашки) , используемой в лабораторной работе по курсу Моделирование ТП и ПП на 9 семестре обучения на кафедре РК-9.

В приведенной ниже таблице можно увидеть, как выглядит модель игры «Пять» в существующей версии РДО и должна выглядеть после внесения изменений в систему моделирования.

| Старая модель «Пять» | Новая модель «Пять» |
|---|---|
| *$Function Где_дырка : such_as Место_дырки*<br>*$Type = algorithmic*<br>*$Parameters*<br>*_Место: such_as Фишка.Местоположение*<br>*$Body*<br>*if (_Место == Дырка.Место + 3) return сверху;*<br>*if (_Место == Дырка.Место - 3) return снизу;*<br>*if (_Место <> 3 and _Место <> 6 and _Место == Дырка.Место - 1) return справа;*<br>*if (_Место <> 1 and _Место <> 4 and _Место == Дырка.Место + 1) return слева;*<br>*if (0 == 0) return дырки_рядом_нет;*<br>*$End*<br><br>*$Function Фишка_на_месте : integer*<br>*$Type = algorithmic*<br>*$Parameters*<br>*_Номер: such_as Фишка.Номер*<br>*_Место: such_as Фишка.Местоположение*<br>*$Body*<br>*if (_Номер == _Место)*<br>*return 1;*<br>*return 0;*<br>*$End*<br><br>*$Function Кол_во_фишек_не_на_месте : integer*<br>*$Type = algorithmic*<br>*$Body*<br>*return 5 - (Фишка_на_месте(Фишка1.Номер, Фишка1.Местоположение)+Фишка_на_месте(Фишка2.Номер, Фишка2.Местоположение)+Фишка_на_месте(Фишка3.Номер, Фишка3.Местоположение)+Фишка_на_месте(Фишка4.Номер, Фишка4.Местоположение)+Фишка_на_месте(Фишка5.Номер, Фишка5.Местоположение));*<br>*$End* | *$Function Где_дырка : such_as Место_дырки*<br>*$Type = algorithmic*<br>*$Parameters*<br>*_Фишка: Фишка*<br>*$Body*<br>*integer Место = _Фишка.Местоположение;*<br>*if (Место== Дырка.Место + 3) return сверху;*<br>*if (Место == Дырка.Место - 3) return снизу;*<br>*if (Место <> 3 and Место <> 6 and Место == Дырка.Место - 1 ) return справа;*<br>*if (Место <> 1 and Место <> 4 and Место == Дырка.Место + 1 ) return слева;*<br>*return дырки_рядом_нет;*<br>*$End*<br><br>*$Function Фишка_на_месте : integer*<br>*$Type = algorithmic*<br>*$Parameters*<br>*_Фишка:Фишка*<br>*$Body*<br>*if (_Фишка.Номер == _Фишка.Местоположение)*<br>*return 1;*<br>*return 0;*<br>*$End*<br><br>*$Function Кол_во_фишек_не_на_месте : integer*<br>*$Type = algorithmic*<br>*$Body*<br>*return 5 - (Фишка_на_месте(Фишка1) + Фишка_на_месте(Фишка2) + Фишка_на_месте(Фишка3) + Фишка_на_месте(Фишка4) + Фишка_на_месте(Фишка5));*<br>*$End* |

***Таблица 1. Сравнение кода модели.***

Как видно в существующей версии приходится вызывать одну функцию с разными параметрами, обращаясь к одному параметру разных ресурсов, чего можно избежать, передавая ресурс в функцию, а так же можно избежать создания нескольких параметров функции, для каждого параметра ресурса.

## 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).

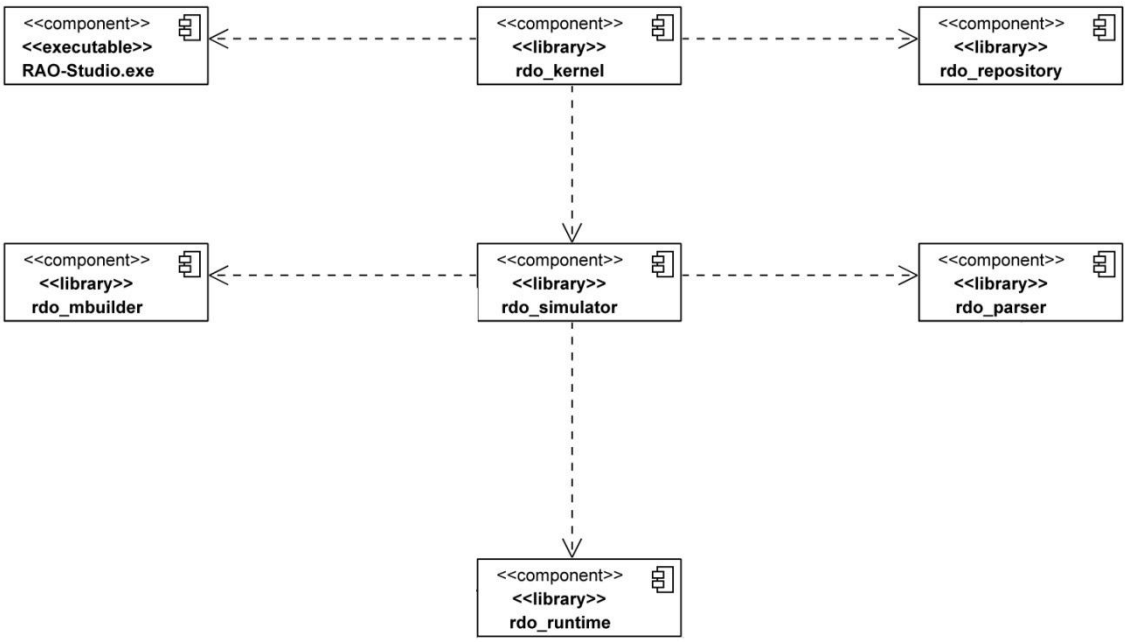


***Рис. 1. Упрощенная диаграмма компонентов.***

Базовый функционал представленных на диаграмме компонентов (**Ошибка! Источник сылки не найден.**):

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:

RDORTPEnumParamType* type = new RDORTPEnumParamType( parent(), enu, dv, such_as_src_info ); которое выделяет место в свободной памяти и инициализирует объект rdoRuntime:: RDORTPEnumParamType, участвующий в дальнейшем процессе имитации.

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.

В систему РДО внедряется возможность работы с ресурсами в качестве параметров функций. Основной разработчик РДО – кафедра РК-9, МГТУ им. Н.Э. Баумана.

### 5.3. Назначение и цели развития системы.

Разработать механизм обращения функций к объектам типа «Ресурс».

### 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 функции работать с параметрами, имеющими тип ресурса;

- обеспечение вызова функции с обычными и релевантными ресурсами, передаваемыми в качестве параметра;

-обеспечение возможности использовать конструкции, указанные в новой модели «Пять» (Таблица 1);

- дополнить документацию новыми возможностями системы.

#### *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*

Программный продукт должен работать на компьютерах со следующими характеристиками:

• объем ОЗУ не менее 256 Мб;

• объем жесткого диска не менее 20 Гб;

• микропроцессор с тактовой частотой не менее 400 МГц;

• монитор не менее 15” с разрешением от 800*600 и выше;

### *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 и приемка передачи параметров должны осуществляться на тестовом примере модели.

# 6. ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

## 6.1. Доработка синтаксиса описания алгоритмической функции

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.

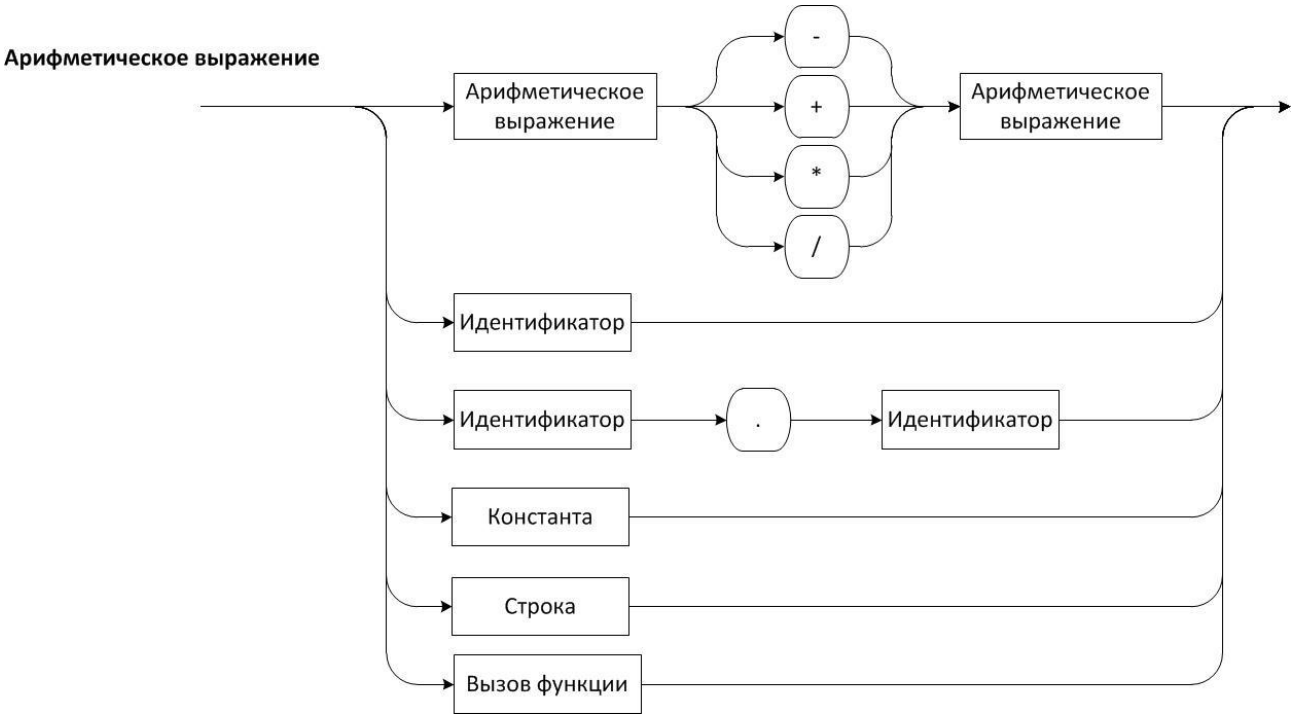


***Рис. 2.Синтаксическая диаграмма группы «арифметическое выражение».***

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

***$Function <имя_функции> : <тип_значения_функции>***
***$Type = algorithmic***
***$Parameters***
***<описание_параметра_функции> { <описание_параметра_функции> }***
***$Body***
***<тело_функции>***
***$End***

имя_функции

Имя функции представляет собой простое имя. Имена должны быть различными для всех функций и не должны совпадать с ранее определенными именами.

тип_значения_функции

Тип значения функции - это один из возможных в языке типов данных. При описании типов значений функции возможны ссылки на типы параметров ресурсов и типы символьных констант. Значения по умолчанию также могут быть указаны при описании типа значения функции, однако они не используются, за исключением функций типа список.

описание_параметра_функции

Описание параметра функции имеет формат:

```
< имя_параметра>:< тип_параметра >
```

*имя_параметра*

Имя параметра - это **простое имя**. Имена параметров должны быть различными для всех параметров данного типа и не должны совпадать с предопределенными и ранее использованными именами. Имя параметра может совпадать с именем параметра другого типа ресурсов.

*тип_параметра*

Тип параметра - это один из возможных *типов данных языка* или ранее описанный *тип ресурса*. При описании типов параметров функции возможны ссылки на типы параметров ресурсов и типы символьных констант. Значения по умолчанию не задают.

*Тип данных языка РДО*

- целый тип - **integer**;
- вещественный тип - **real**;
- строковый тип – **string;**
- логический тип – **bool** ;
- перечислимый тип;
- массив – **array<** *Тип данных языка РДО***>;**
- ссылка на один из выше определенных типов - **such_as**.

Тело алгоритмической функции имеет следующий формат:

<операция>;{<операция>;}

*операция*

Операция представляет собой последовательность операторов языка процедурного программирования, реализующую определенный алгоритм для вычисления значения функции. В операции возможно использование условных операторов, операторов цикла, выражений, операторов возврата значения, оператора прерывания. Значение функции должно быть возвращено оператором return.

Тело функции представляет собой последовательность операций. Операции разделяются между собой разделителем ";".

Значение алгоритмической функции вычисляется следующим образом. Просматриваются в порядке описания в теле функции операции. Значение функции опередеяется первой выполненной операцией, содержащей оператор return, и равно значению выражения после этого оператора.

В арифметических выражениях для вычисления значений функции могут вызываться другие функции, в том числе может рекурсивно вызываться эта же функция, операндами этих выражений могут также являться символьные константы и последовательности.

## 6.2. Разработка архитектуры компонента rdo_parser

Для возможности обработки новой конструкции в коде модели требуют изменений синтаксический анализатор РДО, требуется модифицировать контейнер для значений в системе РДО – RDOValue для возможности хранения в нем произвольных объектов, также требуется добавить наследование классу RDORTPResType от RDOType и Context и реализовать у него соответствующие интерфейсы.

## 6.3. Разработка архитектуры компонента rdo_runtime

В пространстве имен rdoRuntime необходимо модифицировать контейнер RDOValue по аналогии с таковым в компоненте rdo_parser, добавить классы и методы, реализующие логику получения значения параметра ресурса при использовании в теле функции ресурса в качестве параметра.

# 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 описания типа параметра функции*

Описание параметров функции в системе РДО имеет вид:

```
fun_func_params
  : /* empty */
  | fun_func_params RDO_IDENTIF_COLON param_type param_value_default
```

Так как имя параметра менять нет необходимости, то достаточно изменить группу param_type для возможности распознать тип ресурса. К описанию синтаксической группы `param_type был добавлен такой код:`

```
| RDO_IDENTIF
      {
            LPRDOValue pValue = PARSER->stack().pop<RDOValue>($1);
            ASSERT(pValue);

            LPContext pContext = RDOParser::s_parser()->context();
            ASSERT(pContext);

            pContext = pContext->find(pValue);
            ASSERT(pContext);

            LPExpression pExpression = pContext->create(pValue);
            ASSERT(pExpression);

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

Теперь тип параметра может описываться простым именем, являющим собой тип ресурса.

### *7.1.2. Синтаксический анализ вызова параметра у параметра-ресурса*

В файлы синтаксического анализатора было добавлено описание нового арифметического выражения

```
| RDO_IDENTIF '.' RDO_IDENTIF
{ $$ = PARSER->stack().push(
RDOFUNArithm::generateByIdentificator(PARSER->stack().pop<RDOValue>($1),
PARSER->stack().pop<RDOValue>($3))); }
```

## 7.2. Изменения в пространстве имен rdoParser

### *7.2.1. Объявление класса RDORTPResType*

Класс предназначен для хранения типа ресурса, информации о расположении его описания, генерации Expression.

```
CLASS(RDORTPResType):
  INSTANCE_OF     (RDOParserSrcInfo  )
  AND INSTANCE_OF     (boost::noncopyable)
  AND INSTANCE_OF     (RDOType          )
  AND INSTANCE_OF     (Context          )
  AND IMPLEMENTATION_OF(IContextSwitch    )
{
DECLARE_FACTORY(RDORTPResType);
public:
  typedef std::vector<LPRDORTPParam> ParamList;

  enum { UNDEFINED_PARAM = ~0 };

  rsint getNumber  () const   { return m_number;     };
  rbool isPermanent() const   { return m_permanent;  };
  rbool isTemporary() const   { return !m_permanent; };
```

```
LPRDORSSResource createRes(CREF(LPRDOParser) pParser,
CREF(RDOParserSrcInfo) src_info);

void addParam(CREF(LPRDORTPParam) param);
void addParam(CREF(tstring) param_name, rdoRuntime::RDOType::TypeID
param_typeID);
LPRDORTPParam findRTPParam(CREF(tstring) paramName) const;

ruint getRTPParamNumber(CREF(tstring) paramName) const;
CREF(ParamList) getParams () const { return m_params; }

CREF(rdoRuntime::LPIResourceType) getRuntimeResType() const { return
m_pRuntimeResType; }

template<class T>
void end()
{
    rdo::intrusive_ptr<T> pT = rdo::Factory<T>::create(m_number);
    m_pRuntimeResType = pT.template
interface_cast<rdoRuntime::IResourceType>();
    m_pType = m_pRuntimeResType;
    ASSERT(m_pRuntimeResType);
}

void writeModelStructure(REF(std::ostream) stream) const;
```

```
  DECLARE_IType;

private:
  RDORTPResType(CREF(LPRDOParser) pParser, CREF(RDOParserSrcInfo) src_info,
rbool permanent);
  virtual ~RDORTPResType();

  rdoRuntime::LPIResourceType m_pRuntimeResType;
  const ruint              m_number;
  const rbool              m_permanent;
  ParamList                m_params;
  DECLARE_IContextSwitch;
};
```

### *7.2.2. Реализация интерфейса IType*

Данный интерфейс содержит методы кастинга ресурсов, получения значений по умолчанию.

```
tstring RDORTPResType::name() const
{
  static tstring s_name;
  s_name = src_text();
  return s_name;
}
```

```
LPRDOType RDORTPResType::type_cast(CREF(LPRDOType) pFrom,
CREF(RDOParserSrcInfo) from_src_info, CREF(RDOParserSrcInfo) to_src_info,
CREF(RDOParserSrcInfo) src_info) const
	{
	  UNUSED(from_src_info);

	  switch (pFrom->typeID())
	  {
	  case rdoRuntime::RDOType::t_pointer:
		{
			LPRDOType pThisRTPType(const_cast<PTR(RDORTPResType)>(this));

			//! Это один и тот же тип
			if (pThisRTPType == pFrom)
				return pThisRTPType;

			//! Типы разные, сгенерим ошибку
			rdoParse::g_error().push_only(src_info,    _T("Несоответствие типов
ресурсов"));
			rdoParse::g_error().push_only(to_src_info, to_src_info.src_text());
			rdoParse::g_error().push_done();
			break;
		}
	  default:
		{
			rdoParse::g_error().push_only(src_info,    rdo::format(_T("Ожидается
тип ресурса, найдено: %s"), from_src_info.src_text().c_str()));
```

```
                rdoParse::g_error().push_only(to_src_info, rdo::format(_T("См. тип:
%s"), to_src_info.src_text().c_str()));
                rdoParse::g_error().push_done();
                break;
            }
        }

        return LPRDOType(NULL);
    }

    LPRDOValue RDORTPResType::value_cast(CREF(LPRDOValue) pFrom,
CREF(RDOParserSrcInfo) to_src_info, CREF(RDOParserSrcInfo) src_info) const
    {
        ASSERT(pFrom);

        LPRDORTPResType pRTPResType = pFrom->typeInfo()-
>type().object_dynamic_cast<RDORTPResType>();
        if (pRTPResType)
        {
            LPRDOType pThisType = const_cast<PTR(RDORTPResType)>(this);

            //! Это один и тот же тип
            if (pThisType == pRTPResType.object_parent_cast<RDOType>())
                return pFrom;

            //! Типы разные, сгенерим ошибку
```

```
            rdoParse::g_error().push_only(src_info,   _T("Несоответствие типов
ресурсов"));
            rdoParse::g_error().push_only(to_src_info,  rdo::format( _T("Ожидается:
%s"), to_src_info.src_text().c_str()));
            rdoParse::g_error().push_only(src_info,  rdo::format( _T("Пришел: %s"),
pFrom->src_text().c_str()));
            rdoParse::g_error().push_only(to_src_info, to_src_info.src_text());
            rdoParse::g_error().push_done();
          }
          rdoParse::g_error().push_only(src_info,   rdo::format(_T("Ожидается ресурс,
найдено: %s"), pFrom->src_text().c_str()));
          rdoParse::g_error().push_only(to_src_info, rdo::format(_T("См. тип: %s"),
to_src_info.src_text().c_str()));
          rdoParse::g_error().push_done();

          return LPRDOValue(NULL);
        }

        rdoRuntime::LPRDOCalc RDORTPResType::calc_cast(CREF(rdoRuntime::LPRDOCalc)
pCalc, CREF(LPRDOType) pType) const
        {
          return RDOType::calc_cast(pCalc, pType);
        }

        rdoRuntime::RDOValue RDORTPResType::get_default() const
        {
          NEVER_REACH_HERE;
```

```
    return rdoRuntime::RDOValue();
    //return rdoRuntime::RDOValue (pResourceType,pResource);
}
```

### *7.2.3. Реализация интерфейса IContextFind*

Данный интерфейс переключает контекст при нахождении точки между двумя идентификаторами в теле функции и создает Expression содержащий значение параметра ресурса.

```
IContextFind::Result RDORTPResType::onSwitchContext(CREF(LPExpression)
pSwitchExpression, CREF(LPRDOValue) pValue) const
{
    ASSERT(pSwitchExpression);
    ASSERT(pValue         );

    ruint parNumb = getRTPParamNumber(pValue->value().getIdentificator());
    if (parNumb == RDORTPResType::UNDEFINED_PARAM)
    {
        RDOParser::s_parser()->error().error(pValue->src_info(),
rdo::format(_T("Неизвестный параметр ресурса: %s"), pValue-
>value().getIdentificator().c_str()));
    }

    LPRDORTPParam pParam = findRTPParam(pValue->value().getIdentificator());
    ASSERT(pParam);

    LPExpression pExpression = rdo::Factory<Expression>::create(
        pParam->getTypeInfo(),
```

***rdo::Factory<rdoRuntime::RDOCalcGetResParamByCalc>::create(pSwitchExpression->calc(), parNumb),***

***pValue->src_info()***

***);***

***ASSERT(pExpression);***

***return IContextFind::Result(const_cast<PTR(RDORTPResType)>(this), pExpression, pValue);}***

## 7.3. Изменения в пространстве имен rdoRuntime

### *7.3.1. Объявление класса RDOCalcGetResID*

Класс предназначен для получения ID ресурса по пришедшему абстрактному вычислителю, содержащему этот ресурс.

***CALC(RDOCalcGetResID)***

***{***

***DECLARE_FACTORY(RDOCalcGetResID)***

***private:***

***LPRDOCalc m_pCalcGetResource;***

***protected:***

***RDOCalcGetResID(CREF(LPRDOCalc) pCalcGetResource);***

***DECLARE_ICalc;};***

### *7.3.1. Объявление класса RDOCalcGetResParamByCalc*

Класс предназначен для получения параметра ресурса по пришедшему абстрактному вычислителю, содержащему этот ресурс, и номеру параметра этого ресурса.

```
CALC(RDOCalcGetResID)
{
DECLARE_FACTORY(RDOCalcGetResID)

private:
  LPRDOCalc m_pCalcGetResource;
protected:
  RDOCalcGetResID(CREF(LPRDOCalc) pCalcGetResource);

  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, который представлен на синтаксической диаграмме. С помощью диаграммы классов разработана архитектура новой системы. С помощью диаграммы активностей смоделирован механизм передачи ресурсов как параметров функции и работа с ними.

4) На этапе рабочего проектирования написан программный код для реализации спроектированных раннее 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 модель, представленная на этапе постановки задачи, была реализована в системе РДО. Результаты проведения имитационного исследования позволяют сделать вывод об адекватной работе новой функции системы.

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 В МОДЕЛИ ИГРЫ “5”

## 10.1. Текст модели до внедрения разработки

```
$Constant
  Место_дырки: (справа, слева, сверху, снизу, дырки_рядом_нет) =
дырки_рядом_нет
$End

$Function Где_дырка : such_as Место_дырки
$Type = algorithmic
$Parameters
  _Место: such_as Фишка.Местоположение
$Body
  if (_Место == Дырка.Место + 3)  return сверху;
  if (_Место == Дырка.Место - 3)  return снизу;
  if (_Место <> 3  and _Место <> 6 and _Место == Дырка.Место - 1)  return справа;
  if (_Место <> 1  and _Место <> 4 and _Место == Дырка.Место + 1)  return слева;
  if (0 == 0)  return дырки_рядом_нет;
$End

$Function Фишка_на_месте : integer
$Type = algorithmic
$Parameters
  _Номер: such_as Фишка.Номер
  _Место: such_as Фишка.Местоположение
$Body
```

```
  if (_Номер == _Место) return 1;
  if  (0 == 0)         return 0;
$End

$Function Кол_во_фишек_не_на_месте : integer
$Type = algorithmic
$Parameters
  Просто_так: integer
$Body
  if (0 == 0)
    return 5 - (Фишка_на_месте(Фишка1.Номер, Фишка1.Местоположение)+
   Фишка_на_месте(Фишка2.Номер, Фишка2.Местоположение)+
   Фишка_на_месте(Фишка3.Номер, Фишка3.Местоположение)+
   Фишка_на_месте(Фишка4.Номер, Фишка4.Местоположение)+
   Фишка_на_месте(Фишка5.Номер, Фишка5.Местоположение));
$End

$Function Ряд: integer
$Type = algorithmic
$Parameters
  Местоположение: integer
$Body
  if (Местоположение >= 1 and Местоположение <= 3) return 1;
  if (Местоположение >= 4 and Местоположение <= 6) return 2;
  if (Местоположение >= 7 and Местоположение <= 9) return 3;
$End
```

```
$Function Строка: integer
$Type = algorithmic
$Parameters
  Местоположение: integer
$Body
  if ((Местоположение == 1 or Местоположение == 4 or Местоположение == 7))
return 1;
  if ((Местоположение == 2 or Местоположение == 5 or Местоположение == 8))
return 2;
  if ((Местоположение == 3 or Местоположение == 6 or Местоположение == 9))
return 3;
$End

$Function Расстояние_фишеки_до_места : integer
$Type = algorithmic
$Parameters
  Откуда: integer
  Куда  : integer
$Body
  if (Откуда == Куда) return 0;
  if (1 == 1)       return Abs(Ряд(Откуда)-Ряд(Куда)) + Abs(Строка(Откуда)-
Строка(Куда));
$End

$Function Расстояния_фишек_до_мест : integer
$Type = algorithmic
$Parameters
```

```
  Просто_так: integer
$Body
  if (0 == 0)
 return Расстояние_фишеки_до_места(Фишка1.Номер, Фишка1.Местоположение)
+Расстояние_фишеки_до_места(Фишка2.Номер, Фишка2.Местоположение)+
Расстояние_фишеки_до_места(Фишка3.Номер, Фишка3.Местоположение)+
Расстояние_фишеки_до_места(Фишка4.Номер, Фишка4.Местоположение)+
Расстояние_фишеки_до_места(Фишка5.Номер, Фишка5.Местоположение);
$End
```

## 10.2. Текст модели после внедрения разработки

```
$Constant
  Место_дырки: (справа, слева, сверху, снизу, дырки_рядом_нет) =
дырки_рядом_нет
$End


$Function Где_дырка : such_as Место_дырки
$Type = algorithmic
$Parameters
 _Фишка:        Фишка
$Body
   integer Место = _Фишка.Местоположение;
   if (Место== Дырка.Место + 3) return  сверху;
   if (Место == Дырка.Место - 3) return  снизу;
   if (Место <> 3 and Место <> 6 and Место == Дырка.Место - 1     ) return
справа;
   if (Место <> 1 and Место <> 4 and Место == Дырка.Место + 1     ) return
слева;
```

```
  return  дырки_рядом_нет;
$End


$Function Фишка_на_месте : integer
$Type = algorithmic
$Parameters
 _Фишка:         Фишка
$Body
  if (_Фишка.Номер == _Фишка.Местоположение         ) return 1;
  return 0;
$End


$Function Кол_во_фишек_не_на_месте : integer
$Type = algorithmic
$Body
return      5-(Фишка_на_месте(Фишка1) + Фишка_на_месте(Фишка2)
    + Фишка_на_месте(Фишка3) + Фишка_на_месте(Фишка4) +
Фишка_на_месте(Фишка5));
$End


$Function Ряд: integer
$Type = algorithmic
$Parameters
  Местоположение: integer
$Body
  if (Местоположение >= 1 and Местоположение <= 3) return 1;
```

```
    if (Местоположение >= 4 and Местоположение <= 6) return 2;
    if (Местоположение >= 7 and Местоположение <= 9) return 3;
  $End

  $Function Строка: integer
  $Type = algorithmic
  $Parameters
    Местоположение: integer
  $Body
    if (Местоположение == 1 or Местоположение == 4 or Местоположение ==
7) return 1;
    if (Местоположение == 2 or Местоположение == 5 or Местоположение ==
8) return 2;
    if (Местоположение == 3 or Местоположение == 6 or Местоположение ==
9) return 3;
  $End

  $Function Расстояние_фишки_до_места : integer
  $Type = algorithmic
  $Parameters
    _Фишка : Фишка
  $Body
    if (_Фишка.Номер ==_Фишка.Местоположение)      return 0;
    return Abs(Ряд(_Фишка.Номер)-Ряд(_Фишка.Номер)) +
Abs(Строка(_Фишка.Номер)-Строка(_Фишка.Номер));
  $End
```

***$Function Расстояния_фишек_до_мест : integer***

***$Type = algorithmic***

***$Body***

***return Расстояние_фишки_до_места(Фишка1) + Расстояние_фишки_до_места(Фишка2) + Расстояние_фишки_до_места(Фишка3) + Расстояние_фишки_до_места(Фишка4) + Расстояние_фишки_до_места(Фишка5);***

***$End***