

# Оглавление

Введение.....	4
1.Предпроектное исследование .....	5
1.1. Основные подходы к построению ИМ.....	5
1.2. Процесс имитации в РДО .....	6
1.3. Основные положения языка РДО .....	8
1.4. Постановка задачи.....	9
2.Концептуальный этап проектирования.....	11
2.1.Диаграмма компонентов.....	11
2.2.Структура логического вывода РДО .....	12
2.3.Техническое задание .....	13
3.Технический этап проектирования.....	15
3.1.Разработка синтаксиса описания типа ресурса .....	15
3.2.Разработка архитектуры компонента rdo_parser .....	16
3.3.Разработка архитектуры компонента rdo_runtime .....	16
4.Рабочий этап проектирования.....	17
4.1.Синтаксический анализ типов данных.....	17
4.2.Изменения в пространстве имен rdoRuntime.....	19
4.3.Изменения в пространстве имен rdoParser.....	21
Заключение .....	23
Список использованных источников .....	24
Список использованного программного обеспечения .....	24
Приложение 1. Модель гибкой производственной ячейки на языке РДО .....	25
Приложение 2. Полный синтаксический анализ описания типа данных (rdortp.y).....	31
Приложение 3. Полный синтаксический анализ описания ресурса (rdorss.y). ....	42

## Введение

««Сложные системы», «системность», «бизнес-процессы», «управление сложными системами», «модели» – все эти термины в настоящее время широко используются практически во всех сферах деятельности человека». Причиной этого является обобщение накопленного опыта и результатов в различных сферах человеческой деятельности и естественное желание найти и использовать некоторые общесистемные принципы и методы. Именно системность решаемых задач в перспективе должна стать той базой, которая позволит исследователю работать с любой сложной системой, независимо от ее физической сущности. Именно модели и моделирование систем является тем инструментом, которое обеспечивает эту возможность.

Имитационное моделирование (ИМ) на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами в них. К таким системам можно отнести экономические и производственные объекты, морские порты, аэропорты, комплексы перекачки нефти и газа, программное обеспечение сложных систем управления, вычислительные сети и многие другие. Широкое использование ИМ объясняется сложностью (а иногда и невозможностью) применения строгих методов оптимизации, которая обусловлена размерностью решаемых задач и неформализуемостью сложных систем. Так выделяют, например, следующие проблемы в исследовании операций, которые не могут быть решены сейчас и в обозримом будущем без ИМ:

1. Формирование инвестиционной политики при перспективном планировании.
2. Выбор средств обслуживания (или оборудования) при текущем планировании.
3. Разработка планов с обратной информационной связью и операционных предписаний.

Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

1. Сделать выводы о поведении СДС и ее особенностях:
  - без ее построения, если это проектируемая система;
  - без вмешательства в ее функционирование, если это действующая система, проведение экспериментов над которой или слишком дорого, или небезопасно;
  - без ее разрушения, если цель эксперимента состоит в определении пределов воздействия на систему.
2. Синтезировать и исследовать стратегии управления.
3. Прогнозировать и планировать функционирование системы в будущем.
4. Обучать и тренировать управленческий персонал и т.д.

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.

Интеллектуальное ИМ, характеризующееся 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, снимает часть проблем использования ИМ.

## **1.Предпроектное исследование**

### **1.1. Основные подходы к построению ИМ**

Системы имитационного моделирования СДС в зависимости от способов представления процессов, происходящих в моделируемом объекте, могут быть дискретными и непрерывными, пошаговыми и событийными, детерминированными и статистическими, стационарными и нестационарными.

Рассмотрим основные моменты этапа создания ИМ [3]. Чтобы описать функционирование СДС надо описать интересующие нас события и действия, после чего создать алфавит, то есть дать каждому из них уникальное имя. Этот алфавит определяется как природой рассматриваемой СДС, так и целями ее анализа. Следовательно, выбор алфавита событий СДС приводит к ее упрощению – не рассматриваются многие ее свойства и действия не представляющие интерес для исследователя.

Событие СДС происходит мгновенно, то есть это некоторое действие с нулевой длительностью. Действие, требующее для своей реализации определенного времени, имеет собственное имя и связано с двумя событиями – начала и окончания. Длительность действия зависит от многих причин, среди которых время его начала, используемые ресурсы СДС, характеристики управления, влияние случайных факторов и т.д. В течение времени протекания действия в СДС могут возникнуть события, приводящие к преждевременному завершению действия. Последовательность действий образует процесс в СДС (Рис. 1)

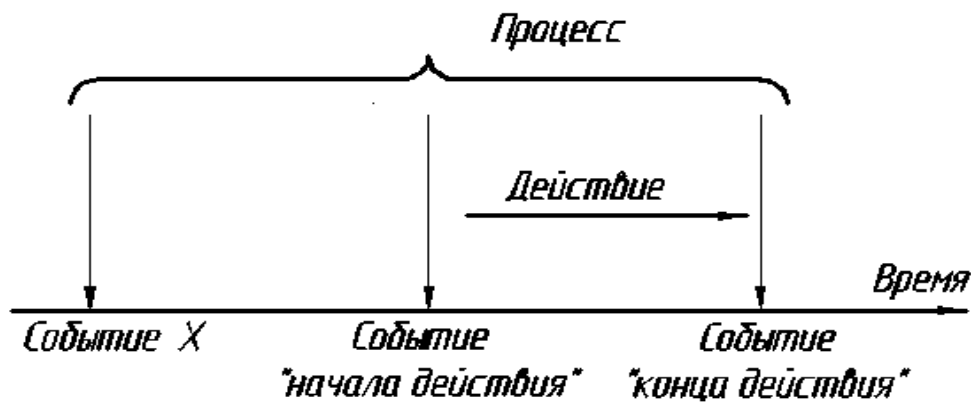


Рис. 1. Взаимосвязь между событиями, действием и процессом.

В соответствии с этим выделяют три альтернативных методологических подхода к построению ИМ: событийный, подход сканирования активностей и процессно-ориентированный.

## 1.2. Процесс имитации в РДО

Для имитации работы модели в РДО [1] реализованы два подхода: событийный и сканирования активностей.

### ***Событийный подход.***

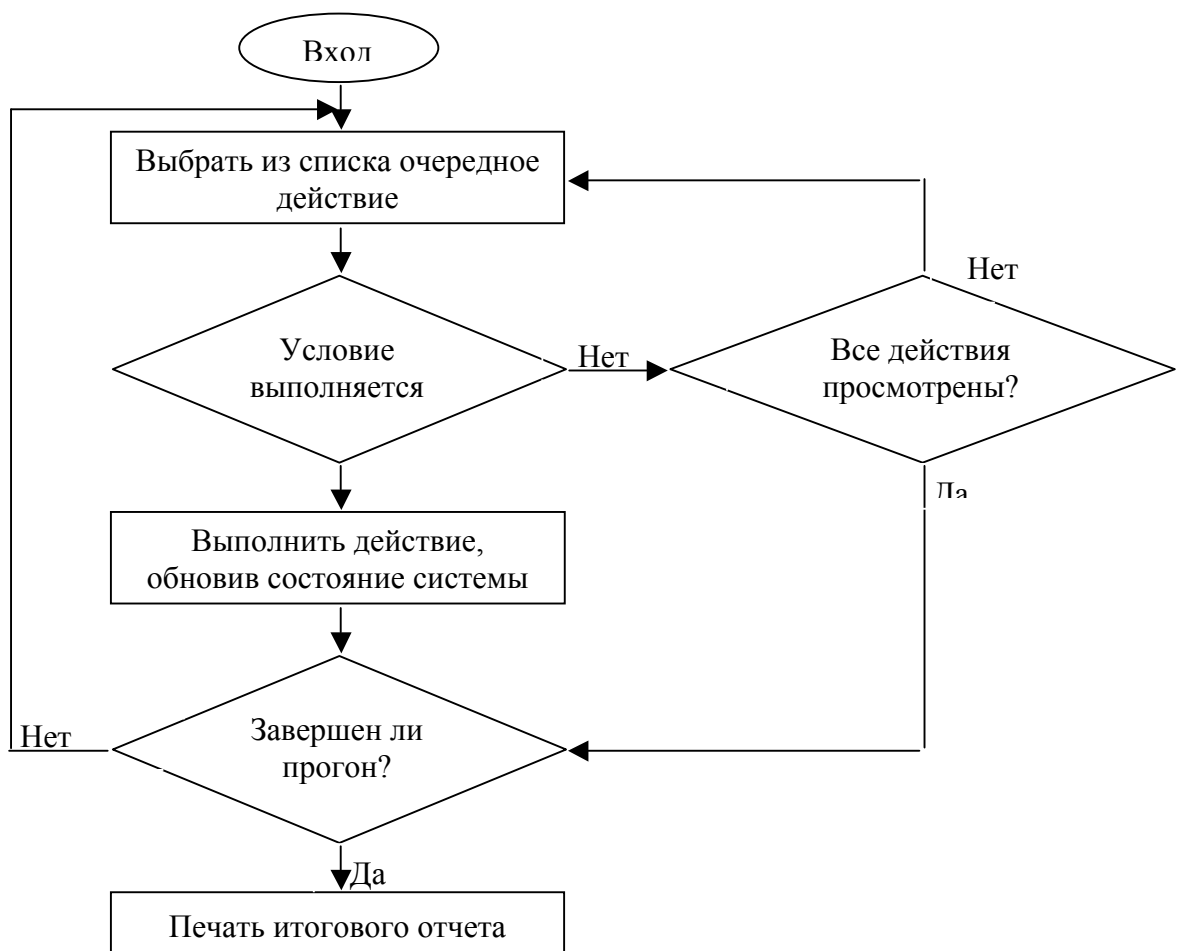
При событийном подходе исследователь описывает события, которые могут изменять состояние системы, и определяет логические взаимосвязи между ними (Рис. 2). Начальное состояние устанавливается путем задания значений переменным модели и параметров генераторам случайных чисел. Имитация происходит путем выбора из списка будущих событий ближайшего по времени и его выполнения. Выполнение события приводит к изменению состояния системы и генерации будущих событий, логически связанных с выполняемым. Эти события заносятся в список будущих событий и упорядочиваются в нем по времени наступления. Например, событие начала обработки детали на станке приводит к появлению в списке будущих событий события окончания обработки детали, которое должно наступить в момент времени равный текущему времени плюс время, требуемое на обработку детали на станке. В событийных системах модельное время фиксируется только в моменты изменения состояний.

### ***Подход сканирования активностей.***



**Рис. 2. Выполнение событий в ИМ.**

При использовании подхода сканирования активностей разработчик описывает все действия, в которых принимают участие элементы системы, и задает условия, определяющие начало и завершение действий (Рис. 3). После каждого продвижения имитационного времени условия всех возможных действий проверяются и если условие выполняется, то происходит имитация соответствующего действия. Выполнение действия приводит к изменению состояния системы и возможности выполнения новых действий. Например, для начала действия обработка детали на станке необходимо наличие свободной детали и наличие свободного станка. Если хотя бы одно из этих условий не выполнено, действие не начинается.



**Рис. 3. Блок-схема реализации подхода сканирования**

### 1.3. Основные положения языка РДО

В основе системы РДО[2] – «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 .trp);
- ресурсы (с расширением .rss);

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

Объекты, создаваемые РДО-имитатором при выполнении прогона:

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

## 1.4. Постановка задачи

Основная идея курсового проекта – добавления в язык имитационного моделирования РДО такого типа данных как массивы. Под массивами, в данном контексте, подразумеваются контейнеры, способные хранить данные других типов, существующих в РДО. Так же данные контейнеры будут способны хранить другие контейнеры, что позволит создавать многомерные массивы, степень вложенности которых будет ограничена только типом файловой системы, в которой запущена RAO Studio. Массивы должны обладать рядом функций, необходимых для работы с ними, таких как добавление элементов в массив, удаление элементов из массива и обращение к элементу массива по индексу.

На данный момент в RAO Studio отсутствует такой тип данных как массивы, это лишает язык РДО необходимой гибкости, а так же делает труднореализуемым или невозможным решение некоторых задач в системе РДО. Рассмотрим данный недостаток на примере модели гибкой производственной ячейки. ГПЯ состоит из трех станков, обслуживаемых промышленным роботом, и четырех накопителей: входной накопитель, промежуточный накопитель, расположенный между первым и вторым станками, промежуточный накопитель, расположенный между вторым и третьим станками, и выходной накопитель. Детали, расположенные во входном накопителе устанавливаются роботом на первый станок, где они обрабатываются. После чего робот помещает их в промежуточный накопитель, расположенный между первым и вторым станками. Потом робот устанавливает детали на второй станок. Детали, обработанные на втором станке, помещаются в промежуточный накопитель, расположенный между вторым и третьим станками. Откуда робот их устанавливает на третий станок, и после окончания обработки робот помещает детали в выходной накопитель.

Модель данной гибкой производственной ячейки на языке РДО представлена в Приложении 1.

Рассмотрим функцию, описывающую последовательность обработки детали, и функцию, описывающую время обработки детали, описанные на закладке FUN.

```
$Function Время_обработки : integer[0..30]
$Type = table
$Parameters
  Номер_станка : integer [1..3]
  Номер_детали : integer [1..5]
$Body
  {Номер_станка}
  {1      2      3      }
  {Номер_детали}
  {1}          5      15      10
  {2}          5      15      10
  {3}          5      15      10
  {4}          5      15      10
  {5}          5      15      10
$End
```

```
$Function Изменить_позицию : such_as Детали.Позиция = Накопитель_входной
$Type = table
$Parameters
  Стадия_обработки: integer[1..13]
$Body
  Накопитель_входной      {Стадия_обработки = 1      }
  Робот Станок_1 Робот    {Стадия_обработки = 2..4  }
  Накопитель_промежуточный_1 {Стадия_обработки = 5      }
  Робот Станок_2 Робот    {Стадия_обработки = 6..8  }
  Накопитель_промежуточный_2 {Стадия_обработки = 9      }
  Робот Станок_3 Робот    {Стадия_обработки = 10..12}
  Накопитель_выходной      {Стадия_обработки = 13    }
$End
```

В случае необходимости описания различных маршрутов обработки для разных деталей, надо будет описывать подобную функцию для каждой детали. Так же подобное описание делает невозможным изменение маршрута обработки и времени обработки детали в процессе прогона детали.

Решением данной проблемы является использование массивов, с их помощью маршрут обработки детали может быть описан как параметр детали, что существенно увеличивает гибкость данной модели.



## 2. Концептуальный этап проектирования

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.

### 2.1. Диаграмма компонентов

Для отображения зависимости между компонентами системы РДО и выделения среди них модернизируемых служит соответствующая диаграмма в нотации UML[6].

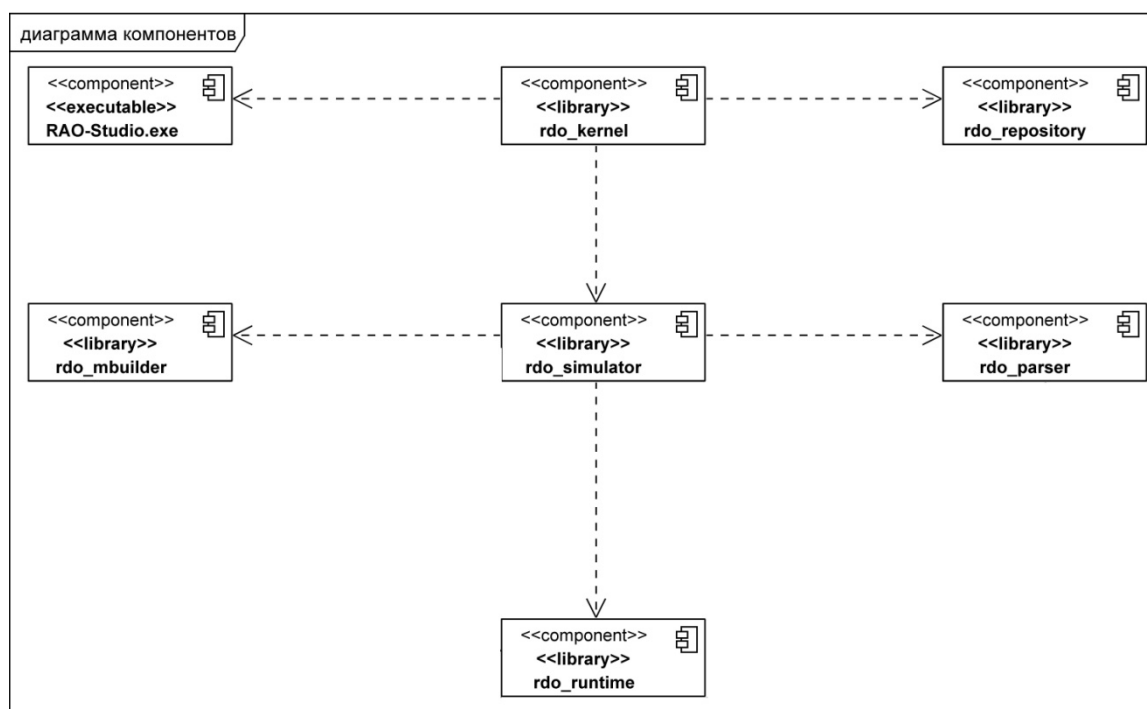


Рис. 4. Упрощенная диаграмма компонентов.

Базовый функционал представленных на диаграмме компонентов (Рис. 4):

rdo\_kernel реализует ядровые функции системы. Не изменяется при разработке системы.

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

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

rdo\_mbuidler 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 инициализируются 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.

## 2.2. Структура логического вывода РДО

Логический вывод системы РДО представляет собой алгоритм, который определяет какое событие в моделируемой системе должно произойти следующим в процессе имитации работы системы.

Во время имитации работы модели в системе существует одна МЕТА-логика. Она является контейнером для хранения разных логик. Сами логики являются контейнерами, в которых хранятся различные атомарные операции (например, нерегулярные события и правила). Таким образом статическое представление БЗ модели на РДО представляет собой трехуровневое дерево, корнем которого является МЕТА-логика, а листьями - атомарные операции.

Интересно отметить, что реализация описанной структуры с помощью наследования – одного из основных механизмов объектно-ориентированного программирования – делает возможным на уровне логики работы РДО рекурсивное вложение логик внутри логик. То есть архитектура имитатора РДО (rdo\_runtime) не запрещает наличие точек принятия решений внутри точек принятия решений с любой глубиной вложенности.

Поиск активности, которая должна быть запущена следующей, начинается с обращения класса RDOSimulator к своему атрибуту m\_logics, в котором хранится описанная выше МЕТА-логика. Далее от корня дерева к листьям распространяется волна вызовов метода onCheckCondition(). Т.е. onCheckCondition() вызывается у МЕТА-логики, затем циклически у ее логик, и наконец, циклически проверяются все атомарные операции каждой логики. Как только найдена активность, которая может быть выполнена, происходит ее кэширование (запоминание) внутри логики и кэширование самой логики внутри МЕТА-логики. После этого управление снова передается в RDOSimulator и найденная активность выполняется.

Для управления поиском очередной активности с помощью приоритетов точек принятия решений необходимо отсортировать список логик внутри МЕТА-логики по убыванию приоритета и в дальнейшем производить поиск в отсортированном списке.

## **2.3.Техническое задание**

### **2.3.1.Введение**

РДО – язык имитационного моделирования, включающий все три основных подхода описания дискретных систем: процессный, событийный и сканирования активностей. В системе РДО разрабатывается новый тип данных - массивы. Начал работу по этой теме студент кафедры РК-9 Чирков Михаил Михайлович. Основной разработчик РДО – кафедра РК-9, МГТУ им. Баумана.

### **2.3.2.Основания для разработки**

1) Разработка ведется на основании следующих документов:

- Задание на выполнение курсового проекта.

2) Документы утверждены « 24 » июня 2010 года.

3) Тема курсового проекта: Добавление массивов в язык имитационного моделирования РДО.

### **2.3.3.Назначение разработки**

Основная цель данного курсового проекта – реализовать синтаксис описания массивов в языке РДО, а так же создать классы описывающие массивы во внутренней структуре RAO Studio.

### **2.3.4.Требования к программе или программному изделию**

#### **2.3.4.1.Требования к функциональным характеристикам**

Требования к составу выполняемых функций реализуемой системы заключаются в возможности использования типа данных «массив» при моделировании в системе РДО. Система моделирования РДО должна выполнять следующие функции:

- Поддержка синтаксиса описания массивов при описании типов ресурсов и при описании ресурсов.

- Создание типа данных «массив» не ограниченной размерности.

- Создание ресурсов типа «массив» не ограниченной размерности.

- Проверка синтаксической верности описания массивов.

- Ввод массивов при анимации модели в текстовом виде.

#### **2.3.4.1.Требования к надежности**

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

#### **2.3.4.1.Условия эксплуатации**

Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости.

Аппаратные средства должны эксплуатироваться в помещениях с

выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

#### **2.3.4.1. 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;

#### **2.3.4.1. Требования к информационной и программной**

##### **совместимости**

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

#### **2.3.4.1. Требования к маркировке и упаковке**

Не предъявляются.

#### **2.3.4.1. Требования к транспортированию и хранению**

Не предъявляются.

#### **2.3.5. Требования к программной документации**

Не предъявляются.

#### **2.3.6. Стадии и этапы разработки**

- Предпроектное исследование .
- Концептуальный этап проектирования.
- Технический этап проектирования.
- Рабочий этап проектирования.

#### **2.3.7. Порядок контроля и приемки**

Контроль и приемка поддержки массивов должны

осуществляться в процессе проверки функциональности (апробирования)

системы имитационного моделирования на тестовом примере модели в соответствии с требованиями к функциональным характеристикам системы.

### 3. Технический этап проектирования

#### 3.1. Разработка синтаксиса описания типа ресурса

**Типы ресурсов** определяют структуру глобальной базы данных программы (модели) и их описывают в отдельном объекте (имеет расширение **.rtp**).

Описание каждого типа ресурса имеет следующий формат:

```
$Resource_type<имя_типа>:<вид_ресурсов>  
$Parameters  
<описание_параметра>{< описание_параметра >}  
$End
```

*имя\_типа*

Имя типа представляет собой простое имя. Имена типов должны быть различными для всех типов и не должны совпадать с предопределенными и ранее использованными именами.

*вид\_ресурсов*

Вид ресурсов данного типа может быть одним из следующих:

**permanent:** Постоянные ресурсы; ресурсы этого вида всегда присутствуют в модели, они не могут быть уничтожены или созданы во время прогона

**temporary:** Временные ресурсы; ресурсы этого вида могут во время прогона создаваться и уничтожаться при выполнении операций, правил и совершении нерегулярных событий

*описание\_параметра*

Описание параметра ресурса имеет формат:

```
< имя_параметра>:< тип_параметра >[=< значение_по_умолчанию>]
```

*имя\_параметра*

Имя параметра - это **простое имя**. Имена параметров должны быть различными для всех параметров данного типа и не должны совпадать 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*. Ссылки возможны на параметры ранее описанных типов ресурсов и на ранее описанные параметры данного типа ресурсов.

*значение\_по\_умолчанию*

Для параметра любого типа может быть задано значение по умолчанию. Это значение указывают после знака равенства целой или вещественной численной константой, либо именем значения для перечислимого параметра. При указании типа ссылкой также возможно задание значения по умолчанию. При этом задаваемое значение может отличаться от значения по умолчанию того параметра, на тип которого проводится ссылка.

*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**.

### 3.2.Разработка архитектуры компонента **rdo\_parser**

Для возможности обработки новой конструкции в коде модели требуют изменений лексический и синтаксический анализаторы РДО, также требуется добавить классы [7] поддерживающие массивы.

### 3.3.Разработка архитектуры компонента **rdo\_runtime**

В пространстве имен **rdoRuntime** необходимо добавить классы описывающие массивы.

## 4. Рабочий этап проектирования

### 4.1. Синтаксический анализ типов данных

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 и нетермальные символы в грамматический анализатор.

В лексическом анализаторе (flex) я добавил новый токен *RDO\_array*, который может быть записан единственным способом:

```
array                                return(RDO_array);
```

Этот токен необходимо также добавить в генератор синтаксического анализатора (bison):

```
%token RDO_array                    379
```

Далее необходимо добавить описание типа массива и описание значения по умолчанию:

```
param_type_array param_value_default
{
    LPRDOArrayType pArray = PARSER->stack().pop<RDOArrayType>($1);
    LPRDOTypeParam pType = rdo::Factory<RDOTypeParam>::create(pArray, RDOVALUE($2),
RDOParserSrcInfo(@1, @2));
    $$ = PARSER->stack().push(pType);
};

param_type_array:  RDO_array '<' param_type '>'
{
    LPRDOTypeParam pParamType = PARSER->stack().pop<RDOTypeParam>($3);
    ASSERT(pParamType);
    LPRDOArrayType pArray = rdo::Factory<RDOArrayType>::create(pParamType->type(),
RDOParserSrcInfo(@1, @4));
    $$ = PARSER->stack().push(pArray);
};

param_value_default:      /* empty */
{
    $$ = (int) PARSER->addValue(new rdoParse::RDOValue());
}
| '=' param_value
{
    $$ = $2;
}
| '=' error
{
    RDOParserSrcInfo src_info(@1, @2, true);
    if (src_info.src_pos().point())
    {
        PARSER->error().error(src_info, _T("Не указано значение по-умолчанию"));
    }
}
```

```

}
else

{
PARSER->error().error(src_info, _T("Неверное значение по-умолчанию"));
}
};

param_value: RDO_INT_CONST      {
                                $$ = $1;
                                }
          | RDO_REAL_CONST {
                                $$ = $1;
                                }
          | RDO_STRING_CONST {
                                $$ = $1;
                                }
          | RDO_IDENTIF {
                                $$ = $1;
                                }
          | RDO_BOOL_CONST {
                                $$ = $1;
                                }
          | param_array_value {
                                $$ = $1;
                                };

param_array_value: '[' array_item ']' {
LPRDOArrayValue pArrayValue = PARSER->stack().pop<RDOArrayValue>($2);
ASSERT(pArrayValue);
$$ = (int)PARSER->addValue(new RDOValue(pArrayValue->getRArray(),
pArrayValue->getArrayType(), RDOParserSrcInfo(@2)));
}
| '[' array_item error {
PARSER->error().error(@2, _T("Массив должен закрываться скобкой"));
};

array_item:      param_value {
LPRDOArrayType pArrayType =
rdo::Factory<RDOArrayType>::create(RDOVALUE($1).type(), RDOParserSrcInfo(@1));
ASSERT(pArrayType);
LPRDOArrayValue pArrayValue = rdo::Factory<RDOArrayValue>::create(pArrayType);
pArrayValue->insertItem(RDOVALUE($1));
$$ = PARSER->stack().push(pArrayValue);
}
| array_item ',' param_value {
LPRDOArrayValue pArrayValue = PARSER->stack().pop<RDOArrayValue>($1);

```



```

ASSERT(pArrayValue);
pArrayValue->insertItem(RDOVALUE($3));
$$ = PARSER->stack().push(pArrayValue);
}
|array_item param_value {
LPRDOArrayValue pArrayValue = PARSER->stack().pop<RDOArrayValue>($1);
ASSERT(pArrayValue);
pArrayValue->insertItem(RDOVALUE($2));
$$ = PARSER->stack().push(pArrayValue);
PARSER->error().warning(@1, rdo::format(_T("Пропущена запятая перед: %s"),
    RDOVALUE($2)->getAsString().c_str()));
};

```

Из этого кода можно сделать вывод, что типом данных содержащихся в массиве могут быть массивы. Это даёт возможность создавать многомерные массивы неограниченной вложенности.

## 4.2.Изменения в пространстве имен rdoRuntime

### Объявление класса RDOArrayType

Класс предназначен для описания типа массива и содержит в себе информацию о типах элементов содержащихся в массивах.

```

class RDOArrayType: public RDOType, public RDORuntimeObject
{
public:
    typedef CREF(RDOType) ArrayType;

    RDOArrayType(PTR(RDORuntimeParent) parent, ArrayType pArrayType);

    ArrayType getArrayType() const;

private:
    ArrayType m_pArrayType;
};

```

- RDOArrayType(PTR(RDORuntimeParent) parent, ArrayType pArrayType)

Конструктор класса, функция предназначенная для инициализации объектов класса.

- ArrayType getArrayType() const

Функция возвращающая переменную m\_pArrayType, описывающую тип элементов содержащихся в массиве.

## Объявление класса RDOArrayValue

Класс предназначен для описания массива и содержит в себе информацию о типе массива и элементы массива.

```
class RDOArrayValue
{
public:
    typedef std::vector<RDOValue> Container;

    RDOArrayValue(CREF(RDOArrayType) type);
    RDOArrayValue(CREF(RDOArrayValue) value);
    ~RDOArrayValue();

    CREF(RDOArrayType) type() const;

    void insertItem(CREF(RDOValue) pArray);
    Container::iterator m_containerBegin();
    Container::iterator m_containerEnd();

    tstring getAsString() const;

private:
    Container m_Container;
    CPtr(RDOArrayType) m_arrayType;
};
```

- RDOArrayValue(CREF(RDOArrayType) type)

Конструктор класса, создает новый объект класса на основе типа массива.

- RDOArrayValue(CREF(RDOArrayValue) value)

Конструктор класса, создает новый объект класса на основе другого массива.

- ~RDOArrayValue()

Деструктор класса, функция, освобождающая память выделенную конструктором.

- void insertItem(CREF(RDOValue) pArray)

Функция, добавляющая элемент в конец массива.

- Container::iterator m\_containerBegin()

Функция, возвращающая итератор указывающий на первый элемент массива.

- Container::iterator m\_containerEnd()

Функция, возвращающая итератор указывающий на элемент следующий за последним

- tstring getAsString() const

Функция, возвращающая массив в виде строки.

## 4.3.Изменения в пространстве имен rdoParser

### Объявление класса RDOArrayType

Класс предназначен для описания типа массива и содержит в себе информацию о положении описания типа массива в тексте модели и типе элементов содержащихся в массиве.

```
class RDOArrayType: public RDOType, public rdo::smart_ptr_counter_reference,
public RDOParserSrcInfo
{
DECLARE_FACTORY(RDOArrayType);
friend class RDOArrayValue;

public:
    CREF(LPRDOType) getItemType() const;

private:
    RDOArrayType (CREF(LPRDOType) pType, CREF(RDOParserSrcInfo) src_info);
    virtual ~RDOArrayType();

    PTR(rdoRuntime::RDOArrayType) __array() const
    {
        return static_cast<PTR(rdoRuntime::RDOArrayType)>(/
(const_cast<PTR(rdoRuntime::RDOType)>(m_type));
    }

    LPRDOType m_pType;

    DECLARE_IType;
    DECLARE_IModelStructure;
};
```

- CREF(LPRDOType) getItemType() const

Функция, возвращающая тип элементов массива.

- RDOArrayType (CREF(LPRDOType) pType, CREF(RDOParserSrcInfo) src\_info)

Конструктор класса, создающий новый объект класса на основе типа элементов и информации о месте описания типа в модели.

- virtual ~RDOArrayType()

Деструктор класса, функция, освобождающая память выделенную конструктором.

- PTR(rdoRuntime::RDOArrayType) \_\_array() const

Функция,возвращает объект класса типа массива из пространства имён rdoRuntime.

## Объявление класса RDOArrayValue

Класс предназначен для описания массива и содержит информацию о типе массива и элементы массива.

```
class RDOArrayValue
{
    DECLARE_FACTORY(RDOArrayValue);
public:
    RDOArrayValue          (LPRDOArrayType pArrayType);

    void insertItem(CREF(RDOValue)    value);
    CREF(LPRDOArrayType) getArrayType() const;
    rdoRuntime::RDOValue getRArray    () const;

private:
    typedef std::vector<RDOValue> Container;

    virtual ~RDOArrayValue();

    Container      m_Container;
    LPRDOArrayType m_arrayType;
};
DECLARE_POINTER(RDOArrayValue);
```

- RDOArrayValue(LPRDOArrayType pArrayType)

Конструктор класса, создающий объект класса на основе типа массива.

- void insertItem(CREF(RDOValue) value);

Функция, добавляющая элемент в конец массива.

- CREF(LPRDOArrayType) getArrayType() const

Функция, возвращающая тип массива.

- rdoRuntime::RDOValue getRArray () const

Функция, создающая объект класса RDOArrayValue в пространстве имён rdoRuntime и возвращающая данный объект.

## **Заключение**

В рамках данного курсового проекта были получены следующие результаты:

- 1) Проведено предпроектное исследование системы имитационного моделирования РДО и сформулированы предпосылки создания в системе типа данных «массив»
- 2) На этапе концептуального проектирования системы с помощью диаграммы компонентов нотации UML укрупнено, показано внутреннее устройство РДО и выделены те компоненты, которые потребуют внесения изменений в ходе этой работы. Разработаны функциональные диаграммы (в нотации IDEF0) процесса компиляции RAO-Studio
- 3) На этапе технического проектирования разработан новый синтаксис для описания типа данных «массив», который представлен на синтаксической диаграмме. С помощью диаграммы классов разработана архитектура новой системы.
- 4) На этапе рабочего проектирования написан программный код для реализации спроектированной архитектуры компонентов `rdo_parser` и `rdo_runtime` системы РДО. Проведены отладка и тестирование новой системы, в ходе которых исправлялись найденные ошибки.

Поставленная цель курсового проекта достигнута.

## **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. Грейди Буч, Джеймс Рамбо, Айвар Джекобсон Язык UML. Руководство пользователя = The Unified Modeling Language user guide. — 2. — М., СПб.: «ДМК Пресс», «Питер», 2004. — 432 с
7. Бьерн Страуструп. Язык моделирования C++. Специальное издание. Пер. с англ. — М.: ООО «Бином-пресс», 2007 г. — 1104 с.: ил.

## **Список использованного программного обеспечения**

1. RAO-Studio.
2. Visual Paradigm © , Visual Paradigm for UML.
3. iGrafx ©, iGrafx.
4. АСКОН ©, Компас3D V11.
5. Microsoft ©, Microsoft office.

## Приложение 1. Модель гибкой производственной ячейки на языке РДО

Mymodel.pat (Образцы):

\$Pattern Установка\_на\_станок : operation trace

\$Parameters

Станок\_номер : such\_as Детали.Позиция

\$Relevant\_resources

Станок : Станки Keep Keep

Деталь : Детали Keep Keep

Накопитель : Накопители Keep Keep

Робот : Роботы Keep Keep

\$Time = Время\_погрузки

\$Body

Станок

Choice from Станок.Состояние = Свободен  
and Станок.Положение = Станок\_номер

Convert\_begin

Состояние set Загружается

Convert\_end

Состояние set Готов\_к\_работе

Деталь

Choice from Деталь.Состояние = Хранится  
and Деталь.Стадия\_обработки < 13  
and Изменить\_позицию(Деталь.Стадия\_обработки + 2) = Станок\_номер

Convert\_begin

Стадия\_обработки set Деталь.Стадия\_обработки + 1

Позиция set Изменить\_позицию(Деталь.Стадия\_обработки)

Состояние set Устанавливается

Convert\_end

Стадия\_обработки set Деталь.Стадия\_обработки + 1

Позиция set Изменить\_позицию(Деталь.Стадия\_обработки)

Состояние set Установлена

Накопитель

Choice from Накопитель.Количество\_деталей > 0  
and Накопитель.Номер = Станок.Номер

Convert\_end

Количество\_деталей set Накопитель.Количество\_деталей - 1

Робот

Choice from Робот.Состояние = Свободен

Convert\_begin

Состояние set Загружает

Convert\_end

Состояние set Свободен

\$End

\$Pattern Обработка\_на\_станке : operation trace

\$Parameters

Станок\_номер : such\_as Детали.Позиция

\$Relevant\_resources

Станок : Станки Keep Keep

Деталь : Детали Keep Keep

\$Time = Время\_обработки(Станок.Номер, Деталь.Номер)

```

$Body
    Станок
        Choice from Станок.Состояние = Готов_к_работе
        Convert_begin
            Состояние set Работает
        Convert_end
            Состояние set Зокончил_обработку

    Деталь
        Choice from Деталь.Состояние = Установлена
            and Деталь.Позиция = Станок_номер
        Convert_begin
            Состояние set Обрабатывается
        Convert_end
            Состояние set Обработана
$End

$Pattern Разгрузка_станка : operation trace
$Parameters
    Станок_номер : such_as Детали.Позиция
$Relevant_resources
    Станок : Станки Keep Keep
    Деталь : Детали Keep Keep
    Накопитель : Накопители Keep Keep
    Робот : Роботы Keep Keep
$Time = Время_разгрузки
$Body
    Станок
        Choice from Станок.Состояние = Зокончил_обработку
        Convert_begin
            Состояние set Разгружается
        Convert_end
            Состояние set Свободен

    Деталь
        Choice from Деталь.Состояние = Обработана
            and Деталь.Позиция = Станок_номер
        Convert_begin
            Стадия_обработки set Деталь.Стадия_обработки + 1
            Позиция set Изменить_позицию(Деталь.Стадия_обработки)
            Состояние set Разгружается
        Convert_end
            Стадия_обработки set Деталь.Стадия_обработки + 1
            Позиция set Изменить_позицию(Деталь.Стадия_обработки)
            Состояние set Хранится

    Накопитель
        Choice from Накопитель.Количество_деталей >= 0
            and Накопитель.Номер = Станок.Номер + 1
        Convert_end
            Количество_деталей set Накопитель.Количество_деталей + 1

    Робот
        Choice from Робот.Состояние = Свободен
        Convert_begin
            Состояние set Разгружает
        Convert_end

```



## Состояние set Свободен

\$End

\$Pattern Работа\_таймера : irregular\_event trace

\$Relevant\_resources

Время : Время Keep

\$Time = Шаг\_таймера

\$Body

Время

Convert\_event

Количество set Время.Количество + Шаг\_таймера

\$End

## Mymodel.rtp (Типы ресурсов):

\$Resource\_type Детали : permanent

\$Parameters

Номер : integer

Состояние : (Хранится, Устанавливается, Установлена, Обрабатывается, Обработана, Разгружается)

Позиция : (Накопитель\_входной, Робот, Станок\_1, Накопитель\_промежуточный\_1, Станок\_2, Накопитель\_промежуточный\_2, Станок\_3, Накопитель\_выходной)

Стадия\_обработки : integer

\$End

\$Resource\_type Накопители : permanent

\$Parameters

Номер : integer

Количество\_деталей : integer

\$End

\$Resource\_type Станки : permanent

\$Parameters

Номер : integer

Состояние : (Свободен, Загружается, Готов\_к\_работе, Работает, Закончил\_обработку, Разгружается)

Положение : such\_as Детали.Позиция

\$End

\$Resource\_type Роботы : permanent

\$Parameters

Номер : integer

Состояние : (Свободен, Загружает, Разгружает)

\$End

\$Resource\_type Время : permanent

\$Parameters

Количество : real

\$End

## Mymodel.rss (Ресурсы):

\$Resources

Время : Время trace 0

Станок\_1 : Станки trace 1 Свободен Станок\_1

Станок\_2 : Станки trace 2 Свободен Станок\_2

Станок\_3 : Станки trace 3 Свободен Станок\_3

Робот : Роботы trace 1 Свободен

Накопитель\_входной : Накопители trace 1 5

Накопитель\_промежуточный\_1 : Накопители trace 2 0

Накопитель\_промежуточный\_2 : Накопители trace 3 0

Накопитель\_выходной : Накопители trace 4 0

Деталь\_1 : Детали 1 Хранится Накопитель\_входной 1

Деталь\_2 : Детали 2 Хранится Накопитель\_входной 1

Деталь\_3 : Детали 3 Хранится Накопитель\_входной 1

Деталь\_4 : Детали 4 Хранится Накопитель\_входной 1

Деталь\_5 : Детали 5 Хранится Накопитель\_входной 1

\$End

### Mymodel.opr (Операции):

\$Operations

Таймер : Работа\_таймера

Установка\_на\_станок\_1 : Установка\_на\_станок Станок\_1

Установка\_на\_станок\_2 : Установка\_на\_станок Станок\_2

Установка\_на\_станок\_3 : Установка\_на\_станок Станок\_3

Обработка\_на\_станке\_1 : Обработка\_на\_станке Станок\_1

Обработка\_на\_станке\_2 : Обработка\_на\_станке Станок\_2

Обработка\_на\_станке\_3 : Обработка\_на\_станке Станок\_3

Разгрузка\_станка\_1 : Разгрузка\_станка Станок\_1

Разгрузка\_станка\_2 : Разгрузка\_станка Станок\_2

Разгрузка\_станка\_3 : Разгрузка\_станка Станок\_3

\$End

### Mymodel.frm (Анимация):

\$Frame fram\_1

\$Back\_picture = <127 127 127> 800 800

Show

text [10, 5, 50, 25, <127 127 127>, <100 255 0>, 'Время:']

text [60, 5, 150, 25, <127 127 127>, <100 255 0>, 'Время.Количество']

text [10,25,350,25, <127 127 127>, <0 0 0>, 'Станок 1 в состоянии:']

text [350,25,350,25, <127 127 127>, <0 0 0>, Станок\_1.Состояние]

text [10,40,350,25, <127 127 127>, <0 0 0>, 'Станок 2 в состоянии:']

text [350,40,350,25, <127 127 127>, <0 0 0>, Станок\_2.Состояние]

text [10,55,350,25, <127 127 127>, <0 0 0>, 'Станок 3 в состоянии:']

text [350,55,350,25, <127 127 127>, <0 0 0>, Станок\_3.Состояние]

text [10,85,350,25, <127 127 127>, <0 0 0>, 'Робот в состоянии:']

text [350,85,350,25, <127 127 127>, <0 0 0>, Робот.Состояние]

text [10,115,350,25, <127 127 127>, <0 0 0>, 'Кол-во детали в входном накопителе:']

text [350,115,350,25, <127 127 127>, <0 0 0>, Накопитель\_входной.Количество\_деталей]

```

text [10,130,350,25,<127 127 127>,<0 0 0>,'Кол-во детали в промежуточном накопителе
№1:']
text [350,130,350,25,<127 127 127>,<0 0 0>,
Накопитель_промежуточный_1.Количество_деталей]
text [10,145,350,25,<127 127 127>,<0 0 0>,'Кол-во детали в промежуточном накопителе
№2']
text [350,145,350,25,<127 127 127>,<0 0 0>,
Накопитель_промежуточный_2.Количество_деталей]
text [10,160,350,25,<127 127 127>,<0 0 0>,'Кол-во детали в выходном накопителе:']
text [350,160,350,25,<127 127 127>,<0 0 0>,Накопитель_выходной.Количество_деталей]

text [10,190,350,25,<127 127 127>,<0 0 0>,'Деталь 1:']
text [250,190,350,25,<127 127 127>,<0 0 0>,Деталь_1.Состояние]
text [380,190,350,25,<127 127 127>,<0 0 0>,Деталь_1.Позиция]

text [10,205,350,25,<127 127 127>,<0 0 0>,'Деталь 2:']
text [250,205,350,25,<127 127 127>,<0 0 0>,Деталь_2.Состояние]
text [380,205,350,25,<127 127 127>,<0 0 0>,Деталь_2.Позиция]

text [10,220,350,25,<127 127 127>,<0 0 0>,'Деталь 3:']
text [250,220,350,25,<127 127 127>,<0 0 0>,Деталь_3.Состояние]
text [380,220,350,25,<127 127 127>,<0 0 0>,Деталь_3.Позиция]

text [10,235,350,25,<127 127 127>,<0 0 0>,'Деталь 4:']
text [250,235,350,25,<127 127 127>,<0 0 0>,Деталь_4.Состояние]
text [380,235,350,25,<127 127 127>,<0 0 0>,Деталь_4.Позиция]

text [10,250,350,25,<127 127 127>,<0 0 0>,'Деталь 5:']
text [250,250,350,25,<127 127 127>,<0 0 0>,Деталь_5.Состояние]
text [380,250,350,25,<127 127 127>,<0 0 0>,Деталь_5.Позиция]

```

\$End

## Мymodel.fun (Константы, последовательности, функции)

\$Constant

Время\_погрузки : real = 5

Время\_разгрузки : real = 5

Шаг\_таймера : real = 0.5

\$End

\$Function Время\_обработки : integer[0..30]

\$Type = table

\$Parameters

Номер\_станка : integer [1..3]

Номер\_детали : integer [1..5]

\$Body

{Номер\_станка}

{1 2 3 }

{Номер\_детали}

{1} 5 15 10

{2} 5 15 10

{3} 5 15 10

{4} 5 15 10

{5} 5 15 10

\$End

```

$Function Изменить_позицию : such_as Детали.Позиция = Накопитель_входной
$Type = table
$Parameters
    Стадия_обработки: integer[1..13]
$Body
    Накопитель_входной      {Стадия_обработки = 1    }
    Робот Станок_1 Робот    {Стадия_обработки = 2..4 }
    Накопитель_промежуточный_1 {Стадия_обработки = 5    }
    Робот Станок_2 Робот    {Стадия_обработки = 6..8 }
    Накопитель_промежуточный_2 {Стадия_обработки = 9    }
    Робот Станок_3 Робот    {Стадия_обработки = 10..12}
    Накопитель_выходной      {Стадия_обработки = 13  }
$End

```

**mymodel.smr (Прогон):**

```
Model_name = mymodel
```

```
Resource_file = mymodel
```

```
Oprlev_file = mymodel
```

```
Statistic_file = mymodel
```

```
Results_file = mymodel
```

```
Trace_file = mymodel
```

```
Frame_file = mymodel
```

```
Frame_number = 1
```

```
Show_mode = Animation
```

```
Show_rate = 1000.0
```

```
Terminate_if Накопитель_выходной.Количество_деталей = 5
```

## Приложение 2. Полный синтаксический анализ описания типа данных (rdortp.y).

```
namespace rdoParse
{
%}

%start rtp_list

%%

rtp_list:                                /* empty */
| rtp_list rtp_res_type
| error {
    PARSE->error( "Ожидается ключевое слово
$Resource_type" );
};

rtp_res_type:                            rtp_header RDO_Parameters rtp_body RDO_End
{
    RDORTPResType* res_type =
reinterpret_cast<RDORTPResType*>($1);
    if ( res_type->getParams().empty() )
    {
        PARSE->warning( @2, rdo::format( "Тип
ресурса '%s' не содержит параметров", res_type->name().c_str() ) );
    }
| rtp_header RDO_Parameters rtp_body {
    PARSE->error( @2, "Не найдено ключевое слово
$End" );
}
| rtp_header error {
    PARSE->error( @2, "Не найдено ключевое слово
$Parameters" );
};

rtp_header:                             RDO_Resource_type RDO_IDENTIF_COLON rtp_vid_res
{
    LEXER->m_enum_param_cnt = 0;
    RDOValue* type_name =
reinterpret_cast<RDOValue*>($2);
    std::string name = type_name-
>value().getIdentificator();
    const RDORTPResType* _rtp = PARSE-
>findRTPResType( name );
    if ( _rtp ) {
        PARSE->error_push_only( type_name-
>src_info(), rdo::format("Тип ресурса уже существует: %s", name.c_str()) );
        PARSE->error_push_only( _rtp->src_info(),
"См. первое определение" );
        PARSE->error_push_done();
    }
    RDORTPResType* rtp = new RDORTPResType( PARSE,
type_name->src_info(), $3 != 0 );
    $$ = (int)rtp;
}
| RDO_Resource_type RDO_IDENTIF_COLON error {
    PARSE->error( @2, "Не указан вид ресурса" );
}
| RDO_Resource_type error {
    std::string str( LEXER->YYText() );
    PARSE->error( @2, rdo::format("Ошибка в
описании имени типа ресурса: %s", str.c_str()) );
};

rtp_vid_res:                             RDO_permanent { $$ = 1; }
| RDO_temporary { $$ = 0; };
```

```

rtp_body:          /* empty */ {
                    }
                    | rtp_body rtp_param {
                        RDORTPPParam* param =
reinterpret_cast<RDORTPPParam*>($2);
                        PARSER->getLastRTPResType()->addParam( param );
                    };

rtp_param:          RDO_IDENTIF_COLON param_type fuzzy_terms_list
                    {
                        RDOValue*          param_name =
reinterpret_cast<RDOValue*>($1);
                        RDORTPPParamType*  param_type =
reinterpret_cast<RDORTPPParamType*>($2);
                        RDORTPFuzzyTermsSet* terms_set =
reinterpret_cast<RDORTPFuzzyTermsSet*>($3);
                        if ( terms_set->empty() )
                        {
                            RDORTPPParam* param = new RDORTPPParam(
PARSER->getLastRTPResType(), param_name->src_info(), param_type );
                            param_type->reparent( param );
                            if ( param_type->typeID() ==
rdoRuntime::RDOTYPE::t_enum ) {

                                static_cast<RDORTPEnumParamType*>(param_type)->enum_name = rdo::format(
"%s.%s", PARSER->getLastRTPResType()->name().c_str(), param_name-
>src_info().src_text().c_str() );

                                }
                                $$ = (int)param;
                            }
                            else
                            {
                                RDORTPFuzzyParam* param = new
RDORTPFuzzyParam( PARSER, param_name->src_info(), terms_set );
                                param_type->reparent( param );
                                $$ = (int)param;
                            }
                        }
                        | RDO_IDENTIF_COLON error {
                            if ( PARSER->lexer_loc_line() == @1.last_line )
{
                                std::string str( LEXER->YYText() );
                                PARSER->error( @2, rdo::format( "Неверный
тип параметра: %s", str.c_str() ) );
                            } else {
                                PARSER->error( @1, "Ожидается тип
параметра" );
                            }
                        }
                        | error {
                            PARSER->error( @1, "Неправильное описание
параметра" );
                        };

fuzzy_terms_list:  /* empty */ {
                    RDORTPFuzzyTermsSet* terms_set = new
RDORTPFuzzyTermsSet( PARSER );
                    $$ = (int)terms_set;
                }
                | fuzzy_terms_list fuzzy_term {
                    RDORTPFuzzyTermsSet* terms_set =
reinterpret_cast<RDORTPFuzzyTermsSet*>($1);
                    RDORTPFuzzyTerm*      term      =
reinterpret_cast<RDORTPFuzzyTerm*>($2);
                    terms_set->add( term );
                    $$ = $1;
                };

```

```

        fuzzy_term:                                RDO_Fuzzy_Term RDO_IDENTIF {
                                                    RDOValue* param_name =
reinterpret_cast<RDOValue*>($2);
        //
        fuzzy_membershift_fun = reinterpret_cast<RDORTPFuzzyMembershiftFun*>($3);
        //
        RDORTPFuzzyTerm( PARSER, param_name->src_info(), fuzzy_membershift_fun );
        //
        fuzzy_term );
        //
        $$ = (int)fuzzy_term;
    };

    fuzzy_membershift_fun: /* empty */ {
        RDORTPFuzzyMembershiftFun* fun = new
RDORTPFuzzyMembershiftFun( PARSER );
        $$ = (int)fun;
    }
    | fuzzy_membershift_fun membershift_point {

        RDORTPFuzzyMembershiftFun* fun =
reinterpret_cast<RDORTPFuzzyMembershiftFun*>($1);
        RDORTPFuzzyMembershiftPoint* point =
reinterpret_cast<RDORTPFuzzyMembershiftPoint*>($2);
        fun->add( point );
        $$ = $1;
        //Задание функции принадлежности точками -
вершинами ломанных кривых
    };

    membershift_point: '(' RDO_REAL_CONST ',' RDO_REAL_CONST ')' {

        double x_value =
reinterpret_cast<RDOValue*>($2)->value().getDouble();
        double y_value =
reinterpret_cast<RDOValue*>($4)->value().getDouble();
        RDORTPFuzzyMembershiftPoint*
fuzzy_membershift_point = new RDORTPFuzzyMembershiftPoint( PARSER,
RDOParserSrcInfo( @1, @5 ), x_value, y_value);
        $$ = (int)fuzzy_membershift_point;
    }
    | '(' RDO_REAL_CONST ',' RDO_REAL_CONST ')' ',' {

        double x_value =
reinterpret_cast<RDOValue*>($2)->value().getDouble();
        double y_value =
reinterpret_cast<RDOValue*>($4)->value().getDouble();
        RDORTPFuzzyMembershiftPoint*
fuzzy_membershift_point = new RDORTPFuzzyMembershiftPoint( PARSER,
RDOParserSrcInfo( @1, @5 ), x_value, y_value);
        $$ = (int)fuzzy_membershift_point;
    }
    | '(' RDO_REAL_CONST ',' RDO_INT_CONST ')' {

        double x_value =
reinterpret_cast<RDOValue*>($2)->value().getDouble();
        double y_value =
reinterpret_cast<RDOValue*>($4)->value().getDouble();
        RDORTPFuzzyMembershiftPoint*
fuzzy_membershift_point = new RDORTPFuzzyMembershiftPoint( PARSER,
RDOParserSrcInfo( @1, @5 ), x_value, y_value);
        $$ = (int)fuzzy_membershift_point;
    }
    | '(' RDO_REAL_CONST ',' RDO_INT_CONST ')' ',' {

        double x_value =
reinterpret_cast<RDOValue*>($2)->value().getDouble();

```

```

double y_value =
reinterpret_cast<RDOValue*>($4)->value().getDouble();
RDORTPFuzzyMembershiftPoint*
fuzzy_membershift_point = new RDORTPFuzzyMembershiftPoint( PARSER,
RDOParserSrcInfo( @1, @5 ), x_value, y_value);
$$ = (int)fuzzy_membershift_point;
};

// -----
// ----- Описание типа параметра
// -----
param_type:
    RDO_integer param_int_diap param_int_default_val
    {
        RDORTPIntDiap*      diap =
reinterpret_cast<RDORTPIntDiap*>($2);
        RDORTPDefVal*      dv =
reinterpret_cast<RDORTPDefVal*>($3);
        RDORTPIntParamType* rp = new RDORTPIntParamType(
PARSER->getLastParsingObject(), diap, dv, RDOParserSrcInfo( @1, @3 ) );
        $$ = (int)rp;
    }
    | RDO_real param_real_diap param_real_default_val
    {
        RDORTPRealDiap*      diap =
reinterpret_cast<RDORTPRealDiap*>($2);
        RDORTPDefVal*      dv =
reinterpret_cast<RDORTPDefVal*>($3);
        RDORTPRealParamType* rp = new RDORTPRealParamType(
PARSER->getLastParsingObject(), diap, dv, RDOParserSrcInfo( @1, @3 ) );
        $$ = (int)rp;
    }
    | RDO_string param_string_default_val
    {
        RDORTPDefVal*      dv =
reinterpret_cast<RDORTPDefVal*>($2);
        RDORTPStringParamType* rp = new RDORTPStringParamType(
PARSER->getLastParsingObject(), dv, RDOParserSrcInfo( @1, @2 ) );
        $$ = (int)rp;
    }
    | RDO_bool param_bool_default_val
    {
        RDORTPDefVal*      dv =
reinterpret_cast<RDORTPDefVal*>($2);
        RDORTPBoolParamType* rp = new RDORTPBoolParamType(
PARSER->getLastParsingObject(), dv, RDOParserSrcInfo( @1, @2 ) );
        $$ = (int)rp;
    }
    | param_enum param_enum_default_val
    {
        LEXER->m_enum_param_cnt = 0;
        RDORTPEnum* enu = reinterpret_cast<RDORTPEnum*>($1);
        RDORTPDefVal* dv =
reinterpret_cast<RDORTPDefVal*>($2);
        if ( dv->isExist() )
        {
            enu->findEnumValueWithThrow( dv-
>value().src_pos(), dv->value().value().getAsString() ); // Если не найдено, то
будет сообщение об ошибке, т.е. throw
        }
        RDORTPEnumParamType* rp = new RDORTPEnumParamType(
PARSER->getLastParsingObject(), enu, dv, RDOParserSrcInfo( @1, @2 ) );
        $$ = (int)rp;
    }
    | param_array /*param_array_default_val*/
    {
        PARSE->warning( @1, rdo::format("create array Done.
Dimension of array: %u" ,LEXER->m_array_param_cnt));
        PARSE->error(@1, "OK");
    }

```



```

LEXER->m_array_param_cnt = 0;
}
| param_such_as
{
    const RDORTPPParam* param =
reinterpret_cast<RDORTPPParam*>($1);
    RDOParserSrcInfo src_info( @1 );
    src_info.setSrcText( "such_as " + (param->getResType()
? param->getResType()->name() + "." : "") + param->name() );
    $$ = (int)param->getType()->constructorSuchAs(
src_info );
}
| param_such_as '=' RDO_INT_CONST
{
    const RDORTPPParam* param =
reinterpret_cast<RDORTPPParam*>($1);
    RDOParserSrcInfo src_info( @1, @3 );
    src_info.setSrcText( "such_as " + (param->getResType()
? param->getResType()->name() + "." : "") + param->name() );
    $$ = (int)param->getType()->constructorSuchAs(
src_info, *reinterpret_cast<RDOValue*>($3) );
}
| param_such_as '=' RDO_REAL_CONST
{
    const RDORTPPParam* param =
reinterpret_cast<RDORTPPParam*>($1);
    RDOParserSrcInfo src_info( @1, @3 );
    src_info.setSrcText( "such_as " + (param->getResType()
? param->getResType()->name() + "." : "") + param->name() );
    $$ = (int)param->getType()->constructorSuchAs(
src_info, *reinterpret_cast<RDOValue*>($3) );
}
| param_such_as '=' RDO_IDENTIF
{
    const RDORTPPParam* param =
reinterpret_cast<RDORTPPParam*>($1);
    RDOParserSrcInfo src_info( @1, @3 );
    src_info.setSrcText( "such_as " + (param->getResType()
? param->getResType()->name() + "." : "") + param->name() );
    $$ = (int)param->getType()->constructorSuchAs(
src_info, *reinterpret_cast<RDOValue*>($3) );
}
| param_such_as '=' error
{
    PARSER->error( "Ожидается значение по-умолчанию" );
};

/*
| RDO_integer error {
    PARSER->error( @2, "Ошибка после ключевого слова
integer. Возможно, не хватает значения по-умолчанию." );
}
| RDO_real error {
    PARSER->error( @2, "Ошибка после ключевого слова real.
Возможно, не хватает значения по-умолчанию." );
}
| param_enum error {
    PARSER->error( @2, "Ошибка после перечислимого типа.
Возможно, не хватает значения по-умолчанию." );
};

*/
param_int_diap: /* empty */ {
    YYLTYPE pos = @0;
    pos.first_line = pos.last_line;
    pos.first_column = pos.last_column;
    RDORTPIntDiap* diap = new RDORTPIntDiap( PARSER, pos
);

    $$ = (int)diap;
}

```

```

        | '[' RDO_INT_CONST RDO_dblpoint RDO_INT_CONST ']' {
            RDORTPIntDiap* diap = new RDORTPIntDiap( PARSER,
reinterpret_cast<RDOValue*>($2)->value().getInt(), reinterpret_cast<RDOValue*>($4)-
>value().getInt(), RDOParserSrcInfo( @1, @5 ), @4 );
            $$ = (int)diap;
        }
        | '[' RDO_REAL_CONST RDO_dblpoint RDO_REAL_CONST {
указан вещественный" );
        }
        | '[' RDO_REAL_CONST RDO_dblpoint RDO_INT_CONST {
указан вещественный" );
        }
        | '[' RDO_INT_CONST RDO_dblpoint RDO_REAL_CONST {
указан вещественный" );
        }
        | '[' RDO_INT_CONST RDO_dblpoint RDO_INT_CONST error {
            PARSER->error( @4, "Диапазон задан неверно" );
        }
        | '[' RDO_INT_CONST RDO_dblpoint error {
            PARSER->error( @4, "Диапазон задан неверно" );
        }
        | '[' error {
            PARSER->error( @2, "Диапазон задан неверно" );
        };

    param_real_diap: /* empty */ {
        YYLTYPE pos = @0;
        pos.first_line = pos.last_line;
        pos.first_column = pos.last_column;
        RDORTPRealDiap* diap = new RDORTPRealDiap( PARSER, pos
);
        $$ = (int)diap;
    }
    | '[' RDO_REAL_CONST RDO_dblpoint RDO_REAL_CONST ']' {
        double min = reinterpret_cast<RDOValue*>($2)-
>value().getDouble();
        double max = reinterpret_cast<RDOValue*>($4)-
>value().getDouble();
        RDORTPRealDiap* diap = new RDORTPRealDiap( PARSER,
min, max, RDOParserSrcInfo( @1, @5 ), @4 );
        $$ = (int)diap;
    }
    | '[' RDO_REAL_CONST RDO_dblpoint RDO_INT_CONST ']' {
        double min = reinterpret_cast<RDOValue*>($2)-
>value().getDouble();
        double max = reinterpret_cast<RDOValue*>($4)-
>value().getDouble();
        RDORTPRealDiap* diap = new RDORTPRealDiap( PARSER,
min, max, RDOParserSrcInfo( @1, @5 ), @4 );
        $$ = (int)diap;
    }
    | '[' RDO_INT_CONST RDO_dblpoint RDO_REAL_CONST ']' {
        double min = reinterpret_cast<RDOValue*>($2)-
>value().getDouble();
        double max = reinterpret_cast<RDOValue*>($4)-
>value().getDouble();
        RDORTPRealDiap* diap = new RDORTPRealDiap( PARSER,
min, max, RDOParserSrcInfo( @1, @5 ), @4 );
        $$ = (int)diap;
    }
    | '[' RDO_INT_CONST RDO_dblpoint RDO_INT_CONST ']' {
        double min = reinterpret_cast<RDOValue*>($2)-
>value().getDouble();
        double max = reinterpret_cast<RDOValue*>($4)-
>value().getDouble();

```

```

RDORTPRealDiap* diap = new RDORTPRealDiap( PARSER,
min, max, RDOParserSrcInfo( @1, @5 ), @4 );
    $$ = (int)diap;
}
| '[' RDO_REAL_CONST RDO_dblpoint RDO_REAL_CONST error {
    PARSER->error( @4, "Диапазон задан неверно" );
}
| '[' RDO_REAL_CONST RDO_dblpoint RDO_INT_CONST error {
    PARSER->error( @4, "Диапазон задан неверно" );
}
| '[' RDO_INT_CONST RDO_dblpoint RDO_REAL_CONST error {
    PARSER->error( @4, "Диапазон задан неверно" );
}
| '[' RDO_INT_CONST RDO_dblpoint RDO_INT_CONST error {
    PARSER->error( @4, "Диапазон задан неверно" );
}
| '[' RDO_REAL_CONST RDO_dblpoint error {
    PARSER->error( @4, "Диапазон задан неверно" );
}
| '[' RDO_INT_CONST RDO_dblpoint error {
    PARSER->error( @4, "Диапазон задан неверно" );
}
| '[' error {
    PARSER->error( @2, "Диапазон задан неверно" );
};

param_int_default_val: /* empty */ {
    $$ = (int)new RDORTPDefVal(PARSER);
}
| '=' RDO_INT_CONST {
    $$ = (int)new RDORTPDefVal(PARSER,
*reinterpret_cast<RDOWalue*>($2) );
}
| '=' RDO_REAL_CONST {
    PARSER->error( @2, rdo::format("Целое число
инициализируется вещественным: %f", reinterpret_cast<RDOWalue*>($2)-
>value().getDouble()) );
}
| '=' error {
    RDOParserSrcInfo _src_info(@1, @2, true);
    if ( _src_info.src_pos().point() )
    {
        PARSER->error( _src_info, "Не указано
значение по-умолчанию для целого типа" );
    }
    else
    {
        PARSER->error( _src_info, "Неверное
значение по-умолчанию для целого типа" );
    }
};

param_real_default_val: /* empty */ {
    $$ = (int)new RDORTPDefVal(PARSER);
}
| '=' RDO_REAL_CONST {
    $$ = (int)new RDORTPDefVal(PARSER,
*reinterpret_cast<RDOWalue*>($2));
}
| '=' RDO_INT_CONST {
    $$ = (int)new RDORTPDefVal(PARSER,
*reinterpret_cast<RDOWalue*>($2));
}
| '=' error {
    RDOParserSrcInfo _src_info(@1, @2, true);
    if ( _src_info.src_pos().point() )
    {

```

```

                                PARSE->error( _src_info, "Не указано
значение по-умолчанию для вещественного типа" );
                                }
                                else
                                {
                                    PARSE->error( _src_info, "Неверное
значение по-умолчанию для вещественного типа" );
                                }
                                };

    param_string_default_val:      /* empty */
    {
        $$ = (int)new RDORTPDefVal (PARSE);
    }
    | '=' RDO_STRING_CONST
    {
        $$ = (int)new RDORTPDefVal (PARSE,
*reinterpret_cast<RDOWalue*>($2));
    }
    | '=' error
    {
        RDOParserSrcInfo _src_info(@1, @2, true);
        if ( _src_info.src_pos().point() )
        {
            PARSE->error( _src_info, "Не указано
значение по-умолчанию для строчного типа" );
        }
        else
        {
            PARSE->error( _src_info, "Неверное
значение по-умолчанию для строчного типа" );
        }
    }
    };

    param_bool_default_val: /* empty */
    {
        $$ = (int)new RDORTPDefVal (PARSE);
    }
    | '=' RDO_BOOL_CONST
    {
        $$ = (int)new RDORTPDefVal (PARSE,
*reinterpret_cast<RDOWalue*>($2));
    }
    | '=' error
    {
        RDOParserSrcInfo _src_info(@1, @2, true);
        if ( _src_info.src_pos().point() )
        {
            PARSE->error( _src_info, "Не указано
значение по-умолчанию для булевского типа" );
        }
        else
        {
            PARSE->error( _src_info, "Неверное
значение по-умолчанию для булевского типа" );
        }
    }
    };

    param_enum:  '(' param_enum_list ')' {
        RDORTPEnum* enu = reinterpret_cast<RDORTPEnum*>($2);
        enu->setSrcPos( @1, @3 );
        enu->setSrcText( enu->getEnums().asString() );
        $$ = $2;
    }
    | '(' param_enum_list error {
        PARSE->error( @2, "Перечисление должно заканчиваться
скобкой" );
    }
    };

```

```

    param_enum_list: RDO_IDENTIF {
        RDORTPEnum* enu = new RDORTPEnum( PARSER-
>getLastParsingObject(), *reinterpret_cast<RDOValue*>($1) );
        enu->setSrcInfo( reinterpret_cast<RDOValue*>($1)-
>src_info() );

        LEXER->m_enum_param_cnt = 1;
        $$ = (int)enu;
    }
    | param_enum_list ',' RDO_IDENTIF {
        if ( LEXER->m_enum_param_cnt >= 1 ) {
            RDORTPEnum* enu =
reinterpret_cast<RDORTPEnum*>($1);
            enu->add( *reinterpret_cast<RDOValue*>($3) );
            $$ = (int)enu;
        } else {
            PARSER->error( @3, "Ошибка в описании значений
перечислимого типа" );
        }
    }
    | param_enum_list RDO_IDENTIF {
        if ( LEXER->m_enum_param_cnt >= 1 ) {
            RDORTPEnum* enu =
reinterpret_cast<RDORTPEnum*>($1);
            enu->add( *reinterpret_cast<RDOValue*>($2) );
            $$ = (int)enu;
            PARSER->warning( @1, rdo::format("Пропущена
запятая перед: %s", reinterpret_cast<RDOValue*>($2)-
>value().getIdificator().c_str()) );
        } else {
            PARSER->error( @2, "Ошибка в описании значений
перечислимого типа" );
        }
    }
    | param_enum_list ',' RDO_INT_CONST {
        PARSER->error( @3, "Значение перечислимого типа не
может быть цифрой" );
    }
    | param_enum_list ',' RDO_REAL_CONST {
        PARSER->error( @3, "Значение перечислимого типа не
может быть цифрой" );
    }
    | param_enum_list RDO_INT_CONST {
        PARSER->error( @2, "Значение перечислимого типа не
может быть цифрой" );
    }
    | param_enum_list RDO_REAL_CONST {
        PARSER->error( @2, "Значение перечислимого типа не
может быть цифрой" );
    }
    | RDO_INT_CONST {
        PARSER->error( @1, "Значение перечислимого типа не
может начинаться с цифры" );
    }
    | RDO_REAL_CONST {
        PARSER->error( @1, "Значение перечислимого типа не
может начинаться с цифры" );
    }
};

param_enum_default_val: /* empty */ {
    $$ = (int)new RDORTPDefVal(PARSER);
}
| '=' RDO_IDENTIF {
    $$ = (int)new RDORTPDefVal(PARSER,
*reinterpret_cast<RDOValue*>($2));
}
| '=' error {
    RDOParserSrcInfo _src_info(@1, @2, true);

```

```

        if ( _src_info.src_pos().point() )
        {
            PARSE->error( _src_info, "Не указано
значение по-умолчанию для перечислимого типа" );
        }
        else
        {
            PARSE->error( _src_info, "Неверное
значение по-умолчанию для перечислимого типа" );
        }
    };

    param_array: RDO_array '<' param_array_type '>'
    {
        LEXER->m_array_param_cnt++;
    }

    param_array_type: RDO_integer
    {
        PARSE->warning(@1, "array integer");
    }
    |
    RDO_real
    {
        PARSE->warning(@1, "array real");
    }
    |
    RDO_string
    {
        PARSE->warning(@1, "array string");
    }
    |
    RDO_bool
    {
        PARSE->warning(@1, "array bool");
    }
    |
    param_such_as
    {
        PARSE->warning(@1, "array enum");
    }
    |
    param_array
    {
        PARSE->warning(@1, "array array");
    }
    };

    param_such_as:      RDO_such_as RDO_IDENTIF '.' RDO_IDENTIF {
        std::string type = reinterpret_cast<RDOValue*>($2)-
>value().getIdentificator();
        std::string param = reinterpret_cast<RDOValue*>($4)-
>value().getIdentificator();
        const RDORTPResType* const rt = PARSE-
>findRTPResType( type );
        if ( !rt ) {
            PARSE->error( @2, rdo::format("Ссылка на
неизвестный тип ресурса: %s", type.c_str()) );
        }
        const RDORTPPParam* const rp = rt->findRTPParam( param
);
        if ( !rp ) {
            PARSE->error( @4, rdo::format("Ссылка на
неизвестный параметр ресурса: %s.%s", type.c_str(), param.c_str()) );
        }
        $$ = (int)rp;
    }
    | RDO_such_as RDO_IDENTIF {

```

```

        std::string constName =
reinterpret_cast<RDOValue*>($2)->value().getIdentificator();
        const RDOFUNConstant* const cons = PARSE-
>findFUNConstant( constName );
        if ( !cons ) {
            PARSE->error( @2, rdo::format("Ссылка на
несуществующую константу: %s", constName.c_str()) );
        }
        $$ = (int)cons->getDescr();
    }
    | RDO_such_as RDO_IDENTIF '.' error {
        std::string type = reinterpret_cast<RDOValue*>($2)-
>value().getIdentificator();
        const RDORTPResType* const rt = PARSE-
>findRTPResType( type );
        if ( !rt ) {
            PARSE->error( @2, rdo::format("Ссылка на
неизвестный тип ресурса: %s", type.c_str()) );
        } else {
            PARSE->error( @4, "Ошибка при указании
параметра" );
        }
    }
    | RDO_such_as error {
        PARSE->error( @2, "После ключевого слова such_as
необходимо указать тип и параметер ресурса для ссылки" );
    };
// -----

```

### Приложение 3. Полный синтаксический анализ описания ресурса (rdorss.y).

```

/* ////////////////////////////////// GENERAL PART ////////////////////////////////// */
dptrtp_main:
| dptrtp_main RDO_Decision_point error RDO_End /* заглущка для
$Decision_point */
| dptrtp_main RDO_Activities error RDO_End /* заглущка для $Activities
*/
| dptrtp_main dpt_process_end;

/* ////////////////////////////////// PROCESS ////////////////////////////////// */

dpt_process:      dpt_process_header dpt_process_input;

dpt_process_header:      dpt_process_begin dpt_process_condition
dpt_process_prior;

dpt_process_begin: RDO_Process;

dpt_process_condition: /* empty */
| RDO_Condition error;

dpt_process_prior: /* empty */
| RDO_Priority error;

dpt_process_input: /* empty */
| dpt_process_input dpt_process_line
| dpt_process_input error dpt_process_line;

dpt_process_line:      RDO_IDENTIF { }
| RDO_GENERATE { }
| RDO_TERMINATE { }
| RDO_ADVANCE { }
| RDO_ASSIGN { }
| RDO_QUEUE dpt_queue_param { }
| RDO_QUEUE error { PARSE-
>error().error(@1, rdo::format("Ожидается имя ресурса для сбора статистики по
очереди")); }
| RDO_DEPART dpt_depart_param { }
| RDO_DEPART error { PARSE-
>error().error(@1, rdo::format("Ожидается имя ресурса для сбора статистики по
очереди")); }
/*| RDO_RELEASE dpt_release_param { }
| RDO_RELEASE error { PARSE-
>error().error(@1, rdo::format("Ожидается имя освобождаемого ресурса"));
}
| RDO_SEIZE dpt_seize_param { }
| RDO_SEIZE error { PARSE-
>error().error(@1, rdo::format("Ожидается имя занимаемого ресурса"));
}*/
| RDO_SEIZE dpt_seize_param { }
| RDO_SEIZE error { PARSE-
>error().error(@1, rdo::format("Ожидается список ресурсов, объединяемых в блок,
через запятую")); }
| RDO_RELEASE dpt_release_param { }
| RDO_RELEASE error { PARSE-
>error().error(@1, rdo::format("Ожидается список ресурсов, объединяемых в блок,
через запятую")); };

dpt_queue_param:      RDO_IDENTIF
{
// Имя ресурса
std::string res_name =
reinterpret_cast<RDOValue*>($1)->value().getIdificator().c_str();
const RDOParserSrcInfo& info = @1;
// Получили список всех ресурсов
rdoMBuilder::RDOResourceList rssList( PARSE-

```



```

// Получили список всех типов ресурсов
rdoMBuilder::RDOResTypeList rtpList( PARSER );
rdoMBuilder::RDOResType rtp;
std::string rtp_name = "QDEPART";
std::string q_name = "Очередь_" + res_name;
// Если ресурс существует, берем его тип и
проверяем

if (rssList[res_name].exist())
{
    rtp = rssList[res_name].getType();
    RDOPROCBlockForQueue::checkType(PARSER,
rtp, info);

    new RDOPMDWatchPar(PARSER,
RDOParserSrcInfo(q_name), 0, RDOParserSrcInfo(res_name),
RDOParserSrcInfo(std::string("длина_очереди")));
}
else
{
    //Если тип "QDEPART" существует
if ( rtpList[rtp_name].exist() )
{
    rdoMBuilder::RDOResType rtp_ =
rtpList[rtp_name];

    if(
RDOPROCBlockForQueue::checkType(PARSER, rtp_, info) )
    {

        RDOPROCBlockForQueue::createRes( PARSER, rtp_, res_name);
        new RDOPMDWatchPar(PARSER,
RDOParserSrcInfo(q_name), 0, RDOParserSrcInfo(res_name),
RDOParserSrcInfo(std::string("длина_очереди")));
    }
    else
    {
        rdoMBuilder::RDOResType rtp_ =
RDOPROCBlockForQueue::createType( PARSER, rtp_name, info );
        if
(RDOPROCBlockForQueue::checkType(PARSER, rtp_, info) )
        {

            RDOPROCBlockForQueue::createRes(PARSER, rtp_, res_name);
            new RDOPMDWatchPar(PARSER,
RDOParserSrcInfo(q_name), 0, RDOParserSrcInfo(res_name),
RDOParserSrcInfo(std::string("длина_очереди")));
        }
    }
}
| RDO_IDENTIF error
{
    PARSER->error().error( @1, "Ошибка в имени
ресурса" );
};

dpt_depart_param: RDO_IDENTIF
{
    // Имя ресурса
std::string res_name =
reinterpret_cast<RDOValue*>($1)->value().getIdificator().c_str();
const RDOParserSrcInfo& info = @1;
// Получили список всех ресурсов
rdoMBuilder::RDOResourceList rssList( PARSER );
// Получили список всех типов ресурсов
rdoMBuilder::RDOResTypeList rtpList( PARSER );
rdoMBuilder::RDOResType rtp;
std::string rtp_name = "QDEPART";
// Если ресурс существует, берем его тип и
проверяем

```

```

        if (rssList[res_name].exist())
        {
            rtp = rssList[res_name].getType();

RDOPROCBlockForQueue::checkType(PARSER, rtp, info);
        }
        else
        {
            if ( rtpList[rtp_name].exist() )
            {
                rdoMBuilder::RDOResType rtp_ =
rtpList[rtp_name];
                if(
RDOPROCBlockForQueue::checkType(PARSER, rtp_, info) )
                {

RDOPROCBlockForQueue::createRes( PARSER, rtp_, res_name );
                }
                else
                {
                    rdoMBuilder::RDOResType rtp_ =
RDOPROCBlockForQueue::createType( PARSER, rtp_name, info );
                    if(
RDOPROCBlockForQueue::checkType(PARSER, rtp_, info) )
                    {

RDOPROCBlockForQueue::createRes( PARSER, rtp_, res_name );
                    }
                }
            }
        }
        | RDO_IDENTIF error
        {
            PARSER->error().error( @1, "Ошибка в имени
ресурса" );
        };
        dpt_seize_param:    RDO_IDENTIF
        {
            // Имя ресурса
            std::string res_name =
reinterpret_cast<RDOValue*>($1)->value().getIdificator().c_str();
            const RDOParserSrcInfo& info = @1;
            // Получили список всех ресурсов
            rdoMBuilder::RDOResourceList rssList( PARSER );
            rdoMBuilder::RDOResType rtp;
            // Если ресурс существует, берем его тип и
проверяем
            if ( rssList[res_name].exist())
            {
                rtp = rssList[res_name].getType();
                if
(RDOPROCBlockForSeize::checkType(PARSER, rtp, info))
                {
                    if
(!rssList[res_name].checkParserResourceType<rdoParse::RDOPROCBResource>(*PARSER))
                    {

RDOPROCBlockForSeize::reobjectRes(PARSER, rtp, res_name);
                    }
                }
            }
            else
            {
                //Ресурс не найден, сформировать имя типа
по имени ресурса
                // Сформировать имя типа по имени ресурса

```

```

std::string rtp_name(
RDOPROCPProcess::s_name_prefix + res_name + RDOPROCPProcess::s_name_sufix );
// Получили список всех типов ресурсов
rdoMBuilder::RDOResTypeList rtpList(
PARSER );

// Нашли тип ресурса
if ( rtpList[rtp_name].exist() )
{
    rdoMBuilder::RDOResType rtp_ =
rtpList[rtp_name];
    if
(RDOPROCPBlockForSeize::checkType(PARSER, rtp_, info))
    {
        RDOPROCPBlockForSeize::createRes(PARSER, rtp_, res_name);
    }
    else
    {
        rtp =
RDOPROCPBlockForSeize::createType(PARSER, rtp_name, info);
        RDOPROCPBlockForSeize::createRes(PARSER, rtp, res_name);
    }
}
| dpt_seize_param ',' RDO_IDENTIF
{
    // Имя ресурса
    std::string res_name =
reinterpret_cast<RDOWalue*>($3)->value().getIdificator().c_str();
    const RDOParserSrcInfo& info = @3;
    // Получили список всех ресурсов
    rdoMBuilder::RDOResourceList rssList( PARSER );
    rdoMBuilder::RDOResType rtp;
    // Если ресурс существует, берем его тип и
проверяем
    if ( rssList[res_name].exist() )
    {
        rtp = rssList[res_name].getType();
        if
(RDOPROCPBlockForSeize::checkType(PARSER, rtp, info))
        {
            if
(!rssList[res_name].checkParserResourceType<rdoParse::RDOPROCPResource>(*PARSER))
            {
                RDOPROCPBlockForSeize::reobjectRes(PARSER, rtp, res_name);
            }
        }
    }
    else
    {
        //Ресурс не найден, сформировать имя типа
по имени ресурса
        // Сформировать имя типа по имени ресурса
        std::string rtp_name(
RDOPROCPProcess::s_name_prefix + res_name + RDOPROCPProcess::s_name_sufix );
        // Получили список всех типов ресурсов
        rdoMBuilder::RDOResTypeList rtpList(
PARSER );

        // Нашли тип ресурса
        if ( rtpList[rtp_name].exist() )
        {
            rdoMBuilder::RDOResType rtp_ =
rtpList[rtp_name];
            if
(RDOPROCPBlockForSeize::checkType(PARSER, rtp_, info))

```

```

        {
            RDOPROCBlockForSeize::createRes(PARSER, rtp_, res_name);
        }
    }
    else
    {
        rtp =
RDOPROCBlockForSeize::createType(PARSER, rtp_name, info);

        RDOPROCBlockForSeize::createRes(PARSER, rtp, res_name);
    }
}
| dpt_seize_param error
{
    PARSER->error().error( @1, "Ошибка в имени
ресурса" );
};
dpt_release_param: RDO_IDENTIF
{
    // Имя ресурса
    std::string res_name =
reinterpret_cast<RDOValue*>($1)->value().getIdentificator().c_str();
    const RDOParserSrcInfo& info = @1;
    // Получили список всех ресурсов
    rdoMBuilder::RDOResourceList rssList( PARSER );
    rdoMBuilder::RDOResType rtp;
    if ( rssList[res_name].exist() )
    {
        rtp = rssList[res_name].getType();
        if
(RDOPROCBlockForSeize::checkType(PARSER, rtp, info))
        {
            if
(!rssList[res_name].checkParserResourceType<rdoParse::RDOPROCResource>(*PARSER))
            {

                RDOPROCBlockForSeize::reobjectRes(PARSER, rtp, res_name);
            }
        }
    }
    else
    {
        // Ресурс не найден, сформировать имя типа
        по имени ресурса
        // Сформировать имя типа по имени ресурса
        std::string rtp_name(
RDOPROCPProcess::s_name_prefix + res_name + RDOPROCPProcess::s_name_sufix );
        // Получили список всех типов ресурсов
        rdoMBuilder::RDOResTypeList rtpList(
PARSER );

        // Нашли тип ресурса
        if ( rtpList[rtp_name].exist() )
        {
            rdoMBuilder::RDOResType rtp_ =
rtpList[rtp_name];

            if
(RDOPROCBlockForSeize::checkType(PARSER, rtp_, info))
            {

                RDOPROCBlockForSeize::createRes(PARSER, rtp_, res_name);
            }
        }
        else
        {

```

```

        rtp =
RDOPROCBlockForSeize::createType(PARSER, rtp_name, info);

        RDOPROCBlockForSeize::createRes(PARSER, rtp, res_name);
    }
}
| dpt_release_param ',' RDO_IDENTIF
{
    // Имя ресурса
    std::string res_name =
reinterpret_cast<RDOValue*>($3)->value().getIdificator().c_str();
    const RDOParserSrcInfo& info = @3;
    // Получили список всех ресурсов
    rdoMBuilder::RDOResourceList rssList( PARSER );
    rdoMBuilder::RDOResType rtp;
    // Если ресурс существует берем его тип и
проверяем

    if ( rssList[res_name].exist() )
    {
        rtp = rssList[res_name].getType();
        if
(RDOPROCBlockForSeize::checkType(PARSER, rtp, info))
        {
            if
(!rssList[res_name].checkParserResourceType<rdoParse::RDOPROCResource>(*PARSER))
            {

                RDOPROCBlockForSeize::reobjectRes(PARSER, rtp, res_name);
            }
        }
    }
    else
    {
        //Ресурс не найден, сформировать имя типа
по имени ресурса

        // Сформировать имя типа по имени ресурса
        std::string rtp_name(
RDOPROCPProcess::s_name_prefix + res_name + RDOPROCPProcess::s_name_sufix );
        // Получили список всех типов ресурсов
        rdoMBuilder::RDOResTypeList rtpList(
PARSER );

        // Нашли тип ресурса
        if ( rtpList[rtp_name].exist() )
        {
            rdoMBuilder::RDOResType rtp_ =
rtpList[rtp_name];

            if
(RDOPROCBlockForSeize::checkType(PARSER, rtp_, info))
            {

                RDOPROCBlockForSeize::createRes(PARSER, rtp_, res_name);
            }
        }
        else
        {
            rtp =
RDOPROCBlockForSeize::createType(PARSER, rtp_name, info);

            RDOPROCBlockForSeize::createRes(PARSER, rtp, res_name);
        }
    }
}
| dpt_release_param error
{
    PARSER->error().error( @1, "Ошибка в имени
ресурса" );
};

```

```
dpt_process_end:    dpt_process RDO_End
                    {
                    };
```

```
%%
```