Оглавление

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. 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. 2.).

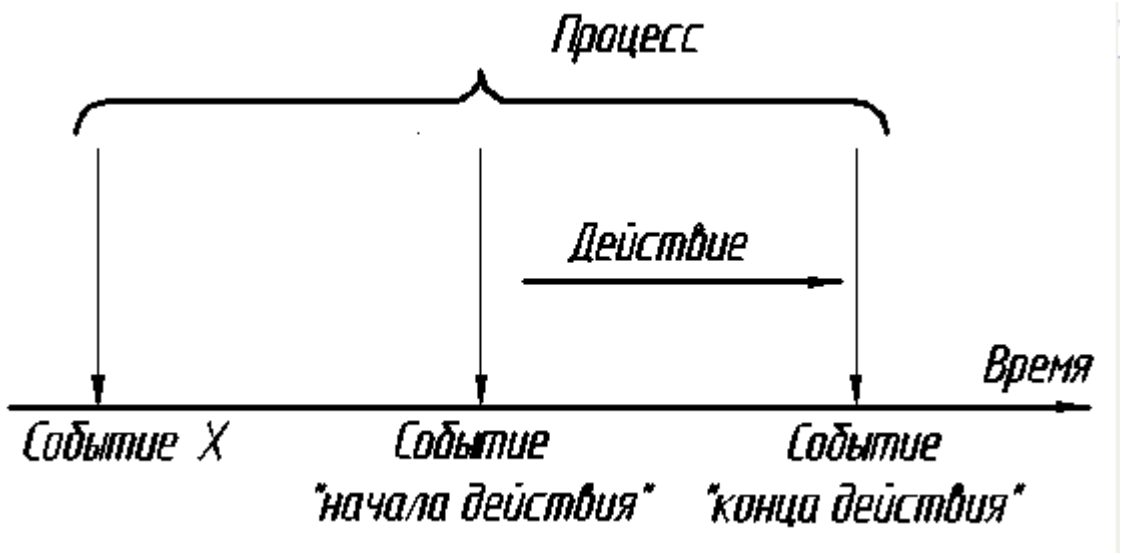


Рис. 1. Взаимосвязь между событиями, действием и процессом.

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 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. 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.

Рис. 2. Выполнение событий в ИМ.

Подход сканирования активностей.

При использовании подхода сканирования активностей разработчик описывает все действия, в которых принимают участие элементы системы, и задает условия, определяющие начало и завершение действий. 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 активностей.

1.3. Основные положения языка РДО.

В основе системы РДО – «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);
- операции (с расширением .opr);
- 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 .smr).

Объекты, создаваемые РДО-имитатором при выполнении прогона:

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

1.4. Постановка задачи.

Основная идея курсового проекта – добавление в систему логического вывода системы имитационного имитирования РДО инструмента, позволяющего решать конфликты, связанные с возможностью запуска более чем одной активности внутри блока активностей точки принятия решений типа some.

Сейчас система логического вывода РДО выполняет первую найденную активность, у которой выполняются условия запуска.

Такой подход не всегда позволяет написать адекватную модель в системе РДО, да и просто лишает язык РДО некоторой гибкости. Для демонстрации этого недостатка рассмотрим модель склада, оборудованного манипулятором, который помещает прибывающие детали в специальный накопитель. По прибытии в рассматриваемую систему детали имеют горизонтальное положение. Манипулятор должен захватить деталь, ориентировать ее вертикально (определенным концом вверх), поместить в накопитель и вернуться за следующей деталью.

Модель данной СМО на языке РДО представлена в Приложении 3.

Обратим внимание как описаны активности, ориентирующие деталь:

```
Ориентирование_детали_против_часовой_стрелки : Образец_ориентирования_детали
против_часовой_стрелки

Ориентирование_детали_по_часовой_стрелке      : Образец_ориентирования_детали
по_часовой_стрелке
```

Эти активности созданы по одному образцу Образец_ориентирования_детали, типа operation, и отличаются только направлением вращения детали (против и по часовой стрелке соответственно). Т.е. обе эти активности в результате ориентируют деталь нужным образом, только одна из них будет поворачивать деталь на 90°, а другая на 270°. Очевидно, что меньший поворот более выгоден (хотябы с точки зрения временных ресурсов). Именно для этого в модели системы предусмотрена возможность вращения детали в разные стороны. Вот только РДО не может сделать объективный выбор между ними и в результате будет выполняться только та, которая записана выше на закладке DPT. Об этом свидетельствует циклограмма, полученная в результате прогона модели.

Очевидно, что пользователю, пишущему модели на РДО, необходим инструмент, с помощью которого можно назначать приоритет для каждой активности.

Таким образом, целью курсового проекта является добавление в систему имитационного моделирования РДО возможности назначать активностям внутри точек принятия решений приоритеты, позволяющие решать подобные конфликты.

2.Концептуальный этап проектирования.

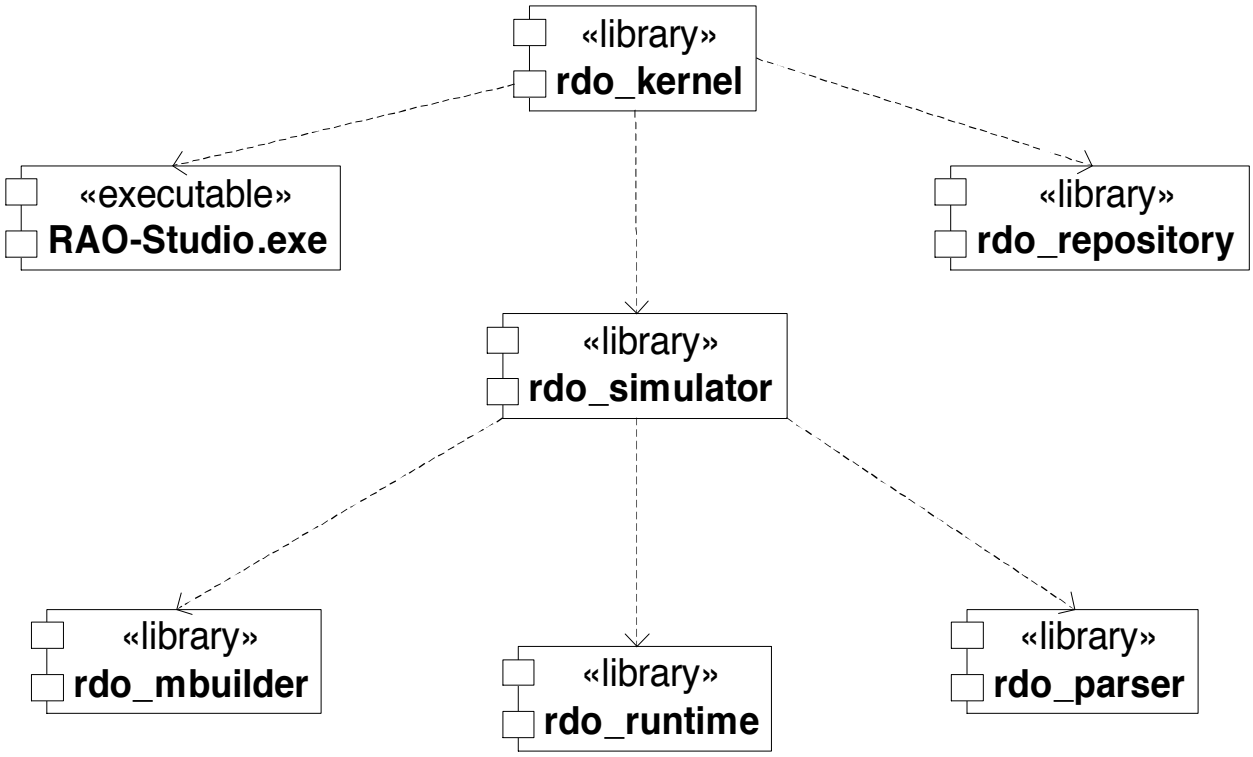
Система имитационного моделирования РДО безусловно является сложной и статически, и динамически. На это указывает сложная иерархическая структура системы со множеством различных связей между компонентами и ее сложное поведение во времени.

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.

Рис. 4. Диаграмма компонентов.



Базовый функционал представленных на диаграмме компонентов:

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::RDODPTSome::RDODPTSome содержит следующее выражение:

```
m_rt_logic = new rdoRuntime::RDODPTSome( parser()->runtime() );
```

которое выделяет место в свободной памяти и инициализирует объект rdoRuntime::RDODPTSome, который в дальнейшем будет учавствовать в процессе имитации работы модели.

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 один СУПЕР-Объект. Он является контейнером для хранения разных операций. Эти операции на данном уровне иерархии будут представлять собой МЕТА-Объекты, например, список операций на закладке OPR, точки принятия решений и блоки свободных активностей на закладке DPT. Все эти МЕТА-Объекты аналогично СУПЕР-Объекту являются контейнерами, в которых хранятся такие объекты РДО, как нерегулярные события, действия, операции, клавиатурные операции.

Интересно отметить, что реализация описанной структуры с помощью наследования – одного из основных механизмов объектно-ориентированного программирования – делает возможным на уровне логики работы РДО рекусивное вложение МЕТА-Объектов внутрь МЕТА-Объектов. То есть на уровне логики РДО разрешено наличие точек принятия решений внутри точки 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 описанный выше СУПЕР-Объект. У этого атрибута вызывается метод onCheckCondition, который по описанной цепочке передает управление аналогичным методам нижележащих объектов. В результате в контейнере МЕТА-Объекта инициализируется атрибут m_first, который будет указывать на активность, которая должна быть выполнена следующей. В свою очередь на данный МЕТА-Объект будет указывать атрибут m_first СУПЕР-Объекта.

2.3.Логика rdoRuntime::RDODPTSome.

Сперва проверяется является ли истинным условие активизации точки. И в случае его истинности начинается циклическая проверка всех активностей, находящихся в контейнере данной точки. Проверка длится до тех пор пока не найдется первая

активность, условия запуска которой являются истинными. После чего эта активность запоминается с помощью указателей m_first.

Для выбора из множества активностей, у которых выполняется условие запуска, активности с наибольшим приоритетом можно поступить двумя способами:

- сначала отсортировать весь список активностей по уменьшению приоритета, а затем двигаясь по отсортированному списку выбрать первую активность, условие запуска которой истинно;

- сначала отобрать из списка всех активностей те, условия запуска которых истинны, а затем выбрать из них одну с наибольшим приоритетом.

Очевидно, что недостатком первого способа является то, что в не зависимости от того, у каких активностей выполняется условие запуска, приоритет необходимо расчитывать для каждой активности.

В свою очередь, недостатком второго способа является необходимость проверки условия запуска каждой активности, несмотря на то, что в итоге будет выбрана лишь одна с наибольшим приоритетом.

Проверка условия запуска активности заключается в проверке всех логических выражений, определяющих предусловия использования ресурсов. Т.е. наличие в образце активности более одного релевантного ресурса делает второй способ более ресурсоемким.

Проанализировав 6 случайно отобранных моделей систем на языке РДО, взятых с сайта http://rdo.rk9.bmstu.ru/, я обнаружил в них 116 патернов и 315 релевантных им ресурса. Т.е. одному патерну релевантно примерно 2,72 ресурса.

Это позволяет сделать выбор в пользу первого способа выбора из множества активностей точки принятия решений типа Some, у которых выполняется условие запуска, активности с наибольшим приоритетом.

2.4.Техническое задание.

2.4.1.Общие сведения.

В системе РДО разрабатывается новый инструмент обработки приоритетов, позволяющий разработчику моделей на РДО решать в явном виде конфликты, связанные с возможностью запуска более чем одной активности внутри точки принятия решений типа Some. Основной разработчик РДО – кафедра РК-9, МГТУ им. Н.Э. Баумана.

2.4.2.Назначение и цели развития системы.

Основная цель данного курсового проекта – разработать механизм логического вывода в системе имитационного моделирования РДО на основе приоритетных правил.

Приоритетное правило (приоритетная активность) – активность точки принятия решений типа Some, созданная по образцу типа operation или rule, и имеющая в своем описании специальную конструкцию – приоритет.

2.4.3.Характеристики объекта автоматизации.

РДО – язык имитационного моделирования, основанный на событийном подходе и подходе сканирования активностей. Знания в системе представляются в виде модифицированных продукционных правил.

2.4.4.Требования к системе.

При описании активности внутри точки принятия решения типа Some пользователь может после всех параметров указать приоритет этой активности. Приоритет должен представлять собой слово “CF”, знак равенства и арифметическое выражение, которое должно возвращать значение целого или вещественного типа данных и находиться в диапазоне [0; 1].

Т.е. с учетом возможности описания приоритетов активностей модель, описывающая работу склада, должна иметь вид:

```
Ориентирование_детали_против_часовой_стрелки : Образец_ориентирования_детали
против_часовой_стрелки CF = Как_ориентирована_деталь(Система.как_ориентированна_текущая_деталь)
```

```
        Ориентирование_детали_по_часовой_стрелке      : Образец_ориентирования_детали
по_часовой_стрелке CF = 1 - Как_ориентирована_деталь(Система.как_ориентированна_текущая_деталь)
```

где Как_ориентирована_деталь() – функция, возвращающая значение 1, от аргумента «против_часовой_стрелки», и значение 0 от аргумента «по_часовой_стрелке» соответственно.

3.Технический этап проектирования.

3.1.Разработка синтаксиса описания точки принятия решения типа Some.

Объект точек принятия решений имеет следующий формат:

<описание_точки_принятия_решений> | <блок_активностей>

{ <описание_точки_принятия_решений> | <блок_активностей> }

Описание каждой точки принятия решений, типа some, имеет следующий формат:

”$Decision_point” <имя_точки> “.” “some”

”$Condition” <условие_активизации_точки>

<блок_активностей>

Имена точек принятия решений должны быть различными для всех точек принятия решений и не должны совпадать с ранее определенными именами.

Условие активизации точки – это логическое выражение. Алгоритм обработки точки принятия решений активизируется как только состояние системы удовлетворяет этому выражению.

Блок активностей имеет следующий формат:

“$Activities” <описание_активности> {<описание_активности>}

“$End”

Описание каждой активности имеет следующий формат:

<имя_активности> “:” <имя_образца> [<значения_параметров_образца>] [<приоритет_активности>]

Имя активности – это любое имя, не совпадающее с ранее определенными в модели именами.

Имя образца – это имя одного из образцов, заданных в объекте образцов.

Значения параметров образца должны быть заданы в позиционном соответствии с параметрами в описании образца. Их значения задают вещественной или целой численной константой, либо именем значения в соответствии с типом параметра. Если для параметра указано значение по умолчанию, то вместо начального значения можно указать символ «*», и параметр примет значение по умолчанию. Если для параметра задан диапазон возможных значений, то осуществляется проверка значения параметра на вхождение в заданный диапазон.

Приоритет активности имеет следующий формат:

“CF” “=” <арифметическое_выражение>

Арифметическое выражение должно иметь целый или вещественный тип. Его значение должно лежать в диапазоне [0; 1].

3.2.Разработка архитектуры компонента rdo_parser.

Для возможности обработки новой конструкции в коде модели требует изменений лексический и синтаксический анализаторы РДО.

В классе RDODPTActivity в пространстве имен rdoParse необходимо добавить функцию-член setPrior(), которая будет передавать приоритет в rdoRuntime. Эту функцию синтаксический должен вызывать после того как найдет корректное описание приоритета у активности точки принятия решений типа Some.

В пространстве имен rdoParse приоритетом активности должен иметь тип RDOFunArithm.

3.3.Разработка архитектуры компонента rdo_runtime.

В пространстве имен rdoRuntime приоритет должен появиться у двух классов RDOOperation и RDORule. Они являются потомками класса RDOActivityPattern<T>, который в свою очередь является шаблоном. Это является аргументом в пользу того, чтобы для классов RDOOperation и RDORule завести еще одного предка (применить множественное наследование) вне цепочки между ними и RDOBaseOperation. Этот класс RDOActivityPatternPrior должен инкапсулировать в себе все необходимое для работы с приоритетами, т.е. атрибут m_prior с приоритетом и методы setPrior() и getPrior() для запоминания и извлечения приоритета соответственно.

В пространстве имен rdoRuntime атрибут с приоритетом должен иметь тип RDOCalc.

Изменений требует метод onCheckCondition класса RDOLogic. В нем необходимо предусмотреть возможность сортировки списка активностей контейнера в методе actionWithRDOOprContainer(). Соответственно в классе RDODPTSome необходимо реализовать новый метод.

Для осуществления сортировки необходимо реализовать функтор RDODPTActivityCompare для сравнения приоритетов у активностей.

Также во время сортировки необходимо исключить из списка нерегулярные события, так как понятие приоритета операции для них неопределено, и проследить, чтобы значение приоритета удовлетворяло диапазону [0; 1].

4.Рабочий этап проектирования.

4.1.Синтаксический анализ приоритета операций.

Для реализации в среде имитационного моделирования нового инструмента способом, выбранным на концептуальном этапе проектирования и подробнее описанным на техническом этапе, 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_CF, который может быть записан двумя разными способами: строчными буквами cf и заглавными CF:

```
cf                                  return(RDO_CF);

CF                                  return(RDO_CF);
```

В генераторе синтаксического анализатора (bison) необходимо добавить, во-первых, новый токен:

```
%token RDO_CF                                           371
```

, во-вторых, описание приоритета активности внутри DPTSome:

```
dpt_some_activity:          /* empty */

                                    | dpt_some_activity dpt_some_name
dpt_some_descr_param dpt_some_descr_prior{
                                          RDODPTSomeActivity* activity =
reinterpret_cast<RDODPTSomeActivity*>($2);
                                          activity->endParam( @3 );
                                    };
```

Этот фрагмент кода имеет следующий смысл:

– активности может просто не быть — на это указывает закоментированное слово empty;

– или данная активность должна состоять из предыдущей активности (это не что иное, как рекурсивное описание возможности существования множества активностей), имени активности, описания параметров активности и описания приоритета активности.

Код, заключенный в фигурные скобки, показывает, что при нахождении новой активности ее имя запоминается и вызывается метод endParam().

Описание всех нетермальных токенов, описывающих DPTSome приведено в Приложении 1, а здесь приведем лишь описание разрабатываемого в рамках данного курсового проекта токена dpt_some_descr_prior:

```
	dpt_some_descr_prior: /* empty */

                                        | RDO_CF '=' fun_arithm
                                        {
                                                if (!PARSER->getLastDPTSome()-
>getLastActivity()->setPrior( reinterpret_cast<RDOFUNArithm*>($3) ))
                                                {
                                                        PARSER->error(@3, _T("Активность не
может иметь приоритет"));
                                                }
                                        }
                                        | RDO_CF '=' error
                                        {
                                                PARSER->error( @1, @2, "Ошибка описания
приоритета активности" )
                                        }
                                        | RDO_CF error
                                        {
                                                PARSER->error( @1, @2, "Ошибка: ожидается
знак равенства" )
                                        };
```

Этот фрагмент кода имеет следующий смысл:

– у активности приоритет может отсутствовать — на это указывает закоментированное слово empty;

– или приоритет представляет из себя простой токен RDO_CF, описанный выше, знака равенства и арифметического выражения. В этом случае у последней активности вызывается метод setPrior(). Если при сохранении приоритета активности возникла ошибка, то пользователь получит сообщение "Активность не может иметь приоритет". В противном случае обработка приоритета активности на этом заканчивается.

Если описание приоритета не соответствует описанным вариантам, то в редакторе модели появляется соответствующая ошибка.

4.2.Изменения в пространстве имен rdoParse.

Объявление класса RDODPTActivity

```
	class RDODPTActivity: public RDOParserObject, public RDOParserSrcInfo

{
public:
	RDODPTActivity( const RDOParserObject* _parent, const RDOParserSrcInfo& _src_info, const
RDOParserSrcInfo& _pattern_src_info );

	const std::string&       name() const     { return src_info().src_text(); }
	rdoRuntime::RDOActivity* activity() const { return m_activity;            }
	const RDOPATPattern*     pattern() const  { return m_pattern;             }

	void addParam( const RDOValue& param  );
	void endParam( const YYLTYPE& _param_pos );

	bool setPrior( RDOFUNArithm* prior );

protected:
	rdoRuntime::RDOActivity* m_activity;

private:
	unsigned int          m_currParam;
	const RDOPATPattern*  m_pattern;
};
```

bool setPrior(RDOFUNArithm* prior) - метод класса RDODPTActivity, сохраняет приоритет активности в классе RDOActivityPatternPrior.

Определение метода bool setPrior(RDOFUNArithm prior)*

```
	bool RDODPTActivity::setPrior(RDOFUNArithm* prior)

{
	rdoRuntime::RDOActivityPatternPrior* prior_activity =
dynamic_cast<rdoRuntime::RDOActivityPatternPrior*>(m_activity);
	if (prior_activity)
	{
		return prior_activity->setPrior(prior->createCalc());
	}
	return false;
}
```

В случае успешного приведения данной активности к классу rdoRuntime::RDOActivityPatternPrior (т.е. если активность образована из образца типа operation или rule) вызывается его метод setPrior() с параметром, в качестве которого выступает распознанный синтаксическим анализатором приоритет активности. В результате в классе rdoRuntime::RDOActivityPatternPrior запоминается приоритет активности, о чем синтаксический анализатор будет оповещен значением true, которое данный метод ему вернет. В случае, если активность не удалось привести к классу rdoRuntime::RDOActivityPatternPrior, синтаксическому анализатору будет передано false, и пользователь получит сообщение о невозможности использования приоритета с данной активностью.

4.3.Изменения в пространстве имен rdoRuntime.

Как было выявлено на техническом этапе проектирования в rdoRuntime должен появиться новый класс, который инкапсулирует в себе все связанное с хранением приоритета активности.

Объявление класса RDOActivityPatternPrior

```
class RDOActivityPatternPrior


{
public:
	RDOActivityPatternPrior()
		: m_prior(NULL)
	{}
	virtual ~RDOActivityPatternPrior()
	{
		if (m_prior)
		{
			delete m_prior;
			m_prior = NULL;
		}
	}

	RDOCalc* getPrior()
	{
		return m_prior;
	}
	bool setPrior(RDOCalc* prior)
	{
		m_prior = prior;
		return true;
	}

private:
	RDOCalc* m_prior;
};
```

RDOCalc* m_prior — атрибут, в котором хранится приоритет в виде арифметического выражения.

bool setPrior(RDOCalc* prior) — метод класса RDOActivityPatternPrior, с помощью которого у объекта класса происходит инициализация атрибута m_prior, и вызывающей функции возвращается значение true.

RDOCalc* getPrior() — интерфейсный метод RDOActivityPatternPrior, возвращает приоритет объекта.

Наиболее важные для понимания работы логического вывода РДО классы RDOLogic и RDOOprContainer<T> представлены в Приложении 2 полностью. Здесь рассмотрим лишь изменения, внесенные в них.

В метод onCheckCondition() класса RDOLogic добавим новый метод actionWithRDOOprContainer(), в котором RDODPTSome будет сортировать активности.

Определение функции-члена onCheckCondition класса RDOLogic

```
virtual bool onCheckCondition( RDOSimulator* sim )
{
	bool condition = checkSelfCondition( sim );
	if ( condition != m_lastCondition )
	{
		m_lastCondition = condition;
		if ( condition )
		{
			start( sim );
		}
		else
		{
			stop( sim );
		}
	}
	if ( condition )
	{
		if ( !m_childLogic.onCheckCondition( sim ) )
		{
			actionWithRDOOprContainer( sim );
			return RDOOprContainer<RDOBaseOperation>::onCheckCondition( sim );
		}
		else
		{
			return true;
		}
	}
	else
	{
		return false;
	}
}
```

Определение функции-члена actionWithRDOOprContainer класса RDOLogic

```
virtual void actionWithRDOOprContainer( RDOSimulator* sim )
{
}
```

virtual void actionWithRDOOprContainer(RDOSimulator* sim) - виртуальный метод класса RDOLogic, для осуществления предварительных действий со списком активностей объекта RDOLogic. Поэтому он располагается непосредственно перед вызовом RDOOprContainer<RDOBaseOperation>::onCheckCondition(sim). Метод пустой.

Для сравнения двух приоритетов во время сортировки списка активностей создадим класс RDODPTActivityCompare с определенным оператором bool operator(), называемый в программировании специальным термином — функтор.

```
class RDODPTActivityCompare
```

```
{
public:
	RDODPTActivityCompare(RDORuntime* runtime)
		: m_runtime(runtime)
	{}
	bool operator() (RDOBaseOperation* opr1, RDOBaseOperation* opr2 )
	{
		RDOActivityPatternPrior* pattern1 = dynamic_cast<RDOActivityPatternPrior*>(opr1);
		RDOActivityPatternPrior* pattern2 = dynamic_cast<RDOActivityPatternPrior*>(opr2);
		if (pattern1 && pattern2)
		{
			RDOCalc* prior1 = pattern1->getPrior();
			RDOCalc* prior2 = pattern2->getPrior();
			RDOValue value1 = prior1 ? prior1->calcValue(m_runtime) : RDOValue(0.0);
			RDOValue value2 = prior2 ? prior2->calcValue(m_runtime) : RDOValue(0.0);
			return value1 > value2;
		}
		return false;
	}

private:
	RDORuntime* m_runtime;
};
```

RDORuntime* m_runtime – атрибут, в котором сохраняется текущее состояние системы, нужен для получения значения арифметического выражения, которым является приоритет активностей.

bool operator() (RDOBaseOperation* opr1, RDOBaseOperation* opr2) - оператор, сравнивающий приоритеты двух передаваемые ему активностей. Сначала каждая активность приводится к объекту типа RDOActivityPatternPrior, в котором хранится его приоритет. Затем методом getPrior() оператор получает арифметическое выражение приоритета, и методом calcValue(m_runtime) вычисляет его значение. После этого сравнивает два полученных значения оператором больше, т.е. operator() вернет true, если приоритет первой активности больше, чем второй. Если активность не может быть приведена к объекту типа RDOActivityPatternPrior (т.е. активность является нерегулярным потоком), то сравнение не осуществляется, а метод sort() получает значение false и не меняет текущие активности местами.

```
	class RDODPTSome: public RDOLogic


{
public:
	RDODPTSome (RDOSimulator* sim);
	virtual ~RDODPTSome();

private:
	virtual void actionWithRDOOprContainer(RDOSimulator* sim)
	{
		RDORuntime* runtime = reinterpret_cast<RDORuntime*>(sim);
		for (CIterator it = begin(); it != end(); ++it)
		{
			RDOActivityPatternPrior* pattern =
dynamic_cast<RDOActivityPatternPrior*>(*it);
			if (pattern)
			{
				RDOCalc* prior = pattern->getPrior();
```

```
                                if (prior)
                                {
                                        RDOValue value = prior->calcValue(runtime);
                                        if (value < 0 || value > 1)
                                                runtime->error(rdo::format(_T("Приоритет активности
вышел за пределы диапазона [0..1]: %s"), value.getAsString().c_str()), prior);
                                }
                        }
                }
                std::sort(begin(), end(),
RDODPTActivityCompare(reinterpret_cast<RDORuntime*>(sim)));
        }
};
```

virtual void actionWithRDOOprContainer(RDOSimulator* sim) - виртуальный метод класса RDODPTSome, сначала производит проверку соответствия приоритетов всех активностей, у которых он есть, диапазону [0; 1], а затем сортировку списка активностей, содержащихся в точке принятии решений, от первой до последней, используя для этого функтор RDOActivityCompare. В итоге список активностей становится отсортированным по уменьшению.

Заключение

В рамках данного курсового проекта были получены следующие результаты:

1) Проведено предпроектное исследование системы имитационного моделирования РДО и сформулированы предпосылки создания в системе инструмента для решения конфликтов, возникающих при состоянии системы в котором может быть запущено более одной активности.

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. Предложены две концепции решения поставленной задачи. Обе конпепции представлены на диаграммах методологии функционального моделирования IDEF0. На основе анализа уже существующих моделей систем на РДО сделан выбор в пользу первой.

3) На этапе технического проектирования разработан синтаксис описания точки принятия решений типа Some, который показан на синтаксической диаграмме. С помощью диаграммы классов разработана архитектура новой системы. На диаграмме блок-схемы алгоритмов показана реализация концепции выбранной на этапе концептуального проектирования.

4) На этапе рабочего проектирования разработан программный код для реализации спроектированных раннее алгоритмов работы и архитектуры компонентов rdo_parser и rdo_runtime системы РДО. Проведены отладка и тестирование новой системы, в ходе которых исправлялись найденные ошибки в программном коде.

5) Для устранения проблем в модели склада, выявленных на этапе предпроектного исследования, она была переписана так, чтобы задействовать новый механизм логического вывода РДО. Результаты проведения имитационного исследования показывают, что время на ориентирование детали роботом действительно сократилось и теперь составляет 13,73% времени работы системы вместо 24,89% раньше.

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. Леоненков. Самоучитель по UML [http://khpi-iip.mipk.kharkiv.edu/library/case/leon/index.html].

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

Приложение 1. Полный синтаксический анализ DPTSome (rdodpt.y).

```
// ----------------------------------------------------------------------------
// ---------- DPT Some
// ----------------------------------------------------------------------------
dpt_some_trace:                     /* empty */ {
                                          $$ = 1;
                                    }
                                    | RDO_no_trace {
                                          $$ = 1;
                                    }
                                    | RDO_trace {
                                          $$ = 2;
                                    }
                                    | RDO_trace_stat {
                                          PARSER->error( @1, "Данный признак
трассировки не используется в точке типа some" );
                                    }
                                    | RDO_trace_tops {
                                          PARSER->error( @1, "Данный признак
трассировки не используется в точке типа some" );
                                    }
                                    | RDO_trace_all {
                                          PARSER->error( @1, "Данный признак
трассировки не используется в точке типа some" );
                                    };

dpt_some_begin:                     RDO_Decision_point RDO_IDENTIF_COLON RDO_some dpt_some_trace
{
                                          RDOValue* name =
reinterpret_cast<RDOValue*>($2);
                                          $$ = (int)new RDODPTSome( PARSER, name-
>src_info() );
                                    };

dpt_some_condition:           dpt_some_begin RDO_Condition fun_logic {
                                          RDODPTSome* dpt =
reinterpret_cast<RDODPTSome*>($1);
                                          dpt-
>setCondition( reinterpret_cast<RDOFUNLogic*>($3) );
                                    }
                                    | dpt_some_begin RDO_Condition RDO_NoCheck {
                                          RDODPTSome* dpt =
reinterpret_cast<RDODPTSome*>($1);
                                          dpt->setCondition();
                                    }
                                    | dpt_some_begin RDO_Condition error {
                                          PARSER->error( @2, @3, "После ключевого слова
$Condition ожидается условие запуска точки" );
                                    }
                                    | dpt_some_begin {
                                          RDODPTSome* dpt =
reinterpret_cast<RDODPTSome*>($1);
                                          dpt->setCondition();
                                    };

dpt_some_name:                RDO_IDENTIF_COLON RDO_IDENTIF {
                                          RDODPTSome* dpt   = PARSER->getLastDPTSome();
                                          RDOValue* name    =
reinterpret_cast<RDOValue*>($1);
                                          RDOValue* pattern =
reinterpret_cast<RDOValue*>($2);
                                          $$ = (int)dpt->addNewActivity( name-
>src_info(), pattern->src_info() );
                                    }
                                    | RDO_IDENTIF_COLON error {
                                          PARSER->error( @1, @2, "Ожидается имя
образца" );
                                    };

dpt_some_descr_param: /* empty */
                                    | dpt_some_descr_param  '*'              { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( RDOValue(RDOParserSrcInfo(@2, "*")) ) }
```

```
                                        | dpt_some_descr_param  RDO_INT_CONST     { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_REAL_CONST    { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_BOOL_CONST    { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_STRING_CONST  { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_IDENTIF       { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }

                                        | dpt_some_descr_param error
                                        {
                                                PARSER->error( @1, @2, "Ошибка описания
параметра образца" )
                                        };

dpt_some_descr_prior: /* empty */
                                        | RDO_CF '=' fun_arithm
                                        {
                                                if (!PARSER->getLastDPTSome()-
>getLastActivity()->setPrior( reinterpret_cast<RDOFUNArithm*>($3) ))
                                                {
                                                        PARSER->error(@3, _T("Активность не
может иметь приоритет"));
                                                }
                                        }
                                        | RDO_CF '=' error
                                        {
                                                PARSER->error( @1, @2, "Ошибка описания
приоритета образца" )
                                        }
                                        | RDO_CF error
                                        {
                                                PARSER->error( @1, @2, "Ошибка: ожидается
знак равенства" )
                                        };

dpt_some_activity:          /* empty */
                                        | dpt_some_activity dpt_some_name
dpt_some_descr_param dpt_some_descr_prior{
                                                RDODPTSomeActivity* activity =
reinterpret_cast<RDODPTSomeActivity*>($2);
                                                activity->endParam( @3 );
                                        };

dpt_some_header:            dpt_some_condition RDO_Activities dpt_some_activity {
                                        }
                                        | dpt_some_condition error {
                                                PARSER->error( @1, @2, "Ожидается ключевое
слово $Activities" );
                                        };

dpt_some_end:               dpt_some_header RDO_End {
                                                RDODPTSome* dpt =
reinterpret_cast<RDODPTSome*>($1);
                                                dpt->end();
                                        }
                                        | dpt_some_header {
                                                PARSER->error( @1, "Ожидается ключевое слово
$End" );
                                        };
```

Приложение 2. Классы RDOLogic и RDOOprContainer<T> (rdo_logic.h).

```
#ifndef RDO_LOGIC_H
#define RDO_LOGIC_H

#pragma warning(disable : 4786)

#include "rdo.h"
#include "rdocalc.h"

namespace rdoRuntime {

// --------------------------------------------------------------------------------
// ---------- RDOOprContainer
// --------------------------------------------------------------------------------
template<class T>
class RDOOprContainer: public RDOBaseOperation
{
public:
	RDOOprContainer( RDORuntimeParent* parent ):
		RDOBaseOperation( parent ),
		m_first( NULL )
	{
	}
	virtual ~RDOOprContainer() {}

	typedef          std::vector< T* >     List;
	typedef typename List::iterator        Iterator;
	typedef typename List::const_iterator  CIterator;

	Iterator  begin()       { return m_items.begin(); }
	Iterator  end()         { return m_items.end();   }
	CIterator begin() const { return m_items.begin(); }
	CIterator end()   const { return m_items.end();   }

	void append( T* item )
	{
		if ( item )
		{
			item->reparent( this );
			m_items.push_back( item );
		}
	}
	virtual void onStart( RDOSimulator* sim )
	{
		Iterator it = begin();
		while ( it != end() )
		{
			(*it)->onStart(sim);
			it++;
		}
	}
	virtual void onStop( RDOSimulator* sim )
	{
		Iterator it = begin();
		while ( it != end() )
		{
			(*it)->onStop(sim);
			it++;
		}
	}
	virtual bool onCheckCondition( RDOSimulator* sim )
	{
		Iterator it = begin();
		while ( it != end() )
		{
			if ( (*it)->onCheckCondition(sim) )
			{
				m_first = *it;
				return true;
			}
			it++;
		}
```

```
                m_first = NULL;
                return false;
        }
        virtual BOResult onDoOperation( RDOSimulator* sim )
        {
                if ( !m_first )
                {
                        if ( !onCheckCondition(sim) || !m_first )
                        {
                                return BOR_cant_run;
                        }
                }
                BOResult result = m_first->onDoOperation(sim);
                if ( result == BOR_must_continue ) {
                        sim->setMustContinueOpr( m_first );
                }
                return result;
        }
        virtual void onMakePlaned( RDOSimulator* sim, void* param = NULL )
        {
                Iterator it = begin();
                while ( it != end() )
                {
                        (*it)->onMakePlaned( sim, param );
                        it++;
                }
        }
        virtual BOResult onContinue( RDOSimulator* sim )
        {
                Iterator it = begin();
                while ( it != end() )
                {
                        if ( (*it)->onContinue( sim ) == BOR_must_continue )
                        {
                                return BOR_must_continue;
                        }
                        it++;
                }
                return BOR_cant_run;
        }

private:
        std::vector< T* > m_items;
        T*                m_first;
};

// ----------------------------------------------------------------------------
// ---------- RDOLogic
// ----------------------------------------------------------------------------
class RDOLogic: public RDOOprContainer< RDOBaseOperation >
{
public:
        RDOLogic( RDOSimulator* sim );
        virtual ~RDOLogic() {}

        RDOOprContainer< RDOBaseOperation > m_childLogic;

        void setCondition( RDOCalc* calc )
        {
                m_condition = calc;
        }

        virtual void onStart( RDOSimulator* sim )
        {
                m_lastCondition = checkSelfCondition( sim );
                if ( m_lastCondition )
                {
                        start( sim );
                }
        }
        virtual void onStop( RDOSimulator* sim )
        {
                m_lastCondition = false;
                stop( sim );
```

```
	}
	virtual bool onCheckCondition( RDOSimulator* sim )
	{
		bool condition = checkSelfCondition( sim );
		if ( condition != m_lastCondition )
		{
			m_lastCondition = condition;
			if ( condition )
			{
				start( sim );
			}
			else
			{
				stop( sim );
			}
		}
		if ( condition )
		{
			if ( !m_childLogic.onCheckCondition( sim ) )
			{
				actionWithRDOOprContainer( sim );
				return RDOOprContainer<RDOBaseOperation>::onCheckCondition( sim );
			}
			else
			{
				return true;
			}
		}
		else
		{
			return false;
		}
	}
	virtual BOResult onDoOperation( RDOSimulator* sim )
	{
		if ( m_lastCondition )
		{
			BOResult result = m_childLogic.onDoOperation( sim );
			if ( result == BOR_cant_run )
			{
				return RDOOprContainer<RDOBaseOperation>::onDoOperation( sim );
			}
			else
			{
				return result;
			}
		}
		else
		{
			return BOR_cant_run;
		}
	}
	virtual void onMakePlaned( RDOSimulator* sim, void* param = NULL )
	{
		m_childLogic.onMakePlaned( sim, param );
		RDOOprContainer<RDOBaseOperation>::onMakePlaned( sim, param );
	}
	virtual BOResult onContinue( RDOSimulator* sim )
	{
		BOResult result = m_childLogic.onContinue( sim );
		if ( result != BOR_must_continue )
		{
			return RDOOprContainer<RDOBaseOperation>::onContinue( sim );
		}
		else
		{
			return result;
		}
	}

private:
	RDOCalc* m_condition;
	bool     m_lastCondition;
```

```
        // в этом методе RDODPTSome сортирует список активностей внутри контейнера
        virtual void actionWithRDOOprContainer( RDOSimulator* sim )
        {
        }
        bool checkSelfCondition( RDOSimulator* sim )
        {
                if ( m_condition )
                {
                        return m_condition->calcValue( (RDORuntime*)sim ).getAsBool();
                }
                else
                {
                        return true;
                }
        }
        void start( RDOSimulator* sim )
        {
                m_childLogic.onStart( sim );
                RDOOprContainer<RDOBaseOperation>::onStart( sim );
        }
        void stop( RDOSimulator* sim )
        {
                m_childLogic.onStop( sim );
                RDOOprContainer<RDOBaseOperation>::onStop( sim );
        }
};

} // namespace rdoRuntime

#endif // RDO_LOGIC_H
```

Приложение 3. Код имитационной модели склада на языке РДО.

СКЛАД.rtp (Типы ресурсов):

```
$Resource_type Детали  : temporary
$Parameters
	номер                  : integer
	ориентация             : (по_часовой_стрелке, против_часовой_стрелки)
	состояние              : ( Свободна, Захват_начат, Захват_закончен, Ориентирование_начато,
Ориентирование_закончено, Транспортировка_начата, Транспортировка_закончена )
$End

$Resource_type Роботы  : permanent
$Parameters
	состояние              : ( Свободен, Захват_начат, Захват_закончен, Ориентирование_начато,
Ориентирование_закончено, Транспортировка_начата, Транспортировка_закончена, Возврат )
$End

$Resource_type Системы : permanent
$Parameters
	как_ориентированна_текущая_деталь : such_as Детали.ориентация
	текущее_число_деталей             : integer
	количество_прибывших_деталей      : integer
	количество_погруженных_деталей    : integer
$End
```

СКЛАД.rss (Ресурсы):

```
$Resources
	Система       : Системы trace по_часовой_стрелке 0 0 0
	Робот         : Роботы  trace Свободен
$End
```

СКЛАД.fun (Константы, последовательности и функции):

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

$Sequence Ориентирование_детали : such_as Детали.ориентация
$Type = by_hist 123456789
$Body
	по_часовой_стрелке     1
	против_часовой_стрелки 1
$End

$Sequence Длительность_захвата_детали : real [0.0..1.0]
$Type = normal 123456789
$End

$Sequence Длительность_транспортировки_детали : real [0.0..1.0]
$Type = normal 123456789
$End

$Sequence Длительность_возврата_робота : real [0.0..1.0]
$Type = normal 123456789
$End

$Sequence Длительность_ориентирования_детали_на_четверть_оборота : real [0.15..0.25]
$Type = normal 123456789
$End

$Function Длительность_ориентирования_детали_по_часовой_стрелке : real = 0
$Type = algorithmic
```

```
$Parameters
        ориентация_детали : such_as Детали.ориентация
$Body
        if ориентация_детали = по_часовой_стрелке     result =
Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )
        if ориентация_детали = против_часовой_стрелки result =
Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )

                + Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )

                + Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )
$End

$Function Длительность_ориентирования_детали_против_часовой_стрелки : real = 0
$Type = algorithmic
$Parameters
        ориентация_детали : such_as Детали.ориентация
$Body
        if ориентация_детали = против_часовой_стрелки result =
Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )
        if ориентация_детали = по_часовой_стрелке     result =
Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )

                + Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )

                + Длительность_ориентирования_детали_на_четверть_оборота( 0.15, 0.25 )
$End

$Function Длительность_ориентирования_детали : real = 0
$Type = algorithmic
$Parameters
        направление_вращения : such_as Детали.ориентация
        ориентация_детали    : such_as Детали.ориентация
$Body
        if направление_вращения = по_часовой_стрелке     result =
Длительность_ориентирования_детали_по_часовой_стрелке( ориентация_детали )
        if направление_вращения = против_часовой_стрелки result =
Длительность_ориентирования_детали_против_часовой_стрелки( ориентация_детали )
$End

$Function Как_ориентирована_деталь : real = 0
$Type = algorithmic
$Parameters
        ориентация_детали    : such_as Детали.ориентация
$Body
        if ориентация_детали = против_часовой_стрелки result = 1
        if ориентация_детали = по_часовой_стрелке     result = 0
$End
```

СКЛАД.pat (Образцы):

```
$Pattern Образец_поступления_новой_детали : irregular_event trace
$Relevant_resources
        _Система        : Система  Keep
        _Деталь         : Детали   Create
$Time = Интервал_поступления( 1.2)
$Body
_Система
        Convert_event
                количество_прибывших_деталей set _Система.количество_прибывших_деталей + 1
                текущее_число_деталей set _Система.текущее_число_деталей + 1

_Деталь
        Convert_event
                номер                 set _Система.количество_прибывших_деталей
                ориентация            set Ориентирование_детали
                состояние             set Свободна
$End
```

```
$Pattern Образец_захвата_детали_роботом : operation trace
$Relevant_resources
        _Система   : Система   Keep Keep
        _Деталь    : Детали    Keep Keep
        _Робот     : Робот     Keep Keep
$Time = Длительность_захвата_детали( 0.2, 0.02 )
$Body
_Система
        Choice from _Система.текущее_число_деталей > 0
        Convert_begin
                текущее_число_деталей  set _Система.текущее_число_деталей - 1
        Convert_end
                как_ориентированна_текущая_деталь  set _Деталь.ориентация

_Деталь
        Choice from _Деталь.состояние = Свободна
        with_min( _Деталь.номер )
        Convert_begin
                состояние  set Захват_начат
        Convert_end
                состояние  set Захват_закончен

_Робот
        Choice from _Робот.состояние = Свободен
        Convert_begin
                состояние  set Захват_начат
        Convert_end
                состояние  set Захват_закончен
$End

$Pattern Образец_ориентирования_детали : operation trace
$Parameters направление_вращения : such_as Детали.ориентация
$Relevant_resources
        _Деталь    : Детали    Keep Keep
        _Робот     : Робот     Keep Keep
$Time = Длительность_ориентирования_детали( направление_вращения, _Деталь.ориентация )
$Body
_Деталь
        Choice from _Деталь.состояние = Захват_закончен
        Convert_begin
                состояние  set Ориентирование_начато
        Convert_end
                состояние  set Ориентирование_закончено

_Робот
        Choice from _Робот.состояние = Захват_закончен
        Convert_begin
                состояние  set Ориентирование_начато
        Convert_end
                состояние  set Ориентирование_закончено
$End

$Pattern Образец_транспортировки_детали_роботом : operation trace
$Relevant_resources
        _Система   : Система   NoChange Keep
        _Деталь    : Детали    Keep     Keep
        _Робот     : Робот     Keep     Keep
$Time = Длительность_транспортировки_детали( 0.4, 0.04 )
$Body
_Система
        Convert_end
                количество_погруженных_деталей set _Система.количество_погруженных_деталей + 1
_Деталь
        Choice from _Деталь.состояние = Ориентирование_закончено
        Convert_begin
                состояние  set Транспортировка_начата
        Convert_end
                состояние  set Транспортировка_закончена

_Робот
        Choice from _Робот.состояние = Ориентирование_закончено
        Convert_begin
                состояние  set Транспортировка_начата
```

```
        Convert_end
                состояние  set Транспортировка_закончена
$End

$Pattern Образец_возврата_робота : operation trace
$Relevant_resources
        _Робот     : Робот     Keep Keep
$Time = Длительность_возврата_робота( 0.4, 0.04 )
$Body
_Робот
        Choice from _Робот.состояние = Транспортировка_закончена
        Convert_begin
                состояние  set Возврат
        Convert_end
                состояние  set Свободен
$End
```

СКЛАД.dpt (Точки принятия решений):

```
$Decision_point generate : some
$Condition Система.количество_прибывших_деталей < 20
$Activities
        Поступление_детали                              : Образец_поступления_новой_детали
$End

$Decision_point simple : some
$Condition NoCheck
$Activities
        Захват_детали                                   : Образец_захвата_детали_роботом
        Ориентирование_детали_против_часовой_стрелки : Образец_ориентирования_детали
против_часовой_стрелки
        Ориентирование_детали_по_часовой_стрелке      : Образец_ориентирования_детали
по_часовой_стрелке
        Транспортировка_детали                          : Образец_транспортировки_детали_роботом
        Возврат_робота                                  : Образец_возврата_робота
$End
```

СКЛАД.smr (Прогон):

```
Model_name     = sklad

Resource_file  = sklad
Statistic_file = sklad
Results_file   = sklad
Trace_file     = sklad
Show_mode      = NoShow
Show_rate      = 3600.0

Terminate_if Система.количество_погруженных_деталей >= 20 and Робот.состояние = Свободен
```

СКЛАД.pmd (Показатели):

```
$Results
        Робот_свободен          : watch_state Робот.состояние = Свободен
        Захват_детали           : watch_state Робот.состояние = Захват_начат
        Ориентирование_детали   : watch_state Робот.состояние = Ориентирование_начато
        Транспортировка_детали  : watch_state Робот.состояние = Транспортировка_начата
        Возврат_робота          : watch_state Робот.состояние = Возврат
        Длина_очереди           : watch_par   Система.текущее_число_деталей
        Поступило_деталей       : get_value   Система.количество_прибывших_деталей
        Погружено_деталей       : get_value   Система.количество_погруженных_деталей
$End
```