Государственное образовательное учреждение высшего профессионального образования

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

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ РК (Робототехника и комплексная автоматизация)

КАФЕДРА РК9 (Компьютерные системы автоматизации производства)

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

к курсовому проекту на тему:

Разработка подсистемы анимации вывода временных ресурсов РДО

Студент группы РК9-101 _____ А. А. Копнин
(Подпись, дата) (И.О.Фамилия)

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

Москва, 2010

Государственное образовательное учреждение высшего профессионального образования

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

УТВЕРЖДАЮ

Заведующий кафедрой __РК__
(Индекс)

_____ _____
(И.О.Фамилия)
« ___ » _____ 20 __ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине__Разработка подсистемы анимации вывода временных ресурсов РДО

(Тема курсового проекта)

Студент_____Копнин А.А_____РК9-101
(Фамилия, инициалы, индекс группы)

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

1. Техническое задание

Разработать подсистему анимации вывода временных ресурсов РДО

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на __44_ листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) Лист 1: А1 Постановка_задачи; Лист 2 А2 Диаграмма компонентов, А2 IDEF0 Функционирование ПАВР; Лист 3: А2 IDEF0 Декомпозиция функционирование ПАВР, А2_ IDEF0 Декомпозиция Произвести отображение временных ресурсов; Лист 4: А2 Диаграмма классов, А2 Синтаксическая диаграмма; Лист 5: А1 Результаты курсового проектирования.

Дата выдачи задания « ____ » _____ 2010г.

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

Студент _____ А. А. Копнин
(Подпись, дата) (И.О.Фамилия)

Примечание:
1. Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

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 (*Рисунок 1*.

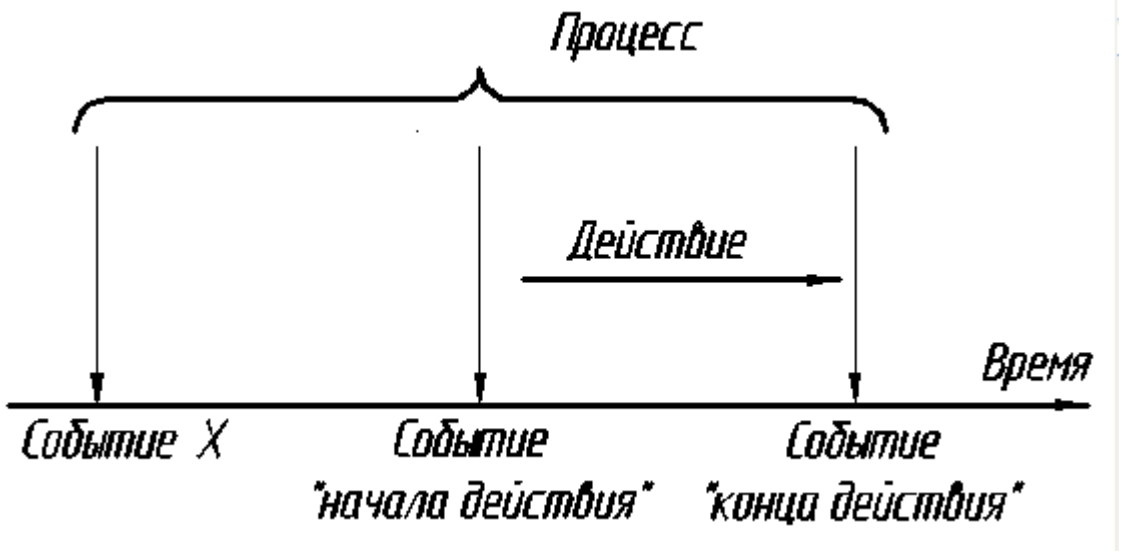

Рисунок 1. Взаимосвязь между событиями, действием и процессом.

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 (Рисунок 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.

Рисунок 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 (Рисунок 3). 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.

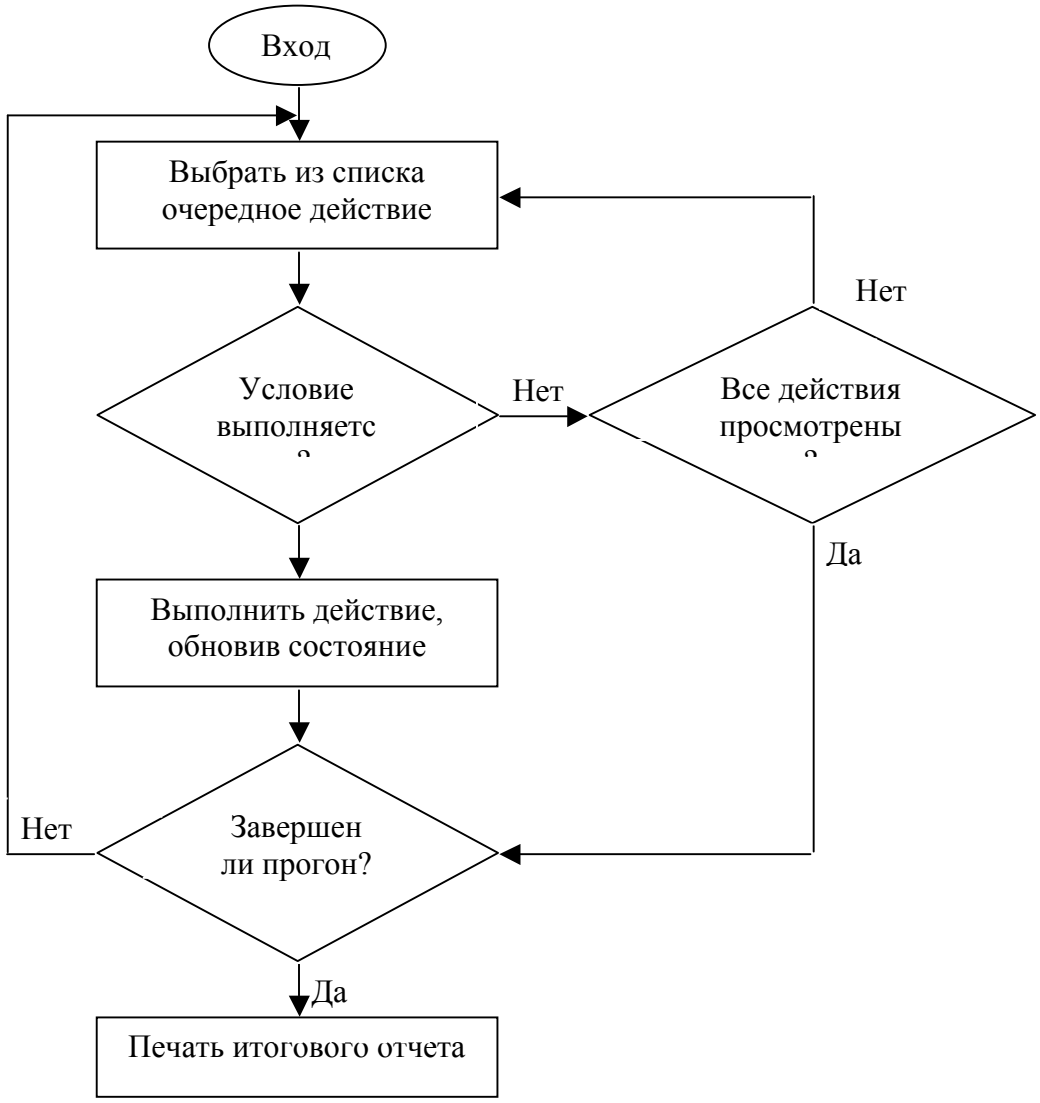

Рисунок 3. Блок-схема реализации подхода сканирования активностей.

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. Состав системы

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

Существующая среда имитационного моделирования RAO-studio имеет в своем составе главное окно, окно объектов и окно вывода.

Рассмотрим эти элементы подробнее.

1.5. Главное окно

Главное окно имеет главное меню, кнопки панели управления, "плавающие" окна объектов и вывода со своими закладками, строку состояния и окно с набором закладок ("PAT", "RTP", "RSS", "OPR", "FRM", "FUN", "DPT", "SMR", "PMD"), соответствующих объектам модели для редактирования новой модели(Рисунок 4).

Переключаясь между закладками, пользователь может редактировать различные элементы модели.

В поле редактирования отображается текст соответствующего объекта. Поле редактирования имеет набор полей для отображения служебной информации (номера строк, границы сворачиваемого фрагмента текста, закладки). Эти поля отображаются слева от редактируемого текста. Поле редактирования имеет выпадающее меню, активизируемое при нажатии на правую кнопку мыши. Это меню содержит функции для вставки синтаксических конструкций языка и шаблонов, работы с буфером обмена, выделения, а также поиска и замены фрагментов текста

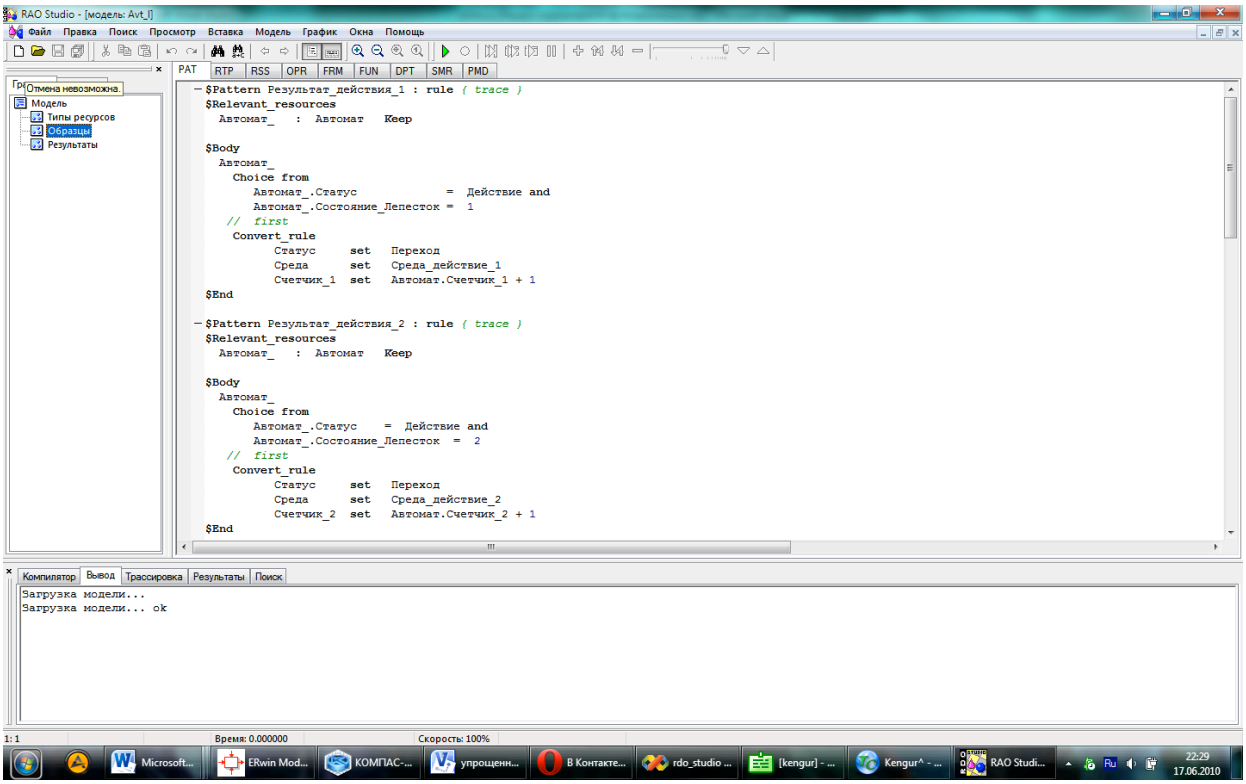

Рисунок 4. Общий вид системы ИМ РДО

1.6. Окно объектов

Сбоку от главного окна располагается плавающее окно объектов модели с закладками. В окне есть следующие закладки:

- Графики - Отображает список трассируемых объектов в процессе моделирования, позволяет отобразить график по выбранному элементу
- Анимация - Отображает список доступных окон анимации в процессе моделирования, позволяет переключиться на выбранное окно

1.7. Окно вывода

Располагается внизу и имеет следующие закладки:

- Компилятор - отображает информацию о процессе компиляции модели, выводит сообщения об ошибках компиляции, позволяет переключиться в текст модели на место возникновения выбранной ошибки
- Вывод - окно, в которое любой модуль может вывести отладочную информацию, например, здесь отображается список закруженных для анимации картинок, при запуске модели на исполнение
- Трассировка - отображает файл трассировки процесса моделирования в режиме реально времени
- Результаты - отображает результаты моделирования по завершении прогона
- Поиск - отображает результаты поиска подстроки по всей модели, позволяет переключиться в текст модели на найденный фрагмент

1.8. Возможности развития

Используя текущие возможности среды имитационного моделирования RAO-studio возможно создавать модели различной степени сложности. Модель создается путем написания текста модели. Программный комплекс решает следующие задачи:

- синтаксический разбор текста модели и настраиваемая подсветка синтаксических конструкций языка РДО;
- открытие и сохранение моделей;
- расширенные возможности для редактирования текстов моделей;
- автоматическое завершение ключевых слов языка;
- поиск и замена фрагментов текста внутри одного модуля модели;
- поиск интересующего фрагмента текста по всей модели;
- навигация по тексту моделей с помощью закладок;
- наличие нескольких буферов обмена для хранения фрагментов текста;
- вставка синтаксических конструкций языка и заготовок (шаблонов) для написания элементов модели;

- настройка отображения текста моделей, в т.ч. скрытие фрагментов текста и масштабирование;
- запуск и остановка процесса моделирования;
- изменение режима моделирования;
- изменение скорости работающей модели;
- переключение между кадрами анимации в процессе моделирования;
- отображение хода работы модели в режиме реального времени;
- построение графиков изменения интересующих разработчика характеристик в режиме реального времени;
- обработка синтаксических ошибок при запуске процесса моделирования;
- обработка ошибок во время выполнения модели;
- обеспечение пользователя справочной информацией.

Добавление элементов визуального программирования на языке РДО позволит упростить создание моделей, сделав этот процесс более наглядным и удобным для пользователя.

1.9. Постановка задачи

Основная идея дипломного проекта – добавление в систему анимации системы имитационного моделирования РДО подсистемы анимации временных ресурсов (далее ПАВР), позволяющей выводить на экран монитора состояние временных ресурсов модели и его изменение в процессе прогона модели. Также планируется добавление возможности проведения аффинных преобразований над объектами анимации.

Основная идея курсового проекта – разобраться в механизмах работы и функционирования системы имитационного моделирования РДО, провести пробное добавление нового элемента анимации – объекта Окружность (Circle), отладка его работы, достичь компиляции и работоспособности программы.

Сейчас отображение временных ресурсов невозможно само по себе, так как существующая подсистема анимации может выводить на экран только те ресурсы, у которых есть имя, а временные ресурсы приходят и уходят из системы (они имеют только тип), занимая лишь какой – то адрес в памяти. Существует механизм вывода параметров временных ресурсов с использованием заранее заведенных постоянных ресурсов, но он неудобен по следующим причинам:

1) Мы значительно увеличиваем размер модели за счет введения дополнительных ресурсов;

2) Увеличивается сложность возможности добавления и вывода временных ресурсов сверх заведенного заранее количества.

Необходимо «научить» подсистему анимации РДО выводить ресурсы и их параметры не только по имени, но и по типу.

Также и аффинные преобразования (рассмотрим на примере масштаба) – в настоящее время вывод происходит следующим образом:

line [Координаты_узлов(X,1) * Const.Scale + Const.dX, Координаты_узлов(У,1) * Const.Scale + Const.dY, Координаты_узлов(X,2) * Const.Scale + Const.dX, Координаты_узлов(У,2) * Const.Scale + Const.dY, <0 0 0>],

т.е. каждая координата умножается на масштаб, что значительно загромождает пространство модели и увеличивает ее объем. Необходимо, чтобы описание масштабных операций в модели происходило по более простому механизму, например:

line [Координаты_узлов(X,1) , Координаты_узлов(У,1), Координаты_узлов(X,2), Координаты_узлов(У,2), <0 0 0>] *[Const.Scale+ Const.dX, Const.Scale+ Const.dY].

Проблема курсового проектирования - построение окружности производится как частный случай построения эллипса, что не всегда удобно. Эллипс описывается следующей строкой в модели:

ellipse [400, 200, 100, 200, <255 0 0>, <255 0 0>]

- первое значение в скобках - координата X верхнего левого угла прямоугольника;

- второе значение в скобках - координата Y верхнего левого угла прямоугольника;

- третье значение в скобках - ширина прямоугольника;

- четвертое значение в скобках - высота прямоугольника;

- пятое значение в скобках - цвет заполнения прямоугольника;

- шестое значение в скобках – цвет границы прямоугольника;

По этим значениям эллипс строится вписыванием в прямоугольник заданных размеров.

Хотелось бы, чтобы окружность описывалась в модели следующим способом:

circle [400, 200, 100, <255 0 0>, <255 0 0>]

- первое значение в скобках - координата X центра окружности;

- второе значение в скобках - координата Y центра окружности;

- третье значение в скобках - радиус окружности;

- пятое значение в скобках - цвет заполнения окружности;

- шестое значение в скобках – цвет границы окружности;

Таким образом, целью курсового проекта является добавление в систему имитационного моделирования РДО возможности описания в модели и построения нового объекта – окружности.

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. 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 (Рисунок
5).

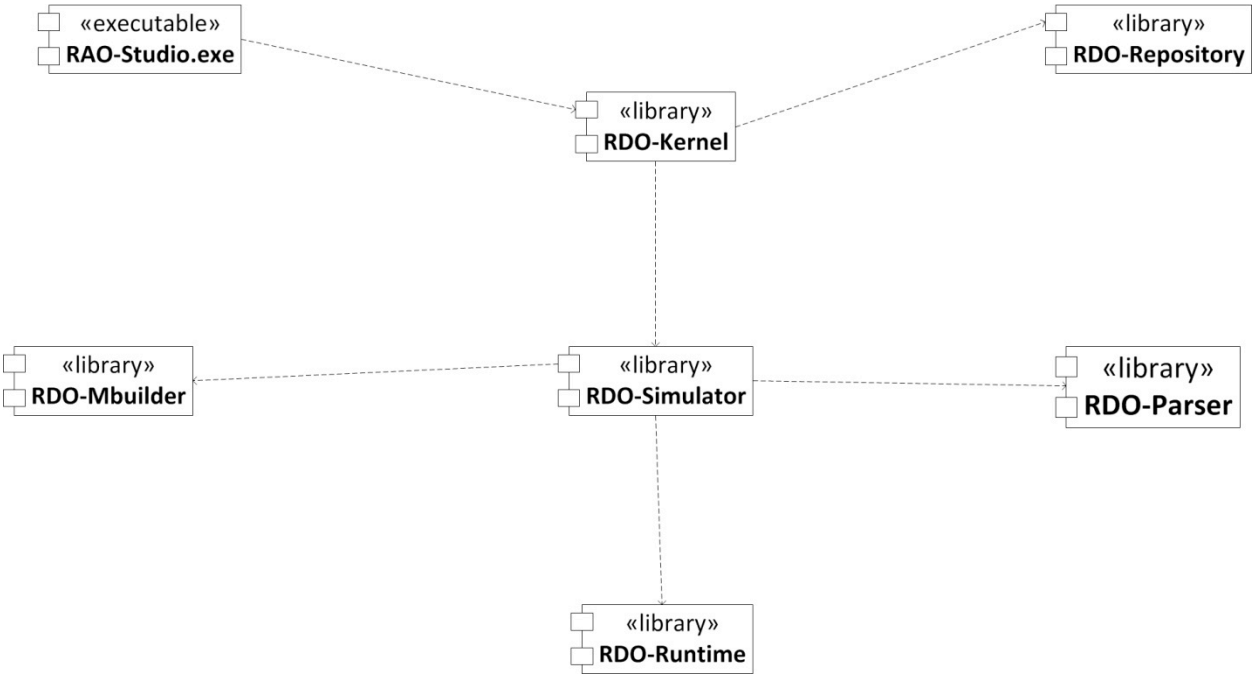

Рисунок 5. Диаграмма компонентов.

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 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 рекусивное вложение логик внутрь 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 META-логика. Далее от корня дерева к листьям распространяется волна вызовов метода onCheckCondition(). Т.е. onCheckCondition() вызывается у META-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 META-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 META-логики по убыванию приоритета и в дальнейшем производить поиск в отсортированном списке.

2.3. Техническое задание

Техническое задание на создание, развитие и модернизацию Автоматизированной системы по [1] и [2].

2.3.1. Общие сведения

Объектом исследования является система имитационного моделирования RAO - Studio.

Сведения о работе данной системы были получены в ходе изучения учебного курса по дисциплине Моделирование ТП и ПП за 9 – й учебный семестр. В качестве источников сведений о данной системе была использована справочная документация.

Работу по исследованию данной системы требуется завершить до окончания 2 - го семестра 2009/2010 учебного года. Как результат работ по исследованию данной системы требуется предъявить комплект документации, а также работающую и выполняющую поставленные перед ней задачи, версию программы.

2.3.2. Назначение и цели развития системы

Требуется усовершенствовать возможности подсистемы анимации РДО для того, чтобы можно было следить за изменением параметров временных ресурсов. Это усовершенствование даст нам возможность расширить наши знания о процессах, происходящих во время работы модели.

В результате необходимо получить выведенную на экран таблицу с параметрами временных ресурсов. Как цель текущего курсового проекта – адекватное восприятие системой ИМ РДО нового объекта Окружность.

2.3.3. Характеристики объекта автоматизации

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

2.3.4. Требования к системе

2.3.4.1. Требования к структуре и функционированию системы

1) ПАВР должна быть реализована в системе ИМ РДО с помощью языка C++;

2) ПАВР должна корректно отображать состояние временных ресурсов на экране монитора.

2.3.4.2. Требования к численности и квалификации персонала на АС

1) Для модуля ПАВР РДО необходимо умение работать с системами ИМ.

2.3.4.3. Требования к применению в системе языков программирования высокого уровня, языков взаимодействия пользователей и технических средств системы

1) ПАВР должна быть написана с использованием синтаксиса C++ и РДО.

2.3.4.4. Требование к функциям (задачам)

1) Необходимо автоматизировать вывод временных ресурсов на экран монитора (исключить участие в выводе постоянных ресурсов);

2) Добавить возможность аффинных преобразований;

2.3.5. Состав и содержание работ по созданию системы

В процессе работы над системой требуется получить следующий список документов:

- Диаграмма IDEF0, описывающая процесс взаимосвязи существующей подсистемы анимации и проектируемой подсистемы анимации временных ресурсов (ПАВР);

- Диаграмма классов, описывающая классы, используемые подсистемой анимации;

- Диаграмма компонентов, описывающая внутреннюю структуру системы и показывающая связь компонентов, которые были внесены изменения.

2.3.6. Порядок контроля и приемки системы

Полученную версию программу необходимо откомпилировать и проверить на работоспособность путем использования в модели новых, добавленных конструкций и получении адекватного результата.

3.Технический этап проектирования

3.1. Разработка синтаксиса описания объекта Окружность

Окружность имеет следующий формат:

circle [арифметическое выражение, арифметическое выражение, арифметическое выражение, цвет, цвет]. (Рисунок 8)

Арифметическое выражение должно иметь целый или вещественный тип.

3.2. Разработка синтаксиса описания ПАВР

В данный момент вывод временных ресурсов осуществляется с использованием постоянных ресурсов системы (имеющих имя). Необходимо убрать все ресурсы «посредники»(Рисунок
6)

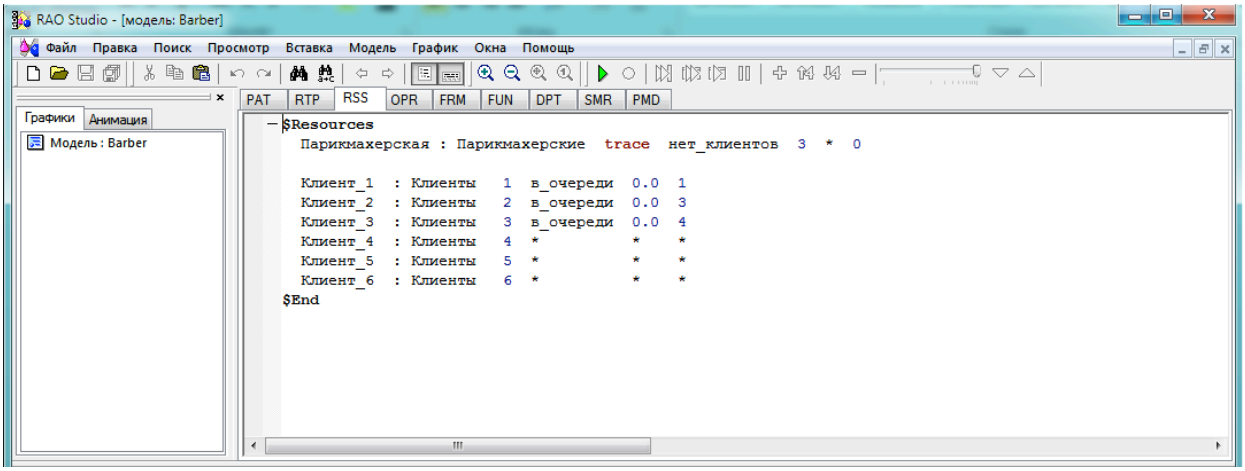

Рисунок 6. Вкладка РДО "Ресурсы"

и научить систему видеть непосредственно временные ресурсы (Рисунок 7) – выводить их на экран через тип.

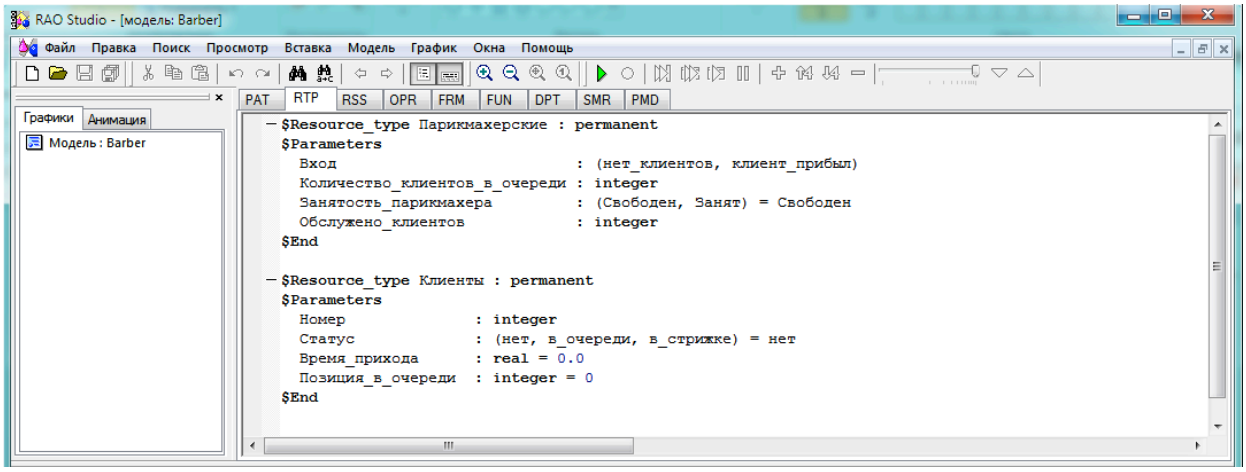

Рисунок 7. Вкладка РДО "Типы ресурсов"

В данный момент система выводит на экран ресурс по следующему принципу (из вкладки FRM): Клиент_1.Статус = в_очереди - Клиент_1-это имя постоянного ресурса. Вывод временных ресурсов будет осуществляться по их типу, т.е. будет существовать временный ресурс – Клиент и для него состояния (а не расписывать для каждого клиента каждую возможность). Т.е. например вместо кода:

```
Show_if Клиент_1.Статус = в_очереди
  bitmap [420-Клиент_1.Позиция_в_очереди*70, 160, man1, man1m]
Show_if Клиент_2.Статус = в_очереди
  bitmap [420-Клиент_2.Позиция_в_очереди*70, 177, man2s, man2m]
будет идти:
Show_if Клиент_i.Статус = в_очереди
  bitmap [420-Клиент_i.Позиция_в_очереди*70,  j, <man1, man2s >, <man1m, man2m >]
```

3.3. Разработка архитектуры компонента rdo_parser

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

Необходимо добавить и зарезервировать ключевое слово **circle,** также зарезервировать место в памяти под новый ресурс.

Для ПАВР необходимо добавить новую структуру типа Sprite. Он будет представлять собой полноценный Frame, давать возможность вывода временных ресурсов с помощью цикла, описываемого в самой системе, а в модели будет прописываться лишь сам временный ресурс и его состояния.

3.4. Разработка архитектуры компонента rdo_runtime

Необходимо добавить и описать само понятие окружности – какими именно параметрами будет характеризоваться это понятие (координаты центра окружности, радиус и цвета).

Sprite - будет представлять собой полноценный Frame, давать возможность вывода временных ресурсов с помощью цикла, описываемого в самой системе, а в модели будет прописываться лишь сам временный ресурс и его состояния.

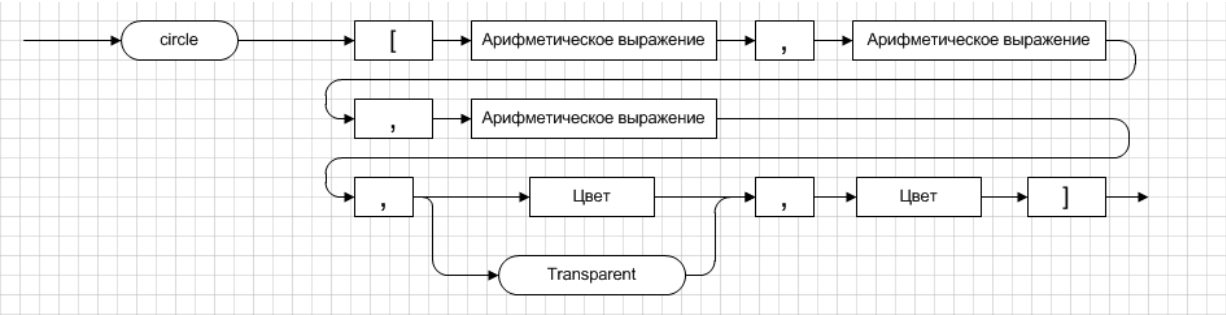

Рисунок 8. Синтаксическая структура объекта Окружность

4. Рабочий этап проектирования

4.1. Изменения в пространстве имен rdoParser

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

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

```
%token RDO_circle                                448
```

в файлы rdodpt.y, rdofrm.y, rdofun.y, rdoopr.y, rdopat.y, rdopmd.y, rdoproc_opr.y, rdoproc_rss.y, rdoproc_rtp.y, rdorss.y, rdortp.y, rdosmr_file.y, rdosmr_sim.y.

во-вторых, описание (в файл rdofrm.y)

```
 frm_circle: RDO_circle '[' frm_position_xy ',' frm_position_xy ','
frm_circle_radius ',' frm_color ',' frm_color ']' {
                        rdoRuntime::RDOFRMFrame::RDOFRMPosition* x      =
reinterpret_cast<rdoRuntime::RDOFRMFrame::RDOFRMPosition*>($3);
                        rdoRuntime::RDOFRMFrame::RDOFRMPosition* y      =
reinterpret_cast<rdoRuntime::RDOFRMFrame::RDOFRMPosition*>($5);
                        rdoRuntime::RDOFRMFrame::RDOFRMPosition* radius  =
reinterpret_cast<rdoRuntime::RDOFRMFrame::RDOFRMPosition*>($7);
                        rdoRuntime::RDOFRMFrame::RDOFRMColor* bg_color =
reinterpret_cast<rdoRuntime::RDOFRMFrame::RDOFRMColor*>($9);
                        rdoRuntime::RDOFRMFrame::RDOFRMColor* fg_color =
reinterpret_cast<rdoRuntime::RDOFRMFrame::RDOFRMColor*>($11);
                        bg_color->setColorType(
rdoRuntime::RDOFRMFrame::RDOFRMColor::color_last_bg );
                        fg_color->setColorType(
rdoRuntime::RDOFRMFrame::RDOFRMColor::color_last_fg );
                        $$ = (int)new rdoRuntime::RDOFRMCircle( RUNTIME-
>lastFrame(), x, y, radius, bg_color, fg_color );
                    }
                    | RDO_circle '[' frm_position_xy ',' frm_position_xy ','
frm_circle_radius ',' frm_color ',' frm_color error {
                        PARSER->error().error( @11, "Ожидается ']'" );
                    }
                    | RDO_circle '[' frm_position_xy ',' frm_position_xy ','
frm_circle_radius ',' frm_color ',' error {
                        PARSER->error().error( @10, @11, "Ожидается цвет линии
круга" );
                    }
                    | RDO_circle '[' frm_position_xy ',' frm_position_xy ','
frm_circle_radius ',' frm_color error {
                        PARSER->error().error( @9, "Ожидается запятая" );
                    }
                    | RDO_circle '[' frm_position_xy ',' frm_position_xy ','
frm_circle_radius ',' error {
```

```
                        PARSER->error().error( @8, @9, "Ожидается цвет фона"
);
                  }
                  | RDO_circle '[' frm_position_xy ',' frm_position_xy ','
frm_circle_radius error {
                        PARSER->error().error( @7, "Ожидается запятая" );
                  }
                  | RDO_circle '[' frm_position_xy ',' frm_position_xy ','
error {
                        PARSER->error().error( @6, @7, "Ожидается радиус" );
                  }
                  | RDO_circle '[' frm_position_xy ',' frm_position_xy error {
                        PARSER->error().error( @5, "Ожидается запятая" );
                  }
                  | RDO_circle '[' frm_position_xy ',' error {
                        PARSER->error().error( @4, @5, "Ожидается координата
по оси Y" );
                  }
                  | RDO_circle '[' frm_position_xy error {
                        PARSER->error().error( @3, "Ожидается запятая" );
                  }
                  | RDO_circle '[' error {
                        PARSER->error().error( @2, @3, "Ожидается координата
по оси X" );
                  }
                  | RDO_circle error {
                        PARSER->error().error( @1, "Ожидается '['" );

                        };
```

Этот фрагмент кода имеет следующий смысл:

– при неправильном описании в тексте модели входных параметров для системы ИМ будут выводится соответствующие ошибки;

– происходит перенаправление параметров в другие файлы системы, занимающиеся непосредственно отрисовкой.

в - третьих, описание (в файл rdofrm.y)

```
frm_circle_radius:    fun_arithm frm_postype_xy
                      {
                            if ($2 !=
rdoRuntime::RDOFRMFrame::RDOFRMPosition::absolute && $2 !=
rdoRuntime::RDOFRMFrame::RDOFRMPosition::delta &&$2 !=
rdoRuntime::RDOFRMFrame::RDOFRMPosition::mult )
                                  PARSER->error().error( @2, "Нельзя
использовать данное выравнивание для радиуса" );

                            rdoRuntime::RDOCalc* calc =
reinterpret_cast<RDOFUNArithm*>($1)->createCalc();
                            $$ = (int)new
rdoRuntime::RDOFRMFrame::RDOFRMPosition( RUNTIME->lastFrame(), calc,
(rdoRuntime::RDOFRMFrame::RDOFRMPosition::PositionType)$2 );
```

```
};
```

Этот фрагмент кода имеет следующий смысл:

- при неправильном описании в тексте модели относительных знаков будет выводиться ошибка.

4.2. Изменения в пространстве имен rdoRuntime

Файл rdoframe.cpp:

```
RDOFRMCircle::RDOFRMCircle( RDOFRMFrame* parent,
RDOFRMFrame::RDOFRMPosition* x, RDOFRMFrame::RDOFRMPosition* y,
RDOFRMFrame::RDOFRMPosition* radius, RDOFRMFrame::RDOFRMColor* bgColor,
RDOFRMFrame::RDOFRMColor* fgColor ):
  RDOFRMItem( parent ),
  RDOFRMColoredItem( bgColor, fgColor ),
  m_x( x ),
  m_y( y ),
  m_radius( radius )
{
  color_reparent( this );
}

rdoAnimation::FrameItem* RDOFRMCircle::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDOCircleElement) );

  rdoAnimation::RDOColor bg = getBg( sim, getFrame() );
  rdoAnimation::RDOColor fg = getFg( sim, getFrame() );
  getFrame()->setColorLastBG( getBgColor()->getColorType(), bg );
  getFrame()->setColorLastFG( getFgColor()->getColorType(), fg );

  return new rdoAnimation::RDOCircleElement(
        rdoAnimation::RDOPoint(m_x->getX( sim, getFrame() ), m_y->getY(
sim, getFrame() )),
        rdoAnimation::RDORadius(m_radius->getX( sim, getFrame() )),
        rdoAnimation::RDOColoredElement(bg, fg)
  );

}
```

Описание класса RDOFRMCircle

```
  class RDOFRMCircle: public RDOFRMItem, public RDOFRMColoredItem
{
private:
  RDOFRMFrame::RDOFRMPosition* m_x;
  RDOFRMFrame::RDOFRMPosition* m_y;
  RDOFRMFrame::RDOFRMPosition* m_radius;
protected:
  virtual rdoAnimation::FrameItem* createElement( RDORuntime* sim );

public:
  RDOFRMCircle( RDOFRMFrame* parent, RDOFRMFrame::RDOFRMPosition* x,
RDOFRMFrame::RDOFRMPosition* y, RDOFRMFrame::RDOFRMPosition* radius,
RDOFRMFrame::RDOFRMColor* bgColor, RDOFRMFrame::RDOFRMColor* fgColor );
};
```

Данным программным кодом мы описываем характеристики нашего нового элемента, его составляющие.

4.3. Изменения в пространстве имен rdoStudio

Резервируем системные ресурсы, необходимые для нормального функционирования системы (файл resource.h)

```
#define ID_INSERT_FRM_CIRCLE            33017
```

Файл rdostudioframemanager.cpp. В данном файле происходит обращение к функциям WinApi.

```
case rdoAnimation::FrameItem::FIT_CIRCLE:
                                {
                                      rdoAnimation::RDOCircleElement* element =
static_cast<rdoAnimation::RDOCircleElement*>(currElement);
                                      HBRUSH brush = ::CreateSolidBrush(
RGB(element->m_background.m_r, element->m_background.m_g, element-
>m_background.m_b) );
                                      HBRUSH pOldBrush;
                                      if( !element->m_background.m_transparent )
{
                                            pOldBrush =
static_cast<HBRUSH>(::SelectObject( hdc, brush ));
                                      } else {
                                            pOldBrush =
static_cast<HBRUSH>(::GetStockObject( NULL_BRUSH ));
                                      }

                                      HPEN pen     = NULL;
                                      HPEN pOldPen = NULL;
                                      if( !element->m_foreground.m_transparent )
{
                                            pen     = ::CreatePen( PS_SOLID, 0,
RGB(element->m_foreground.m_r, element->m_foreground.m_g, element-
>m_foreground.m_b) );
                                            pOldPen =
static_cast<HPEN>(::SelectObject( hdc, pen ));
                                      }

                                      ::Ellipse( hdc, (int)(element-
>m_center.m_x - element->m_radius.m_radius), (int)(element->m_center.m_y -
element->m_radius.m_radius), (int)(element->m_center.m_x + element-
>m_radius.m_radius), (int)(element->m_center.m_y + element-
>m_radius.m_radius) );

                                      ::SelectObject( hdc, pOldBrush );
                                      ::DeleteObject( brush );
                                      if ( pen ) {
                                            ::SelectObject( hdc, pOldPen );
                                            ::DeleteObject( pen );
                                            }
```

4.4. Изменения в пространстве имен rdoCommon

Файл rdoanimation.inl

```
inline RDOCircleElement::RDOCircleElement(CREF(RDOPoint) center,
CREF(RDORadius) radius, CREF(RDOColoredElement) color)
   : FrameItem        (FIT_CIRCLE)
   , RDOColoredElement(color     )
   , m_center         (center    )
   , m_radius         (radius    )
{}
```

5. Результаты

Скриншот работающей версии программы (модель строит окружность по заданным параметрам – координаты центра (X, Y) = 200, 100, радиус = 50, цвет заливки = <255 0 0>, цвет границы = <0 0 255>) (Рисунок 9)

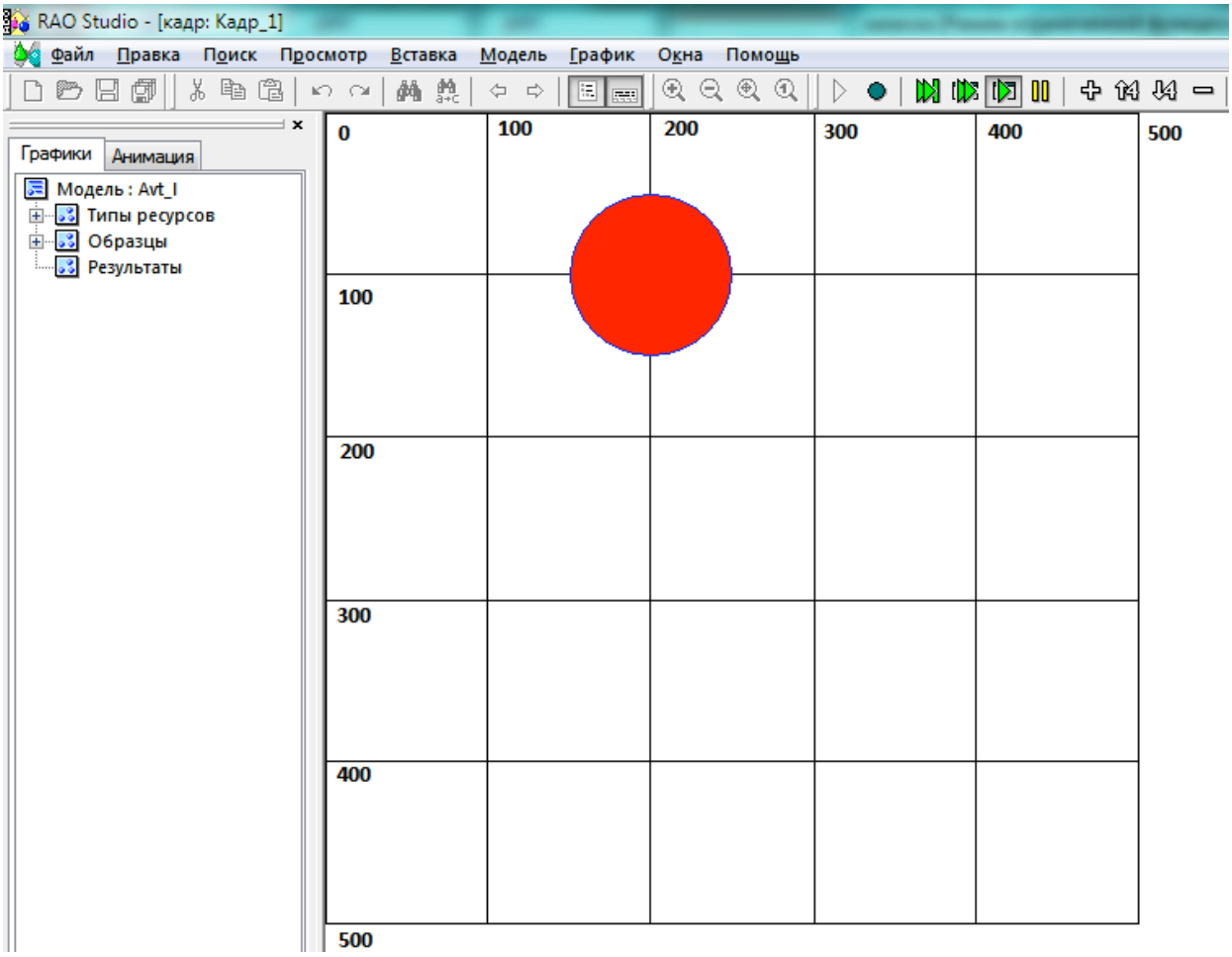

Рисунок 9. Экранная форма работающей программы

6. Выводы

Было проведено изучение принципов работы системы ИМ РДО и принципов добавления новых объектов в нее. После этого была проведены успешная компиляция, запуск программы и тестирование правильности построения системой ИМ нового типа объектов. Отображение объекта Окружность в точности соответствует поставленным требованиям. Поставленная цель курсового проекта достигнута.

7. Заключение

В рамках данного курсового проекта были получены следующие результаты:

1) Проведено предпроектное исследование системы имитационного моделирования РДО и сформулированы предпосылки дополнения существующей подсистемы анимации.

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

3) На этапе технического проектирования разработан синтаксис описания нового объекта, который приведен на синтаксической диаграмме.

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) Добавлен новый объект Окружность (Circle), который характеризуется следующими параметрами – координаты X и Y центра окружности, радиус окружности, цвет заполнения окружности и границы окружности.

6) Проведено успешное тестирование нового объекта Окружность (Circle) в модели (Рисунок 10).

7) Проведено изучение структуры и механизмов работы системы ИМ РДО,

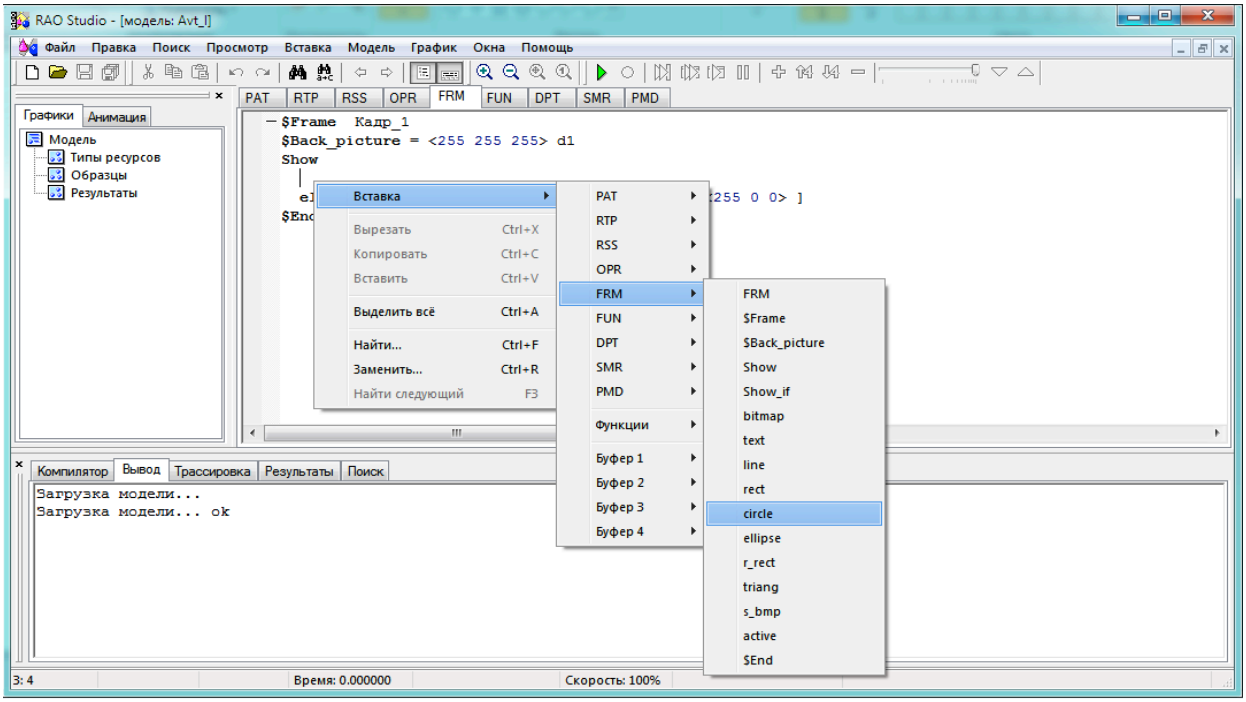

Рисунок 10. Возможность применения нового ключевого слова Окружность (Circle)

8. Приложение

8.1. Листинг файла rdoframe.cpp

```
#include "rdo_lib/rdo_runtime/pch.h"
#include "rdo_lib/rdo_runtime/rdoframe.h"
#include "rdo_lib/rdo_runtime/rdo_runtime.h"

namespace rdoRuntime
{

// --------------------------------------------------------------------
-----
// ---------- RDOFRMColor - объект-цвет
// --------------------------------------------------------------------
-----
RDOFRMFrame::RDOFRMColor::RDOFRMColor( RDOFRMFrame* _parent, ColorType
_type ):
   RDORuntimeObject( _parent ),
   color_type( _type ),
   red_calc( NULL ),
   green_calc( NULL ),
   blue_calc( NULL )
{
}

RDOFRMFrame::RDOFRMColor::RDOFRMColor( RDOFRMFrame* _parent, int _red, int
_green, int _blue ):
   RDORuntimeObject( _parent ),
   color_type( color_rgb )
{
   red_calc   = new RDOCalcConst( _parent, _red );
   green_calc = new RDOCalcConst( _parent, _green );
   blue_calc  = new RDOCalcConst( _parent, _blue );
   red_calc->setSrcText( rdo::format("%d, _red") );
   green_calc->setSrcText( rdo::format("%d, _green") );
   blue_calc->setSrcText( rdo::format("%d, _blue") );
}

RDOFRMFrame::RDOFRMColor::RDOFRMColor( RDOFRMFrame* _parent, RDOCalc*
_red_calc, RDOCalc* _green_calc, RDOCalc* _blue_calc ):
   RDORuntimeObject( _parent ),
   color_type( color_rgb ),
   red_calc( _red_calc ),
   green_calc( _green_calc ),
   blue_calc( _blue_calc )
{
}

RDOFRMFrame::RDOFRMColor::~RDOFRMColor()
{
}

rdoAnimation::RDOColor RDOFRMFrame::RDOFRMColor::getColor(PTR(RDORuntime)
sim, PTR(RDOFRMFrame) frame) const
{
   switch (color_type)
   {
         case color_none        : return rdoAnimation::RDOColor(50, 200,
50);
```

```
        case color_rgb         : return rdoAnimation::RDOColor(red_calc-
>calcValue(sim).getInt(), green_calc->calcValue(sim).getInt(), blue_calc-
>calcValue(sim).getInt());
        case color_transparent : return rdoAnimation::RDOColor();
        case color_last_bg     : return frame->color_last_bg;
        case color_last_fg     : return frame->color_last_fg;
        case color_last_bg_text: return frame->color_last_bg_text;
        case color_last_fg_text: return frame->color_last_fg_text;
        default                : NEVER_REACH_HERE;
   }
   return rdoAnimation::RDOColor();
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMFrame
// ----------------------------------------------------------------------
-----
RDOFRMFrame::RDOFRMFrame( RDORuntime* _runtime, const RDOSrcInfo&
_src_info, RDOCalc* _conditionCalc ):
   RDORuntimeParent( _runtime ),
   RDOSrcInfo( _src_info ),
   conditionCalc( _conditionCalc ),
   background_color( NULL ),
   picFileName(_T("")),
   width( 800 ),
   height( 600 ),
   last_x( 0 ),
   last_y( 0 ),
   last_width( 0 ),
   last_height( 0 )
{
   _runtime->addRuntimeFrame( this );
   color_last_bg      = rdoAnimation::RDOColor(50, 200, 50);
   color_last_fg      = color_last_bg;
   color_last_bg_text = color_last_bg;
   color_last_fg_text = color_last_bg;
}

RDOFRMFrame::~RDOFRMFrame()
{
   std::vector< RDOFRMRulet* >::iterator it = rulets.begin();
   while ( it != rulets.end() ) {
        delete *it;
        it++;
   }
}

void RDOFRMFrame::setColorLastBG( RDOFRMColor::ColorType type, const
rdoAnimation::RDOColor& _last_bg )
{
   if ( type == RDOFRMColor::color_rgb ) {
        color_last_bg = _last_bg;
   }
}

void RDOFRMFrame::setColorLastFG( RDOFRMColor::ColorType type, const
rdoAnimation::RDOColor& _last_fg )
{
   if ( type == RDOFRMColor::color_rgb ) {
        color_last_fg = _last_fg;
   }
}
```

```
void RDOFRMFrame::setColorLastBGText( RDOFRMColor::ColorType type, const
rdoAnimation::RDOColor& _last_bg_text )
{
  if ( type == RDOFRMColor::color_rgb ) {
        color_last_bg_text = _last_bg_text;
  }
}

void RDOFRMFrame::setColorLastFGText( RDOFRMColor::ColorType type, const
rdoAnimation::RDOColor& _last_fg_text )
{
  if ( type == RDOFRMColor::color_rgb ) {
        color_last_fg_text = _last_fg_text;
  }
}

void RDOFRMFrame::setBackgroundColor( RDOFRMColor* _background_color )
{
  background_color = _background_color;
}

void RDOFRMFrame::setBackPicture( const std::string& _picFileName )
{
  picFileName = _picFileName;
}

void RDOFRMFrame::setBackPicture( int _width, int _height )
{
  picFileName = _T("");
  width = _width;
  height = _height;
}

void RDOFRMFrame::startShow( RDOCalc* calc )
{
  shows.push_back( new RDOFRMShow( this, calc ) );
}

void RDOFRMFrame::addItem( RDOFRMItem* item )
{
  if ( shows.empty() ) startShow();
  item->reparent( shows.back() );
}

void RDOFRMFrame::addRulet( RDOFRMRulet* rulet )
{
  rulets.push_back( rulet );
}

bool RDOFRMFrame::checkCondition( RDORuntime* sim )
{
  if ( !conditionCalc ) return true;
  return conditionCalc->calcValue( sim ).getAsBool();
}

rdoAnimation::RDOFrame* RDOFRMFrame::prepareFrame( rdoAnimation::RDOFrame*
frame, RDORuntime* sim )
{
  if (background_color)
  {
        if (background_color->getColorType() == RDOFRMColor::color_rgb)
        {
```

```
            rdoAnimation::RDOColor bg_color = background_color-
>getColor(sim, this);
            frame->m_bgColor = bg_color;
        }
        else
        {
            frame->m_bgColor = rdoAnimation::RDOColor();
        }
  }
  else
  {
        frame->m_bgColor = rdoAnimation::RDOColor();
  }
  frame->m_bgImageName   = picFileName;
  frame->m_size.m_width  = width;
  frame->m_size.m_height = height;

  last_x      = 0;
  last_y      = 0;
  last_width  = 0;
  last_height = 0;

  if ( checkCondition( sim ) ) {
        std::list< RDOFRMShow* >::iterator it_show = shows.begin();
        while ( it_show != shows.end() ) {
            if ( (*it_show)->checkCondition(sim) ) {
                ChildList::iterator it_obj = (*it_show)-
>m_childList.begin();
                while ( it_obj != (*it_show)->m_childList.end() ) {
                    rdoAnimation::FrameItem* element =
static_cast<RDOFRMItem*>(*it_obj)->createElement(sim);
                    if ( element ) {
                        frame->m_elements.push_back( element );
                    }
                    it_obj++;
                }
            }
            it_show++;
        }
  }

  return frame;
}

void RDOFRMFrame::getBitmaps( std::list< std::string >& list ) const
{
  if (!picFileName.empty())
        list.push_back(picFileName);

  std::list< RDOFRMShow* >::const_iterator it = shows.begin();
  while ( it != shows.end() ) {
        (*it)->getBitmaps( list );
        it++;
  }
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMText
// ----------------------------------------------------------------------
-----
RDOFRMText::RDOFRMText( RDOFRMFrame* _parent, RDOFRMFrame::RDOFRMPosition*
_x, RDOFRMFrame::RDOFRMPosition* _y, RDOFRMFrame::RDOFRMPosition* _width,
```

```
RDOFRMFrame::RDOFRMPosition* _height, RDOFRMFrame::RDOFRMColor* bgColor,
RDOFRMFrame::RDOFRMColor* fgColor ):
  RDOFRMItem( _parent ),
  RDOFRMBoundingItem( _x, _y, _width, _height ),
  RDOFRMColoredItem( bgColor, fgColor ),
  align(rdoAnimation::RDOTextElement::TETA_LEFT),
  value( NULL ),
  txt( "" ),
  isTextString( true )
{
  color_reparent( this );
}

void RDOFRMText::setText( rdoAnimation::RDOTextElement::TextAlign _align,
RDOCalc* _value )
{
  align = _align;
  value = _value;

  isTextString = false;
}

void RDOFRMText::setText( rdoAnimation::RDOTextElement::TextAlign _align,
const std::string& _txt )
{
  align = _align;
  txt   = _txt;
  isTextString = true;
}

rdoAnimation::FrameItem* RDOFRMText::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDOTextElement) );

  rdoAnimation::RDOColor bg = getBg( sim, getFrame() );
  rdoAnimation::RDOColor fg = getFg( sim, getFrame() );
  getFrame()->setColorLastBGText( getBgColor()->getColorType(), bg );
  getFrame()->setColorLastFGText( getFgColor()->getColorType(), fg );

  std::string t;
  if ( isTextString ) {
        t = txt;
  } else {
        RDOValue val = value->calcValue( sim );
        t = val.getAsString();
  }

  int _x      = getX( sim, getFrame() );
  int _y      = getY( sim, getFrame() );
  int _width  = getWidth( sim, getFrame() );
  int _height = getHeight( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, _width, _height );

  return new rdoAnimation::RDOTextElement(
        rdoAnimation::RDOBoundedElement(rdoAnimation::RDOPoint(_x, _y),
rdoAnimation::RDOSize(_width, _height)),
        rdoAnimation::RDOColoredElement(bg, fg),
        t, align
  );
}

// ----------------------------------------------------------------------
-----
```

```
// ---------- RDOFRMBitmap
// ----------------------------------------------------------------------
-----
RDOFRMBitmap::RDOFRMBitmap( RDOFRMFrame* _parent,
RDOFRMFrame::RDOFRMPosition* _x, RDOFRMFrame::RDOFRMPosition* _y, const
std::string& _pict_filename, const std::string& _mask_filename ):
  RDOFRMBitmapBase( _parent, _pict_filename, _mask_filename ),
  x( _x ),
  y( _y )
{
}

rdoAnimation::FrameItem* RDOFRMBitmap::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDOBmpElement) );

  int _x = x->getX( sim, getFrame() );
  int _y = y->getY( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, 0, 0 );

  return new rdoAnimation::RDOBmpElement(
        rdoAnimation::RDOPoint(_x, _y),
        pict_filename, mask_filename
  );
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMBitmapStretch
// ----------------------------------------------------------------------
-----
RDOFRMBitmapStretch::RDOFRMBitmapStretch( RDOFRMFrame* _parent,
RDOFRMFrame::RDOFRMPosition* _x, RDOFRMFrame::RDOFRMPosition* _y,
RDOFRMFrame::RDOFRMPosition* _width, RDOFRMFrame::RDOFRMPosition* _height,
const std::string& _pict_filename, const std::string& _mask_filename ):
  RDOFRMBitmapBase( _parent, _pict_filename, _mask_filename ),
  RDOFRMBoundingItem( _x, _y, _width, _height )
{
}

rdoAnimation::FrameItem* RDOFRMBitmapStretch::createElement( RDORuntime*
sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDOSBmpElement) );

  int _x      = getX( sim, getFrame() );
  int _y      = getY( sim, getFrame() );
  int _width  = getWidth( sim, getFrame() );
  int _height = getHeight( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, _width, _height );

  return new rdoAnimation::RDOSBmpElement(
        rdoAnimation::RDOBoundedElement(rdoAnimation::RDOPoint(_x, _y),
rdoAnimation::RDOSize(_width, _height)),
        pict_filename, mask_filename
  );
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMRect
// ----------------------------------------------------------------------
-----
```

```
RDOFRMRect::RDOFRMRect( RDOFRMFrame* _parent, RDOFRMFrame::RDOFRMPosition*
_x, RDOFRMFrame::RDOFRMPosition* _y, RDOFRMFrame::RDOFRMPosition* _width,
RDOFRMFrame::RDOFRMPosition* _height, RDOFRMFrame::RDOFRMColor* bgColor,
RDOFRMFrame::RDOFRMColor* fgColor ):
  RDOFRMItem( _parent ),
  RDOFRMBoundingItem( _x, _y, _width, _height ),
  RDOFRMColoredItem( bgColor, fgColor )
{
  color_reparent( this );
}

rdoAnimation::FrameItem* RDOFRMRect::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDORectElement) );

  rdoAnimation::RDOColor bg = getBg( sim, getFrame() );
  rdoAnimation::RDOColor fg = getFg( sim, getFrame() );
  getFrame()->setColorLastBG( getBgColor()->getColorType(), bg );
  getFrame()->setColorLastFG( getFgColor()->getColorType(), fg );
  int _x      = getX( sim, getFrame() );
  int _y      = getY( sim, getFrame() );
  int _width  = getWidth( sim, getFrame() );
  int _height = getHeight( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, _width, _height );

  return new rdoAnimation::RDORectElement(
        rdoAnimation::RDOBoundedElement(rdoAnimation::RDOPoint(_x, _y),
rdoAnimation::RDOSize(_width, _height)),
        rdoAnimation::RDOColoredElement(bg, fg)
  );
}

// --------------------------------------------------------------------
-----
// ---------- RDOFRMRectRound
// --------------------------------------------------------------------
-----
RDOFRMRectRound::RDOFRMRectRound( RDOFRMFrame* _parent,
RDOFRMFrame::RDOFRMPosition* _x, RDOFRMFrame::RDOFRMPosition* _y,
RDOFRMFrame::RDOFRMPosition* _width, RDOFRMFrame::RDOFRMPosition* _height,
RDOFRMFrame::RDOFRMColor* bgColor, RDOFRMFrame::RDOFRMColor* fgColor ):
  RDOFRMItem( _parent ),
  RDOFRMBoundingItem( _x, _y, _width, _height ),
  RDOFRMColoredItem( bgColor, fgColor )
{
  color_reparent( this );
}

rdoAnimation::FrameItem* RDOFRMRectRound::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDORRectElement) );

  rdoAnimation::RDOColor bg = getBg( sim, getFrame() );
  rdoAnimation::RDOColor fg = getFg( sim, getFrame() );
  getFrame()->setColorLastBG( getBgColor()->getColorType(), bg );
  getFrame()->setColorLastFG( getFgColor()->getColorType(), fg );
  int _x      = getX( sim, getFrame() );
  int _y      = getY( sim, getFrame() );
  int _width  = getWidth( sim, getFrame() );
  int _height = getHeight( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, _width, _height );

  return new rdoAnimation::RDORRectElement(
```

```
        rdoAnimation::RDOBoundedElement(rdoAnimation::RDOPoint(_x, _y),
rdoAnimation::RDOSize(_width, _height)),
        rdoAnimation::RDOColoredElement(bg, fg)
  );
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMEllipse
// ----------------------------------------------------------------------
-----
RDOFRMEllipse::RDOFRMEllipse( RDOFRMFrame* _parent,
RDOFRMFrame::RDOFRMPosition* _x, RDOFRMFrame::RDOFRMPosition* _y,
RDOFRMFrame::RDOFRMPosition* _width, RDOFRMFrame::RDOFRMPosition* _height,
RDOFRMFrame::RDOFRMColor* bgColor, RDOFRMFrame::RDOFRMColor* fgColor ):
  RDOFRMItem( _parent ),
  RDOFRMBoundingItem( _x, _y, _width, _height ),
  RDOFRMColoredItem( bgColor, fgColor )
{
  color_reparent( this );
}

rdoAnimation::FrameItem* RDOFRMEllipse::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDOEllipseElement) );

  rdoAnimation::RDOColor bg = getBg( sim, getFrame() );
  rdoAnimation::RDOColor fg = getFg( sim, getFrame() );
  getFrame()->setColorLastBG( getBgColor()->getColorType(), bg );
  getFrame()->setColorLastFG( getFgColor()->getColorType(), fg );
  int _x      = getX( sim, getFrame() );
  int _y      = getY( sim, getFrame() );
  int _width  = getWidth( sim, getFrame() );
  int _height = getHeight( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, _width, _height );

  return new rdoAnimation::RDOEllipseElement(
        rdoAnimation::RDOBoundedElement(rdoAnimation::RDOPoint(_x, _y),
rdoAnimation::RDOSize(_width, _height)),
        rdoAnimation::RDOColoredElement(bg, fg)
  );
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMCircle
// ----------------------------------------------------------------------
-----
RDOFRMCircle::RDOFRMCircle( RDOFRMFrame* parent,
RDOFRMFrame::RDOFRMPosition* x, RDOFRMFrame::RDOFRMPosition* y,
RDOFRMFrame::RDOFRMPosition* radius, RDOFRMFrame::RDOFRMColor* bgColor,
RDOFRMFrame::RDOFRMColor* fgColor ):
  RDOFRMItem( parent ),
  RDOFRMColoredItem( bgColor, fgColor ),
  m_x( x ),
  m_y( y ),
  m_radius( radius )
{
  color_reparent( this );
}

rdoAnimation::FrameItem* RDOFRMCircle::createElement( RDORuntime* sim )
{
```

```
  sim->memory_insert( sizeof(rdoAnimation::RDOCircleElement) );

  rdoAnimation::RDOColor bg = getBg( sim, getFrame() );
  rdoAnimation::RDOColor fg = getFg( sim, getFrame() );
  getFrame()->setColorLastBG( getBgColor()->getColorType(), bg );
  getFrame()->setColorLastFG( getFgColor()->getColorType(), fg );

  return new rdoAnimation::RDOCircleElement(
        rdoAnimation::RDOPoint(m_x->getX( sim, getFrame() ), m_y->getY(
sim, getFrame() )),
        rdoAnimation::RDORadius(m_radius->getX( sim, getFrame() )),
        rdoAnimation::RDOColoredElement(bg, fg)
  );
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMLine
// ----------------------------------------------------------------------
-----
RDOFRMLine::RDOFRMLine( RDOFRMFrame* _parent, RDOFRMFrame::RDOFRMPosition*
_x1, RDOFRMFrame::RDOFRMPosition* _y1, RDOFRMFrame::RDOFRMPosition* _x2,
RDOFRMFrame::RDOFRMPosition* _y2, RDOFRMFrame::RDOFRMColor* _color ):
  RDOFRMItem( _parent ),
  RDOFRMBoundingItem( _x1, _y1, _x2, _y2 ),
  color( _color )
{
  color->reparent( this );
}

rdoAnimation::FrameItem* RDOFRMLine::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDOLineElement) );

  rdoAnimation::RDOColor fg = color->getColor( sim, getFrame() );
  getFrame()->setColorLastFG( color->getColorType(), fg );
  int _x1 = getX( sim, getFrame() );
  int _y1 = getY( sim, getFrame() );
  int _x2 = getWidthAsX( sim, getFrame() );
  int _y2 = getHeightAsY( sim, getFrame() );
  getFrame()->setLastXYWH( _x1, _y1, _x2 - _x1, _y2 - _y1 );

  return new rdoAnimation::RDOLineElement(
        rdoAnimation::RDOPoint(_x1, _y1),
        rdoAnimation::RDOPoint(_x2, _y2),
        fg
  );
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMTriang
// ----------------------------------------------------------------------
-----
RDOFRMTriang::RDOFRMTriang( RDOFRMFrame* _parent,
RDOFRMFrame::RDOFRMPosition* _x1, RDOFRMFrame::RDOFRMPosition* _y1,
RDOFRMFrame::RDOFRMPosition* _x2, RDOFRMFrame::RDOFRMPosition* _y2,
RDOFRMFrame::RDOFRMPosition* _x3, RDOFRMFrame::RDOFRMPosition* _y3,
RDOFRMFrame::RDOFRMColor* bgColor, RDOFRMFrame::RDOFRMColor* fgColor ):
  RDOFRMItem( _parent ),
  RDOFRMColoredItem( bgColor, fgColor ),
  x1( _x1 ),
  y1( _y1 ),
```

```
  x2( _x2 ),
  y2( _y2 ),
  x3( _x3 ),
  y3( _y3 )
{
  color_reparent( this );
}

rdoAnimation::FrameItem* RDOFRMTriang::createElement( RDORuntime* sim )
{
  sim->memory_insert( sizeof(rdoAnimation::RDOTriangElement) );

  rdoAnimation::RDOColor bg = getBg( sim, getFrame() );
  rdoAnimation::RDOColor fg = getFg( sim, getFrame() );
  getFrame()->setColorLastBG( getBgColor()->getColorType(), bg );
  getFrame()->setColorLastFG( getFgColor()->getColorType(), fg );
  int _x1 = x1->getX( sim, getFrame() );
  int _y1 = y1->getY( sim, getFrame() );
  int min_x = _x1;
  int max_x = _x1;
  int min_y = _y1;
  int max_y = _y1;
  getFrame()->setLastXYWH( _x1, _y1, _x1, _y1 );
  int _x2 = x2->getX( sim, getFrame() );
  int _y2 = y2->getY( sim, getFrame() );
  if ( min_x > _x2 ) min_x = _x2;
  if ( max_x < _x2 ) max_x = _x2;
  if ( min_y > _y2 ) min_y = _y2;
  if ( max_y < _x2 ) max_y = _y2;
  getFrame()->setLastXYWH( _x2, _y2, _x2 - _x1, _y2 - _y1 );
  int _x3 = x3->getX( sim, getFrame() );
  int _y3 = y3->getY( sim, getFrame() );
  if ( min_x > _x3 ) min_x = _x3;
  if ( max_x < _x3 ) max_x = _x3;
  if ( min_y > _y3 ) min_y = _y3;
  if ( max_y < _x3 ) max_y = _y3;
  getFrame()->setLastXYWH( min_x, min_y, max_x - min_x, max_y - min_y );

  return new rdoAnimation::RDOTriangElement(
        rdoAnimation::RDOPoint(_x1, _y1),
        rdoAnimation::RDOPoint(_x2, _y2),
        rdoAnimation::RDOPoint(_x3, _y3),
        rdoAnimation::RDOColoredElement(bg, fg)
  );
}

// --------------------------------------------------------------------
-----
// ---------- RDOFRMActive
// --------------------------------------------------------------------
-----
RDOFRMActive::RDOFRMActive( RDOFRMFrame* _parent,
RDOFRMFrame::RDOFRMPosition* _x, RDOFRMFrame::RDOFRMPosition* _y,
RDOFRMFrame::RDOFRMPosition* _width, RDOFRMFrame::RDOFRMPosition* _height,
const std::string& _operName ):
  RDOFRMItem( _parent ),
  RDOFRMBoundingItem( _x, _y, _width, _height ),
  operName( _operName )
{
}

rdoAnimation::FrameItem* RDOFRMActive::createElement( RDORuntime* sim )
{
```

```
  sim->memory_insert( sizeof(rdoAnimation::RDOActiveElement) );

  int _x      = getX( sim, getFrame() );
  int _y      = getY( sim, getFrame() );
  int _width  = getWidth( sim, getFrame() );
  int _height = getHeight( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, _width, _height );

  return new rdoAnimation::RDOActiveElement(
        rdoAnimation::RDOBoundedElement(rdoAnimation::RDOPoint(_x, _y),
rdoAnimation::RDOSize(_width, _height)),
        operName
  );
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMSpace
// ----------------------------------------------------------------------
-----
RDOFRMSpace::RDOFRMSpace( RDOFRMFrame* _parent,
RDOFRMFrame::RDOFRMPosition* _x, RDOFRMFrame::RDOFRMPosition* _y,
RDOFRMFrame::RDOFRMPosition* _width, RDOFRMFrame::RDOFRMPosition* _height
):
  RDOFRMItem( _parent ),
  RDOFRMBoundingItem( _x, _y, _width, _height )
{
}

rdoAnimation::FrameItem* RDOFRMSpace::createElement( RDORuntime* sim )
{
  int _x      = getX( sim, getFrame() );
  int _y      = getY( sim, getFrame() );
  int _width  = getWidth( sim, getFrame() );
  int _height = getHeight( sim, getFrame() );
  getFrame()->setLastXYWH( _x, _y, _width, _height );
  return NULL;
}

// ----------------------------------------------------------------------
-----
// ---------- RDOFRMShow
// ----------------------------------------------------------------------
-----
RDOFRMShow::RDOFRMShow( RDOFRMFrame* _parent, RDOCalc* _conditionCalc ):
  RDORuntimeParent( _parent ),
  conditionCalc( _conditionCalc )
{
}

RDOFRMShow::~RDOFRMShow()
{
}

void RDOFRMShow::getBitmaps(REF(std::list<tstring>) list)
{
  STL_FOR_ALL(ChildList, m_childList, it)
  {
        static_cast<PTR(RDOFRMItem)>(*it)->getBitmaps(list);
  }
}

bool RDOFRMShow::checkCondition( RDORuntime* sim )
```

```
{
  if ( !conditionCalc ) return true;
  return conditionCalc->calcValue( sim ).getAsBool();
}

} // namespace rdoRuntime
```

Список литературы

1) ГОСТ 34.602-89.. - [б.м.] : Техническое задание на создание автоматизированной системы..

2) ГОСТ 19.201-78.. - [б.м.] : Единая система программной документации. Техническое задание. Требования к содержанию и оформлению..

3) Емельянов В.В. Ясиновский С.И. Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО. [Книга]. - Москва : «АНВИК», 1998.