1. Введение

Имитационное моделирование(ИМ) на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами(СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, транспортные системы (морские порты, аэропорты) и комплексы перекачки нефти и газа, программное обеспечение сложных систем управления и вычислительные сети, а также многие другие.

Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. 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.

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском государственном техническом университете (МГТУ им.Н.Э. Баумана) на кафедре "Компьютерные системы автоматизации производства". Причинами ее проведения и создания РДО явились требования универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей, моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в реальном масштабе времени) и ряд других, сформировавшихся у разработчиков при выполнении работ, связанных с системным анализом и организационным управлением сложными системами различной природы.

2. Предпроектное исследование объекта автоматизации

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.

В соответствии с основной целью 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;
- анимация работы модели и переключение между кадрами анимации;
- 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. [2]

Подробнее остановимся на анимации, ведь именно графическая составляющая модели наглядно показывает процесс работы модели, а так же позволяет эффективно оценивать её эффективность.

2.1 Кадры анимации в комплексе RAO-studio

Описание кадров производится в отдельном объекте, который является исходным для системы отображения. Этот объект имеет расширение .frm. Кадр представляет собой прямоугольную область экрана, в которой производится отображение. Он состоит из фоновой картинки и переменных элементов (элементов отображения или спрайтов),

состав, форма, размеры и расположение которых определяются состоянием системы и, следовательно, могут изменяться во время просмотра кадра.
Описание кадра имеет следующий формат:

$Frame <*имя_кадра*>
$Back_picture = <*описание_фоновой_картинки*>
[<*описание_элементов_отображения*>]
$End

имя_кадра:

Имя кадра представляет собой простое имя (последовательность русских или латинских строчных или прописных букв и цифр, а также символов "$" и "_".). Имена должны быть различными для всех кадров и не должны совпадать с ранее определенными именами.

описание_фоновой_картинки:

Описание фоновой картинки имеет следующий формат:

[<*цвет_фона*>] (<*размеры_кадра*> | <*имя_файла_фона*>)

цвет_фона:

Цвет фона задает цвет части кадра, которая находится за пределами фоновой картинки. Цвет задается тремя численными константами целого типа, разделенными пробелами и заключенными в угловые скобки. Каждое число должно находиться в диапазоне 0..255, оно задает интенсивность одной из трех цветовых составляющих: первое – красной, второе – зеленой и третье – синей.

Цвет фона является необязательным параметром. Если он не задан, используется значение по умолчанию, равное <0 100 0> (это значение соответствует темно-зеленому цвету).

размеры_кадра, имя_файла_фона:

Для описания фоновой картинки задают **либо** имя файла, содержащего фоновое изображение, **либо** размер фоновой картинки. Файл фоновой картинки должен быть растровым изображением, сохраненным в формате независимой от устройства битовой карты (BMP - формате) и иметь расширение .bmp.

Если вместо имени файла указаны размеры фоновой картинки, РДО-имитатор сам создает фоновую картинку, которая представляет собой прямоугольник указанного размера с цветом фона и границей черного цвета толщиной в один пиксел. Размеры задают двумя численными константами целого типа. Первое число задает ширину фоновой картинки и должно быть в диапазоне 1..800, второе число задает высоту и должно находиться в диапазоне 1..600.

описание_элементов_отображения:

Элемент отображения имеет следующий формат:

<*тип_элемента*> [<*свойства_элемента*>]

тип_элемента:

Тип элемента задают одним из следующих зарезервированных слов:

Тип элемента	Описание
text	Текстовый элемент
bitmap	Битовая карта
rect	Прямоугольник
line	Отрезок прямой
circle	Окружность

ellipse	Эллипс
r_rect	Прямоугольник со скругленными углами
triang	Треугольник
s_bmp	Масштабируемая битовая карта

Таблица 1. Соответствие типов элементов и зарезервированных слов

свойства_элемента:

Порядок записи, количество и смысл свойств элемента зависят от типа элемента. Свойства элементов записываются в прямых скобках и разделяются запятыми.

При изменении состояния модели значения выражений, определяющих координаты и размеры элемента, могут изменяться, то есть элемент может перемещаться по экрану и меняться в размерах. Если при этом элемент выходит за границы кадра, то он автоматически усекается. Прорисовка элементов происходит в том порядке, как они описаны в объекте кадров, то есть при наложении элементов тот из них, который описан в объекте раньше, будет скрыт (полностью или частично) под элементом, описанным позже. [1]

При исследовании программного комплекса RAO-studio, и процесса описания кадров в частности, выяснилось, что он не лишен недостатков. Так, при таком построении кадра отсутствует возможность, кроме как непосредственное копирование-вставка, повторного использования кода.

Если принять во внимание, что модель может меняться в процессе разработки, становится понятным, что такого рода недостаток является критичным при частом использовании сложных составных частей кадра (таблицы или сложные графические конструкции) в различных частях кадра или разных кадрах. Использование модульного подхода позволяет локализовать место ошибки, обычно исправление ошибки внутри одного модуля не влечет за собой исправление программы.

Часто при исправлении текста куска программы в одном месте разработчик забывает изменить этот же текст в другом, что порождает трудно отслеживаемые ошибки. Использование модульного подхода защищает от них. Другие преимущества использования модульного подхода к программированию, моделированию в данном случае, это возможность придать модели (кадру анимации) иерархическую структуру, что положительно сказывается на ее восприятии; обеспечение независимости компонентов кадра, т.е. возможность их независимой разработки и отладки. [4]

3. Концептуальный этап проектирования системы.

3.1 Выбор общесистемной методологии проектирования

Проектирование любой системы начинается с выявления проблемы, для которой она создается. Под проблемой понимается несовпадение характеристик состояния систем, существующей и желаемой. Одна из них была выявлена при предпроектном исследовании и заключается в невозможности повторного использования части программы. Проблема может быть решена на основе следующих концепций:

1. Модульность
2. Объектная ориентированность

Модульность — это свойство системы, связанное с возможностью ее декомпозиции на ряд внутренне связанных между собой модулей. Применительно к конструированию технических систем модульность — принцип, согласно которому функционально связанные части группируются в законченные узлы — модули. В свою очередь модульность в программировании — принцип, согласно которому программное средство(ПС) разделяется на отдельные именованные сущности, называемые модулями. Модульность часто является средством упрощения задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков. При разбиении ПС на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями.

Объектно-ориентированное программирование (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. Объект — это сущность, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм; то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

3.2 Диаграмма компонентов

На рисунке 1 изображена диаграмма компонентов системы. Базовый функционал этих пакетов**Ошибка! Источник ссылки не найден.**:

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

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

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

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

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

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

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

Объекты компонента **rdo_runtime** инициализируются при разборе исходного текста модели компонентом **rdo_parser**.

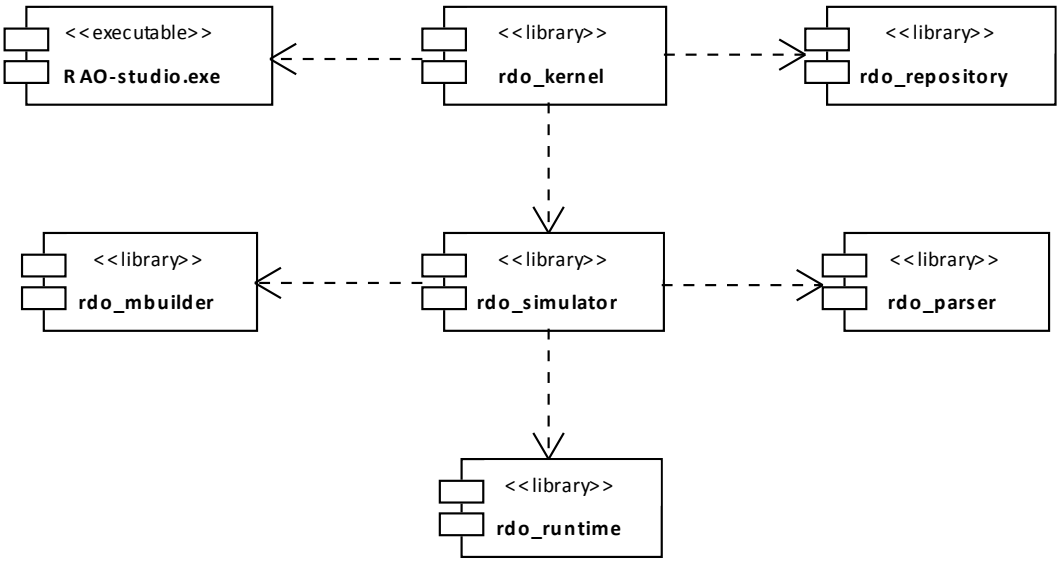

Рисунок 1

4. Формирование ТЗ

4.1 Введение

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.

В соответствии с основной целью программный комплекс решает множество задач, одна из которых анимация в процессе моделирования.

4.2 Общие сведения

- Полное наименование темы разработки: «Добавление в анимацию системы дискретного имитационного моделирования РДО объекта спрайт»
- Заказчик: Кафедра "Компьютерные системы автоматизации производства " МГТУ им. Н.Э.Баумана"
- Разработчик: студент кафедры "Компьютерные системы автоматизации производства" Чернов А. О.
- Основание для разработки: Задание на курсовой проект
- Плановые сроки начала работы: 10 сентября 2012г.
- Плановые сроки окончания работы по созданию системы: 21 декабря 2012г.

4.3 Назначение разработки

Разработать объект «Спрайт» и реализовать его вызов в кадре анимации на базе рабочей версии RAO-studio.

Функциональным назначением объекта является предоставление пользователю возможности повторного использования кода. Сам объект к тому же является базой для дальнейшего развития системы РДО, а именно передачи объектов типа «Ресурс» как параметра объекта «Спрайт», а так же добавление возможности аффинных преобразований элементов кадра анимации.

Изменения должны эксплуатироваться разработчиками моделей со сложной графической составляющей, что позволит им многократно использовать определенный текст программы и обезопаситься от ошибок, вызванных постоянным копирование куска программы, а так же повысить модульность своей модели.

4.4 Требования к программе или программному изделию

Требования к функциональным характеристикам:

- обеспечить в РДО возможности создания объекта «Спрайт»
- обеспечить вызов спрайта в кадре анимации;
- дополнить документацию новыми возможностями системы.

Требования к надежности:

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

Условия эксплуатации:

- Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости.
- Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

Требования к составу и параметрам технических средств:

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 256 Мб;
- микропроцессор с тактовой частотой не менее 400 МГц;
- не менее 40 Мб свободного места;

Требования к информационной и программной совместимости:

- операционная система WINDOWS 2000, WINDOWS XP, WINDOWS 2003, WINDOWS VISTA, WINDOWS 2008 или WINDOWS 7[1];

Требования к маркировке и упаковке

Не предъявляются

Требования к транспортированию и хранению

Не предъявляются

4.5 Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

- техническое задание;
- технический и рабочий проекты;
- внедрение.

На стадии «Техническое задание» должен быть выполнен этап разработки и согласования настоящего технического задания.

На стадии «Технический и рабочий проект» должны быть выполнены перечисленные ниже этапы работ:

- разработка программы;
- разработка программной документации;
- испытания программы;

На стадии «Внедрение» должен быть выполнен этап разработки «Подготовка и передача программы».

[1] Microsoft, Windows являются зарегистрированными торговыми марками или торговыми марками Microsoft Corporation (в США и/или других странах).

4.6. Порядок контроля и приемки

Контроль и приемка работоспособности объекта «Спрайт» должны осуществляться в процессе проверки функциональности (апробирования) системы имитационного моделирования на тестовом примере модели в соответствии с требованиями к функциональным характеристикам системы.

5. Технический этап проектирования системы

5.1 Разработка синтаксиса описания спрайта и вызова спрайта

Для создания лексического анализатора в системе РДО используется генератор лексических анализаторов общего назначения Bison, который преобразует описание контекстно-свободной LALR(1) грамматики в программу на языке C++ для разбора этой грамматики. Для того чтобы Bison мог разобрать программу на каком-то языке, этот язык должен быть описан контекстно-свободной грамматикой. Это означает, необходимо определить одну или более синтаксических групп и задать правила их сборки из составных частей. Например, в языке C одна из групп называется `выражение'. Правило для составления выражения может выглядеть так: "Выражение может состоять из знака `минус' и другого выражения". Другое правило: "Выражением может быть целое число". Правила часто бывают рекурсивными, но должно быть, по крайней мере, одно правило, выводящее из рекурсии. [7]

Наиболее распространённой формальной системой для представления таких правил в удобном для человека виде является форма Бэкуса-Наура (БНФ). Любая грамматика, выраженная в форме Бэкуса-Наура, является контекстно-свободной грамматикой. Bison принимает на вход, в сущности, особый вид БНФ, адаптированный для машинной обработки.

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

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

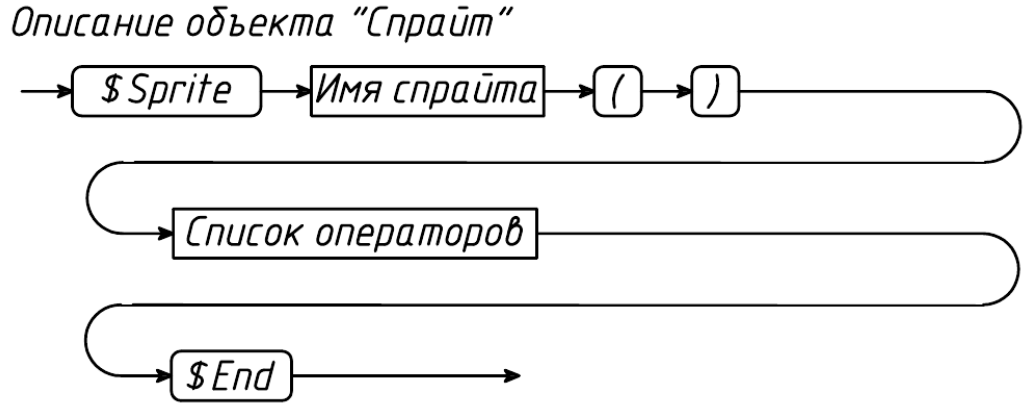

Рисунок 2

На рисунке 2 изображена разработанная грамматика описания объекта «Спрайт» в БНФ. Сам спрайт описывается в объекте исходной информации кадров анимации (вкладка с расширением .frm) до описания самого кадра (кадров). Вызов, согласно грамматике, должен выглядеть следующим образом:

```
$Sprite <имя_спрайта> ()
[ <описание_элементов_отображения> ]
$End
```

имя_ спрайта

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

описание_элементов_отображения

Описание элементов отображения происходит так же как во фрейме. Текст описания в спрайте ничем не отличается от того что ранее писался в тексте программы описания кадра анимации, однако, теперь при необходимости его повторного использования можно вызывать спрайт, а не копировать/вставлять его. Список элементов можно посмотреть выше, он рассматривался при предпроектном исследовании.

Спрайт может быть пустым. При описании спрайта может быть использован процедурный язык. Это дает возможность отображать часть элементов по заданному условию или сокращать объём кода модели за счет циклов.

Вызов объекта «Спрайт» производится в описании кадра подобно тому, как происходит описание элементов отображения. Следовательно, необходимо добавить специальную конструкцию вызова в группе «оператор элементов анимации». Для того чтобы Bison мог разобрать эту конструкцию, он должен быть описан контекстно-свободной грамматикой. Изменения в грамматике группы «оператор элементов анимации» представлена на рисунке 3 в БНФ. А грамматика элемента вызова спрайта в форме Бэкуса-Наура показана на рисунке 4.

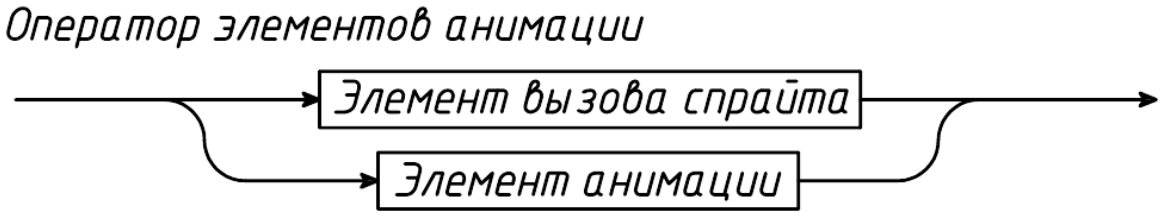

Рисунок 3

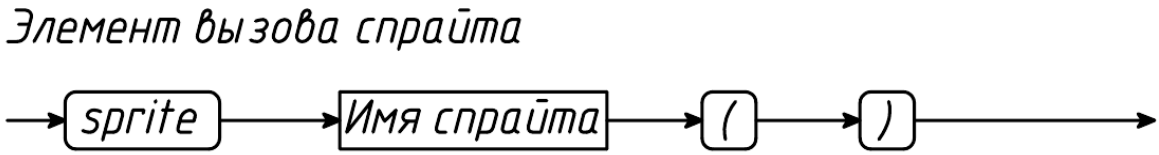

Рисунок 4

5.2 Разработка архитектуры компонента rdo_runtime

rdo_runtime отвечает за непосредственное выполнение модели, управление базой данных и базой знаний.

В пространстве имен rdo::runtime необходимо создать новый класс RDOFRMSprite, сделав новый класс базовым для существующего RDOFRMFrame. Создаваемый класс владеет и запускает на исполнение основные команды анимации, а класс RDOFRMFrame необходимо изменить, оставив ему лишь сугубо «фреймовые» методы.

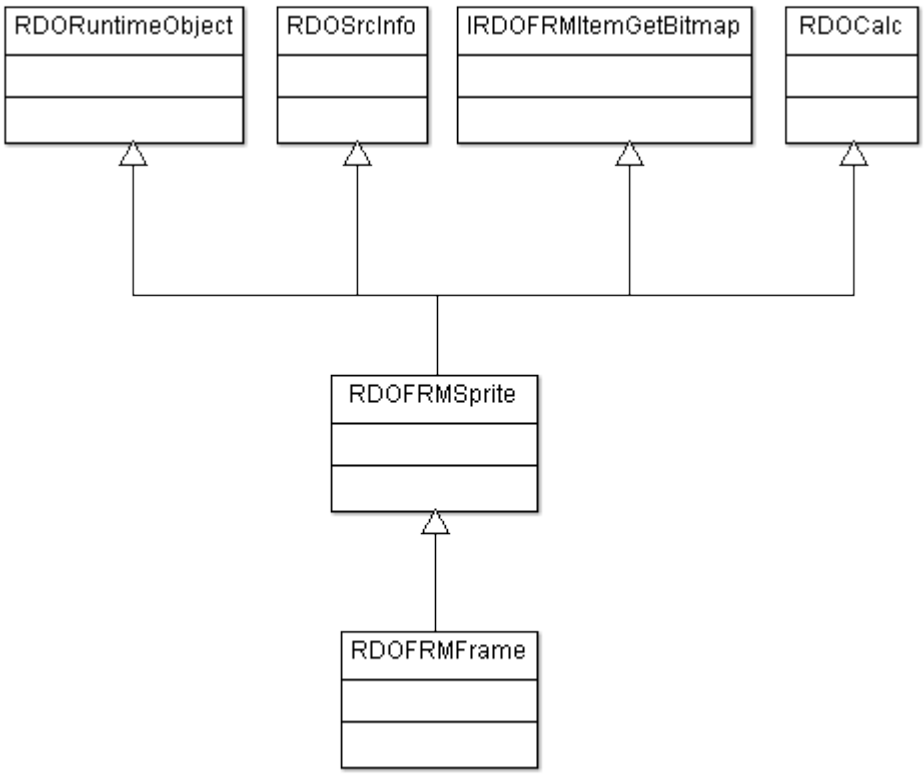

Рисунок 5

На рисунке 5 представлена упрощенная диаграмма классов, отображающая наследование. Полная диаграмма классов представлена на этапе рабочего проектирования.

5.3 Разработка архитектуры компонента rdo_parser

rdo_parser производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО.

Для возможности обработки новой конструкции в коде модели требуют изменений синтаксический анализатор РДО. В пространстве имен rdo:: parser требуется создать класс RDOFRMSprite, который должен создавать свой рантайм-аналог. Также требуется создать класс RDOFRMCommandList и добавить классам RDOFRMSprite и RDOFRMFrame наследование от него. Каждый из объектов этих трех классов должен быть зарегистрирован в парсере Это позволит в различных местах rdofrm.y (файл правила грамматики) обращаться к нужному объекту. При этом, RDOFRMCommandList должен содержать абсолютно виртуальный метод list(), который будет реализован в каждом из потомков.

На рисунке 6 представлена упрощенная диаграмма классов, отображающая наследование. Полная диаграмма классов представлена на этапе рабочего проектирования.

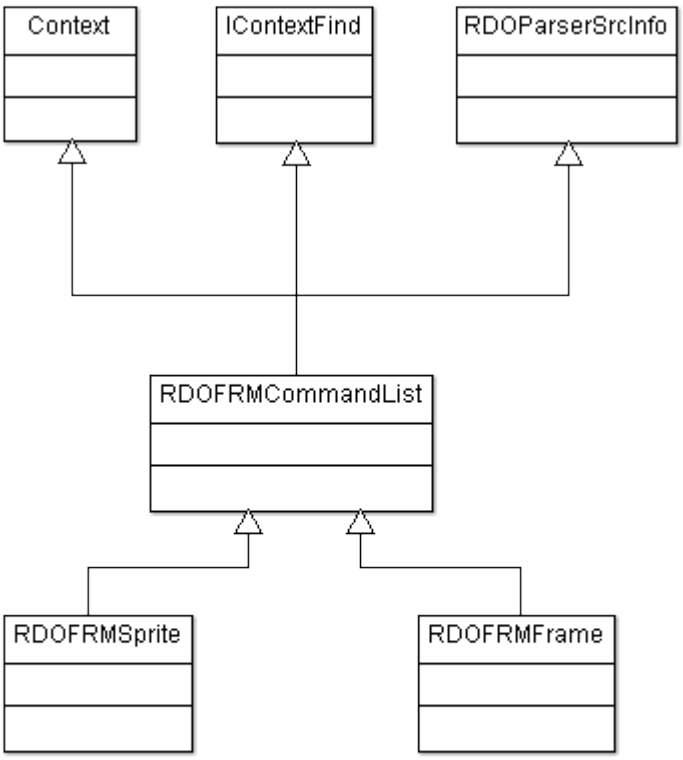
Context
IContextFind
RDOParserSrcInfo
RDOFRMCommandList
RDOFRMSprite
RDOFRMFrame

Рисунок 6

6. Рабочий этап проектирования системы

6.1 Изменения в файлах синтаксического анализатора

Для реализации в среде имитационного моделирования грамматики, разработанной на техническом этапе проектирования, необходимо добавить новые терминальные символы в лексический анализатор РДО и нетерминальные символы в грамматический анализатор.

В лексическом анализаторе (\RDO\simulator\compiler\parser\grammar\rdo_lexer.l) добавлены новые токены, записанные следующим образом:

```
$Sprite return(RDO_Sprite);
$sprite return(RDO_Sprite);
sprite  return(RDO_sprite_call);
```

Эти токены необходимо также добавить в секцию объявлений Bison файла грамматики (\RDO\simulator\compiler\parser\grammar\rdofrm.y):

```
%token RDO_Sprite
%token RDO_sprite_call
```

Далее необходимо разработать правила грамматики для группы “Описание объекта «Спрайт»”

```
frm_sprite_end
     : frm_sprite_begin RDO_End
     {
        LPRDOFRMSprite pSprite = PARSER->stack().pop<RDOFRMSprite>($1);
        ASSERT(pSprite);
        pSprite->end();
     }
     ;
frm_sprite_begin
     : frm_sprite_header statement_list
     {
        LPExpression pExpressionSpriteBody = PARSER-
        >stack().pop<Expression>($2);
        ASSERT(pExpressionSpriteBody);

        rdo::runtime::LPRDOCalcStatementList pCalcStatementList =
        pExpressionSpriteBody-
        >calc().object_dynamic_cast<rdo::runtime::RDOCalcStatementList>();
        ASSERT(pCalcStatementList);

        rdo::runtime::LPRDOCalcBaseStatementList pCalcBaseStatementList =
        rdo::Factory<rdo::runtime::RDOCalcBaseStatementList>::create();
        ASSERT(pCalcBaseStatementList);

        rdo::runtime::LPRDOCalcOpenBrace pCalcOpenBrace =
        rdo::Factory<rdo::runtime::RDOCalcOpenBrace>::create();
        ASSERT(pCalcOpenBrace);

        rdo::runtime::LPRDOCalcCloseBrace pCalcCloseBrace =
        rdo::Factory<rdo::runtime::RDOCalcCloseBrace>::create();
        ASSERT(pCalcCloseBrace);

        pCalcBaseStatementList->addCalcStatement(pCalcOpenBrace);
        pCalcBaseStatementList->addCalcStatement(pCalcStatementList);
        pCalcBaseStatementList->addCalcStatement(pCalcCloseBrace);
```

```
            LPExpression pExpressionSprite =
            rdo::Factory<Expression>::create(pExpressionSpriteBody->typeInfo(),
            pCalcBaseStatementList, pCalcStatementList->srcInfo());
            ASSERT(pExpressionSprite);

            LPRDOFRMSprite pSprite = PARSER->stack().pop<RDOFRMSprite>($1);
            ASSERT(pSprite);

            PARSER->getLastFRMCommandList()->list()-
            >setSpriteCalc(pExpressionSprite->calc());

            $$ = PARSER->stack().push(pSprite);
         }
         ;

      frm_sprite_header
         : RDO_Sprite RDO_IDENTIF '(' param_list ')'
         {
            LPRDOFRMSprite pSprite = rdo::Factory<RDOFRMSprite>::create(PARSER-
            >stack().pop<RDOValue>($2)->src_info());
            ASSERT(pSprite);
            $$ = PARSER->stack().push(pSprite);
         }
         | RDO_Sprite RDO_IDENTIF '(' param_list error
         {
            PARSER->error().error(@5, _T("Ожидается закрывающая скобка"));
         }
         | RDO_Sprite RDO_IDENTIF '(' error
         {
            PARSER->error().error(@4, _T("Ошибка задания параметров"));
         }
         | RDO_Sprite RDO_IDENTIF error
         {
            PARSER->error().error(@3, _T("Ожидается открывающая скобка"));
         }
         ;
```

Итак, в группе “Описание объекта «Спрайт»” в парсере создается объект RDOFRMSprite. В нетерминальном символе frm_sprite_header есть символ param_list, но на данном этапе разработки объекта спрайт он не умеет её обрабатывать. Это задел на будущее и дальнейшие развитие объекта «Спрайт». Имя спрайта передается в парсер с помощью лексемы RDO_IDENTIF. В нетерминальном символе frm_sprite_begin токен frm_sprite_header принимает заготовку спрайта с иминем из прошлой группы, а с помощью токена statement_list «обрастает» операторами, подаваемыми на выполнение при вызове спрайта.

6.2 Изменения в пространстве имен rdo::runtime

6.2.1 Реализация класса RDOFRMSprite

Как уже говорилось на этапе технического проектирования, разрабатываемый класс должен владеть и запускать на исполнение основные команды анимации. Описание класса производится в \RDO\simulator\runtime\rdoframe.h, а его методов в соотвествующих .cpp и .inl файлах.

```
CALC(RDOFRMSprite)
     IS  INSTANCE_OF       (RDORuntimeObject    )
     AND INSTANCE_OF       (RDOSrcInfo          )
     AND IMPLEMENTATION_OF(IRDOFRMItemGetBitmap)
{
DECLARE_FACTORY(RDOFRMSprite)
public:
```

```
    OBJECT(RDOFRMPosition) IS INSTANCE_OF(RDORuntimeObject) {};
    friend class RDOFRMPosition;
    OBJECT(RDOFRMColor) IS INSTANCE_OF(RDORuntimeObject) {};
    friend class RDOFRMColor;
    CALC(RDOFRMRulet) {};


public:
    CREF(tstring) name          () const;
    void          insertItem    (CREF(LPRDOCalc) pItem      );
    void          setSpriteCalc(CREF(LPRDOCalc) pSpriteCalc);

    void setColorLastBG    (RDOFRMColor::ColorType type,
                                        CREF(rdo::animation::Color) lastBg);
    void setColorLastFG    (RDOFRMColor::ColorType type,
                                        CREF(rdo::animation::Color) lastFg);
    void setColorLastBGText(RDOFRMColor::ColorType type,
                                      CREF(rdo::animation::Color) lastBgText);
    void setColorLastFGText(RDOFRMColor::ColorType type,
                                      CREF(rdo::animation::Color) lastFgText);
    void setLastXY         (double x, double y);
    void setLastXYWH       (double x, double y, double width, double height);

    int getRuletX(CREF(LPRDORuntime) pRuntime, ruint ruletID) const;
    int getRuletY(CREF(LPRDORuntime) pRuntime, ruint ruletID) const;
    LPRDOFRMRulet findRulet(ruint ruletID) const;
}
```

Итак, класс RDOFRMSprite является потомком от четырех классов RDORuntimeObject, RDOSrcInfo, IRDOFRMItemGetBitmap, RDOCalc. Объект класса спрайт владеет и запускает на исполнение основные команды анимации. В классе RDOFRMSprite определены три класса членов (так называемые вложенные члены): RDOFRMPosition – объект позиция, класс-потомок RDORuntimeObject; RDOFRMColor – объект цвет, класс-потомок RDORuntimeObject; RDOFRMRulet – рулетка, используемая для позиционирования, класс-потомок RDOCalc. Запись friend class RDOFRMPosition (RDOFRMColor) означает, что все функции этих классов являются друзьями RDOFRMSprite, т.е. эти функции имеют право доступа к закрытой части объявления класса. Методы класса RDOFRMSprite формируют спрайт: имя, отображаемые элементы, а так же запоминают последние значения их свойств.

6.2.2 Реализация класса RDOFRMFrame

Как уже говорилось на этапе технического проектирования, класс RDOFRMFrame должен претерпеть значительные изминения. Фрейм формирует кадр анимации, следовательно, класс должен иметь только фреймовые методы и стать потомком нового RDOFRMSprite. Описание класса производится в \RDO\simulator\runtime\rdoframe.h, а его методов в соответствующем .cpp файле.

```
CLASS(RDOFRMFrame):
    INSTANCE_OF(RDOFRMSprite)
{
DECLARE_FACTORY(RDOFRMFrame)
public:
    void setBackPicture(CREF(tstring) picFileName);
    void setBackPicture(int width, int height);

    void prepareFrame(PTR(rdo::animation::Frame) pFrame,
                                                 CREF(LPRDORuntime) pRuntime);

    void setBackgroundColor(CREF(LPRDOFRMColor) pBgColor);
```

```
    DECLARE_IRDOFRMItemGetBitmap;

    DECLARE_ICalc;
};
```

После рефакторинга у RDOFRMFrame остались только сугубо фреймовые методы, такие как установка цвета (или картинка) заднего фона и подготовка самого кадра, которая теперь осуществляется с помощью калков.

Макрос DECLARE_ICalc определен как:

```
#define DECLARE_ICalc \
private:              \
      RDOValue doCalc(CREF(LPRDORuntime) pRuntime);
```

doCalc – это виртуальный метод класса RDOCalc, однако, несомтря на то, что фрейм не имеет наследования от него, доступ к нему предоставляется через RDOFRMSprite, насредником которого является RDOFRMFrame. Этот метод вызывается при подготовки кадра. Следует отметить, что теперь фрейм это спрайт с фоном.

Подобно doCalc, виртуальный метод класса IRDOFRMItemGetBitmap getBitmaps объявляется шаблоном DECLARE_IRDOFRMItemGetBitmap. Он также получает доступ через наследование RDOFRMSprite и добавляет имя картинки, если таковая имеется, заднего фона в список картинок.

6.2.3 Диаграммы классов архитектуры системы анимации

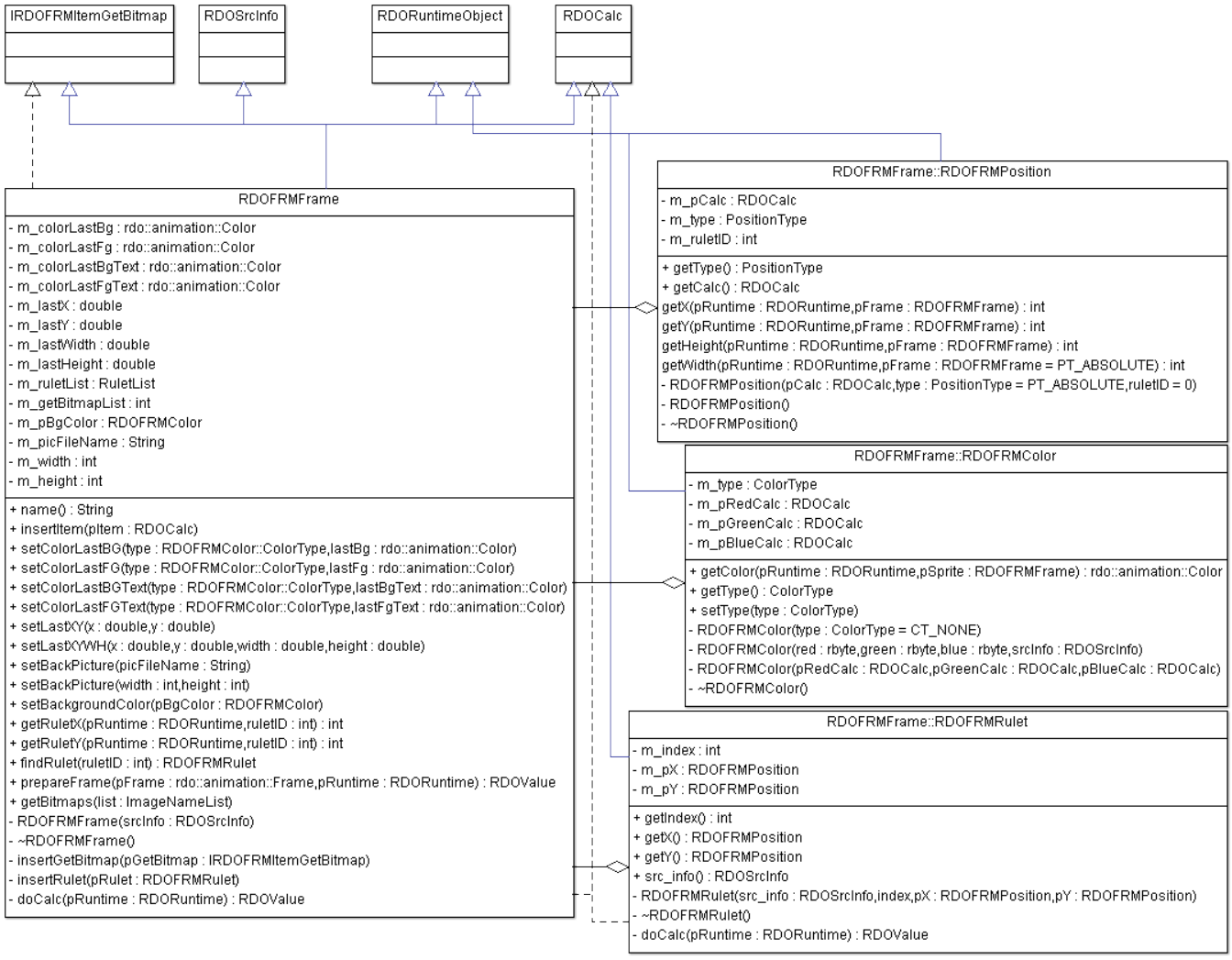

Рисунок 7

На рисунке 7 изображена диаграмма классов до внесения изменений, а полная диаграмма классов новой архитектуры приведена на листе. Сравнивая эти диаграммы, можно отметить, что произведена декомпозиция (объектная). Объектная декомпозиция – процесс представления предметной области в виде совокупности объектов, обменивающихся сообщениями, что является следствием выбранной концепции модульности и объектной ориентированности.

Проведенная декомпозиция позволяет построить четкую иерархию новой структуры анимации, что упрощает чтение исходного кода и изучение системы в целом и также является следствием решения проблемы на основе выбранных концепций.

6.3 Изменения в пространстве имен rdo::parser

6.3.1 Реализация класса RDOFRMCommandList

Как уже говорилось на этапе технического проектирования, разрабатываемый класс RDOFRMCommandList должен стать базовым для классов RDOFRMSprite и RDOFRMFrame. Каждый из объектов этих трех классов должен быть зарегистрирован в парсере, поэтому RDOFRMCommandList должен содержать абсолютно виртуальный метод list(), который будет реализован в каждом из потомков. Это позволит в различных местах rdofrm.y (файл правила грамматики) обращаться к нужному объекту. Описание нового класса производится в \RDO\simulator\compiler\parser\rdofrm.h, а его методов в соотвествующем .cpp файле.

```
CLASS(RDOFRMCommandList):
          INSTANCE_OF      (RDOParserSrcInfo)
      AND INSTANCE_OF      (Context          )
      AND IMPLEMENTATION_OF(IContextFind     )
{
DECLARE_FACTORY(RDOFRMCommandList);
public:
      CREF(tstring) name() const  { return src_info().src_text(); }
      virtual rdo::runtime::LPRDOFRMSprite list() const = 0;

      static LPExpression generateExpression(
      CREF(rdo::runtime::LPRDOCalc) pCalc, CREF(RDOParserSrcInfo) srcInfo);
}
```

Класс RDOFRMCommandList наследник от трех классов RDOParserSrcInfo, Context, IContextFind. Обладает требуемым виртуальным методом list(). Кроме того обладает методом name, ведь и спрайт и для фрейм обладают именами.

Так как метод generateExpression объявлен как статичный, он имеет право доступа к закрытой части объявления класса и находится в области видимости класса, но вызывается не для объекта класса.(отсутсвует указатель this)[6]. Метод пораждает выражение, вызывается для элементов отображения.

6.3.2 Реализация класса RDOFRMSprite

Как уже было сказано на техническом этапе проектирования для возможности обработки новой конструкции в коде модели, в пространстве имен rdo::parser требуется создать класс RDOFRMSprite, который должен создавать свой рантайм-аналог. Этот класс должен быть потомком созданного класса RDOFRMCommandList. При этом, RDOFRMSprite должен содержать описание метода list(). Описание нового класса производится в \RDO\simulator\compiler\parser\rdofrm.h, а его методов в соотвествующем .cpp файле.

```
CLASS(RDOFRMSprite):
      INSTANCE_OF (RDOFRMCommandList)
{
DECLARE_FACTORY(RDOFRMSprite);
```

```
public:
      void end();

      CREF(rdo::runtime::LPRDOFRMSprite) sprite() const;

      LPExpression expression() const;

private:
      rdo::runtime::LPRDOFRMSprite list() const }

      DECLARE_IContextFind;
};
     DECLARE_POINTER(RDOFRMSprite);
```

Виртуальный метод класса IContextFind onFindContext объявляется шаблоном DECLARE_IContextFind. Несомтря на то, что спрайт не имеет наследования от этого класса, доступ к нему предоставляется через RDOFRMCommandList, насредником которого является RDOFRMSprite.

Метод list() возвращает умный указатель на объект rdo::runtime::RDOFRMSprite и является реализацией виртуального метода класса RDOFRMCommandList.

Метод sprite() возвращает умный указатель на объект rdo::runtime::RDOFRMSprite и применяется для конструктора парсеровского RDOFRMSprite.

Метод end() отчищает ContextMemory и contextStack. Функция вызывается при окончании формирования спрайта.

6.3.3 Реализация класса RDOFRMFrame

Как уже было сказано на техническом этапе проектирования требуется изменение класса rdo::parser::RDOFRMFrame, который должен создавать свой рантайм-аналог. Этот класс должен быть потомком созданного класса RDOFRMCommandList. При этом, RDOFRMFrame должен содержать описание метода list(). Описание класса производится в \RDO\simulator\compiler\parser\rdofrm.h, там же где и до изменений, а его методов в соотвествующем .cpp файле.

```
CLASS(RDOFRMFrame):
      INSTANCE_OF (RDOFRMCommandList)
{
DECLARE_FACTORY(RDOFRMFrame);
public:
      void                                end   ();
      CREF(rdo::runtime::LPRDOFRMFrame) frame() const { return m_pFrame; }

private:
      RDOFRMFrame(CREF(RDOParserSrcInfo) srcInfo);

      rdo::runtime::LPRDOFRMSprite list() const { return m_pFrame; }

      DECLARE_IContextFind;
};
```

Виртуальный метод класса IContextFind onFindContext объявляется шаблоном DECLARE_IContextFind. Несомтря на то, что спрайт не имеет наследования от этого класса, доступ к нему предоставляется через RDOFRMCommandList, насредником которого является RDOFRMFrame.

Метод list() возвращает умный указатель на объект rdo::runtime::RDOFRMFrame и является реализацией виртуального метода класса RDOFRMCommandList.

Метод frame() возвращает умный указатель на объект rdo::runtime::RDOFRMSprite и применяется, когда нет необходимости использовать функцию list(), например задание фона спрайта.

Метод end() отчищает ContextMemory и contextStack. Функция вызывается при окончании формирования фрейма.

6.3.4 Диаграммы классов компилятора системы анимации

На рисунке 8 изображена диаграмма компилятора системы анимации до внесения изменений, аналогичная диаграмма для новой системы изображена на листе.

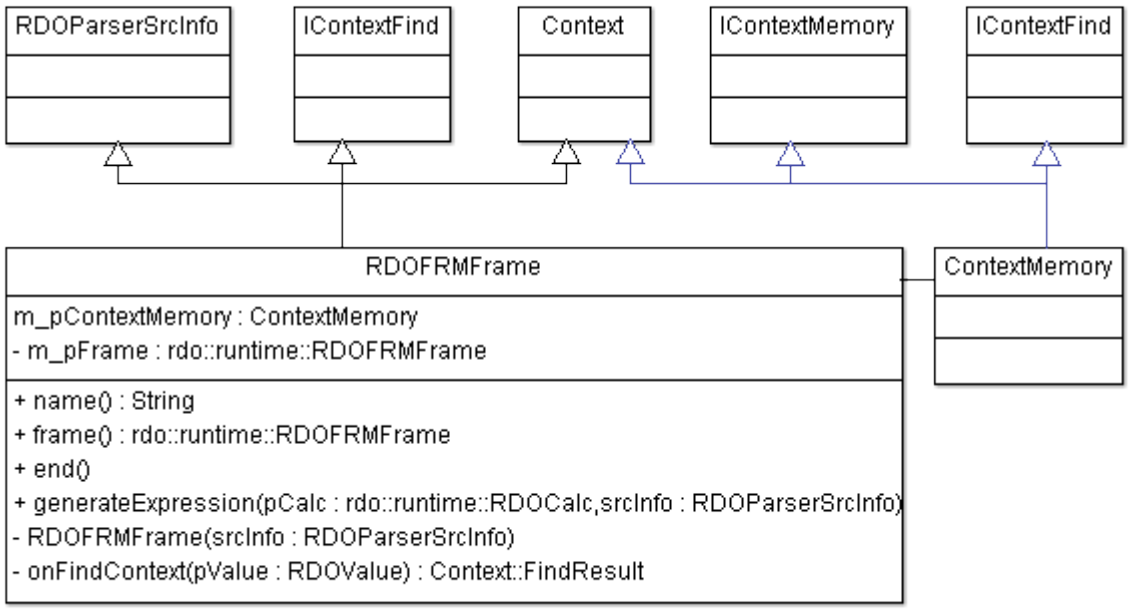

Рисунок 8

Сравнивая эти диаграммы, можно заметить, что класс RDOFRMFrame почти не понес изменений. Он обзавелся новым методом list() необходимым для реализации вызова спрайта и стал наследником нового класса RDOFRMCommandList, через который он получает доступ к классам, от которых он зависел ранее. Так как RDOFRMCommandList стал базовым методы name и generateExpression определяются в теле этого класса.

7. Апробирование разработанной системы для модельных условий

Для контроля работоспособности объекта «Спрайт» разработан тестовый пример модели в соответствии с требованиями к функциональным характеристикам системы. Модель «Тир» показывает удобство использования спрайтов.

Текст модели с применением изменений и без них отличается лишь во вкладке с расширением .frm, поэтому в приложении представлен полный текст модели и отдельно вкладка с расширением .frm для модели без использования спрайтов.

На листе представлены снимки работающей модели.

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

1. Проведено предпроектное исследование системы имитационного моделирования РДО.
2. На этапе концептуального проектирования системы с помощью диаграммы компонентов нотации UML укрупнено показано внутреннее устройство РДО и выделены те компоненты, которые потребуют внесения изменений в ходе этой работы.
3. На этапе технического проектирования доработана грамматика синтаксиса, которая представлена на синтаксической диаграмме. С помощью упрощенной диаграммы классов разработана архитектура новой системы.
4. На этапе рабочего проектирования написан программный код для реализации спроектированных раннее алгоритмов работы и архитектуры компонента rdo_parser и rdo_runtime системы РДО. Вновь разработанные классы показаны с помощью подробных диаграмм классов. Написан программный код, реализующий разработанную грамматику. На диаграмме активностей показан механизм создания объекта спрайта. Проведены отладка и тестирование нового функционала системы, в ходе которых исправлялись найденные ошибки.
5. Для демонстрации новых возможностей системы РДО была разработана модель. Результаты проведения имитационного исследования позволяют сделать вывод об адекватной работе новой функции системы.
6. Все внесенные в систему изменения отражены в справочной информации по системе РДО, что позволяет пользователям оперативно получать справку по новым функциям системы.

Список используемых источников

1. Справка по языку РДО [http://rdo.rk9.bmstu.ru/help/].
2. Справка по RAO-studio [http://rdo.rk9.bmstu.ru/help/].
3. Емельянов В. В., Ясиновский С. И. Имитационное моделирование систем: Учеб. Пособие – М.: Издательство МГТУ им. Н. Э. Баумана, 2009. – 584 с.: ил. (Информатика в техническом университете)
4. Мартин Р. Чистый код. Создание, анализ и рефакторинг / пер. с анг. Е. Матвеев – СПб.: Питер, 2010. – 464 стр.
5. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78.
6. Б. Страуструп Язык программирования C++. Специальное издание / пер. с англ. – М.: ООО «»Бином-Пресс, 2006. – 1104 с.: ил.
7. Чарльз Доннелли. Ричард Столлмен. Bison. Генератор синтаксических анализаторов, совместимый с YACC. / пер. с анг. [http://www.opennet.ru/docs/RUS/bison_yacc/]

Список использованного программного обеспечения

1. RAO-Studio.
2. ArgoUML.
3. Autodesk®, Inc., AutoCAD® Mechanical 2009.
4. Microsoft®, Office Word 2003 SP3.
5. Microsoft®, Visual Studio 2008 SP

Приложение

Приложение 1. Полный текст модели «Тир»

Вкладка с расширением .rtp

```
$Resource_type Тиры : permanent
$Parameters
     Возможность_стрелять : integer
$End

$Resource_type Стрелки : permanent
$Parameters
     Выстрел_1_r        : integer
     Выстрел_1_f        : real
     Выстрел_1_m        : integer
     Выстрел_2_r        : integer
     Выстрел_2_f        : real
     Выстрел_2_m        : integer
     Выстрел_3_r        : integer
     Выстрел_3_f        : real
     Выстрел_3_m        : integer
     Среднее            : real
     Максимальное       : integer
     Сумма              : integer
     количестов_попыток : integer
     состояние_стрелка  : ( Стреляет, Ожидает ) = Ожидает
$End

$Resource_type Стрелы : temporary
$Parameters
     Состояние   : ( Используется, Не_используется )
     Стрела_1_r  : integer
     Стрела_1_f  : real
     Стрела_2_r  : integer
     Стрела_2_f  : real
     Стрела_3_r  : integer
     Стрела_3_f  : real
$End
```

Вкладка с расширением .rss

```
$Resources
     Тир    : Тиры trace 0
     Петр   : Стрелки trace 210 0 0 210 0 0 210 0 0 0 0 0 0 *
     Иван   : Стрелки trace 210 0 0 210 0 0 210 0 0 0 0 0 0 *
     Степан : Стрелки trace 210 0 0 210 0 0 210 0 0 0 0 0 0 *
$End
```

Вкладка с расширением .evn

```
$Pattern Образец_стрельбы : event
$Relevant_resources
     _Тир         : Тир      Keep
     _Набор_стрел : Стрелы   Create

$Body
_Тир
     Convert_event
           Образец_стрельбы.planning(time_now + Интервал_подготовки ( 30 ) );
           Возможность_стрелять++;

_Набор_стрел
     Convert_event
```

```
            Состояние = Не_используется;
            Стрела_1_r = rnd(0,2*R - 5);
            Стрела_1_f = rnd(0,2*R - 5);
            Стрела_2_r = rnd(0,2*R - 5);
            Стрела_2_f = rnd(0,2*R - 5);
            Стрела_3_r = rnd(0,2*R - 5);
            Стрела_3_f = rnd(0,2*R - 5);
$End
```

Вкладка с расширением .pat

```
$Pattern Образец_подсчета : operation
$Relevant_resources
     _Тир               : Тир              Keep NoChange
     _Набор_стрел       : Стрелы           Keep Erase
     _Стрелки           : Стрелки          Keep Keep
$Time = Длительность_стрельбы ( 20, 40 )
$Body
_Тир
     Choice from _Тир.Возможность_стрелять > 0
     Convert_begin
            Возможность_стрелять--;

_Набор_стрел
     Choice from _Набор_стрел.Состояние == Не_используется
     Convert_begin
            Состояние = Используется;

_Стрелки
     Choice from _Стрелки.состояние_стрелка == Ожидает
     with_min(_Стрелки.количестов_попыток)
     Convert_begin
            состояние_стрелка = Стреляет;
            Выстрел_1_r = _Набор_стрел.Стрела_1_r;
            Выстрел_2_r = _Набор_стрел.Стрела_2_r;
            Выстрел_3_r = _Набор_стрел.Стрела_3_r;
            Выстрел_1_f = _Набор_стрел.Стрела_1_f;
            Выстрел_2_f = _Набор_стрел.Стрела_2_f;
            Выстрел_3_f = _Набор_стрел.Стрела_3_f;
            Выстрел_1_m =num - (_Стрелки.Выстрел_1_r - 3)/(R*2/num);
            Выстрел_2_m =num - (_Стрелки.Выстрел_2_r - 3)/(R*2/num);
            Выстрел_3_m =num - (_Стрелки.Выстрел_3_r - 3)/(R*2/num);
            Сумма=(_Стрелки.Выстрел_1_m + _Стрелки.Выстрел_2_m +
_Стрелки.Выстрел_3_m);
            Среднее=_Стрелки.Сумма/3;
            Максимальное=IMax(
                        IMax(_Стрелки.Выстрел_1_m,_Стрелки.Выстрел_2_m),
                  IMax(_Стрелки.Выстрел_1_m,_Стрелки.Выстрел_3_m));
     Convert_end
            состояние_стрелка = Ожидает;
            количестов_попыток++;
$End
```

Вкладка с расширением .dpt

```
$Decision_point model: some
$Condition NoCheck
$Activities
     Подсчет: Образец_подсчета
$End
```

Вкладка с расширением .frm

```
$Sprite StrelokInfo()
     integer center = 2*R + 30;
     integer step = R*2/num;
```

```
    for (integer i=0;i<num/2;i++)
    {
          circle (center,center,2*R - 2*step*i + 2          ,black,black);
          circle (center,center,2*R - 2*step*i              ,red  ,black);
          circle (center,center,2*R - 2*R/num - 2*step*i + 2,black,black);
          circle (center,center,2*R - 2*R/num - 2*step*i    ,white,black);
    }

    for (integer j=1;j<num;j++)
    {
          text (center - step*j - 20,center - 10,20,20,transparent,black,= num - j);
          text (center + step*j     ,center - 10,20,20,transparent,black,= num - j);
    }
    text (center - 10,center - 10,20,20,transparent,black,= num);

    rect (550, 50, 210,  30,transparent,black);
    text (550, 50, 210,  30,transparent,black,= 'Таблица результатов');
    rect (760,230,-210,-180,transparent,black);

    rect (550,80 , 70, 30,transparent,black);
    text (550,80 , 70, 30,transparent,black,= 'Выстрел 1');
    rect (550,110, 70, 30,transparent,black);

    rect (620, 80, 70, 30,transparent,black);
    text (620, 80, 70, 30,transparent,black,= 'Выстрел 2');
    rect (620,110, 70, 30,transparent,black);

    rect (690, 80, 70, 30,transparent,black);
    text (690, 80, 70, 30,transparent,black,= 'Выстрел 3');
    rect (690,110, 70, 30,transparent,black);

    rect (550,140,140, 30,transparent,black);
    text (550,140,140, 30,transparent,black,= 'Максимальное');
    rect (690,140, 70, 30,transparent,black);

    rect (550,170,140, 30,transparent,black);
    text (550,170,140, 30,transparent,black,= 'Среднее');
    rect (690,170, 70, 30,transparent,black);

    rect (550,200,140, 30,transparent,black);
    text (550,200,140, 30,transparent,black,= 'Сумма');
    rect (690,200, 70, 30,transparent,black);
$End

$Frame Petr_frame
$Back_picture = <23, 124, 253> 1148 480
    sprite StrelokInfo();
    integer center =2*R + 30;
    integer step = R*2/num;
    text (550,110,70,30,transparent,black,= Петр.Выстрел_1_m);
    text (620,110,70,30,transparent,black,= Петр.Выстрел_2_m);
    text (690,110,70,30,transparent,black,= Петр.Выстрел_3_m);
    text (690,170,70,30,transparent,black,= Петр.Среднее);
    text (690,140,70,30,transparent,black,= Петр.Максимальное);
    text (690,200,70,30,transparent,black,= Петр.Сумма);
    circle (center + Петр.Выстрел_1_r*Cos(Петр.Выстрел_1_f),
            center + Петр.Выстрел_1_r*Sin(Петр.Выстрел_1_f),
                                                    5,black,black);
    circle (center + Петр.Выстрел_2_r*Cos(Петр.Выстрел_2_f),
            center + Петр.Выстрел_2_r*Sin(Петр.Выстрел_2_f),
                                                    5,black,black);
```

```
    circle (center + Петр.Выстрел_3_r*Cos(Петр.Выстрел_3_f),
            center + Петр.Выстрел_3_r*Sin(Петр.Выстрел_3_f),
                                                        5,black,black);
$End
$Frame Ivan_frame
$Back_picture = <58, 204, 147> 1148 480
    sprite StrelokInfo();
    integer center = 2*R + 30;
    integer step = R*2/num;
    text (550,110,70,30,transparent,black,= Иван.Выстрел_1_m);
    text (620,110,70,30,transparent,black,= Иван.Выстрел_2_m);
    text (690,110,70,30,transparent,black,= Иван.Выстрел_3_m);
    text (690,170,70,30,transparent,black,= Иван.Среднее);
    text (690,140,70,30,transparent,black,= Иван.Максимальное);
    text (690,200,70,30,transparent,black,= Иван.Сумма);
    circle (center + Иван.Выстрел_1_r*Cos(Иван.Выстрел_1_f),
            center + Иван.Выстрел_1_r*Sin(Иван.Выстрел_1_f),
                                                        5,black,black);
    circle (center + Иван.Выстрел_2_r*Cos(Иван.Выстрел_2_f),
            center + Иван.Выстрел_2_r*Sin(Иван.Выстрел_2_f),
                                                        5,black,black);
    circle (center + Иван.Выстрел_3_r*Cos(Иван.Выстрел_3_f),
            center + Иван.Выстрел_3_r*Sin(Иван.Выстрел_3_f),
                                                        5,black,black);
$End

$Frame Stepan_frame
$Back_picture = <213, 187, 66> 1148 480
    sprite StrelokInfo();
    integer center = 2*R + 30;
    integer step = R*2/num;
    text (550,110,70,30,transparent,black,= Степан.Выстрел_1_m);
    text (620,110,70,30,transparent,black,= Степан.Выстрел_2_m);
    text (690,110,70,30,transparent,black,= Степан.Выстрел_3_m);
    text (690,170,70,30,transparent,black,= Степан.Среднее);
    text (690,140,70,30,transparent,black,= Степан.Максимальное);
    text (690,200,70,30,transparent,black,= Степан.Сумма);
    circle (center + Степан.Выстрел_1_r*Cos(Степан.Выстрел_1_f),
            center + Степан.Выстрел_1_r*Sin(Степан.Выстрел_1_f),
                                                        5,black,black);
    circle (center + Степан.Выстрел_2_r*Cos(Степан.Выстрел_2_f),
            center + Степан.Выстрел_2_r*Sin(Степан.Выстрел_2_f),
                                                        5,black,black);
    circle (center + Степан.Выстрел_3_r*Cos(Степан.Выстрел_3_f),
            center + Степан.Выстрел_3_r*Sin(Степан.Выстрел_3_f),
                                                        5,black,black);
$End
```

Вкладка с расширением .fun

```
$Constant
    R : integer = 100
    num : integer = 10
$End

$Sequence rnd : integer
$Type = uniform 123456789
$End

$Sequence rndr : real
$Type = uniform 123456789
$End

$Sequence Интервал_подготовки : real
$Type = exponential 123456789
```

```
$End

$Sequence Длительность_стрельбы : real
$Type = uniform 123456789
$End
```

Вкладка с расширением .smr

```
Show_mode = Animation
Show_rate = 100000

Образец_стрельбы.planning( time_now + Интервал_подготовки( 30 ))

Terminate_if Time_now >= 12 * 7 * 60
```

Приложение 2. Вкладка с расширением .frm без использования спрайтов

```
$Frame Petr_frame
$Back_picture = <23, 124, 253> 1148 480
     integer center = 2*R + 30;
     integer step = R*2/num;

     for (integer i=0;i<num/2;i++)
     {
           circle (center,center,2*R - 2*step*i + 2          ,black,black);
           circle (center,center,2*R - 2*step*i              ,red  ,black);
           circle (center,center,2*R - 2*R/num - 2*step*i + 2,black,black);
           circle (center,center,2*R - 2*R/num - 2*step*i    ,white,black);
     }

     for (integer j=1;j<num;j++)
     {
           text (center - step*j - 20,center - 10,20,20,transparent,black,= num - j);
           text (center + step*j     ,center - 10,20,20,transparent,black,= num - j);
     }
     text (center - 10,center - 10,20,20,transparent,black,= num);

     rect (550, 50, 210,  30,transparent,black);
     text (550, 50, 210,  30,transparent,black,= 'Таблица результатов');
     rect (760,230,-210,-180,transparent,black);

     rect (550,80 , 70, 30,transparent,black);
     text (550,80 , 70, 30,transparent,black,= 'Выстрел 1');
     rect (550,110, 70, 30,transparent,black);

     rect (620, 80, 70, 30,transparent,black);
     text (620, 80, 70, 30,transparent,black,= 'Выстрел 2');
     rect (620,110, 70, 30,transparent,black);

     rect (690, 80, 70, 30,transparent,black);
     text (690, 80, 70, 30,transparent,black,= 'Выстрел 3');
     rect (690,110, 70, 30,transparent,black);

     rect (550,140,140, 30,transparent,black);
     text (550,140,140, 30,transparent,black,= 'Максимальное');
     rect (690,140, 70, 30,transparent,black);

     rect (550,170,140, 30,transparent,black);
     text (550,170,140, 30,transparent,black,= 'Среднее');
     rect (690,170, 70, 30,transparent,black);

     rect (550,200,140, 30,transparent,black);
     text (550,200,140, 30,transparent,black,= 'Сумма');
     rect (690,200, 70, 30,transparent,black);
```

```
     text (550,110,70,30,transparent,black,= Петр.Выстрел_1_m);
     text (620,110,70,30,transparent,black,= Петр.Выстрел_2_m);
     text (690,110,70,30,transparent,black,= Петр.Выстрел_3_m);
     text (690,170,70,30,transparent,black,= Петр.Среднее);
     text (690,140,70,30,transparent,black,= Петр.Максимальное);
     text (690,200,70,30,transparent,black,= Петр.Сумма);
     circle (center + Петр.Выстрел_1_r*Cos(Петр.Выстрел_1_f),
             center + Петр.Выстрел_1_r*Sin(Петр.Выстрел_1_f),
                                                    5,black,black);
     circle (center + Петр.Выстрел_2_r*Cos(Петр.Выстрел_2_f),
             center + Петр.Выстрел_2_r*Sin(Петр.Выстрел_2_f),
                                                    5,black,black);
     circle (center + Петр.Выстрел_3_r*Cos(Петр.Выстрел_3_f),
             center + Петр.Выстрел_3_r*Sin(Петр.Выстрел_3_f),
                                                    5,black,black);
$End

$Frame Ivan_frame
$Back_picture = <58, 204, 147> 1148 480
     integer center = 2*R + 30;
     integer step = R*2/num;

     for (integer i=0;i<num/2;i++)
     {
           circle (center,center,2*R - 2*step*i + 2          ,black,black);
           circle (center,center,2*R - 2*step*i              ,red  ,black);
           circle (center,center,2*R - 2*R/num - 2*step*i + 2,black,black);
           circle (center,center,2*R - 2*R/num - 2*step*i    ,white,black);
     }

     for (integer j=1;j<num;j++)
     {
           text (center - step*j - 20,center - 10,20,20,transparent,black,= num - j);
           text (center + step*j     ,center - 10,20,20,transparent,black,= num - j);
     }
     text (center - 10,center - 10,20,20,transparent,black,= num);

     rect (550, 50, 210,  30,transparent,black);
     text (550, 50, 210,  30,transparent,black,= 'Таблица результатов');
     rect (760,230,-210,-180,transparent,black);

     rect (550,80 , 70, 30,transparent,black);
     text (550,80 , 70, 30,transparent,black,= 'Выстрел 1');
     rect (550,110, 70, 30,transparent,black);

     rect (620, 80, 70, 30,transparent,black);
     text (620, 80, 70, 30,transparent,black,= 'Выстрел 2');
     rect (620,110, 70, 30,transparent,black);

     rect (690, 80, 70, 30,transparent,black);
     text (690, 80, 70, 30,transparent,black,= 'Выстрел 3');
     rect (690,110, 70, 30,transparent,black);

     rect (550,140,140, 30,transparent,black);
     text (550,140,140, 30,transparent,black,= 'Максимальное');
     rect (690,140, 70, 30,transparent,black);

     rect (550,170,140, 30,transparent,black);
     text (550,170,140, 30,transparent,black,= 'Среднее');
     rect (690,170, 70, 30,transparent,black);

     rect (550,200,140, 30,transparent,black);
     text (550,200,140, 30,transparent,black,= 'Сумма');
     rect (690,200, 70, 30,transparent,black);
```

```
     text (550,110,70,30,transparent,black,= Иван.Выстрел_1_m);
     text (620,110,70,30,transparent,black,= Иван.Выстрел_2_m);
     text (690,110,70,30,transparent,black,= Иван.Выстрел_3_m);
     text (690,170,70,30,transparent,black,= Иван.Среднее);
     text (690,140,70,30,transparent,black,= Иван.Максимальное);
     text (690,200,70,30,transparent,black,= Иван.Сумма);
     circle (center + Иван.Выстрел_1_r*Cos(Иван.Выстрел_1_f),
             center + Иван.Выстрел_1_r*Sin(Иван.Выстрел_1_f),
                                                    5,black,black);
     circle (center + Иван.Выстрел_2_r*Cos(Иван.Выстрел_2_f),
             center + Иван.Выстрел_2_r*Sin(Иван.Выстрел_2_f),
                                                    5,black,black);
     circle (center + Иван.Выстрел_3_r*Cos(Иван.Выстрел_3_f),
             center + Иван.Выстрел_3_r*Sin(Иван.Выстрел_3_f),
                                                    5,black,black);
$End

$Frame Stepan_frame
$Back_picture = <213, 187, 66> 1148 480
     integer center = 2*R + 30;
     integer step = R*2/num;

     for (integer i=0;i<num/2;i++)
     {
           circle (center,center,2*R - 2*step*i + 2          ,black,black);
           circle (center,center,2*R - 2*step*i              ,red  ,black);
           circle (center,center,2*R - 2*R/num - 2*step*i + 2,black,black);
           circle (center,center,2*R - 2*R/num - 2*step*i    ,white,black);
     }

     for (integer j=1;j<num;j++)
     {
           text (center - step*j - 20,center - 10,20,20,transparent,black,= num - j);
           text (center + step*j     ,center - 10,20,20,transparent,black,= num - j);
     }
     text (center - 10,center - 10,20,20,transparent,black,= num);

     rect (550, 50, 210,  30,transparent,black);
     text (550, 50, 210,  30,transparent,black,= 'Таблица результатов');
     rect (760,230,-210,-180,transparent,black);

     rect (550,80 , 70, 30,transparent,black);
     text (550,80 , 70, 30,transparent,black,= 'Выстрел 1');
     rect (550,110, 70, 30,transparent,black);

     rect (620, 80, 70, 30,transparent,black);
     text (620, 80, 70, 30,transparent,black,= 'Выстрел 2');
     rect (620,110, 70, 30,transparent,black);

     rect (690, 80, 70, 30,transparent,black);
     text (690, 80, 70, 30,transparent,black,= 'Выстрел 3');
     rect (690,110, 70, 30,transparent,black);

     rect (550,140,140, 30,transparent,black);
     text (550,140,140, 30,transparent,black,= 'Максимальное');
     rect (690,140, 70, 30,transparent,black);

     rect (550,170,140, 30,transparent,black);
     text (550,170,140, 30,transparent,black,= 'Среднее');
     rect (690,170, 70, 30,transparent,black);

     rect (550,200,140, 30,transparent,black);
     text (550,200,140, 30,transparent,black,= 'Сумма');
```

```
    rect (690,200, 70, 30,transparent,black);

    text (550,110,70,30,transparent,black,= Степан.Выстрел_1_m);
    text (620,110,70,30,transparent,black,= Степан.Выстрел_2_m);
    text (690,110,70,30,transparent,black,= Степан.Выстрел_3_m);
    text (690,170,70,30,transparent,black,= Степан.Среднее);
    text (690,140,70,30,transparent,black,= Степан.Максимальное);
    text (690,200,70,30,transparent,black,= Степан.Сумма);
    circle (center + Степан.Выстрел_1_r*Cos(Степан.Выстрел_1_f),
            center + Степан.Выстрел_1_r*Sin(Степан.Выстрел_1_f),
                                                        5,black,black);
    circle (center + Степан.Выстрел_2_r*Cos(Степан.Выстрел_2_f),
            center + Степан.Выстрел_2_r*Sin(Степан.Выстрел_2_f),
                                                        5,black,black);
    circle (center + Степан.Выстрел_3_r*Cos(Степан.Выстрел_3_f),
            center + Степан.Выстрел_3_r*Sin(Степан.Выстрел_3_f),
                                                        5,black,black);
$End
```