Оглавление

1 Введение

Имитационное моделирование — метод, позволяющий строить модели, описывающие процессы так, как они проходили бы в действительности. Такую модель можно «проиграть» во времени как для одного испытания, так и заданного их множества. При этом результаты будут определяться случайным характером процессов. По этим данным можно получить достаточно устойчивую статистику.

Имитационное моделирование — это метод исследования, при котором изучаемая система заменяется моделью с достаточной точностью описывающей реальную систему и с ней проводятся эксперименты с целью получения информации об этой системе. Экспериментирование с моделью называют имитацией (имитация — это постижение сути явления, не прибегая к экспериментам на реальном объекте).

Имитационное моделирование — это частный случай математического моделирования. Существует класс объектов, для которых по различным причинам не разработаны аналитические модели, либо не разработаны методы решения полученной модели. В этом случае математическая модель заменяется имитатором или имитационной моделью.

Имитационное моделирование является мощным инструментом исследования поведения реальных систем. Методы имитационного моделирования позволяют собрать необходимую информацию о поведении системы путем создания ее компьютерной модели. Эта информация используется затем для проектирования системы.

С помощью имитационного моделирования можно ответить на множество вопросов, возникающих в момент принятия решения об изменениях в процессах, происходящих в бизнесе:

Как изменится рентабельность бизнеса?

Как проведенные изменения отразятся на производительности технологического оборудования и персонала?

Какие дополнительные инвестиции потребуется сделать?

Каков срок окупаемости производимых инвестиций?

Очень часто эти вопросы остаются без ответов до тех пор, пока предполагаемые изменения не будут осуществлены. Но после того как изменения проделаны, цена исправления ошибочных решений становится значительно выше. Имитационное моделирование дает возможность тестировать разные идеи, «проигрывая» их на

компьютерной модели, что намного дешевле, чем проводить множество испытаний и исправлений ошибок на реальных процессах.

Типичные примеры, где может быть с выгодой применено имитационное моделирование:

1. Строительство нового производства любой отрасли: машиностроение, металлургия, нефтехимическая промышленность, деревообработка и др.
2. Расширение и модернизация существующего производства.
3. Постановка на производство новой продукции.
4. Проектирование системы транспортировки угля, руды из шахты на поверхность и далее к потребителям.
5. Организация логистической системы, состоящей из дистрибутивных центров, складов, транспортных средств.
6. Строительство транспортного узла.
7. Технико-экономическое обоснование внедрения автоматизированных систем оперативным управлением производством, складом, транспортным предприятием.

Одна из причин для того, чтобы использовать имитационное моделирование – это повышение уровня автоматизации производства, нацеленное на повышение производительности, качества продукции и на снижение затрат, приведшее к увеличению сложности производственных систем. А проблемы, возникающие в системах такой сложности, могут быть проанализированы только с применением компьютерного моделирования. Еще одна причина в том, что применение анимации в моделировании повысило возможность большего понимания имитационных моделей неспециалистами в моделировании, т.е. руководителями, менеджерами и инженерами-производственниками.

Компьютерное моделирование включает в себя построение модели отдельного агрегата, технологического процесса, всего производства в целом, логистической системы с применением специализированного программного обеспечения. Эта модель будет полностью воспроизводить все процессы, происходящие в реальности на производстве, складе, в любой логистической системе. Используя модель, можно экспериментировать, проверять разные идеи для понимания того, как реальная система будет вести себя в разных ситуациях. Результаты имитации могут быть использованы при решении оптимизационных задач в качестве оценки значений функциональных характеристик моделируемой системы

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

Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. 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, характеризующиеся 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, использовании нечетких данных, снимает часть проблем использования ИМ.

2 Предпроектное исследование

2.1 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.

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

2.2 Процесс имитации в РДО

Для имитации работы модели в РДО реализованы два подхода:

- событийный
- сканирования активностей.

2.2.1 Событийный подход.

При событийном подходе исследователь описывает события, которые могут изменять состояние системы, и определяет логические взаимосвязи между ними. Начальное состояние устанавливается путем задания значений переменным модели и параметров генераторам случайных чисел. Имитация происходит путем выбора из списка будущих событий ближайшего по времени и его выполнения. Выполнение события приводит к изменению состояния системы и генерации

будущих событий, логически связанных с выполняемым. Эти события заносятся в список будущих событий и упорядочиваются в нем по времени наступления. Например, событие начала обработки детали на станке приводит к появлению в списке будущих событий события окончания обработки детали, которое должно наступить в момент времени равный текущему времени плюс время, требуемое на обработку детали на станке. В событийных системах модельное время фиксируется только в моменты изменения состояний(Рисунок 1)

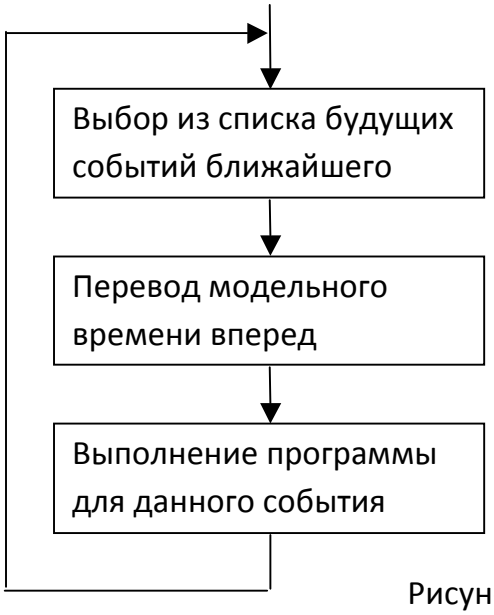


Рисунок 1

2.2.2 Подход сканирования активностей.

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

2.3 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)

2.4 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.

2.5 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 2).

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

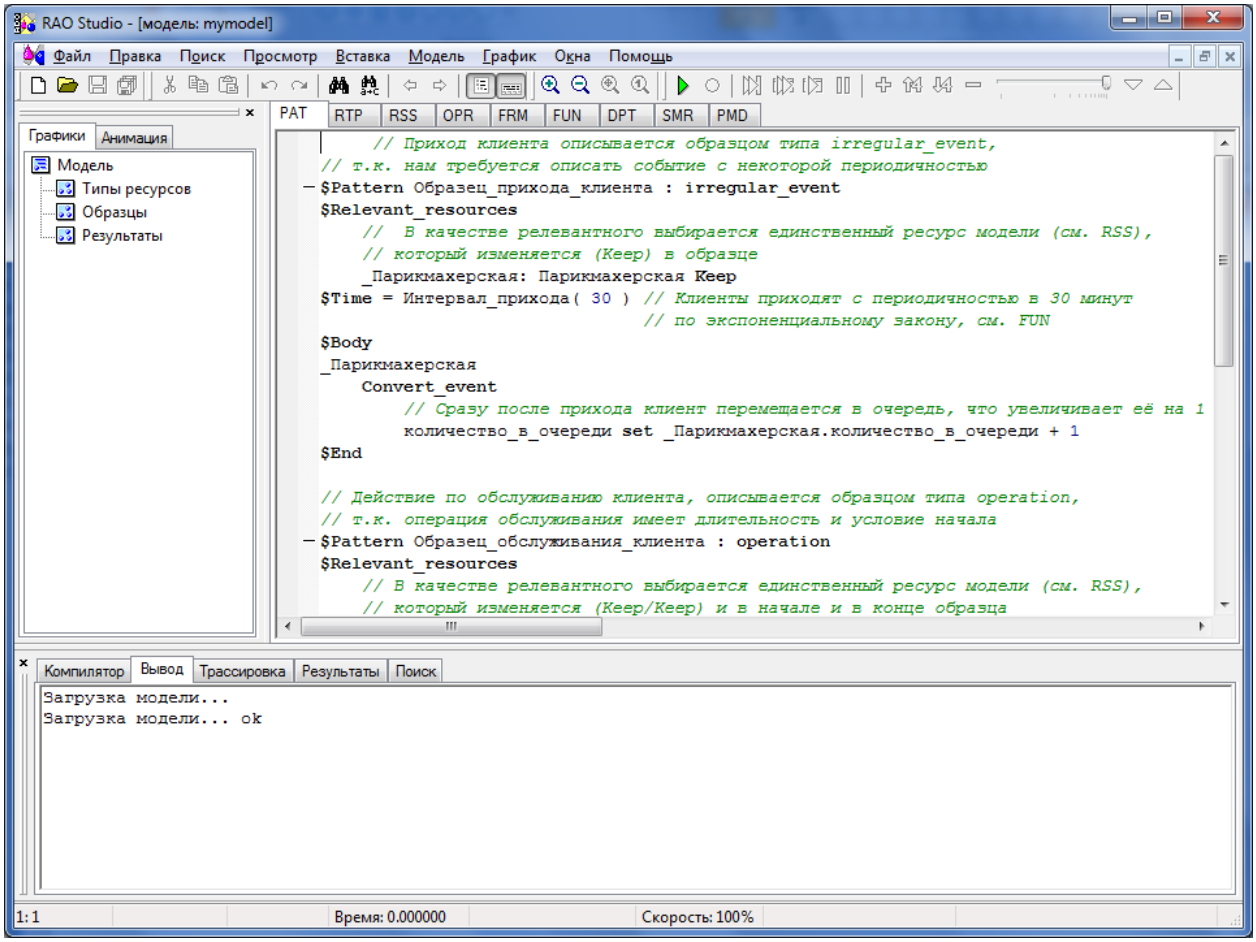


Рисунок 2

2.6 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

2.7 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

2.8 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.

2.9 Исследование структуры ПО

Ниже приведена UML диаграмма компонентов, отображающая структуру программного комплекса RAO-studio(Рисунок 3):

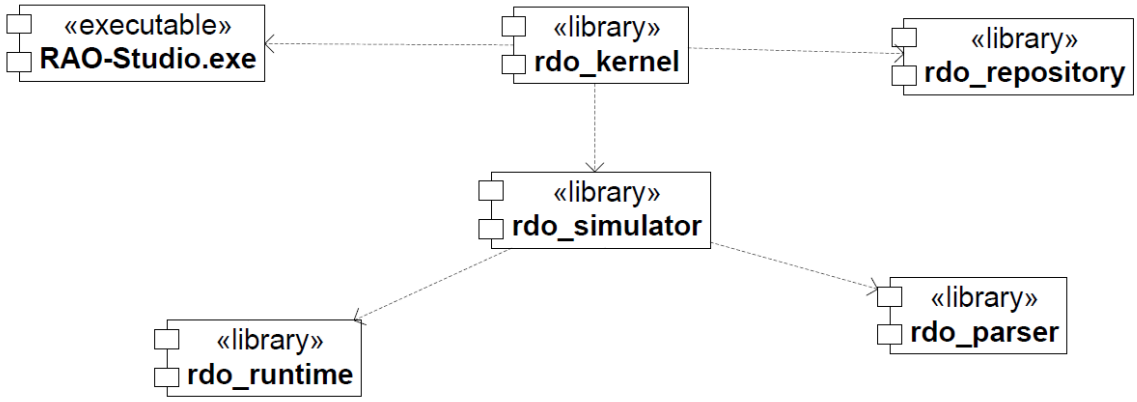


Рисунок 3

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_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.

2.10 Исследование принципа работы ПО

Ниже приведена IDEF0-диаграмма (Рисунок 4), на которой отображен принцип функционирования RAO-studio

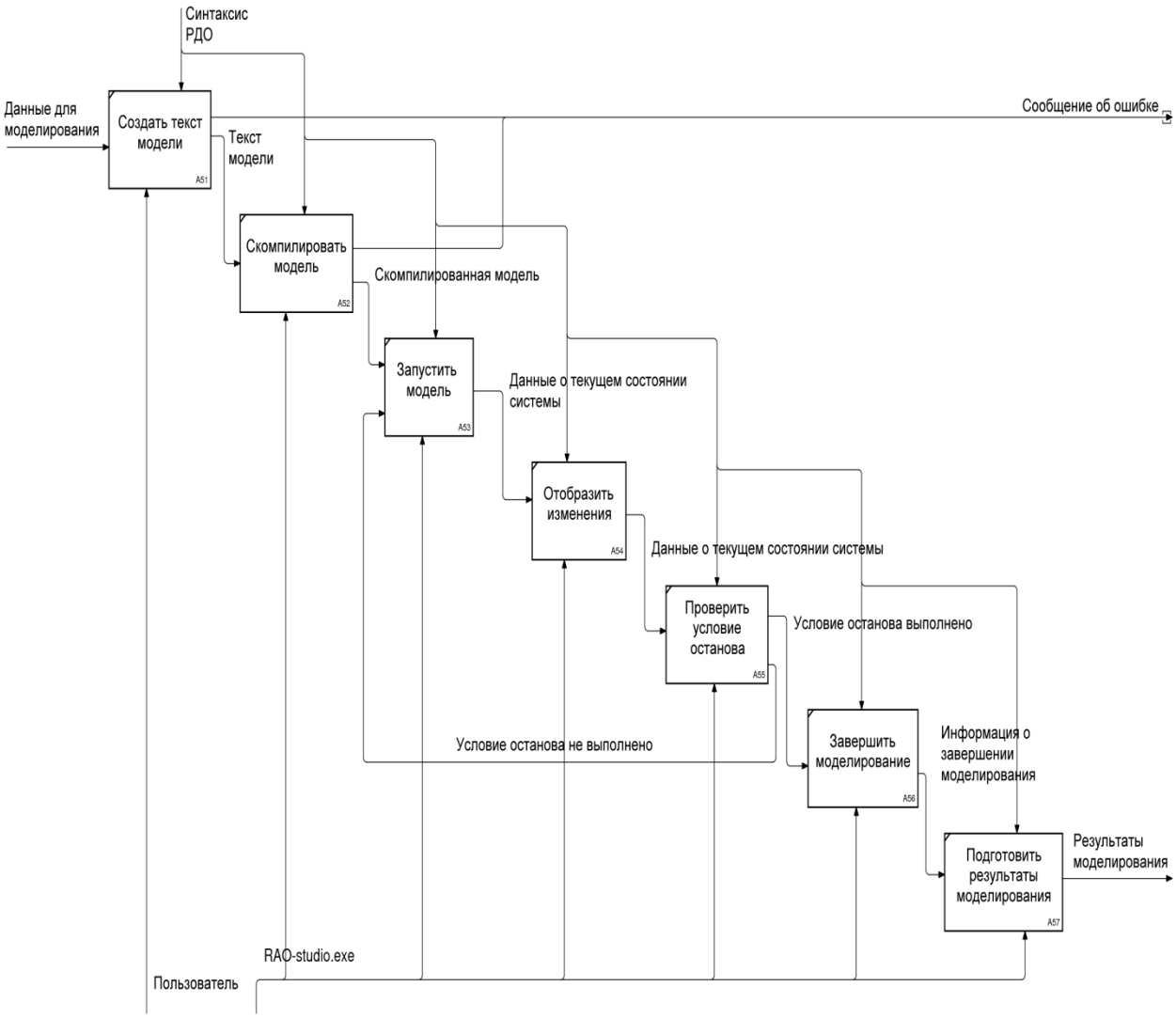


Рисунок 4

На вход поступает текст модели, который обрабатывается компилятором. Если выявляются ошибки – то система выдает сообщение об ошибке. Существует 2 типа ошибок – 1 рода и 2 рода. Первый тип ошибок подразумевает ошибки в лексике РДО, второй тип ошибок подразумевает непосредственно ошибки написания модели. После компиляции модель запускается. В момент когда необходимо отрисовать анимацию модель останавливается и в систему поступает информация о текущем состоянии модели и проверяется выполнение условия останова моделирования. Этот процесс повторяется до тех пор, пока не будет выполнено условие останова.

3 Концептуальное проектирование

3.1 Выбор интерфейса

Так как существующая система имитационного моделирования РДО уже имеет достаточно удобный интерфейс взаимодействия с пользователем, то добавление новых возможностей разумно производить на существующей основе.

Основными элементами, которые позволят пользователю пользоваться возможностями визуального программирования на РДО, являются:

- Меню выбора типа графического блока
- Окно для создания графической модели
- Панель инструментов для работы с графической моделью

Меню выбора типа графического блока возможно создать как дополнительную закладку в окно объектов, либо как дополнительную панель объектов. В случае создания дополнительной панели объектов, произойдет потеря рабочего места. Перед глазами пользователя всегда будет графическая составляющая. Наиболее удобно будет интегрировать меню выбора типа графического блока в качестве еще одной закладки в окно объектов. Таким образом, пользователь не будет отвлекаться на графическую составляющую, если будет работать с текстовой составляющей.

Существует две потенциальные возможности внедрения окна для создания графической модели. В первом случае в этом окне будет создаваться как дополнительная вкладка в главном окне. В этом случае для переключения в режим визуального программирования пользователю необходимо просто переключиться из одной вкладки в другую. Плюсами данной реализации является сохранение главного окна. Так же она имеет ряд недостатков. Текст модели уже разнесен в разные вкладки, добавление еще одной будет дезориентировать пользователя. Во втором случае будет создаваться еще одно окно параллельно с главным окном. Недостатком в данной реализации является собственно создание еще одного окна. С другой стороны это будет отдельное окно для визуального программирования, в котором не будет элементов, которые не имеют отношения к визуальному программированию. Переключаться между окнами так же удобно, как и между вкладками в существующем главном окне.

Существует возможность внедрения панели инструментов для работы с графической моделью рядом с уже существующими панелями, либо непосредственно в новом окне, которое будет создано специально для визуального программирования на РДО.

3.2 Принципы построения системы

При создании модуля визуального программирования на РДО будем руководствоваться тем, что код системы имитационного моделирования РДО написан на языке C++. Так же следует учесть то, что модели созданные в графической среде должны запускаться в РДО, значит необходимо учесть синтаксис РДО (Рисунок 5).

Рисунок 5

4 Техническое задание

1. Общие сведения

1.1. Полное наименование системы и ее условное обозначение

Разрабатываемый программный продукт носит название «Модуль визуального программирования на РДО».

1.2. Наименование предприятий разработчика и заказчика системы

Разрабатывается на кафедре РК9 Московского Государственного Университета им. Баумана.

1.3. Перечень документов, на основании которых создается система

Разрабатывается на основании учебного плана для 5-го курса X семестр от 2009 года.

1.4. Плановые сроки начала и окончания работы по созданию системы

Сроки начала и окончания работы принимаются на основании учебного плана и внутренних распоряжений кафедры РК9.

1.5. Порядок оформления и предъявления заказчику результатов работ по созданию системы, по изготовлению и наладке отдельных средств и программно-технических комплексов системы.

Порядок оформления и предъявления результатов работ по созданию системы устанавливается в соответствии с правилами выполнения и сдачи курсового проекта.

2. Назначение и цели создания системы

2.1. Назначение системы

Модуль визуального программирования на РДО, выполнять создание модели на РДО, соответствующей наглядному представлению модели и заданным в ней параметрам, из системы имитационного моделирования РДО.

2.2. Цели создания системы.

Целью создания является добавление в систему имитационного моделирования РДО возможности визуального программирования моделей.

3. Требования к системе

3.1. Требования к системе в целом

3.1.1. Требования к структуре и функционированию системы

Модуль визуального программирования на РДО должен быть написан на языке C++

Модели, созданные с помощью визуального программирования должны проигрываться в системе имитационного моделирования РДО.

Использование графического Модуль визуального программирования на РДО должно быть возможным на обычных персональных компьютерах без использования специальной аппаратной части.

3.1.2. Требования к численности и квалификации персонала системы и режиму его работы

Для модуля визуального программирования на РДО необходимо умение работать со средами моделирования.

3.1.3. Требования к надежности системы

Во время функционирования системы пользователь может изменять геометрическое положение блоков, менять связи между ними и добавлять новые блоки. В случае некорректности соединения блоков между собой должно быть выдано сообщение об ошибке соединения блоков между собой.

3.1.4. Требования к защите информации от несанкционированного доступа

Модуль визуального программирования на РДО и документация к нему распространяется на условиях лицензирования РДО.

3.1.5. Требования по стандартизации и унификации

Модуль визуального программирования на РДО должен быть совместим с современными программными продуктами, написанными с использованием современных языков программирования. Обращение к модулю визуального программирования на РДО должно осуществляться с использованием стандартных интерфейсов языков программирования, родственных языку C++.

3.2. Требования к функциям, выполняемым системой

Модуль визуального программирования на РДО должен:

- создавать полноценные модели
- созданные модели должны удовлетворять синтаксису РДО
- созданные модели должны запускаться из системы имитационного моделирования РДО

4. Состав и содержание работ по созданию системы

Разработка модуль визуального программирования на РДО производится в четыре этапа.

1) Обзор аналогичных программных продуктов.
2) Проектирование программного продукта
3) Написание и отладка программного продукта
4) Документирование программного продукта

Сроки выполнения всех этапов принимаются в соответствии и с учебным планом.

5. **Порядок контроля и приемки системы**

   Корректность работы модуля визуального программирования на РДО тестируется на простых моделях, имеющих конкретное теоретическое решение. Сдача модуля визуального программирования на РДО производится в соответствии с учебным планом и внутренними распоряжениями кафедры.

6. **Требования к документированию**

   В состав программной документации должны всходить: Описание работы модуля визуального программирования на РДО, описание типов и структуры входных и выходных параметров, описание границ применения модуля визуального программирования на РДО.

5 Техническое проектирование

Для обеспечения полной функциональности в соответствии с предыдущими утверждениями блок модуль визуального программирования на РДО должен состоять из:

1) Базовые классы
2) Окна для визуального программирования
3) Панель инструментов

Далее назначение каждого пункта будет рассмотрено более детально

5.1 Базовые классы

Назначение: организация взаимодействия модуля визуального программирования и системы ИМ РДО

К базовым относим классы, отвечающие за:

- Создание меню для выбора типа графических объектов
- Создание главного окна визуального программирования
- Создание панели инструментов
- Генерация кода

5.2 Окна для визуального программирования

Назначение: интерфейс для обеспечения взаимодействия пользователя и РДО

Выделяем 2 окна:

1) Меню – содержит все типовые объекты, которые являются конструкциями для создания графических моделей (Рисунок 6) В дальнейшем эти объекты будут называться блоками.
   На данный момент в системе существует пять блоков:
   - Генератор
   - Терминатор
   - Процесс
   - Ресурс
   - Десайд

   Далее назначение каждого блока мы рассмотрим подробнее.

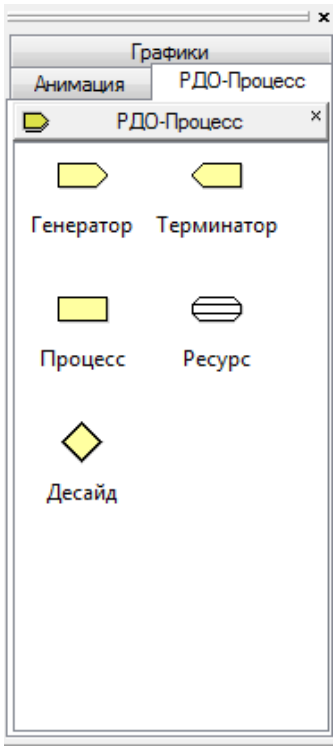


Рисунок 6

2) Главное окно визуального программирования – представляет собой поле для визуального создания модели (Рисунок 7). В это поле из меню объектов перетаскиваются(методом drag&drop) интересующие нас блоки.

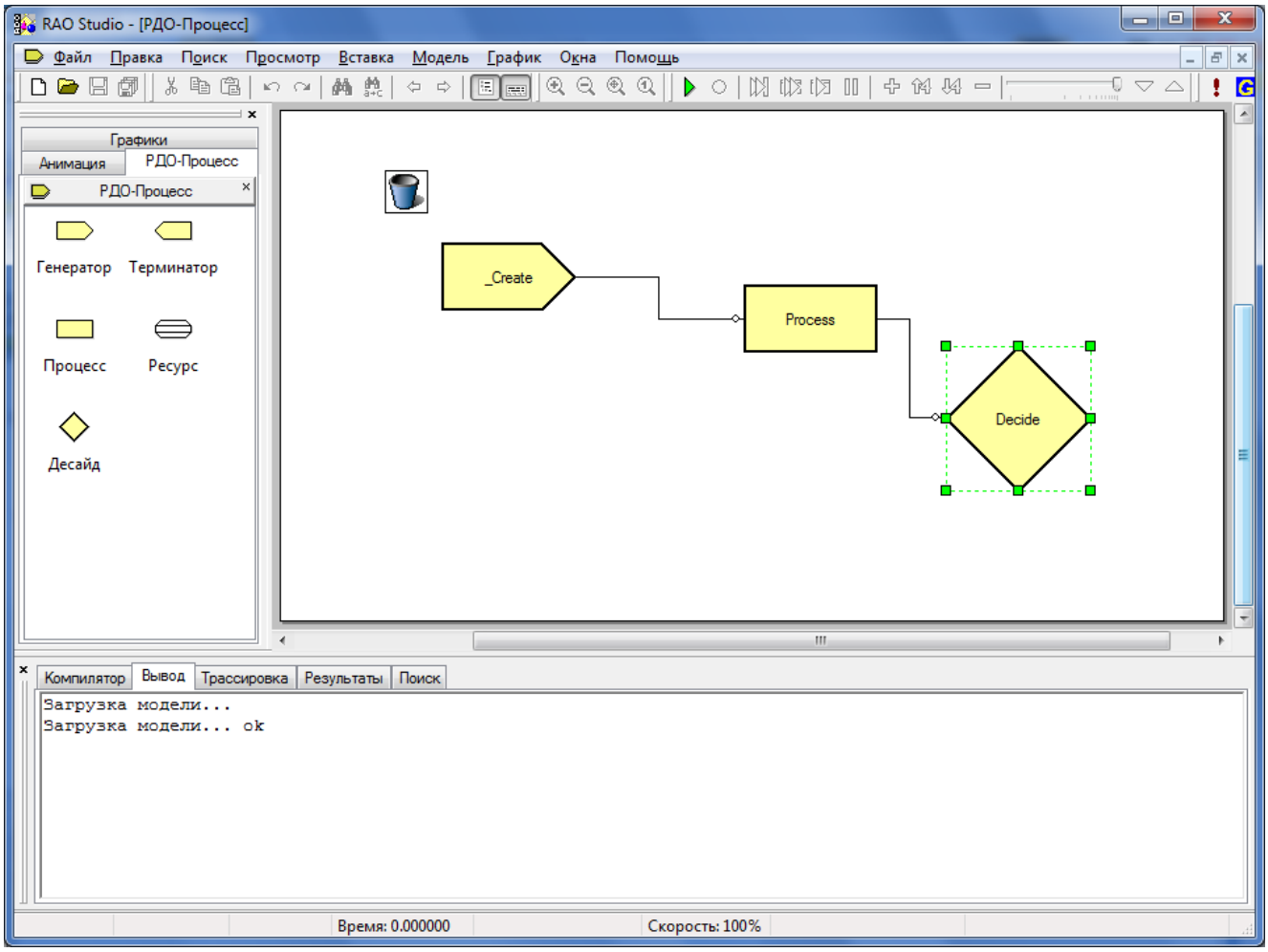


Рисунок 7

5.2.1 Генератор

Это блок, который генерирует появление транзактов в модели. Является необходимым для работы модели. Имеет диалоговое окно настроек создания транзактов. Существует возможность изменять такие параметры создания транзактов как количество создаваемых транзактов, время прибытия первого транзакта, и интервал между поступлениями транзактов. Интервал в свою очередь может быть представлен в виде одного из законов распределения (константа, нормальный, равномерный, экспоненциальный)

5.2.2 Терминатор

Это блок, который выводит транзакты из системы. Так же как и Генератор имеет диалоговое окно. Предусмотрена опция присвоения данному блоку произвольного имени в модели.

5.2.3 Процесс

Блок Процесс моделирует какие-либо процессы. В диалоговом окне предоставлена возможность выбрать действия с ресурсами, если такие имеются в модели, а так же дисциплину очереди поступления транзактов и интервал поступления транзактов. Позволяет моделировать разнообразные ситуации.

5.2.4 Ресурс

Позволяет пользователю программы добавлять в модель дополнительные ресурсы, по своему усмотрению.

5.2.5 Десайд

Осуществляет вероятностный выбор дальнейшего поведения транзакта в системе. Предусмотрена возможность движения транзакта по двум разным напрвлениям

5.3 Панель инструментов

Состоит из двух кнопок – кнопки создания модели и кнопки определения условия останова моделирования (Рисунок 8). Существует два условия останова:

- по времени
- по количеству транзактов в системе

Рисунок 8

После срабатывания кнопки создания модели, произойдет генерация кода графической модели и открытие этого кода в главном окне системы ИМ РДО.

После этого данную модель можно запустить и получить результаты моделирования.

5.4 Перспективы развития

На данном этапе развития модуль визуального программирования работает следующим образом:

1) Визуальное программирование процесса

2) Перевод визуальной модели в текстовую модель
3) Компиляция модели
4) Запуск модели и получение результатов моделирования

Данный механизм имеет ряд неудобств. Хотелось бы проводить компиляцию и узнавать об ошибках создания модели, если таковые имеются, без создания текстового кода модели.

Тогда механизм создания модели можно представить следующим образом:

1) Визуальное программирование процесса
2) Компиляция модели
3) Запуск модели и получение результатов моделирования

Также на данном этапе развития отсутствует как таковая обратная связь окна с визуальной моделью и ядром системы. Это означает, что мы не имеем возможности «пронаблюдать» поведение нашей модели непосредственно в главном окне визуального программирования.

Итак, можно наметить основные цели по продолжению работы в области визуального программирования на РДО:

- Обеспечение обратной связью ядра системы и модуля для визуального программирования
- Компилирование модели «напрямую», без генерации текста модели.

6 Этап использования визуального программирования на РДО

6.1 Создание меню выбора типа объектов

Меню выбора представляет собой класс `RPPageCtrl`, который является потомком класса `CWnd`.(Страуструп, 2008)

Вкладка же создается в окне класса `RDOStudioWorkspace`

Пример создания меню выбора:(Центр разработки на C++)

```
	pagectrl = new RPPageCtrl;
	pagectrl->Create( "", "", 0 , CRect(0, 0, 0, 0), &tab, 0);

	tab.insertItem(pagectrl,rdo::format(IDS_TAB_PAGECTRL).c_str());

	studioApp.mainFrame->registerCmdWnd( pagectrl )
```

tab.insertItem(что вставляем, название вкалдки)

Так же данную вкладку необходимо заполнить данными о типах объектов. (Центр разработки на C++)Заполнение происходит вызовом метода `insertMethod` класса `RPMethodManager(Страуструп, 2008)`

```
	void RPMethodManager::init()
{
	insertMethod( RPMethodProc2RDO_MJ::registerMethod() );
#endif
}
```

6.2 Создание главного окна визуального программирования

Главное окно создается в классе `RPProjectMFC` методом `makeFlowChartWnd`.(Страуструп, 2008)

Пример:(Центр разработки на C++)

```
void RPProjectMFC::makeFlowChartWnd( RPObjectFlowChart* flowobj )
{
	BOOL maximized = false;
	studioApp.mainFrame->MDIGetActive( &maximized );
	RPDoc* doc = static_cast<RPDoc*>(model->flowchartDocTemplate-
>OpenDocumentFile( NULL ));
	RPChildFrame* mdi = static_cast<RPChildFrame*>(doc->getView()-
>GetParent());
	mdi->SetIcon( flowobj->getMethod()->getPixmap()->getIcon(), true
);
	if ( maximized ) {
		mdi->ShowWindow( SW_HIDE );
		mdi->MDIRestore();
		mdi->ShowWindow( SW_HIDE );
	}
	doc->getView()->makeFlowChartWnd( flowobj );
	flowobj->setCorrectName( flowobj->getClassInfo()->getLabel() );
	if ( maximized ) {
```

```
            mdi->MDIMaximize();
      }
```

Вызов метода `makeFlowChartWnd` происходит при инициализации модели.(Страуструп, 2008) То есть при открытии уже существующей модели или создании новой модели, наряду с главным окном системы ИМ РДО создается главное окно для визуального программирования:(Центр разработки на C++)

```
if ( modelDocTemplate ) {

            std::vector< rpMethod::RPMethod* >::const_iterator it =
studioApp.getMethodManager().getList().begin();
            while ( it != studioApp.getMethodManager().getList().end()
) {
                  rpMethod::RPMethod* method = *it;
                  if(method-
>getClassName()==_T("RPMethodProc2RDO_MJ")){
                        method->makeFlowChart(rpMethod::project);
                  }
                  it++;
            }
. . . .

}
```

6.3 Создание панели инструментов

Создается панель инструментов в классе `RPCtrlToolbar`:(Центр разработки на C++)

```
RPCtrlToolbar* toolbar = rpMethod::project->createToolBar(_T("РДО-
Процесс") );
btn_generate       = toolbar->insertButton( this, generate_xpm,
_T("Создать модель") );
btn_generate_setup = toolbar->insertButton( this, generate_setup_xpm,
_T("Настройки") );


RPCtrlToolbar* RPProjectMFC::createToolBar(const rp::string& caption )
{
      RPCtrlToolbarMFC* toolbar = new RPCtrlToolbarMFC(
studioApp.mainFrame );
      toolbars.push_back( toolbar );
      studioApp.mainFrame->insertToolBar( &toolbar->toolbar );
      toolbar->setCaption( caption );
      return toolbar;
}
```

6.4 Инициализация генерируемого кода

Происходит в методе generate()

Сначала находятся все блоки которые ест ьв системе, после чего система проверяет наличие каждого из этих блоков в системе. На этом этапе происходит проверка на структурную целостность системы, а именно система проверяет - все ли блоки соединены между собой. В случае ошибки соединения блоков получаем сообщение.

После того как было определено наличие блоков в системе запускается генерация кода каждого блока методом generate():(Страуструп, 2008)

```
all_child.clear();
rpMethod::project->getAllChildByClass( all_child, "RPShape_MJ", true );
  std::list< RPObject* >::iterator shape_it = all_child.begin();
  while ( shape_it != all_child.end() ) {
      dynamic_cast<RPObject_MJ*>(static_cast<RPShape_MJ*>(*shape_it))-
>generate();
      shape_it++;
}
```

Метод generate() каждого блока создает код РДО. Создаваемый код полностью удовлетворяет синтаксису РДО: (Емельянов В.В., Ясиновский С.И.)

```
RDOfiles->pattern <<std::endl
<<std::endl<<"{блок десайд"<<getName().c_str()<<"}"
<<std::endl<<"$Pattern Блок_десайд_"<<getName().c_str()<<" : rule
{срабатывание закона}trace"
<<std::endl<<"$Relevant_resources"
<<std::endl<<"_transact_X : Group_of_transacts_X keep"
<<std::endl<<"$Body"
<<std::endl<<" _transact_X"
<<std::endl<<"Choice from _transact_X.Состояние_транспортировки =
ожидает and"
<<std::endl<<"_transact_X.Место_нахождения_будущее="<<getName().c_str()
<<std::endl<<"first"
<<std::endl<<"Convert_rule"
<<std::endl<<"Место_нахождения set "<<getName().c_str()
<<std::endl<<"                Место_нахождения_будущее    set
ГИСТ_"<<getName().c_str()
<<std::endl<<"$End";
RDOfiles->function<<std::endl<<"{блок десайд" <<getName().c_str()<<"}"
<<std::endl<<"$Sequence ГИСТ_"<<getName().c_str()<<": such_as
Group_of_transacts_X.Место_нахождения"
      <<std::endl<<"$Type = by_hist"
      <<std::endl<<"$Body"
<<std::endl<<id_next.c_str()<<"  "<<ptrue
<<std::endl<<id_next2.c_str()<<"  "<<pfalse
      <<std::endl<<"    $End";
```

7 Апробирование системы

7.1 Модель для тестов

Для апробирования системы была смоделирована простейшая ситуация.

Заготовки подвергаются обработке, после чего происходит операция контроля обработанных деталей. Известно, что вероятность появления брака равна 5%. Успешно прошедшие контроль заготовки отправляются на склад. Бракованные заготовки проверяются на возможность устранения брака. Брак может быть устранен с вероятностью 50%. На обработку поступает партия из 100 заготовок. В случае если брак устраним, заготовки отправляются на исправление брака, а затем на склад. Если брак не устраним, заготовки утилизируются.

7.2 Ожидаемые результаты

100 транзактов в системе:

После первой операции контроля 5 заготовок оказываются бракованными (100*0,95=95) После второй операции контроля на восстановление будет отправлено 2,5 заготовки (5*0,5=2,5). Итого на складе должно оказаться 97,5 заготовок.

Итого:

Утилизировано 100*0,05 *0,5=2,5 заготовок

Отправлено на склад 100*0,95+100*0,05*0,5=197,5 заготовок

200 транзактов в системе:

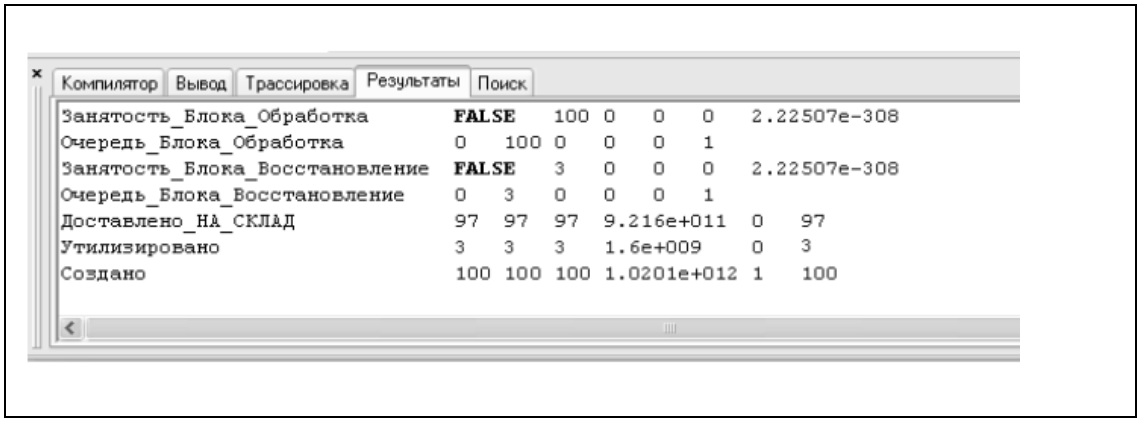
Ожидаемые результаты:

Утилизировано 200*0,05 *0,5=5 заготовок

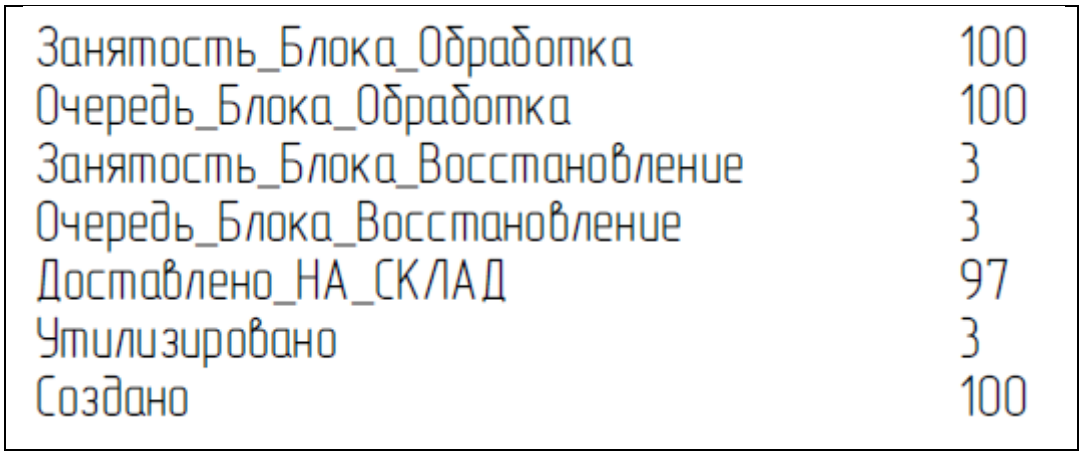
Отправлено на склад 200*0,95+200*0,05*0,5=195 заготовок

7.3 Результаты моделирования

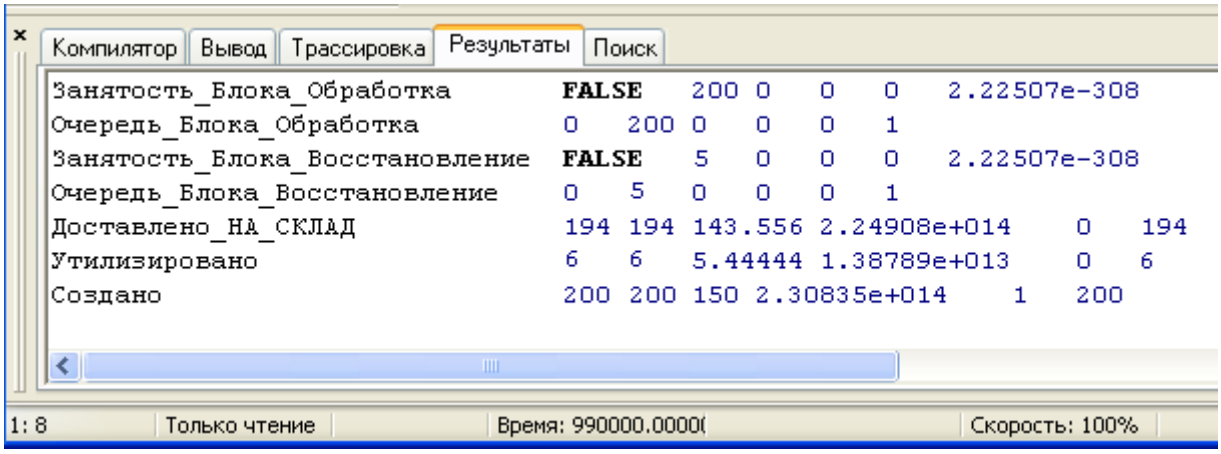
После прогона модели были получены следующие результаты:



И получены следующие результаты



Создается 200 транзактов, процентные соотношения такие же:



Занятость_блока_Обработка	200
Очередь_блока_Обработка	200
Занятость_блока_Восстановление	5
Очередь _блока_Восстановление	5
Доставлено_НА_СКЛАД	194
Утилизировано	6
Создано	200

Как видно из итоговых таблиц и ожидаемых данных, модель работает и соответствует ожидаемым результатам.

8 Заключение

Был проведен анализ системы имитационного моделирования РДО, и на его основе, было принято решение добавить в РДО возможности визуального описания процессов на языке РДО или визуальное программирование на языке РДО.

Функционал системы имитационного моделирования РДО был расширен за счет добавления в систему возможностей визуального программирования. Были намечены этапы разработки системы визуального программирования.

На данной промежуточной стадии, система ИМ РДО в режиме графического создания моделей умеет создавать визуальные модели и генерировать код моделей, который впоследствии может быть обработан в РДО с целью получения результатов моделирования.

Следующим этапом планируется введение «прямой» обработки визуальной модели, без генерации кода модели с использованием обратной связи от ядра РДО для отображения изменений в модели в режиме визуального программирования.

9 Приложения

9.1 Листинг файла заголовков генерации кода модели

```
#ifndef RDO_PROCESS_METHOD_PROC2RDO_H
#define RDO_PROCESS_METHOD_PROC2RDO_H

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "rdo_studio/rdo_process/rp_method/rdoprocess_method.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_creation_RDO_files_MJ.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_shape_MJ.h"
#include "rdo_common/rdotypes.h"

class RPMethodProc2RDO_MJ: public rpMethod::RPMethod, public RPObject_MJ
{
private:
	virtual ~RPMethodProc2RDO_MJ();
	tstring getDirectory() const;
	tstring getName() const;

protected:
	void registerObject();

	CToolBar toolbar;
	void blank_rdo_MJ();

	int btn_generate;
	int btn_generate_setup;
	virtual void buttonCommand( int button_id );
	virtual void buttonUpdate( RPCtrlToolbar::ButtonUpdate& button_update );

public:
	RPMethodProc2RDO_MJ( RPObject* _parent );
	static rpMethod::RPMethod* registerMethod();
	virtual rp::string getVersionDesc() const { return "альфа"; }
	virtual rp::string getDescription() const { return "Переводит квадратики
в паттерны"; }
	virtual rp::string getClassName() const { return "RPMethodProc2RDO_MJ";
}

	virtual RPObjectFlowChart* makeFlowChart( RPObject* parent );

	double generate_time_MJ;
	RPCreationRDOFilesMJ* RDOfiles;
	std::list< CString > list_pattern_names; // MJ 7.04.06 хранятся имена
всех паттернов для записи в файл *.opr generate() заполняет его

	virtual void generate();
};

extern RPMethodProc2RDO_MJ* proc2rdo;

#endif // RDO_PROCESS_METHOD_PROC2RDO_H
```

9.2 Листинг основного файла генерации кода модели

```
#include "stdafx.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_method_proc2rdo_MJ.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_shape_create_MJ.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_shape_decide.h"
```

```
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_shape_resource.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_shape_process_MJ.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_shape_terminate_MJ.h"
#include "rdo_studio/rdo_process/proc2rdo/rdoprocess_generation_type_MJ.h"
#include "rdo_studio/rdo_process/proc2rdo/res/method_big.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/method_small.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/generate.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/generate_setup.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/block_create.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/block_decide.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/block_terminate.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/block_process.xpm"
#include "rdo_studio/rdo_process/proc2rdo/res/block_resource.xpm"
#include "rdo_studio/rdo_process/rp_method/rdoprocess_object_chart.h"
#include "rdo_studio/rdo_process/rp_method/rdoprocess_shape.h"
#include <afxole.h>          // для rdostudioapp
#include <afxpriv.h>    // для rdostudioapp
#include "rdo_studio/rdostudioapp.h"
#include "rdo_repository/rdorepository.h"
#include "rdo_studio/stdafx.h"
#include "rdo_studio/rdostudiomodel.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////
RPMethodProc2RDO_MJ* proc2rdo = NULL;

RPMethodProc2RDO_MJ::RPMethodProc2RDO_MJ( RPObject* _parent ):
      rpMethod::RPMethod( _parent, "РДО-Процесс" ),
      RPObject_MJ( get_this() ),
      btn_generate( -1 ),
      btn_generate_setup( -1 ),
      RDOfiles( NULL ) // инициализация
{
      proc2rdo = this;
      generate_time_MJ = 100;
      RDOfiles = new RPCreationRDOFilesMJ();
      pixmap_big   = rpMethod::project->createBitmap( method_big_xpm );
      pixmap_small = rpMethod::project->createBitmap( method_small_xpm );
}

RPMethodProc2RDO_MJ::~RPMethodProc2RDO_MJ()
{
      delete RDOfiles;
      proc2rdo = NULL;
}

void RPMethodProc2RDO_MJ::registerObject()
{
      // Базовый класс
      rpMethod::factory->insertFactory( new RPObjectClassInfo( "RPObject_MJ",
NULL ) );

      // Потомки RPMethod
      RPObjectClassInfo* class_info = new RPObjectClassInfo(
"RPMethodProc2RDO_MJ", "RPMethod" );
      class_info->multiParent( "RPObject_MJ" );
```

```
	rpMethod::factory->insertFactory( class_info );

	// Потомки RPObjectFlowChart
	class_info = new RPObjectClassInfo( "RPObjectFlowChart_MJ",
"RPObjectFlowChart", RPObjectFlowChart_MJ::newObject, this, _T("РДО-Процесс")
);
	class_info->multiParent( "RPObject_MJ" );
	rpMethod::factory->insertFactory( class_info );

	// Потомки RPShape
	class_info = new RPObjectClassInfo( "RPShape_MJ", "RPShape" );
	class_info->multiParent( "RPObject_MJ" );
	rpMethod::factory->insertFactory( class_info );

	// Потомки RPShape_MJ
	rpMethod::factory->insertFactory( new RPObjectClassInfo(
"RPShapeCreateMJ", "RPShape_MJ", RPShapeCreateMJ::newObject, this,
_T("Генератор"), block_create_xpm, 0 ) );

	rpMethod::factory->insertFactory( new RPObjectClassInfo(
"RPShapeTerminateMJ", "RPShape_MJ", RPShapeTerminateMJ::newObject, this,
_T("Терминатор"), block_terminate_xpm, 1 ) );
	rpMethod::factory->insertFactory( new RPObjectClassInfo(
"RPShapeProcessMJ", "RPShape_MJ", RPShapeProcessMJ::newObject, this,
_T("Процесс"), block_process_xpm, 2 ) );
	rpMethod::factory->insertFactory( new RPObjectClassInfo(
"RPShapeResource_MJ", "RPShape_MJ", RPShapeResource_MJ::newObject, this,
_T("Ресурс"), block_resource_xpm, 3 ) );
	rpMethod::factory->insertFactory( new RPObjectClassInfo(
"RPShapeDecide", "RPShape_MJ", RPShapeDecide::newObject, this, _T("Десайд"),
block_decide_xpm, 4 ) );

	RPCtrlToolbar* toolbar = rpMethod::project->createToolBar( _T("РДО-
Процесс") );
	btn_generate       = toolbar->insertButton( this, generate_xpm,
_T("Создать модель") );
	btn_generate_setup = toolbar->insertButton( this, generate_setup_xpm,
_T("Настройки") );
}

rpMethod::RPMethod* RPMethodProc2RDO_MJ::registerMethod()
{
	new RPMethodProc2RDO_MJ( NULL );
	proc2rdo->registerObject();

	return proc2rdo;
}

RPObjectFlowChart* RPMethodProc2RDO_MJ::makeFlowChart( RPObject* parent )
{
	return new RPObjectFlowChart_MJ( parent );
}

void RPMethodProc2RDO_MJ::buttonCommand( int button_id )
{
	if ( button_id == btn_generate ) {
		generate();
		model->closeModel();
		model->openModel();
	} else if ( button_id == btn_generate_setup ) {
		RP_GENERATION_TYPE_MJ dlg( AfxGetMainWnd() );
		dlg.DoModal();
	}
```

```
}

void RPMethodProc2RDO_MJ::buttonUpdate( RPCtrlToolbar::ButtonUpdate&
button_update )
{
	button_update.enable = true;
}

tstring RPMethodProc2RDO_MJ::getDirectory() const
{
	rdoRepository::RDOThreadRepository::FileInfo data(rdoModelObjects::PAT);
	studioApp.studioGUI->sendMessage(kernel->repository(),
RDOThread::RT_REPOSITORY_MODEL_GET_FILEINFO, &data);

	tstring::size_type pos = data.m_full_name.find(data.m_extention);
	ASSERT(pos != tstring::npos);

	tstring result(data.m_full_name.substr(0, pos));
	return result;
}

tstring RPMethodProc2RDO_MJ::getName() const
{
	rdoRepository::RDOThreadRepository::FileInfo data(rdoModelObjects::PAT);
	studioApp.studioGUI->sendMessage(kernel->repository(),
RDOThread::RT_REPOSITORY_MODEL_GET_FILEINFO, &data);

	tstring result = data.m_name;
	return result;
}

void RPMethodProc2RDO_MJ::generate()
{
	RDOfiles->pattern.open  (rdo::format(_T("%s.pat"),
getDirectory().c_str()).c_str());
	RDOfiles->pattern.is_open();
	RDOfiles->resourse.open  (rdo::format(_T("%s.rss"),
getDirectory().c_str()).c_str());
	RDOfiles->function.open  (rdo::format(_T("%s.fun"),
getDirectory().c_str()).c_str());
	RDOfiles->typeres.open   (rdo::format(_T("%s.rtp"),
getDirectory().c_str()).c_str());
	RDOfiles->operations.open(rdo::format(_T("%s.opr"),
getDirectory().c_str()).c_str());
	RDOfiles->smr.open       (rdo::format(_T("%s.smr"),
getDirectory().c_str()).c_str());
	RDOfiles->statistic.open (rdo::format(_T("%s.pmd"),
getDirectory().c_str()).c_str());

	blank_rdo_MJ();

	// Заполнили ID
	std::list< RPObject* > all_child;
	std::list< rp::string > class_names;
	class_names.push_back( "RPShapeCreateMJ" );
	class_names.push_back( "RPShapeProcessMJ" );
	class_names.push_back( "RPShapeTerminateMJ" );
	class_names.push_back( "RPShapeDecide" );
	rpMethod::project->getAllChildByClasses( all_child, class_names, true );
	std::list< RPObject* >::const_iterator block_it = all_child.begin();
	while ( block_it != all_child.end() ) {
		std::list< RPShape* > list = static_cast<RPShape*>(*block_it)-
>getNextBlock();
```

```
            if ( !list.empty() ){
                  dynamic_cast<RPObject_MJ*>(*block_it)->id_next =
list.front()->getName();
                  rp::string name1 = dynamic_cast<RPObject_MJ*>(*block_it)-
>id_next;
                  if((*block_it)->getClassName()=="RPShapeDecide"){
                        if(list.size()==2){
                              dynamic_cast<RPObject_MJ*>(*block_it)->id_next2
= list.back()->getName();
                              rp::string name2 =
dynamic_cast<RPObject_MJ*>(*block_it)->id_next2;
                        }else{
                                    TRACE1( "%s\n", "ДЕСАЙД ДОЛЖЕН ИМЕТЬ 2
ВЕТВИ!!\nА НЕ ОДНУ!!" );
                        }
                  }
            }else{
                  if((*block_it)->getClassName()!="RPShapeTerminateMJ")
                  {
                        TRACE1( "%s\n", "БЛОК ДОЛЖЕН ИМЕТЬ КОННЕКТОР" );
                  }
            }
            block_it++;
      }

      // Вызвали генерацию у объектов
      all_child.clear();
      rpMethod::project->getAllChildByClass( all_child, "RPShape_MJ", true );
      std::list< RPObject* >::iterator shape_it = all_child.begin();
      while ( shape_it != all_child.end() ) {
            dynamic_cast<RPObject_MJ*>(static_cast<RPShape_MJ*>(*shape_it))-
>generate();
            shape_it++;
      }

      RDOfiles->pattern.close();

      RDOfiles->resourse<<std::endl<<std::endl<<"$End";
      RDOfiles->resourse.close();

      RDOfiles->function.close();
      RDOfiles->typeres.close();

      RDOfiles->smr.close();

      RDOfiles->statistic<<
      std::endl<<"$End";
      RDOfiles->statistic.close();

      //ГЕНЕРИРОВАНИЕ  ОПЕРАЦИЙ *.opr тут т.к. должны быть уже сгенерированны
все файлы

      //list_pattern_names.push_back("{Блок_останова_моделирования_по_времени}
");

                        RDOfiles->operations<<"$Operations";

                  std::list<CString>::iterator it =
list_pattern_names.begin();
                    int amount = 1;

                    while( it != list_pattern_names.end() )
                        {
```

```
                                    TRACE1( "%s\n", (*it) );
                                    RDOfiles->operations<<std::endl
                                    <<std::endl<<"operation_"<<amount<<" : "<<(*it);

                                    it++; amount++;
                                    }


                                    RDOfiles->operations
                                    <<std::endl<<"$End";
list_pattern_names.clear();

      RDOfiles->operations.close();
}

void RPMethodProc2RDO_MJ::blank_rdo_MJ()
{
//ГЕНЕРИРОВАНИЕ ТИПОВ РЕССУРСОВ *.rtp
RDOfiles->typeres
<<std::endl
<<std::endl<<"{ res }"
<<std::endl<<"$Resource_type Resource : permanent {Параметры блока Create}"
<<std::endl<<"$Parameters"
<<std::endl<<"Состояние : (занят,свободен)"
<<std::endl<<"Колличество_транзактов : integer {сколько обслуживается
транзактов в текущий момент}"
<<std::endl<<"Максим_колличество : integer {макс колличество которое может
обслуживаться"
<<std::endl<<"после которого рессурс переходит в состяоние занят}"
<<std::endl<<"$End"
<<std::endl
<<std::endl
<<std::endl
<<std::endl
<<std::endl
<<std::endl<<"{1}"
<<std::endl<<"$Resource_type Creates : permanent {Параметры блока Create}"
<<std::endl<<"$Parameters"
<<std::endl<<"  par_1 : (_true, _false) {false} {связанно с созданием первого
только}"
<<std::endl<<"  par_amount: integer   {1}     {соклько выполнилось раз} "
<<std::endl<<"$End"
<<std::endl
<<std::endl
<<std::endl
<<std::endl<<"$Resource_type Group_of_transacts_X : temporary    {Параметры
создаваемого транзакта}"
<<std::endl<<"$Parameters"
<<std::endl<<"   {параметры транзакта задаются пользователем"
<<std::endl<<"    и генерируются автоматически}"
<<std::endl
<<std::endl<<"{ ПАРАМЕТРЫ МЕСТО НАХОЖДЕНИЯ }"
<<std::endl
<<std::endl<<"    Место_нахождения :(";

      std::list< RPObject* > blocks;
      rpMethod::project->getAllChildByClass( blocks, "RPShape_MJ", true );
      std::list< RPObject* >::const_iterator it = blocks.begin();
      while ( it != blocks.end() ) {
            RDOfiles->typeres << (*it)->getName() << ", ";
            it++;
      }
```

```
RDOfiles->typeres<<")"


<<std::endl<<"    Место_нахождения_будущее : such_as
Group_of_transacts_X.Место_нахождения {ID модуля, куда планируется отправится
на след шаге}"
<<std::endl
<<std::endl
<<std::endl
<<std::endl<<"{ СОСТОЯНИЕ ТРАНЗАКТА}      "
<<std::endl<<"    Состояние_транзакта : integer {}"
<<std::endl
<<std::endl
<<std::endl<<"{PRIMARY KEY}"
<<std::endl<<"    Идентификационный_номер_транзакта : integer"
<<std::endl<<"    Идентификационный_номер_группы_транзактов : such_as
Group_of_transacts_X.Место_нахождения"
<<std::endl<<"{-----------}"
<<std::endl
<<std::endl<<"    Место_в_очереди : integer"
<<std::endl<<"    Состояние_транспортировки : (ожидает, захвачен, в_очереди,
в_очереди_выход)"
<<std::endl
<<std::endl<<"{ ДОПОЛНИТЕЛЬНЫЕ ПАРАМЕТРЫ ОПРЕДЕЛЯЮЩИЕСЯ ПОЛЬЗОВАТЕЛЕМ}"
<<std::endl
<<std::endl<<"{"
<<std::endl<<"... : ..."
<<std::endl<<" }"
<<std::endl<<"$End "
<<std::endl
<<std::endl<<" {2 очередь}"
<<std::endl<<" $Resource_type Queue : permanent"
<<std::endl<<"$Parameters"
<<std::endl<<"  Размер_очереди : integer"
<<std::endl<<"     Выход_очереди: (свободен, занят)"
<<std::endl<<"$End"
<<std::endl
<<std::endl
<<std::endl<<"{3 процесс}"
<<std::endl<<"$Resource_type Block_process : permanent {Параметры блока
Create}"
<<std::endl<<"$Parameters"
<<std::endl<<"  Состояние : (занят,свободен)"
<<std::endl<<"$End"
<<std::endl
<<std::endl<<"{4del}"
<<std::endl<<" $Resource_type Block_Del : permanent"
<<std::endl<<"$Parameters"
<<std::endl<<"  Колличество_удаленных : integer"
<<std::endl
<<std::endl<<"$End";


//ГЕНЕРИРОВАНИЕ  РЕССУРСОВ *.rtp
RDOfiles->resourse<<"$Resources";


//ГЕНЕРИРОВАНИЕ  СТАТИСТИКИ *.pmd
RDOfiles->statistic<<

std::endl<<"$Results";
```

```
//ГЕНЕРИРОВАНИЕ  SMR *.smr
RDOfiles->smr<<
std::endl<<"Model_name = "<< getName()
<<std::endl
<<std::endl<<"Resource_file  = "<< getName()
<<std::endl<<"OprIev_file    = "<< getName() // OPR
<<std::endl<<"Trace_file     = "<< getName()
<<std::endl<<"Statistic_file = "<< getName() //PMD
<<std::endl<<"Results_file   = "<< getName()
//<<std::endl<<"Frame_file     = RDO_PROCESS"
//<<std::endl<<"Frame_number   = 1"
//<<std::endl<<"Show_mode      = Animation"
//<<std::endl<<"Show_rate      = 10000000.0"
<<std::endl
<<std::endl<<"Terminate_if  Time_now >= "<< generate_time_MJ;



// ГЕНЕРИРОВАНИЕ ОПРЕЦИЙ в *.pat генерация блока перемещения
list_pattern_names.push_back("Блок_перемещения");// добавляем имя для *.opr

RDOfiles->pattern
<<"{----------------ГЕНЕРАЦИЯ БЛОКА ПЕРЕМЕЩЕНИЯ-----------------------------
-------------}"
<<std::endl<<"$Pattern Блок_перемещения : rule  {срабатывание закона}trace"
<<std::endl<<"    $Relevant_resources"
<<std::endl<<"          _transact_X : Group_of_transacts_X Keep"
<<std::endl
<<std::endl<<"    $Body"
<<std::endl
<<std::endl<<"    _transact_X"
<<std::endl<<"          Choice from _transact_X.Состояние_транспортировки =
ожидает  and"
<<std::endl<<"
      _transact_X.Место_нахождения_будущее<>_transact_X.Место_нахождения "
<<std::endl<<"    first"
<<std::endl
<<std::endl<<"    Convert_rule"
<<std::endl<<"          Место_нахождения set
_transact_X.Место_нахождения_будущее"
<<std::endl
<<std::endl<<"    $End";


RDOfiles->pattern
<<std::endl<<"{----------------ГЕНЕРАЦИЯ если останавливается по времени-----
-------------------------------------}"
<<std::endl<<"{"
<<std::endl<<"$Pattern  Блок_останова_моделирования_по_времени :
irregular_event trace "
<<std::endl<<"$Relevant_resources"
<<std::endl<<"_parameter : Group_of_transacts_X  NoChange"
<<std::endl<<"{"
<<std::endl<<"$Time ="<< generate_time_MJ
<<std::endl<<"$Body"
<<std::endl<<"_parameter"
<<std::endl<<"$End"
<<std::endl<<"}";
}
```

9.3 Листинг основного файла создания меню блоков

```
#include "rdo_studio/stdafx.h"
#include "rdo_studio/rdostudioworkspace.h"
#include "rdo_studio/rdostudioapp.h"
#include "rdo_studio/rdostudiomainfrm.h"
#include "rdo_studio/rdo_tracer/rdotracer.h"
#include "rdo_studio/rdo_tracer/tracer_ctrls/rdotracertreectrl.h"
#include "rdo_studio/resource.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
// ----------------------------------------------------------------------------
--
// ---------- RDOStudioWorkspace
// ----------------------------------------------------------------------------
--
BEGIN_MESSAGE_MAP(RDOStudioWorkspace, RDOStudioDockWnd)
      //{{AFX_MSG_MAP(RDOStudioWorkspace)
      ON_WM_CREATE()
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

RDOStudioWorkspace::RDOStudioWorkspace():
      RDOStudioDockWnd(),
      frames( NULL )
{
}

RDOStudioWorkspace::~RDOStudioWorkspace()
{
}

int RDOStudioWorkspace::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
      if (RDOStudioDockWnd::OnCreate(lpCreateStruct) == -1)
            return -1;

      tab.Create( NULL, NULL, 0, CRect(0, 0, 100, 100), this, 0 );
      tab.modifyTabStyle( 0, TCS_MULTILINE );

      RDOTracerTreeCtrl* trace = tracer->createTree();
      trace->Create( 0, CRect(0, 0, 0, 0), &tab, 0 );

      frames = new RDOStudioFrameTreeCtrl;
      frames->Create( 0, CRect(0, 0, 0, 0), &tab, 0 );

      pagectrl = new RPPageCtrl;
      pagectrl->Create( "", "", 0 , CRect(0, 0, 0, 0), &tab, 0);

      tab.insertItem( trace, rdo::format( IDS_TAB_TRACER ).c_str() );
      tab.insertItem( frames, rdo::format( IDS_TAB_FRAMES ).c_str() );
      tab.insertItem( pagectrl, rdo::format( IDS_TAB_PAGECTRL ).c_str() );

      studioApp.mainFrame->registerCmdWnd( trace );
      studioApp.mainFrame->registerCmdWnd( frames );
      studioApp.mainFrame->registerCmdWnd( pagectrl );

      return 0;
}
```

10 Список литературы

Емельянов В.В., Ясиновский С.И. (1998). *Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО.* Москва: «АНВИК».

Страуструп, Б. (2008). *Язык программирования C++. Специальное издание.* Санкт-Петербург: Бином.

Центр разработки на C++. (б.д.). Получено из http://msdn.microsoft.com/ru-ru/visualc/default.aspx