Оглавление

Введение

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

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

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

Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную

взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

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

ИМ является эффективным, но и не лишенным недостатков, методом. Трудности использования ИМ, связаны с обеспечением адекватности описания системы, интерпретацией результатов, обеспечением стохастической сходимости процесса моделирования, решением проблемы размерности и т.п. К проблемам применения ИМ следует отнести также и большую трудоемкость данного метода.

Интеллектуальное ИМ, характеризующееся возможностью использования методов искусственного интеллекта и, прежде всего, знаний, при принятии решений в процессе имитации, при управлении имитационным экспериментом, при реализации интерфейса пользователя, создании информационных банков ИМ, снимает часть проблем использования ИМ.

1.Предпроектное исследование.

1.1. Основные подходы к построению ИМ.

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

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

Событие СДС происходит мгновенно, то есть это некоторое действие с нулевой длительностью. Действие, требующее для своей реализации определенного времени, имеет собственное имя и связано с двумя событиями – начала и окончания. Длительность действия зависит от многих причин, среди которых время его начала, используемые ресурсы СДС, характеристики управления, влияние случайных факторов и т.д. В течение времени протекания действия в СДС могут возникнуть события, приводящие к преждевременному завершению действия. Последовательность действий образует процесс в СДС (Рис. 2.).

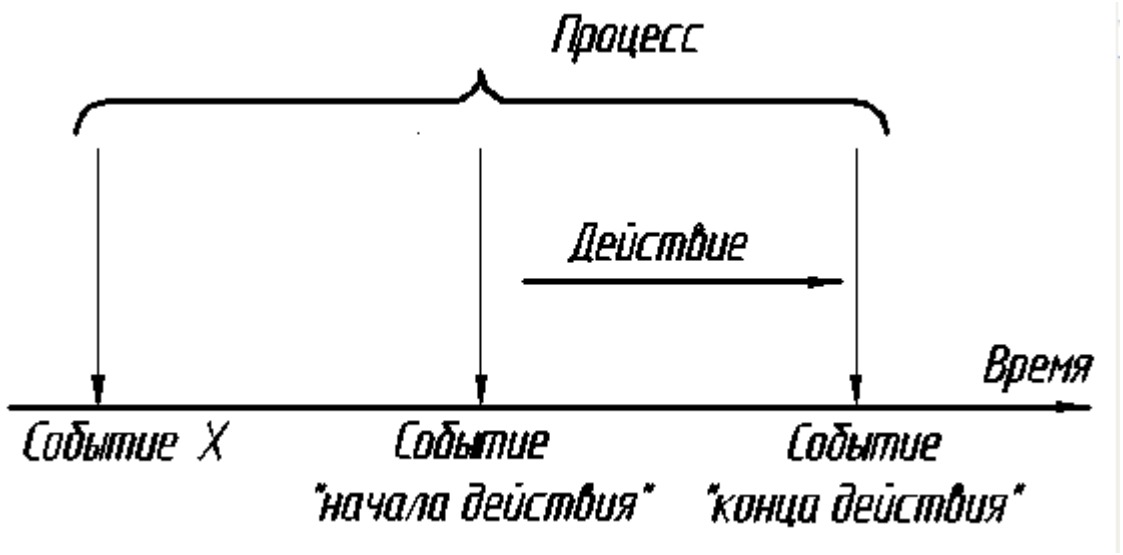


Рис. 1. Взаимосвязь между событиями, действием и процессом.

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

1.2. Процесс имитации в РДО.

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

Событийный подход.

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

времени плюс время, требуемое на обработку детали на станке. В событийных системах модельное время фиксируется только в моменты изменения состояний.

Рис. 2. Выполнение событий в ИМ.

Подход сканирования активностей.

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

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

1.3. Основные положения языка РДО.

В основе системы РДО – «Ресурсы, Действия, Операции» – лежат следующие положения:

- Все элементы сложной дискретной системы (СДС) представлены как ресурсы, описываемые некоторыми параметрами.
- Состояние ресурса определяется вектором значений всех его параметров; состояние СДС – значением всех параметров всех ресурсов.
- Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояния ресурсов; действия ограничены во времени двумя событиями: событиями начала и конца.

- Нерегулярные события описывают изменение состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.
- Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения ресурсов в начале и конце соответствующего действия.

При выполнении работ, связанных с созданием и использованием ИМ в среде РДО, пользователь оперирует следующими основными понятиями:

Модель - совокупность объектов РДО-языка, описывающих какой-то реальный объект, собираемые в процессе имитации показатели, кадры анимации и графические элементы, используемые при анимации, результаты трассировки.

Прогон - это единая неделимая точка имитационного эксперимента. Он характеризуется совокупностью объектов, представляющих собой исходные данные и результаты, полученные при запуске имитатора с этими исходными данными.

Проект - один или более прогонов, объединенных какой-либо общей целью. Например, это может быть совокупность прогонов, которые направлены на исследование одного конкретного объекта или выполнение одного контракта на имитационные исследования по одному или нескольким объектам.

Объект - совокупность информации, предназначенной для определенных целей и имеющая смысл для имитационной программы. Состав объектов обусловлен РДО-методом, определяющим парадигму представления СДС на языке РДО.

Объектами исходных данных являются:

- типы ресурсов (с расширением .rtp);
- ресурсы (с расширением .rss);

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

Объекты, создаваемые РДО-имитатором при выполнении прогона:

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

1.4. Постановка задачи.

Основная идея курсового проекта – добавление в механизм логического вывода РДО приоритетов на уровне логик. Под логикой, в данном контексте, понимается такая сущность системы, которая может объединять в себе простые атомарные активности и имеет свой алгоритм работы. Т.е. логиками являются точки принятия решений всех видов (some, prior, free и search) и процессы (process). Однако процессы далее рассматриваться не будут. Это связано с тем, что из-за специфики их работы, существуют трудности с определением понятия приоритета для процесса в целом. Поэтому для них в рамках данного курсового проекта поддержка приоритета была реализована лишь частично. Подробно этот аспект описан в документации к РДО.

Другими словами, в РДО должна появиться возможность управлять на верхнем уровне иерархии логик направлением продукционного поиска.

Сейчас механизм логического вывода РДО проверяет точки принятия решений в той последовательности, в которой они встречаются в модели.

Такой подход не всегда позволяет написать адекватную модель в системе РДО, да и просто лишает язык РДО некоторой гибкости. Для демонстрации этого недостатка

рассмотрим модель некоторого производственного участка, обрабатывающего детали трех типов. Технологический процесс обработки каждого типа деталей состоит из 3 операций. Участок оборудован тремя станками, каждый из которых выполняет одну операцию технологического процесса изготовления детали, и двумя станками, которые могут выполнять все три операции. Производственный участок управляется MES-системой, которая принимает решения о приоритете выпуска деталей каждого типа. В модели роль MES-системы выполняет пользователь, нажимая во время прогона на соответствующие клавиши, увеличивающие и уменьшающие приоритет деталей определенного типа. Кроме этого различные операции также имеют приоритет, смысл которого сводится к уравниванию количества деталей с различной степенью готовности. Это позволяет делать очереди перед станками равномерными и наиболее синхронно заканчивать обработку деталей определенного типа.

Модель данной СМО на языке РДО представлена в Приложении 2.

Обратим внимание на закладку DPT:

```
$Decision_point обработка_деталей : prior
$Activities
      обработка_детали_1: образец_операции_1 1 CF = приоритет_операции(1, 1)
      обработка_детали_2: образец_операции_2 1 CF = приоритет_операции(1, 2)
      обработка_детали_3: образец_операции_3 1 CF = приоритет_операции(1, 3)
      обработка_детали_4: образец_операции_1 2 CF = приоритет_операции(2, 1)
      обработка_детали_5: образец_операции_2 2 CF = приоритет_операции(2, 2)
      обработка_детали_6: образец_операции_3 2 CF = приоритет_операции(2, 3)
      обработка_детали_7: образец_операции_1 3 CF = приоритет_операции(3, 1)
      обработка_детали_8: образец_операции_2 3 CF = приоритет_операции(3, 2)
      обработка_детали_9: образец_операции_3 3 CF = приоритет_операции(3, 3)
$End
```

Этот код содержит существенные недостатки, главным из которых является отсутствие управления производственным участком MES-системой. Это связано с тем, что текущая версия РДО не позволяет адекватно описать оба приоритета, описанных выше. Пользователь мог бы назначить активностям приоритет, связанный с типом детали, но в таком случае, соответственно, не был бы учтен приоритет операций:

```
$Decision_point обработка_деталей : prior
$Activities
      обработка_детали_1: образец_операции_1 1 CF = приоритет_типа_детали(1)
      обработка_детали_2: образец_операции_2 1 CF = приоритет_типа_детали(1)
      обработка_детали_3: образец_операции_3 1 CF = приоритет_типа_детали(1)
      обработка_детали_4: образец_операции_1 2 CF = приоритет_типа_детали(2)
```

```
    обработка_детали_5: образец_операции_2 2 CF = приоритет_типа_детали(2)
    обработка_детали_6: образец_операции_3 2 CF = приоритет_типа_детали(2)
    обработка_детали_7: образец_операции_1 3 CF = приоритет_типа_детали(3)
    обработка_детали_8: образец_операции_2 3 CF = приоритет_типа_детали(3)
    обработка_детали_9: образец_операции_3 3 CF = приоритет_типа_детали(3)
$End
```

Также к недостаткам этого подхода можно отнести необходимость размещения всех актиностей внутри одной точки принятия решений и повторение приоритетов активностей, из-за невозможности присвоить приоритет сразу нескольким активностям.

Эту проблему можно решить путем назначения приоритета точкам принятия решений целиком.

Таким образом, целью курсового проекта является разработка логического вывода в системе имитационного моделирования РДО, основанного на приоритетных логиках.

2.Концептуальный этап проектирования.

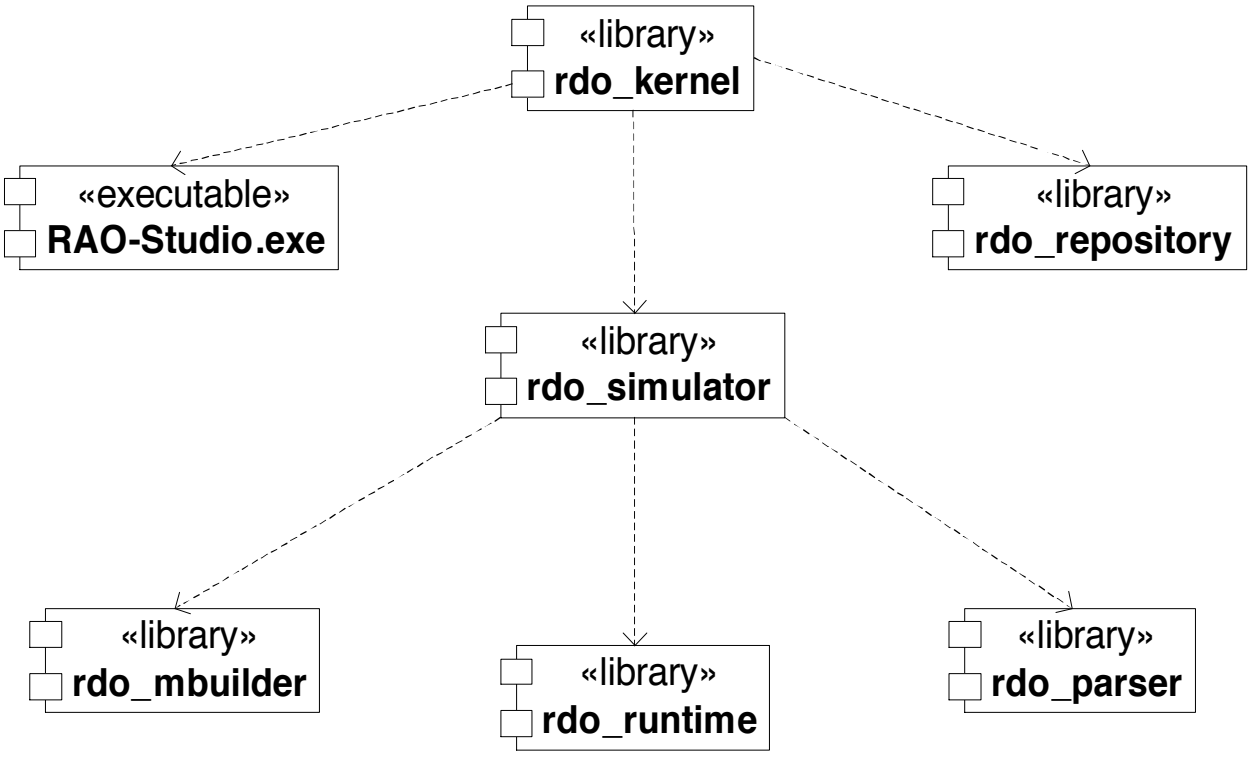
Система имитационного моделирования РДО безусловно является сложной и статически, и динамически. На это указывает сложная иерархическая структура системы со множеством различных связей между компонентами и ее сложное поведение во времени.

Ярко выраженная иерархическая структура и модульность системы определяют направление изучения системы сверху вниз. Т.е. мне необходимо применять принцип декомпозиции нужных модулей до тех пор, пока не будет достигнут уровень абстракции, представление на котором нужных объектов не нуждается в дальнейшей детализации для решения данной задачи.

2.1.Диаграмма компонентов.

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

Рис. 4. Упращенная диаграмма компонентов.



Базовый функционал представленных на диаграмме компонентов:

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

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

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

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

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

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

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

Объекты компонента rdo_runtime инициализуруются при разборе исходного текста модели компонентом rdo_parser. Например, конструктор rdoParse::RDODPTSome::RDODPTSome содержит следующее выражение:

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

которое выделяет место в свободной памяти и инициализирует объект rdoRuntime::RDODPTSome, учавствующий в дальнейшем процессе имитации.

В дальнейшем компоненты rdo_parser и rdo_runtime описываются более детально.

2.2.Структура логического вывода РДО.

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

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

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

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

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

2.3.Техническое задание.

2.3.1.Общие сведения.

В системе РДО разрабатывается новый инструмент обработки приоритетов на уровне логик, позволяющий управлять направлением поиска активности для запуска. Основной разработчик РДО – кафедра РК-9, МГТУ им. Н.Э. Баумана.

2.3.2.Назначение и цели развития системы.

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

2.3.3.Характеристики объекта автоматизации.

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

2.3.4.Требования к системе.

При описании точки принятия решения пользователь может после условия запуска точки указать ее приоритет - слово “$Priority” и арифметическое выражение, которое должно возвращать значение целого или вещественного типа данных и находиться в диапазоне [0; 1].

Т.е. с учетом возможности описания приоритетов точек принятия решений, модель производственного участка должна иметь вид:

```
$Decision_point обработка_деталей_типа_1 : prior
$Priority приоритет_типа_детали(1)
$Activities
      обработка_детали_1: образец_операции_1 1 CF = приоритет_операции(1, 1)
      обработка_детали_2: образец_операции_2 1 CF = приоритет_операции(1, 2)
      обработка_детали_3: образец_операции_3 1 CF = приоритет_операции(1, 3)
$End

$Decision_point обработка_деталей_типа_2 : prior
$Priority приоритет_типа_детали(2)
$Activities
      обработка_детали_4: образец_операции_1 2 CF = приоритет_операции(2, 1)
      обработка_детали_5: образец_операции_2 2 CF = приоритет_операции(2, 2)
      обработка_детали_6: образец_операции_3 2 CF = приоритет_операции(2, 3)
$End
```

```
$Decision_point обработка_деталей_типа_3 : prior
$Priority приоритет_типа_детали(3)
$Activities
      обработка_детали_7: образец_операции_1 3 CF = приоритет_операции(3, 1)
      обработка_детали_8: образец_операции_2 3 CF = приоритет_операции(3, 2)
      обработка_детали_9: образец_операции_3 3 CF = приоритет_операции(3, 3)
$End
```

3.Технический этап проектирования.

3.1.Разработка синтаксиса точки принятия решения.

Объект точек принятия решений имеет следующий формат:

<описание_точки_принятия_решений> { <описание_точки_принятия_решений> }

Описание точки принятия решений имеет следующий формат:

[<заголовок_точки_принятия_решений>] <блок_активностей>

У точки принятия решений заголовок может состоять лишь из описания приоритета точки (или в частном случае отсутствовать вовсе). В таком случае она называется свободной точкой принятия решений или блоком свободных активностей.

Заголовок точки принятия решений имеет следующий формат:

$Decision_point <имя_точки> : <тип_точки> [<признак_трассировки>]

$Condition <условие_активизации_точки>

$Priority <приоритета_точки>

[$Term_condition <терминальное_условие>

$Evaluate_by <оценка_стоимости_оставшегося_пути_на_графе>

$Compare_tops = <признак_сравнения_вершин>]

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

Тип точки может быть одним из следующих: some, prior или search (подробнее см. в справке РДО).

Признак трассировки может быть одним из следующих: no_trace, trace_stat, trace_tops, trace_all (подробнее см. в справке РДО).

Условие активизации точки – это логическое выражение. Алгоритм обработки точки принятия решений активизируется только в том случае, если состояние системы удовлетворяет этому выражению.

Приоритет точки - арифметическое выражение целого или вещественного типа данных, ограниченное диапазоном [0; 1].

Терминальное условие поиска, эвристическая оценочная функция стоимости оставшегося пути до целевой вершины и признак трассировки вершин записывается только для точек типа search (подробнее см. в справке РДО).

Блок активностей имеет следующий формат:

$Activities

<описание_активности> {<описание_активности>}

$End

Описание каждой активности подробно описано в справке РДО.

3.2.Разработка архитектуры компонента rdo_parser.

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

В классе-шаблоне RDOLogicActivity<RTLogic, Activity>, от которого наследуются RDODPTFree, RDODPTPrior, RDODPTSearch, RDODPTSome и RDOOperations, в пространстве имен rdoParse необходимо добавить функцию-член setPrior(), которая будет передавать приоритет в rdoRuntime. Эту функцию синтаксический должен вызывать после того как найдет корректное описание приоритета у точки принятия решений.

В пространстве имен rdoParse приоритет активности должен иметь тип RDOFunArithm.

3.3.Разработка архитектуры компонента rdo_runtime.

В пространстве имен rdoRuntime приоритет должен появиться у классов RDODPTFree, RDODPTPrior, RDODPTSearch, RDODPTSome, RDOOperations и RDOPROCProcess. Все они являются наследниками класса RDOPatternPrior, который инкапсулирует в себе все необходимое для работы с приоритетами, т.е. атрибут m_prior с приоритетом и методы setPrior() и getPrior() для запоминания и извлечения приоритета соответственно.

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

Каждая из описанных логик представляется в компоненте rdoRuntime в виде контейнера с дисциплиной поиска внутри него. Это реализовано с помощью класса-шаблона RDOLogic<Order>, в который в качестве параметра Order передается дисциплина логики. С помощью этого шаблона и трех дисциплин необходимо реализовать три разные логики: RDOLogicDPTPrior - для точек принятия решений типа Prior, RDOLogicMeta - для META-логики и RDOLogicSimple - для простых логик с обычным циклическим перебором активностей. Хотя логики RDOLogicMeta и RDOLogicDPTPrior обе являются приоритетными, их алгоритм работы различается. Алгоритм поиска внутри DPTPrior более сложный и состоит из большего числа итераций из-за возможности использовать в выражении приоритета активности точки типа Prior и временных, и постоянных релевантных ресурсов.

Таким образом необходимо определить три дисциплины RDOOrderDPTPrior, RDOOrderMeta и RDOOrderSimple, которые должны реализовывать метод sort().

Для осуществления сортировки необходимо воспользоваться функтором RDODPTActivityCompare для сравнения приоритетов у активностей.

Также перед выполнением сортировки приоритетных логик необходимо проследить, чтобы значение их приоритета удовлетворяло диапазону [0; 1].

4.Рабочий этап проектирования.

4.1.Синтаксический анализ приоритета логик.

Для реализации в среде имитационного моделирования нового инструмента разработанного на концептуальном и техническом этапах проектирования, в первую очередь необходимо добавить новые термальные символы в лексический анализатор РДО и нетермальные символы в грамматический анализатор.

В лексическом анализаторе (flex) я добавил новый токен RDO_Priority, который может быть записан двумя разными способами (с заглавной и строчной буквы):

```
$Priority           return(RDO_Priority);

$priority           return(RDO_Priority);
```

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

```
%token RDO_ Priority                                        372
```

Далее нужно добавить описание приоритета точки:

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

dpt_some_prior:
        dpt_some_condition
        | dpt_some_condition RDO_Priority fun_arithm
        {
                if (!PARSER->getLastDPTSome()->setPrior( reinterpret_cast<RDOFUNArithm*>($3) ))
                {
                        PARSER->error(@3, _T("Ошибка использования приоритета"));
                }
        }
        | dpt_some_condition RDO_Priority error
        {
                PARSER->error( @1, @2, "Ошибка описания приоритета точки принятия решений" )
        }
        | dpt_some_condition error
        {
                PARSER->error( @1, @2, "Ожидается ключевое слово $Priority" )
        };
```

Из этого кода можно сделать вывод, что точка не обязательно должна иметь приоритет. Это сделано для поддержки уже существующих моделей, написанных на РДО. Если пользователь опустит приоритет, то система самостоятельно назначит данной точке минимальный приоритет, равный нулю.

При нахождении описания приоритета для его сохранения у данной точки вызывается метод setPrior(), в качестве параметра которому передается арифметическое выражение приоритета точки.

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

Подробное описание всех точек принятия решений приведено в Приложении 1.

4.2.Изменения в пространстве имен rdoParse.

Объявление класса RDOLogicActivity

```
template< class RTLogic, class Activity >
class RDOLogicActivity: public RDOParserObject, public RDOParserSrcInfo
{
public:
	RDOLogicActivity( RDOParser* _parser, const RDOParserSrcInfo& _src_info ):
		RDOParserObject( _parser ),
		RDOParserSrcInfo( _src_info )
	{}

	const std::string& name() const { return src_info().src_text(); }

	Activity* addNewActivity( const RDOParserSrcInfo& activity_src_info, const
RDOParserSrcInfo& pattern_src_info )
	{
		Activity* activity = new Activity( this, activity_src_info, pattern_src_info );
		m_activities.push_back( activity );
		return activity;
	}

	Activity* getLastActivity() const
	{
		return !m_activities.empty() ? m_activities.back() : NULL;
	}
	const std::vector< Activity* >& getActivities() const { return m_activities; }

	bool setPrior(RDOFUNArithm* prior)
	{
		LPIPriority priority = m_rt_logic;
		if (priority)
		{
			return priority->setPrior(prior->createCalc());
		}
		return false;
	}

protected:
	LPILogic		m_rt_logic;
```

```
private:
	std::vector< Activity* > m_activities;
};
```

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

В случае успешного приведения данной точки к классу rdoRuntime::RDOActivityPatternPrior вызывается его метод setPrior() с параметром, в качестве которого выступает распознанный синтаксическим анализатором приоритет точки. В результате в классе rdoRuntime::RDOActivityPatternPrior запоминается приоритет точки, о чем синтаксический анализатор будет оповещен значением true, которое данный метод ему вернет. В случае, если точку не удалось привести к классу rdoRuntime::RDOActivityPatternPrior, синтаксическому анализатору будет передано false, и пользователь получит сообщение о невозможности использования приоритета с данной точкой.

4.3.Изменения в пространстве имен rdoRuntime.

Как было выявлено на техническом этапе проектирования RDOLogic является шаблоном-контейнером, параметр которого Order - это дисциплина поиска внутри этого контейнера.

Объявление класса RDOLogic

```
template <class Order>
class RDOLogic: public IBaseOperation, public IBaseOperationContainer, public ILogic,
CAST_TO_UNKNOWN
{
QUERY_INTERFACE_BEGIN
	QUERY_INTERFACE(IBaseOperation)
	QUERY_INTERFACE(IBaseOperationContainer)
	QUERY_INTERFACE(ILogic)
QUERY_INTERFACE_END

public:
	typedef  BaseOperationList                 ChildList;
	typedef  BaseOperationList::iterator       Iterator;
	typedef  BaseOperationList::const_iterator CIterator;

protected:
	RDOLogic();
	virtual ~RDOLogic();

	DECLARE_IBaseOperationContainer;

	PTR(RDOCalc)      m_condition;
	rbool             m_lastCondition;
	ChildList         m_childList;
```

```
        LPIBaseOperation  m_first;

private:
        rbool checkSelfCondition(PTR(RDOSimulator) sim);
        void  start             (PTR(RDOSimulator) sim);
        void  stop              (PTR(RDOSimulator) sim);

        DECLARE_IBaseOperation;
        DECLARE_ILogic;
};
```

Определение функции-члена onCheckCondition класса RDOLogic

```
template <class Order>
inline rbool RDOLogic<Order>::onCheckCondition(PTR(RDOSimulator) sim)
{
        rbool condition = checkSelfCondition(sim);
        if (condition != m_lastCondition)
        {
                m_lastCondition = condition;
                if (condition)
                        start(sim);
                else
                        stop(sim);
        }

        if (!condition)
                return false;

        m_first = Order::sort(sim, m_childList);
        return m_first ? true : false;
}
```

m_first = Order::sort(sim, m_childList); - Функции кэширования берет на себя дисциплина Order.

Объявление класса-дисциплины RDOOrderMeta

```
class RDOOrderMeta
{
public:
        static LPIBaseOperation sort(PTR(RDOSimulator) sim, REF(BaseOperationList) container);
};
```

Объявление класса-дисциплины RDOOrderSimple

```
class RDOOrderSimple
{
public:
        static LPIBaseOperation sort(PTR(RDOSimulator) sim, REF(BaseOperationList) container);
};
```

Объявление класса-дисциплины RDOOrderDPTPrior

```
class RDOOrderDPTPrior
{
public:
        static LPIBaseOperation sort(PTR(RDOSimulator) sim, REF(BaseOperationList) container);
};
```

Все эти классы-дисциплины имеют лишь одну статическую функцию-член sort(), возвращающую активность или логику, которую необходимо выполнить следующей.

Определение функции-члена sort() класса RDOOrderMeta

```
inline LPIBaseOperation RDOOrderMeta::sort(PTR(RDOSimulator) sim, REF(BaseOperationList) container)
{
	if (container.empty())
		return NULL;

	PTR(RDORuntime) runtime = static_cast<PTR(RDORuntime)>(sim);
	STL_FOR_ALL_CONST(BaseOperationList, container, it)
	{
		LPIPriority pattern = *it;
		if (pattern)
		{
			PTR(RDOCalc) prior = pattern->getPrior();
			if (prior)
			{
				RDOValue value = prior->calcValue(runtime);
				if (value < 0 || value > 1)
					runtime->error(rdo::format(_T("Приоритет логики вышел за пределы диапазона [0..1]: %s"), value.getAsString().c_str()), prior);
			}
		}
	}
	std::sort(container.begin(), container.end(), RDODPTActivityCompare(static_cast<PTR(RDORuntime)>(sim)));
	STL_FOR_ALL(BaseOperationList, container, it)
	{
		if ((*it)->onCheckCondition(sim))
		{
			return *it;
		}
	}
	return NULL;
}
```

Сперва в целях оптимизации кода проверяется не пуст ли контейнер логик, и если это так, то вместо указателя на логику возвращается неопределенное значение NULL, сигнализирующее о том, что отсутствует логика для запуска. Далее, если контейнер не является пустым, то каждая логика в нем проверяется на соответствие ее приоритета диапазону значений [0..1]. В случае, если значение приоритета какой-нибудь из них выйдет за пределы указанного диапазона, пользователь получит соответствующее уведомление и на этом прогон закончится. Если значения приоритетов всех логик верны, то они сортируются в порядке уменьшения приоритета. Далее в цикле у каждой логики из отсортированного списка вызывается метода onCheckCondition() до тех пор, пока либо не будет найдена и закэширована логика (и соответственно, активность в ней), которая может быть запущена, либо не будет установлено отсутствие такой логики. В итоге в вызывающую функцию вернется либо указатель на логику. либо NULL.

Определение функции-члена sort() класса RDOOrderSimple

```
inline LPIBaseOperation RDOOrderSimple::sort(PTR(RDOSimulator) sim, REF(BaseOperationList) container)
{
	STL_FOR_ALL(BaseOperationList, container, it)
	{
		if ((*it)->onCheckCondition(sim))
		{
			return *it;
		}
	}
	return NULL;
}
```

Алгоритм работы этой функции достаточно прост. Без каких-либо предварительных действий сразу происходит циклический поиск активности, которая может быть выполнена, путем вызова у всех активностей логики метода onCheckCondition(). Аналогично, если нужная активность найдена, то в вызвавшую функцию вернется указатель на найденную активность, иначе NULL.

Определение функции-члена sort() класса RDOOrderDPTPrior

```
inline LPIBaseOperation RDOOrderDPTPrior::sort(PTR(RDOSimulator) sim, REF(BaseOperationList) container)
{
	BaseOperationList priorContainer;
	STL_FOR_ALL_CONST(BaseOperationList, container, it)
	{
		if (it->query_cast<IBaseOperation>()->onCheckCondition(sim))
		{
			priorContainer.push_back(*it);
		}
	}

	if (priorContainer.empty())
		return NULL;

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

```
        return priorContainer.front();
}
```

В данном алгоритме сначала происходит формирование списка конфликтных активностей точки DPTPrior. Этот список представляет собой контейнер из тех активностей точки, условия запуска которых выполнены. Далее над этим контейнером производятся действия аналогичные действиям с контейнером META-логики.

На основании описанных дисциплин в компоненте rdoRuntime заводятся три класса:

Объвление класса RDOLogicMeta

```
class RDOLogicMeta: public RDOLogic<RDOOrderMeta>
{
protected:
        DEFINE_FACTORY(RDOLogicMeta);

        RDOLogicMeta()
                : RDOLogic<RDOOrderMeta>()
        {}
        virtual ~RDOLogicMeta()
        {}
};
```

Объвление класса RDOLogicSimple

```
class RDOLogicSimple: public RDOLogic<RDOOrderSimple>
{
protected:
        DEFINE_FACTORY(RDOLogicSimple);

        RDOLogicSimple()
                : RDOLogic<RDOOrderSimple>()
        {}
        virtual ~RDOLogicSimple()
        {}
};
```

Объвление класса RDOLogicDPTPrior

```
class RDOLogicDPTPrior: public RDOLogic<RDOOrderDPTPrior>
{
protected:
        DEFINE_FACTORY(RDOLogicDPTPrior);

        RDOLogicDPTPrior()
                : RDOLogic<RDOOrderDPTPrior>()
        {}
        virtual ~RDOLogicDPTPrior()
        {}
};
```

Класс RDOLogicMeta служит для создания в компоненте rdoRuntime META-логики (внутри класса RDOSimulator):

Объявление класса RDOSimulator

```
class RDOSimulator: public RDOSimulatorBase
{
//friend class RDOTrace;

public:
	RDOSimulator();
	virtual ~RDOSimulator();

	void appendLogic(CREF(LPIBaseOperation) logic);

	LPIBaseOperation getMustContinueOpr() const           { return opr_must_continue;  }
	void setMustContinueOpr(CREF(LPIBaseOperation) value) { opr_must_continue = value; }

	virtual void onPutToTreeNode() = 0;

	std::string writeActivitiesStructure( int& counter );

	RDOSimulator* createCopy();
	// Для DPT необходимо перекрыть две нижеследующие функции:
	// 1. Создает клон RDOSimulator с копиями всех ресурсов, но не более
	virtual RDOSimulator* clone()                    = 0;
	// 2. Сравнение двух симуляторов по ресурсам
	virtual bool operator == ( RDOSimulator& other ) = 0;

	ruint getSizeofSim() const
	{
		return m_sizeof_sim;
	}

	LPIBaseOperationContainer m_metaLogic;

protected:
	void appendBaseOperation(CREF(LPIBaseOperation) op)
	{
		ASSERT(op);
		ASSERT(!m_metaLogic->empty());
		LPIBaseOperationContainer logic = m_metaLogic->back();
		ASSERT(logic);
		logic->append(op);
	}

	// Инициализирует нерегулярные события и блоки GENERATE: задает время первого срабатывания
	virtual void preProcess();

	virtual void onResetPokaz()      = 0;
	virtual void onCheckPokaz()      = 0;
	virtual void onAfterCheckPokaz() = 0;

	ruint m_sizeof_sim;

private:
	LPIBaseOperation opr_must_continue;
	virtual bool doOperation();
};
```

Определение конструктора RDOSimulator

```
RDOSimulator::RDOSimulator()
	: RDOSimulatorBase( )
```

```
        , m_sizeof_sim    (0)
{
        m_metaLogic = F(RDOLogicMeta)::create();
}
```

Класс RDOLogicDPTPrior служит для создания точки принятия решений типа Prior:

Объвление класса RDODPTPrior

```
class RDODPTPrior: public RDOLogicDPTPrior, public RDOPatternPrior
{
DEFINE_FACTORY(RDODPTPrior);
QUERY_INTERFACE_BEGIN
QUERY_INTERFACE_PARENT(RDOLogicDPTPrior)
QUERY_INTERFACE_PARENT(RDOPatternPrior)
QUERY_INTERFACE_END

private:
        RDODPTPrior(RDOSimulator* sim);
        virtual ~RDODPTPrior();
};
```

Класс RDOLogicSimpler служит для создания всех остальных логик в РДО:

Объвление класса RDOOperations

```
class RDOOperations: public RDOLogicSimple, public RDOPatternPrior
{
DEFINE_FACTORY(RDOOperations);
QUERY_INTERFACE_BEGIN
QUERY_INTERFACE_PARENT(RDOLogic)
QUERY_INTERFACE_PARENT(RDOPatternPrior)
QUERY_INTERFACE_END

private:
        RDOOperations (RDOSimulator* sim);
        virtual ~RDOOperations();
};
```

Объвление класса RDODPTSome

```
class RDODPTSome: public RDOLogicSimple, public RDOPatternPrior
{
DEFINE_FACTORY(RDODPTSome);
QUERY_INTERFACE_BEGIN
QUERY_INTERFACE_PARENT(RDOLogic)
QUERY_INTERFACE_PARENT(RDOPatternPrior)
QUERY_INTERFACE_END

private:
        RDODPTSome (RDOSimulator* sim);
        virtual ~RDODPTSome();
};
```

Объвление класса RDODPTSearch

```
class RDODPTSearch: public RDOLogicSimple, public IDPTSearchLogic, public RDOPatternPrior
{
QUERY_INTERFACE_BEGIN
```

```
QUERY_INTERFACE_PARENT(RDOLogic)
QUERY_INTERFACE_PARENT(RDOPatternPrior)
QUERY_INTERFACE(IDPTSearchLogic)
QUERY_INTERFACE_END

friend class RDOSimulator;
friend class TreeNode;

protected:
	RDODPTSearch( RDOSimulator* sim );
	virtual ~RDODPTSearch();

	typedef std::list<LPIDPTSearchActivity> ActivityList;
	ActivityList	m_activityList;

	virtual bool	TermCondition( RDOSimulator* sim )                              = 0;
	virtual double	EvaluateBy( RDOSimulator* sim )                                 = 0;
	virtual void	onSearchBegin( RDOSimulator* sim )                              = 0;
	virtual void	onSearchDecisionHeader( RDOSimulator* sim )                     = 0;
	virtual void	onSearchDecision( RDOSimulator* sim, TreeNode* node )           = 0;
	virtual void	onSearchResultSuccess( RDOSimulator* sim, TreeRoot* treeRoot )  = 0;
	virtual void	onSearchResultNotFound( RDOSimulator* sim, TreeRoot* treeRoot ) = 0;
	virtual bool	NeedCompareTops()                                               = 0;
	virtual TreeRoot* createTreeRoot( RDOSimulator* sim )                           = 0;
	virtual BOResult  onContinue( RDOSimulator* sim );

private:
	TreeRoot* treeRoot;
	virtual BOResult onDoOperation( RDOSimulator* sim );

	DECLARE_IDPTSearchLogic;
};
```

Объявление класса RDODPTFree

```
class RDODPTFree: public RDOLogicSimple, public RDOPatternPrior

{
DEFINE_FACTORY(RDODPTFree);
QUERY_INTERFACE_BEGIN
QUERY_INTERFACE_PARENT(RDOLogic)
QUERY_INTERFACE_PARENT(RDOPatternPrior)
QUERY_INTERFACE_END

private:
	RDODPTFree (RDOSimulator* sim);
	virtual ~RDODPTFree();
};
```

Объявление класса RDOPROCProcess

```
class RDOPROCProcess: public RDOLogicSimple, public IPROCProcess, public RDOPatternPrior

{
DEFINE_FACTORY(RDOPROCProcess)
QUERY_INTERFACE_BEGIN
	QUERY_INTERFACE       (IPROCProcess)
	QUERY_INTERFACE_PARENT(RDOLogic    )
	QUERY_INTERFACE_PARENT(RDOPatternPrior)
QUERY_INTERFACE_END
friend class RDOPROCBlock;

protected:
	tstring                    m_name;
	LPIPROCProcess             m_parent;
	std::list<LPIPROCProcess>  m_child;

private:
	RDOPROCProcess(CREF(tstring) _name, PTR(RDOSimulator) sim);
```

```
        DECLARE_IPROCProcess;
};
```

Таким образом, каждая логика имеет в качестве предка класс RDOPatternPrior, что позволяет РДО во время прогона модели изменять очередность проверки ветвей дерева БЗ при поиске активности для запуска.

Заключение

В рамках данного курсового проекта были получены следующие результаты:

1) Проведено предпроектное исследование системы имитационного моделирования РДО и сформулированы предпосылки создания в системе инструмента для управления поиском активности для запуска на уровне точек принятия решений.

2) На этапе концептуального проектирования системы с помощью диаграммы компонентов нотации UML укрупненно показано внутреннее устройство РДО и выделены те компоненты, которые потребуют внесения изменений в ходе этой работы. Разработаны функциональные диаграммы (в нотации IDEF0) выбора нужной логики внутри META-логики и выбора нужной активности внутри точки принятия решений типа Prior.

3) На этапе технического проектирования разработан новый синтаксис точек принятия решений, который представлен на синтаксической диаграмме. С помощью диаграммы классов разработана архитектура новой системы. С помощью блок-схемы разработаны алгоритмы, реализующие в системе РДО механизм логического вывода на основе приоритетных логик.

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

5) Модель, представленная на этапе постановки задачи, была переписана так, чтобы задействовать новый механизм логического вывода РДО. Результаты проведения имитационного исследования позволяют сделать вывод об адекватности новой модели и, соответственно, об ошибках старой.

Поставленная цель курсового проекта достигнута.

Список использованных источников

1. RAO-Studio – Руководство пользователя, 2007 [http://rdo.rk9.bmstu.ru/forum/viewtopic.php?t=900].

2. Справка по языку РДО (в составе программы) [http://rdo.rk9.bmstu.ru/forum/viewforum.php?f=15].

3. Емельянов В.В., Ясиновский С.И. Имитационное моделирование систем: Учеб. пособие. – М.: Изд-во МГТУ им.Н.Э. Баумана, 2009. – 584 с.: ил. (Информатика в техническом университете).

4. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78.

5. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-90. Условные обозначения и правила выполнения.

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

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

Приложение 1. Полный синтаксический анализ точек принятия решений (rdodpt.y).

```
// --------------------------------------------------------------------------------
// ---------- DPT
// --------------------------------------------------------------------------------
dpt_main:
               | dpt_main dpt_search_end
               | dpt_main dpt_some_end
               | dpt_main dpt_prior_end
               | dpt_main dpt_free_end
               | dpt_main dpt_process_end
               | error {
                     PARSER->error( @1, "Ожидается описание точки или свободного блока
активностей" );
                };

// --------------------------------------------------------------------------------
// ---------- DPT Search
// --------------------------------------------------------------------------------
dpt_search_trace:            /* empty */ {
                                                $$ =
rdoRuntime::RDODPTSearchTrace::DPT_no_trace;
                                          }
                                          | RDO_no_trace {
                                                $$ =
rdoRuntime::RDODPTSearchTrace::DPT_no_trace;
                                          }
                                          | RDO_trace {
                                                PARSER->error( @1, "Данный признак
трассировки не используется в точке типа search" );
                                          }
                                          | RDO_trace_stat {
                                                $$ =
rdoRuntime::RDODPTSearchTrace::DPT_trace_stat;
                                          }
                                          | RDO_trace_tops {
                                                $$ =
rdoRuntime::RDODPTSearchTrace::DPT_trace_tops;
                                          }
                                          | RDO_trace_all {
                                                $$ =
rdoRuntime::RDODPTSearchTrace::DPT_trace_all;
                                          };

dpt_search_begin:            RDO_Decision_point RDO_IDENTIF_COLON RDO_search dpt_search_trace {
                                                RDOValue* identificator =
reinterpret_cast<RDOValue*>($2);
                                                $$ = (int)new RDODPTSearch( PARSER,
identificator->src_info(),
*reinterpret_cast<rdoRuntime::RDODPTSearchTrace::DPT_TraceFlag*>(&$4) );
                                          }
                                          | RDO_Decision_point RDO_IDENTIF_COLON error {
                                                PARSER->error( @2, @3, "Ожидается тип
точки" );
                                          }
                                          | RDO_Decision_point RDO_IDENTIF error {
                                                PARSER->error( @2, "Ожидается двоеточие" );
                                          }
                                          | RDO_Decision_point error {
                                                PARSER->error( @1, @2, "После ключевого слова
$Decision_point ожидается имя точки" );
                                          };
```

```
dpt_search_condition: dpt_search_begin RDO_Condition fun_logic {
                                                RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                                dpt->setCondition((RDOFUNLogic *)$3);
                                        }
                                        | dpt_search_begin RDO_Condition RDO_NoCheck {
                                                RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                                dpt->setCondition();
                                        }
                                        | dpt_search_begin RDO_Condition error {
                                                PARSER->error( @2, @3, "После ключевого слова
$Condition ожидается условие начала поиска (начальная вершина)" );
                                        }
                                        | dpt_search_begin error {
                                                PARSER->error( @2, "Ожидается ключевое слово
$Condition" );
                                        };

dpt_search_prior:         dpt_search_condition
                                        | dpt_search_condition RDO_Priority fun_arithm
                                        {
                                                if (!PARSER->getLastDPTSearch()-
>setPrior( reinterpret_cast<RDOFUNArithm*>($3) ))
                                                {
                                                        PARSER->error(@3, _T("Точка принятия
решений пока не может иметь приоритет"));
                                                }
                                        }
                                        | dpt_search_condition RDO_Priority error
                                        {
                                                PARSER->error( @1, @2, "Ошибка описания
приоритета точки принятия решений" )
                                        }
                                        | dpt_search_condition error
                                        {
                                                PARSER->error( @1, @2, "Ожидается ключевое
слово $Priority" )
                                        };

dpt_search_term:              dpt_search_prior RDO_Term_condition fun_logic {
                                                RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                                dpt->setTermCondition((RDOFUNLogic *)$3);
                                        }
                                        | dpt_search_prior RDO_Term_condition RDO_NoCheck {
                                                RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                                dpt->setTermCondition();
                                        }
                                        | dpt_search_prior RDO_Term_condition error {
                                                PARSER->error( @2, @3, "После ключевого слова
$Term_condition ожидается условие остановки поиска (конечная вершина)" );
                                        }
                                        | dpt_search_prior error {
                                                PARSER->error( @2, "Ожидается ключевое слово
$Term_condition" );
                                        };

dpt_search_evaluate:  dpt_search_term RDO_Evaluate_by fun_arithm {
                                                RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                                dpt->setEvaluateBy((RDOFUNArithm *)$3);
```

```
                                        }
                                        | dpt_search_term RDO_Evaluate_by error {
                                              PARSER->error( @2, @3, "После ключевого слова
$Evaluate_by ожидается оценочная функция, например, 0 для поиска в ширину" );
                                        }
                                        | dpt_search_term error {
                                              PARSER->error( @2, "Ожидается ключевое слово
$Evaluate_by" );
                                        };

dp_searcht_compare:          dpt_search_evaluate RDO_Compare_tops '=' RDO_NO {
                                              RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                              dpt->setCompareTops(false);
                                        }
                                        | dpt_search_evaluate RDO_Compare_tops '=' RDO_YES {
                                              RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                              dpt->setCompareTops(true);
                                        }
                                        | dpt_search_evaluate RDO_Compare_tops '=' error {
                                              PARSER->error( @3, @4, "Ожидается режим
запоминания пройденных вершин (YES или NO)" );
                                        }
                                        | dpt_search_evaluate RDO_Compare_tops error {
                                              PARSER->error( @2, @3, "Ожидается знак
равенства" );
                                        }
                                        | dpt_search_evaluate error {
                                              PARSER->error( @2, "Ожидается ключевое слово
$Compare_tops" );
                                        };

dpt_search_descr_param:      /* empty */
                                        | dpt_search_descr_param  '*'               { PARSER-
>getLastDPTSearch()->getLastActivity()->addParam( RDOValue(RDOParserSrcInfo(@2, "*")) ) }
                                        | dpt_search_descr_param  RDO_INT_CONST     { PARSER-
>getLastDPTSearch()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_search_descr_param  RDO_REAL_CONST    { PARSER-
>getLastDPTSearch()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_search_descr_param  RDO_BOOL_CONST    { PARSER-
>getLastDPTSearch()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_search_descr_param  RDO_STRING_CONST  { PARSER-
>getLastDPTSearch()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_search_descr_param  RDO_IDENTIF       { PARSER-
>getLastDPTSearch()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) };

dpt_search_descr_value:      RDO_value_before fun_arithm {
                                              RDODPTSearch* dpt = PARSER-
>getLastDPTSearch();
                                              dpt->getLastActivity()-
>setValue( IDPTSearchActivity::vt_before, reinterpret_cast<RDOFUNArithm*>($2), @1 );
                                        }
                                        | RDO_value_after fun_arithm {
                                              RDODPTSearch* dpt = PARSER-
>getLastDPTSearch();
                                              dpt->getLastActivity()-
>setValue( IDPTSearchActivity::vt_after, reinterpret_cast<RDOFUNArithm*>($2), @1 );
                                        }
                                        | RDO_value_before error {
                                              PARSER->error( @1, @2, "Ошибка в
арифметическом выражении" );
                                        }
                                        | RDO_value_after error {
```

```
                                                PARSER->error( @1, @2, "Ошибка в
арифметическом выражении" );
                                        };

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

dpt_searcht_activity: /* empty */
                                        | dpt_searcht_activity dpt_search_name
dpt_search_descr_param dpt_search_descr_value {
                                                RDODPTSearchActivity* activity =
reinterpret_cast<RDODPTSearchActivity*>($2);
                                                activity->endParam( @3 );
                                        }
                                        | dpt_searcht_activity dpt_search_name
dpt_search_descr_param error {
                                                PARSER->error( @3, @4, "Ожидаются ключевые
слова value before или value after и стоимость применения правила" );
                                        };

dpt_search_header:            dp_searcht_compare RDO_Activities dpt_searcht_activity {
                                        }
                                        | dp_searcht_compare error {
                                                PARSER->error( @1, @2, "После режима
запоминания пройденных вершин ожидается ключевое слово $Activities" );
                                        };

dpt_search_end:                       dpt_search_header RDO_End {
                                                RDODPTSearch* dpt =
reinterpret_cast<RDODPTSearch*>($1);
                                                dpt->end();
                                        }
                                        | dpt_search_header {
                                                PARSER->error( @1, "Ожидается ключевое слово
$End" );
                                        };

// --------------------------------------------------------------------------
// ---------- DPT Some
// --------------------------------------------------------------------------
dpt_some_trace:                       /* empty */ {
                                                $$ = 1;
                                        }
                                        | RDO_no_trace {
                                                $$ = 1;
```

```
                                        }
                                        | RDO_trace {
                                                $$ = 2;
                                        }
                                        | RDO_trace_stat {
                                                PARSER->error( @1, "Данный признак
трассировки не используется в точке типа some" );
                                        }
                                        | RDO_trace_tops {
                                                PARSER->error( @1, "Данный признак
трассировки не используется в точке типа some" );
                                        }
                                        | RDO_trace_all {
                                                PARSER->error( @1, "Данный признак
трассировки не используется в точке типа some" );
                                        };

dpt_some_begin:                     RDO_Decision_point RDO_IDENTIF_COLON RDO_some dpt_some_trace
{
                                                // TODO: а где признак трассировки для some ?
                                                RDOValue* name =
reinterpret_cast<RDOValue*>($2);
                                                $$ = (int)new RDODPTSome( PARSER, name-
>src_info() );
                                        };

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

dpt_some_prior:               dpt_some_condition
                                        | dpt_some_condition RDO_Priority fun_arithm
                                        {
                                                if (!PARSER->getLastDPTSome()-
>setPrior( reinterpret_cast<RDOFUNArithm*>($3) ))
                                                {
                                                        PARSER->error(@3, _T("Точка принятия
решений пока не может иметь приоритет"));
                                                }
                                        }
                                        | dpt_some_condition RDO_Priority error
                                        {
                                                PARSER->error( @1, @2, "Ошибка описания
приоритета точки принятия решений" )
                                        }
                                        | dpt_some_condition error
                                        {
```

```
                                                PARSER->error( @1, @2, "Ожидается ключевое
слово $Priority" )
                                        };

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

dpt_some_descr_param: /* empty */
                                        | dpt_some_descr_param  '*'              { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( RDOValue(RDOParserSrcInfo(@2, "*")) ) }
                                        | dpt_some_descr_param  RDO_INT_CONST    { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_REAL_CONST   { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_BOOL_CONST   { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_STRING_CONST { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                        | dpt_some_descr_param  RDO_IDENTIF      { PARSER-
>getLastDPTSome()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }

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

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

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

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

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

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

dpt_prior_begin:              RDO_Decision_point RDO_IDENTIF_COLON RDO_prior dpt_prior_trace {
                                          RDOValue* name =
reinterpret_cast<RDOValue*>($2);
                                          $$ = (int)new RDODPTPrior( PARSER, name-
>src_info() );
                                    };

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

dpt_prior_prior:              dpt_prior_condition
                                    | dpt_prior_condition RDO_Priority fun_arithm
                                    {
                                          if (!PARSER->getLastDPTPrior()-
>setPrior( reinterpret_cast<RDOFUNArithm*>($3) ))
                                          {
                                                PARSER->error(@3, _T("Точка принятия
решений пока не может иметь приоритет"));
                                          }
                                    }
                                    | dpt_prior_condition RDO_Priority error
                                    {
```

```
                                        PARSER->error( @1, @2, "Ошибка описания
приоритета точки принятия решений" )
                                    }
                                    | dpt_some_condition error
                                    {
                                        PARSER->error( @1, @2, "Ожидается ключевое
слово $Priority" )
                                    };

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

dpt_prior_descr_param:/* empty */
                                    | dpt_prior_descr_param  '*'              { PARSER-
>getLastDPTPrior()->getLastActivity()->addParam( RDOValue(RDOParserSrcInfo(@2, "*")) ) }
                                    | dpt_prior_descr_param  RDO_INT_CONST    { PARSER-
>getLastDPTPrior()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                    | dpt_prior_descr_param  RDO_REAL_CONST   { PARSER-
>getLastDPTPrior()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                    | dpt_prior_descr_param  RDO_BOOL_CONST   { PARSER-
>getLastDPTPrior()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                    | dpt_prior_descr_param  RDO_STRING_CONST { PARSER-
>getLastDPTPrior()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                    | dpt_prior_descr_param  RDO_IDENTIF      { PARSER-
>getLastDPTPrior()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }

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

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

```
dpt_prior_activity:          /* empty */
                                        | dpt_prior_activity dpt_prior_name
dpt_prior_descr_param dpt_prior_activ_prior{
                                                RDODPTPriorActivity* activity =
reinterpret_cast<RDODPTPriorActivity*>($2);
                                                activity->endParam( @3 );
                                        };

dpt_prior_header:            dpt_prior_prior RDO_Activities dpt_prior_activity {
                                        }
                                        | dpt_prior_prior error {
                                                PARSER->error( @1, @2, "Ожидается ключевое
слово $Activities" );
                                        };

dpt_prior_end:               dpt_prior_header RDO_End {
                                                RDODPTPrior* dpt =
reinterpret_cast<RDODPTPrior*>($1);
                                                dpt->end();
                                        }
                                        | dpt_prior_header {
                                                PARSER->error( @1, "Ожидается ключевое слово
$End" );
                                        };

// ----------------------------------------------------------------------------
// ---------- DPT Free
// ----------------------------------------------------------------------------
dpt_free_prior:                         dpt_free_header
                                                | RDO_Priority fun_arithm dpt_free_header
                                                {
                                                        if (!PARSER->getLastDPTFree()-
>setPrior( reinterpret_cast<RDOFUNArithm*>($2) ))
                                                        {
                                                                PARSER->error(@3, _T("Точка
принятия решений пока не может иметь приоритет"));
                                                        }
                                                }
                                                | RDO_Priority error dpt_free_header
                                                {
                                                        PARSER->error( @1, @2, "Ошибка
описания приоритета точки принятия решений" )
                                                }
                                                | error dpt_free_header
                                                {
                                                        PARSER->error( @1, @2, "Ожидается
ключевое слово $Priority" )
                                                };

dpt_free_header:                    RDO_Activities {
                                                        $$ = (int)new RDODPTFree( PARSER,
@1 );
                                                };

dpt_free_activity:                  /* empty */
                                                | dpt_free_activity dpt_free_activity_name
dpt_free_activity_param dpt_free_activity_keys {
                                                };

dpt_free_activity_name:             RDO_IDENTIF_COLON RDO_IDENTIF {
                                                RDODPTFree* dpt     = PARSER-
>getLastDPTFree();
```

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

dpt_free_activity_param:     /* empty */
                                                | dpt_free_activity_param  '*'
{ PARSER->getLastDPTFree()->getLastActivity()->addParam( RDOValue(RDOParserSrcInfo(@2, "*")) ) }
                                                | dpt_free_activity_param  RDO_INT_CONST
{ PARSER->getLastDPTFree()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                                | dpt_free_activity_param  RDO_REAL_CONST
{ PARSER->getLastDPTFree()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                                | dpt_free_activity_param  RDO_BOOL_CONST
{ PARSER->getLastDPTFree()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                                | dpt_free_activity_param  RDO_STRING_CONST
{ PARSER->getLastDPTFree()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }
                                                | dpt_free_activity_param  RDO_IDENTIF
{ PARSER->getLastDPTFree()->getLastActivity()->addParam( *reinterpret_cast<RDOValue*>($2) ) }

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

dpt_free_activity_keys:             /* empty */
                                                | dpt_free_activity_keys RDO_STRING_CONST {
                                                        RDODPTFreeActivity* activity = PARSER-
>getLastDPTFree()->getLastActivity();
                                                        std::string         key      =
reinterpret_cast<RDOValue*>($2)->value().getString();
                                                        activity->addHotKey( key, @2 );
                                                }
                                                | dpt_free_activity_keys '+' RDO_STRING_CONST
{
                                                        RDODPTFreeActivity* activity = PARSER-
>getLastDPTFree()->getLastActivity();
                                                        std::string         key      =
reinterpret_cast<RDOValue*>($3)->value().getString();
                                                        activity->addHotKey( key, @3 );
                                                };

dpt_free_end:                       dpt_free_prior dpt_free_activity RDO_End {
                                                }
                                                | dpt_free_header error {
                                                        PARSER->error( @1, "Ожидается ключевое
слово $End" );
                                                };
```

Приложение 2. Код имитационной модели производственного участка на языке РДО.

Цех.rtp (Типы ресурсов):

```
$Resource_type системы: permanent
$Parameters
	детали_типа_1: integer = 0
	детали_типа_2: integer = 0
	детали_типа_3: integer = 0
$End

$Resource_type приоритеты : permanent
$Parameters
	номер   : integer
	значение: integer = 10
$End

$Resource_type станки : permanent
$Parameters
	номер : integer
	статус: (занят, свободен) = свободен
$End

$Resource_type детали : temporary
$Parameters
	тип     : integer[1..3]
	статус  : (обрабатывается, свободна) = свободна
	операция: integer = 0
$End
```

Цех.rss (Ресурсы):

```
$Resources
	система    : системы    trace * * *
	приоритет_1: приоритеты trace 1 1
	приоритет_2: приоритеты trace 2 2
	приоритет_3: приоритеты trace 3 3
	станок_1   : станки     trace 1 *
	станок_2   : станки     trace 2 *
	станок_3   : станки     trace 3 *
	станок_4   : станки     trace 4 *
	станок_5   : станки     trace 5 *
$End
```

Цех.fun (Константы, последовательности и функции):

```
$Function время_операции : integer[0..10]
$Type = table
$Parameters
	номер_операции: integer[1..3]
	номер_типа_детали: integer[1..3]
	номер_станка: integer[1..5]
$Body
	7	0	0
	8	0	0
	4	0	0

	0	8	0
	0	9	0
	0	9	0

	0	0	8
	0	0	9
```

```
        0       0       7

        7       7       7
        8       8       8
        8       8       7

        7       7       7
        8       8       8
        8       8       7
$End

$Function приоритет_операции : real
$Type = algorithmic
$Parameters
        тип_детали     : integer
        номер_операции: integer
$Body
        result = ( select(детали: детали.тип = тип_детали and детали.операция = (номер_операции -
1)).size() /
                                (       select(детали: детали.тип = тип_детали and детали.операция =
0).size()
                                  + select(детали: детали.тип = тип_детали and детали.операция =
1).size()
                                  + select(детали: детали.тип = тип_детали and детали.операция =
2).size() ))
$End

$Function приоритет_типа_детали : real
$Type = algorithmic
$Parameters
        тип_детали     : integer
$Body
        if тип_детали = 1 result = (приоритет_1.значение / ( приоритет_1.значение +
приоритет_2.значение + приоритет_3.значение ))
        if тип_детали = 2 result = (приоритет_2.значение / ( приоритет_1.значение +
приоритет_2.значение + приоритет_3.значение ))
        if тип_детали = 3 result = (приоритет_3.значение / ( приоритет_1.значение +
приоритет_2.значение + приоритет_3.значение ))
$End

$Function деталь_готова : real
$Type = algorithmic
$Parameters
        контейнер      : integer
        тип_детали     : integer
$Body
        if контейнер =  тип_детали result = 1
        if контейнер <> тип_детали result = 0
$End
```

Цех.pat (Образцы):

```
$Pattern образец_прихода_детали : irregular_event trace
$Parameters
        тип_детали: integer
$Relevant_resources
        _деталь: детали Create
$Time = 3
$Body
_деталь
        Convert_event
        trace
                тип set тип_детали
$End

$Pattern образец_операции_1 : operation trace
$Parameters
        тип_детали: integer
$Relevant_resources
        _деталь: детали Keep Keep
        _станок: станки Keep Keep
```

```
        _система: системы Keep Keep
$Time = время_операции(1, тип_детали, _станок.номер)
$Body
_деталь
        Choice from _деталь.статус = свободна and _деталь.операция = 0 and _деталь.тип =
тип_детали
        Convert_begin
                статус set обрабатывается
        Convert_end
                статус set свободна
                операция set _деталь.операция + 1
_станок
        Choice from _станок.статус = свободен and время_операции(1, тип_детали, _станок.номер) > 0
        Convert_begin
                статус set занят
        Convert_end
                статус set свободен
_система
        Choice from true
        Convert_end
                детали_типа_1 set _система.детали_типа_1 + деталь_готова(1, тип_детали)
                детали_типа_2 set _система.детали_типа_2 + деталь_готова(2, тип_детали)
                детали_типа_3 set _система.детали_типа_3 + деталь_готова(3, тип_детали)
$End

$Pattern образец_операции_2 : operation trace
$Parameters
        тип_детали: integer
$Relevant_resources
        _деталь: детали Keep Keep
        _станок: станки Keep Keep
        _система: системы Keep Keep
$Time = время_операции(2, тип_детали, _станок.номер)
$Body
_деталь
        Choice from _деталь.статус = свободна and _деталь.операция = 1 and _деталь.тип =
тип_детали
        Convert_begin
                статус set обрабатывается
        Convert_end
                статус set свободна
                операция set _деталь.операция + 1
_станок
        Choice from _станок.статус = свободен and время_операции(2, тип_детали, _станок.номер) > 0
        Convert_begin
                статус set занят
        Convert_end
                статус set свободен
_система
        Choice from true
        Convert_end
                детали_типа_1 set _система.детали_типа_1 + деталь_готова(1, тип_детали)
                детали_типа_2 set _система.детали_типа_2 + деталь_готова(2, тип_детали)
                детали_типа_3 set _система.детали_типа_3 + деталь_готова(3, тип_детали)
$End

$Pattern образец_операции_3 : operation trace
$Parameters
        тип_детали: integer
$Relevant_resources
        _деталь : детали  Keep Keep
        _станок : станки  Keep Keep
        _система: системы Keep Keep
$Time = время_операции(3, тип_детали, _станок.номер)
$Body
_деталь
        Choice from _деталь.статус = свободна and _деталь.операция = 2 and _деталь.тип =
тип_детали
        Convert_begin
                статус set обрабатывается
        Convert_end
                статус set свободна
                операция set _деталь.операция + 1
_станок
```

```
        Choice from _станок.статус = свободен and время_операции(3, тип_детали, _станок.номер) > 0
        Convert_begin
                статус set занят
        Convert_end
                статус set свободен
_система
        Choice from true
        Convert_end
                детали_типа_1 set _система.детали_типа_1 + деталь_готова(1, тип_детали)
                детали_типа_2 set _система.детали_типа_2 + деталь_готова(2, тип_детали)
                детали_типа_3 set _система.детали_типа_3 + деталь_готова(3, тип_детали)
$End

$Pattern образец_увеличения_приоритета : keyboard trace
$Parameters
        номер_типа_деталей : integer
$Relevant_resources
        _приоритет: приоритеты NoChange Keep
$Time = 0
$Body
_приоритет
        Choice from _приоритет.номер = номер_типа_деталей
        Convert_end
                значение set _приоритет.значение + 1
$End

$Pattern образец_уменьшения_приоритета : keyboard trace
$Parameters
        номер_типа_деталей : integer
$Relevant_resources
        _приоритет: приоритеты Keep NoChange
$Time = 0
$Body
_приоритет
        Choice from _приоритет.номер = номер_типа_деталей and _приоритет.значение > 1
        Convert_begin
                значение set _приоритет.значение - 1
$End
```

Цех.opr (Операции):

```
$Operations
        приход_детали_1         : образец_прихода_детали 1
        приход_детали_2         : образец_прихода_детали 2
        приход_детали_3         : образец_прихода_детали 3
        увеличение_приоритета_1: образец_увеличения_приоритета 'NUMPAD1' 1
        увеличение_приоритета_2: образец_увеличения_приоритета 'NUMPAD2' 2
        увеличение_приоритета_3: образец_увеличения_приоритета 'NUMPAD3' 3
        уменьшение_приоритета_1: образец_уменьшения_приоритета 'CONTROL' + 'NUMPAD1' 1
        уменьшение_приоритета_2: образец_уменьшения_приоритета 'CONTROL' + 'NUMPAD2' 2
        уменьшение_приоритета_3: образец_уменьшения_приоритета 'CONTROL' + 'NUMPAD3' 3
$End
```

Цех.dpt (Точки принятия решений):

```
$Decision_point обработка_деталей_типа_1 : prior
$Priority приоритет_типа_детали(1)
$Activities
        обработка_детали_1: образец_операции_1 1 CF = приоритет_операции(1, 1)
        обработка_детали_2: образец_операции_2 1 CF = приоритет_операции(1, 2)
        обработка_детали_3: образец_операции_3 1 CF = приоритет_операции(1, 3)
$End

$Decision_point обработка_деталей_типа_2 : prior
$Priority приоритет_типа_детали(2)
$Activities
        обработка_детали_4: образец_операции_1 2 CF = приоритет_операции(2, 1)
        обработка_детали_5: образец_операции_2 2 CF = приоритет_операции(2, 2)
        обработка_детали_6: образец_операции_3 2 CF = приоритет_операции(2, 3)
```

```
$End

$Decision_point обработка_деталей_типа_3 : prior
$Priority приоритет_типа_детали(3)
$Activities
       обработка_детали_7: образец_операции_1 3 CF = приоритет_операции(3, 1)
       обработка_детали_8: образец_операции_2 3 CF = приоритет_операции(3, 2)
       обработка_детали_9: образец_операции_3 3 CF = приоритет_операции(3, 3)
$End
```

Цех.frm (Анимация):

```
$Frame кадр
$Back_picture = <100 100 100> 800 600
Show
       text [10, 10, 225, 20, <255 255 255>, <0 0 0>, 'Приоритет деталей типа 1 равен ' ]
       text [10, 30, 225, 20, <255 255 255>, <0 0 0>, 'Приоритет деталей типа 2 равен ' ]
       text [10, 50, 225, 20, <255 255 255>, <0 0 0>, 'Приоритет деталей типа 3 равен ' ]
       text [235, 10, 30, 20, <255 255 255>, <0 0 0>, =приоритет_1.значение ]
       text [235, 30, 30, 20, <255 255 255>, <0 0 0>, =приоритет_2.значение ]
       text [235, 50, 30, 20, <255 255 255>, <0 0 0>, =приоритет_3.значение ]
$End
```

Цех.smr (Прогон):

```
Model_name     = sample

Resource_file  = sample
OprIev_file    = sample
Statistic_file = sample
Results_file   = sample
Trace_file     = sample
Frame_file     = sample
Show_mode      = Animation
Show_rate      = 10000.0

Terminate_if Time_now >= 60*1
```

Цех.pmd (Показатели):

```
$Results
       загрузка_станка_1: watch_state станок_1.статус = занят
       загрузка_станка_2: watch_state станок_2.статус = занят
       загрузка_станка_3: watch_state станок_3.статус = занят
       загрузка_станка_4: watch_state станок_4.статус = занят
       загрузка_станка_5: watch_state станок_5.статус = занят
       количество_деталей_типа_1: watch_quant детали детали.тип = 1
       количество_деталей_типа_2: watch_quant детали детали.тип = 2
       количество_деталей_типа_3: watch_quant детали детали.тип = 3
       обработано_деталей_типа_1: watch_quant детали детали.тип = 1 and детали.операция = 3
       обработано_деталей_типа_2: watch_quant детали детали.тип = 2 and детали.операция = 3
       обработано_деталей_типа_3: watch_quant детали детали.тип = 3 and детали.операция = 3
$End
```