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.).

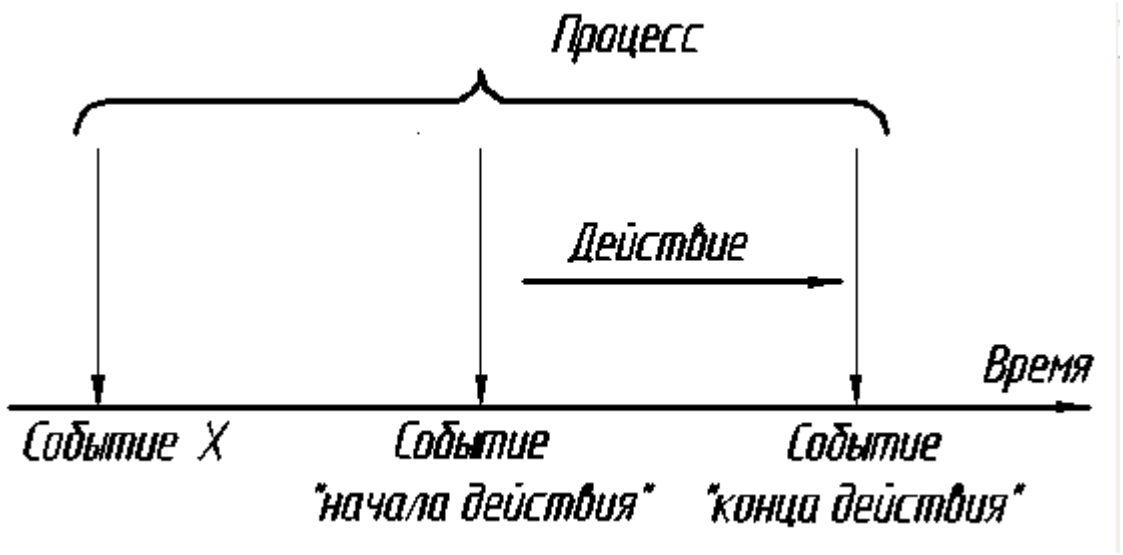

Рис. 1. Взаимосвязь между событиями, действием и процессом.

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.

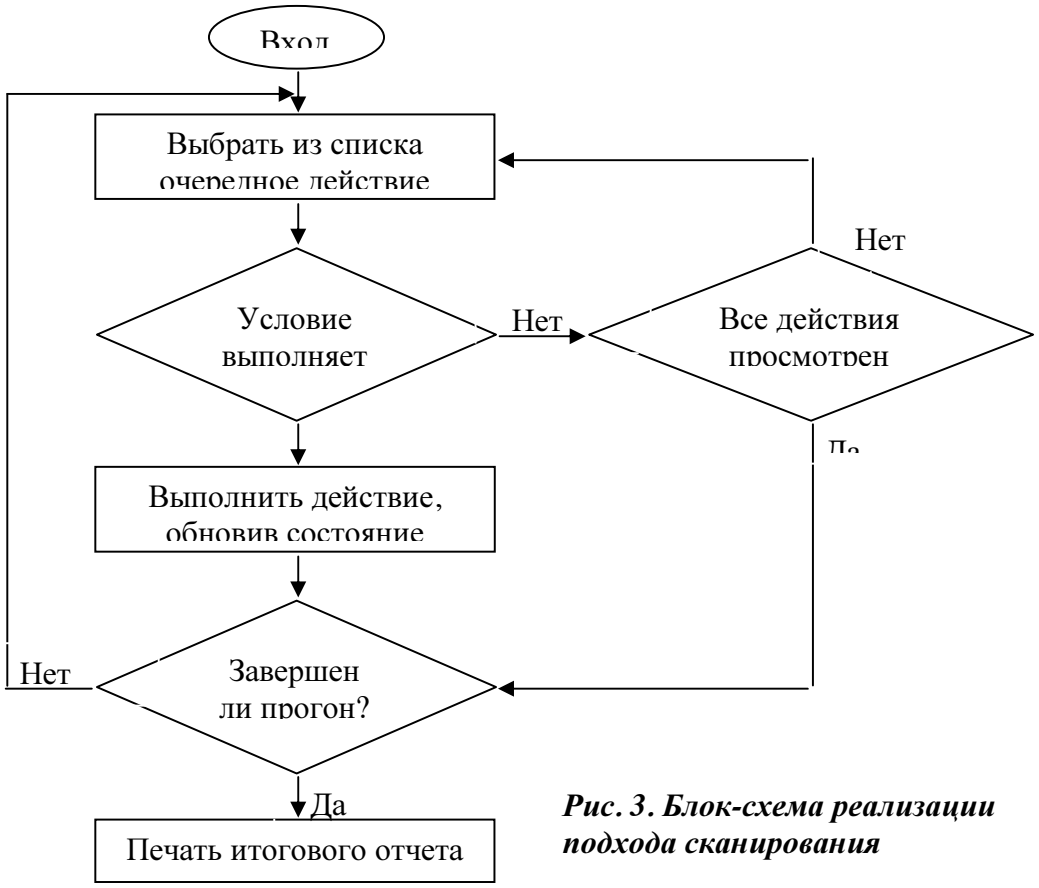

Рис. 3. Блок-схема реализации подхода сканирования

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.

РДО изначально основывался на подходе сканирования активностей и на событийном подходе. Но событийный подход поддерживался лишь частично, потому что у пользователей не было возможности в явном виде планировать события, а были только нерегулярные события, которые при выполнении автоматически планировали себя вновь.

Основная идея курсового проекта – добавление в РДО поддержки полноценного событийного подхода к написанию имитационных моделей.

Другими словами, у пользователей РДО должна появитьяс возможность в явном виде управлять планированием событий с помощью функции Planning().

Это представляет интерес для пользователей, так как событийный подход является самым гибким способом описания моделируемой системы.

Кроме этого, внедрение в РДО событийного подхода позволит более детально подойти к его изучению в рамках курса лекций «Моделирование ТП и ПП», потому что сейчас за неимением инструментальных средств имитация работы модели осуществляется

в ручном режиме, что, конечно же, не так показательно, как применение полноценного компилятора и имитатора моделей.

В качестве примера модели, который система РДО должна поддерживать в явном виде имеет смысл выбрать пример из курса лекций ввиду его универсальности. Эта модель состоит из двух событий:

Событие «Приход клиента»:

- Планирование следующего события "Приход клиента" в момент времени =

Текущее время + Время между приходами;

- Если Парикмахер "Занят":

Число ожидающих = число ожидающих + 1;

Возврат.

- Если Парикмахер "Свободен":

Перевод Парикмахера в состояние "Занят";

Планирование события "Окончание обслуживания" в момент времени =

Текущее время + Длительность обслуживания;

Возврат.

Конец события.

Событие «Окончание обслуживания»:

- Если Число ожидающих больше нуля:

Число ожидающих = число ожидающих - 1;

Планирование события "Окончание обслуживания" в момент времени =

Текущее временя + Длительность обслуживания;

Возврат.

- Если Число ожидающих равно нулю:

Перевод Парикмахера в состояние "Свободен"

Возврат.

Конец.

Для полноценного использования нового подхода в системе РДО соответствующих изменений требует и справочный материал, встроенный в систему. Т.е. пользователи РДО должны иметь возможность прочитать во встроенной справке о всех нововведениях и найти там соответствующие примеры.

2.Концептуальный этап проектирования.

2.1.Диаграмма компонентов.

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.

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

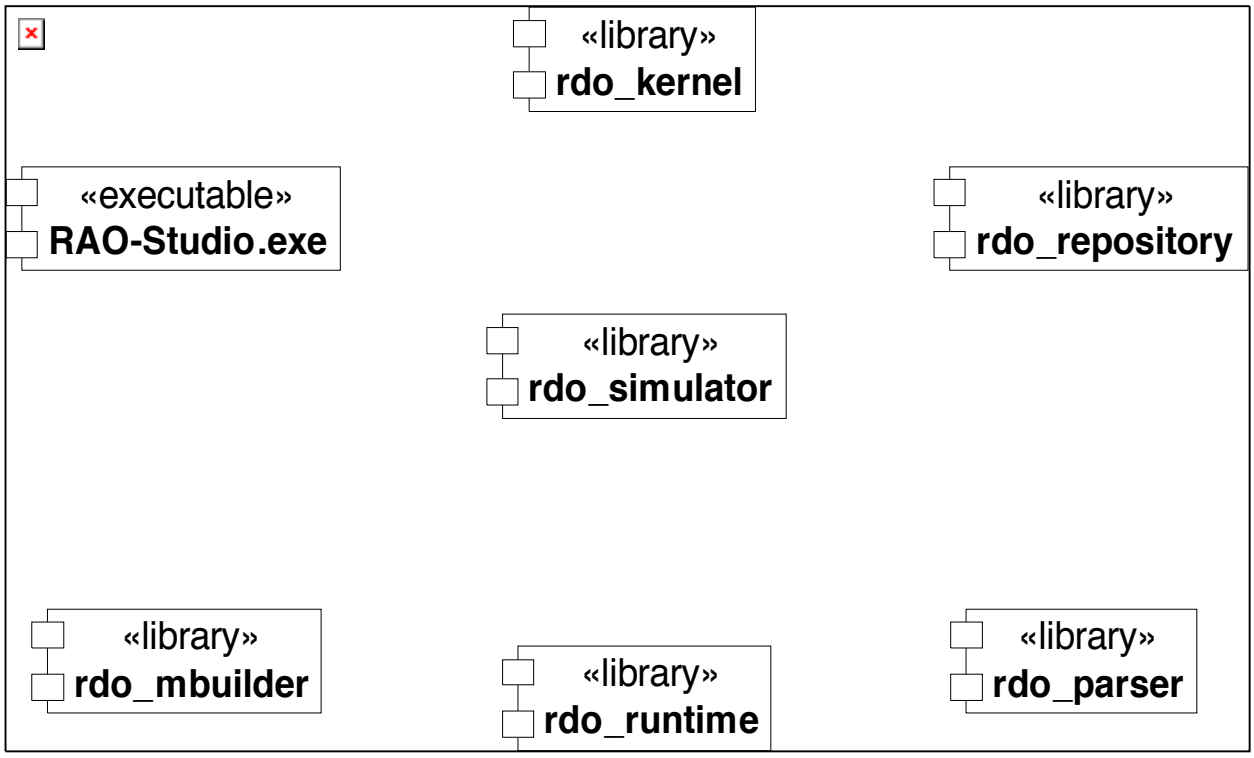

Рис. 4. Упращенная диаграмма компонентов.

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 или активности в моделируемой системе должны выполниться следующими в процессе имитации работы системы.

Подробно алгоритм работы логического вывода представлен на диаграмме с блок-схемой. Здесь же кратко опишем его работу.

Во время имитации работы модели в системе существует одна МЕТА-логика. Она является контейнером для хранения разных логик. Сами логики являются одновременно и контейнерами, в которых хранятся различные атомарные активности (например, нерегулярные события и правила) и атомарной (базовай) операцией. Помимо этого в системе есть ассоциативный массив, ключевым значением, которого является модельное время, а значением — контейнер событий, которые должны произойти в данный момент модельного времени. В этот массив попадают нерегулярные события при своем автопланировании во время его выполнения и также события окончания операций (operation и keyboard), а также их аналоги в процессном подходе — GENERATE и ADVANCE.

Логический вывод работает следующим образом: сначала выполняются все активности БЗ, потом управление передается событийному подходу. Он работает следующим образом: при необходимости происходит продвижение модельного времени вперед (до наступления ближайшего события). Затем найденное событие выполняется. После выполнения первого события событийный подход блокируется и управление снова забирает сканирование активностей. Окончание моделирования происходит одним из двух образов: либо выполняется терминальное условие, его проверка осуществляется в начале каждого цикла управления, либо после того, как сканирование активностей не смогло найти активность, предусловия срабатывания которой выполнены, а список запланированных событий пуст. Второй вариант останвки модели сопровождается сообщением «больше нет событий для моделирования».

2.3.Техническое задание.

2.3.1.Общие сведения.

В систему РДО внедряется поддержка полноценного событийного подхода. 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 явного планирования событий при имитации работы системы.

2.3.3.Характеристики объекта автоматизации.

РДО – язык имитационного моделирования, включающий на данный момент подход сканирования активностей, процессный подход и частично событийный подход.

2.3.4.Требования к системе.

При описании образца активности или события пользователь может в блоке инструкций данного образца писать специальную инструкцию планирования, что приведет к тому, что при срабатывания этого образца в список будущих запланированных событий на нужныу временную метку будет добавлено нужное событие.

Таким образом, рассмотренная раннее модель должна описываться такими событиями:

```
$Pattern Событие_прихода_клиента: event trace
$Relevant_resources
        _Парикмахерская: Парикмахерская Keep
$Body
_Парикмахерская
        Convert_event
                Событие_прихода_клиента.Planning(Time_now + Время_между_приходами(30));
                if (состояние_парикмахера == Занят)
                {
                        число_ожидающих += 1;
                }
                else
                {
                        состояние_парикмахера = Занят;
                        Событие_окончания_обслуживания_клиента.planning(Time_now +
Длительность_обслуживания( 20, 40 ));
                }
$End

$Pattern Событие_окончания_обслуживания_клиента: event trace
$Relevant_resources
        _Парикмахерская: Парикмахерская Keep
$Body
_Парикмахерская
```

```
        Convert_event
                количество_обслуженных += 1;
                if (число_ожидающих>0)
                {
                        число_ожидающих -= 1;
                        Событие_окончания_обслуживания_клиента.planning(Time_now +
Длительность_обслуживания( 20, 40 ));
                }
                else
                {
                        состояние_парикмахера = Свободен;
                }

$End
```

3.Технический этап проектирования.

3.1.Разработка синтаксиса планирования событий.

В качестве синтаксиса для инструкций планирования событий было предложено использовать классический объектно-ориентированный синтаксис:

<имя_события>.Planning (<арифметическое_выражение>)

имя события – это имя одного из событий, описанных в модели, которое должно быть запланировано;

арифметическое выражение – обычное арифметическое выражение, в состав которого могут входить параметры релевантных образцу и глобальных ресурсов.

3.3.Разработка алгоритма планирования событий.

Из-за того, чтобы не накладывать ограничения на порядок следования событий, т.е. чтобы можно было в каком либо образце планировать события, описанные в модели позже, необходимо перед основным разбором текста образцов модели произвести предварительный, целью которого является сбор минимальной информации о событиях, которые могут быть запланированы.

Далее при основном разборе текста паттернов нужно проверять, существует ли событие, которое пользователь решил запланировать, и имеет ли оно нужный тип event. В случае, если нужное событие не найдено необходимо сообщить об этом пользователю.

Далее необходимо запомнить внутри образца в компоненте rdoParse ссылку на соответствующее ему событие из компонента rdoRuntime. Сложность заключается в том, что события в rdoRuntime создаются после создания их в rdoParse. Т.е. на момент создания образцов, в которых используются инструкции планирования еще нет данных о событиях, которые будут планироваться при работе данного образца. Решением этой проблемы может быть позднее связывание.

Таким образом, планирование сбор всей информации, необходимой для планирования событий будет проходить в три этапа.

3.2.Разработка архитектуры компонента rdo_parser.

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

В пространстве имен rdoParse необходимо завести новый класс, который будет хранить имя события, указатель на все объекты-вычислители rdoCalc, в которых планируются данное событие, и указатель на данное событие в rdoRuntime.

3.3.Разработка архитектуры компонента rdo_runtime.

В пространстве имен rdoRuntime необходимо реализовать новые объекты-вычислители, которые будут планировать события, т.е. добавлять в список будущих запланированных событий новые события, которые будут осуществлять условную обработку инструкций, так как без возможности выполнять инструкции по определенному условию событийный подход потеряет изрядную часть своей гибкости. Также необходимо реализовать объект-вычислитель, который будет группировать в одну инструкцию список инструкций, заключенный в фигурные скобки, потому что, практика использования таких конструкций в объектно-ориентированных языках программирования себя хорошо зарекомендовала.

4.Рабочий этап проектирования.

4.1.Синтаксический анализ инструкций планирования событий.

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 новые токены RDO_Event, RDO_Planning, RDO_Else (RDO_If в системе уже присутствует):

```
$Planning           return(RDO_Planning);

$planning           return(RDO_Planning);

$event              return(RDO_Event);

$Else               return(RDO_Else);

$else               return(RDO_Else);
```

Эти токены необходимо также добавить в генератор синтаксического анализатора (bison):

```
%token RDO_Planning                                          377

%token RDO_Event                                             378

%token RDO_Else                                              379
```

Далее нужно реализовать предварительный сбор данных о событиях модели:

```
pat_header

        : RDO_Pattern RDO_IDENTIF_COLON RDO_operation       pat_trace {}

        | RDO_Pattern RDO_IDENTIF_COLON RDO_irregular_event pat_trace {}

        | RDO_Pattern RDO_IDENTIF_COLON RDO_event           pat_trace

        {
```

```
                    LPRDOEvent    pEvent    =    rdo::Factory<RDOEvent>::create(RDOVALUE($2)-
>getIdentificator());

                    ASSERT(pEvent);

                    PARSER->insertEvent(pEvent);

            }

            | RDO_Pattern RDO_IDENTIF_COLON RDO_rule           pat_trace {}

            | RDO_Pattern RDO_IDENTIF_COLON RDO_keyboard       pat_trace {}

            | RDO_Pattern RDO_IDENTIF_COLON error                        {}

            | RDO_Pattern error                                          {}

            ;
```

Этот код создает объекты RDOEvent и сохраняет их в контейнер rdoParse RDOEvents.

Далее необходимо запомнить «привязать» все найденные инструкции планирования к своему событию, т.е. к тому событию, которое они планируют:

```
      planning_statement

            : RDO_IDENTIF '.' RDO_Planning '(' fun_arithm ')' ';'

            {

                    tstring           eventName   = RDOVALUE($1)->getIdentificator();

                    PTR(RDOFUNArithm) pTimeArithm = P_ARITHM($5);

                    LPRDOEvent        pEvent      = PARSER->findEvent(eventName);

                    if (!pEvent)

                    {

                          PARSER->error().error(@1,    rdo::format(_T("Попытка    запланировать
неизвестное событие: %s"), eventName.c_str()));

                    }
```

```
                PTR(rdoRuntime::RDOCalc) pCalcTime = pTimeArithm->createCalc(NULL);
                ASSERT(pCalcTime);

                PTR(rdoRuntime::RDOCalcEventPlan) pCalc = new
rdoRuntime::RDOCalcEventPlan(RUNTIME, pCalcTime);
                ASSERT(pCalc);
                pEvent->attachCalc(pCalc);

                $$ = reinterpret_cast<int>(pCalc);
        }
        | RDO_IDENTIF '.' RDO_Planning '(' fun_arithm ')' error
        {
                PARSER->error().error(@7, _T("Не найден символ окончания инструкции - точка
с запятой"));
        }
        | RDO_IDENTIF '.' RDO_Planning '(' error
        {
                PARSER->error().error(@5, _T("Ошибка в арифметическом выражении"));
        }
        | RDO_IDENTIF '.' RDO_Planning error
        {
                PARSER->error().error(@4, _T("Ожидается открывающая скобка"));
        }
        | RDO_IDENTIF '.' RDO_Planning '(' fun_arithm error
        {
                PARSER->error().error(@6, _T("Ожидается закрывающая скобка"));
```

```
		}

		;
```

Подробное описание разбора всего синтаксиса образцов приведено в Приложении 1.

4.2.Изменения в пространстве имен rdoParse.

Позднее связывание реализуется таким образом:

```
// --------------------------------------------------------------------------------
// ---------- RDOParserPATPost
// --------------------------------------------------------------------------------
void RDOParserPATPost::parse()
{
	//! Позднее связывание для планирования событий
	STL_FOR_ALL_CONST(RDOParser::EventList, m_parser->getEvents(), eventIt)
	{
		LPRDOEvent pEvent = *eventIt;

		CPTR(RDOPATPattern) pPattern = m_parser->findPATPattern(pEvent->name());
		if (!pPattern)
		{
			STL_FOR_ALL_CONST(RDOEvent::CalcList, pEvent->getCalcList(), calcIt)
			{
				m_parser->error().push_only((*calcIt)->src_info(),
rdo::format(_T("Попытка запланировать неизвестное событие: %s"), pEvent->name().c_str()));
			}
			m_parser->error().push_done();
		}
		if (pPattern->getType() != RDOPATPattern::PT_Event)
		{
			STL_FOR_ALL_CONST(RDOEvent::CalcList, pEvent->getCalcList(), calcIt)
			{
				m_parser->error().push_only((*calcIt)->src_info(),
rdo::format(_T("Паттерн %s не является событием: %s"), pEvent->name().c_str()));
			}
			m_parser->error().push_done();
		}

		LPIBaseOperation pRuntimeEvent =
static_cast<PTR(rdoRuntime::RDOPatternEvent)>(pPattern->getPatRuntime())-
>createActivity(m_parser->runtime()->m_metaLogic, m_parser->runtime(), pEvent->name());
		ASSERT(pRuntimeEvent);
		pEvent->setRuntimeEvent(pRuntimeEvent);

		STL_FOR_ALL_CONST(RDOEvent::CalcList, pEvent->getCalcList(), calcIt)
		{
			(*calcIt)->setEvent(pRuntimeEvent);
		}
	}
}
```

Здесь сначала, создается событие в пространстве имен rdoRuntime, а затем ссылка на вновь созданное событие сохраняется во всех объектах-вычислителях, которые хранятся в объекте rdoParse::RDOEvent.

4.3.Изменения в пространстве имен rdoRuntime.

Объявление класса RDOLogic

```
// --------------------------------------------------------------------------
// ---------- RDOCalcEventPlan
// --------------------------------------------------------------------------
class RDOCalcEventPlan: public RDOCalc
{
public:
	RDOCalcEventPlan(PTR(RDORuntimeParent) parent, PTR(RDOCalc) timeCalc);

	void setEvent(CREF(LPIBaseOperation) event);

private:
	LPIBaseOperation m_event;
	PTR(RDOCalc)     m_timeCalc;

	virtual REF(RDOValue) doCalc(PTR(RDORuntime) runtime);
};
```

В атрибуте m_event хранится указатель на событие в имитаторе, а в m_timeCalc – значение времени, в которое должно выполниться событие.

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 планирования событий в явном виде, что позволит РДО поддерживать событийный подход в полной мере.

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

3) На этапе технического проектирования разработан новый синтаксис образцов активностей и событий, который представлен на синтаксической диаграмме. С помощью диаграммы классов разработана архитектура новой системы. С помощью блок-схемы разработаны алгоритмы, реализующие в системе РДО механизм многоэтапного планирования событий.

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) Для демонстрации новых возможностей системы модель, представленная на этапе постановки задачи, была реализована на в системе РДО. Результаты проведения имитационного исследования позволяют сделать вывод об адекватной работе нового событийного подхода.

6) Все внесенные в систему изменения справочной информации по системе РДО, что позволяет пользователям оперативно получать справку по новым функциям системы.

Поставленная цель работы достигнута в полном объеме.

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 (rdopat.y).

```
/*
 * copyright: (c) RDO-Team, 2009
 * filename : rdopat.y
 * author   : Александ Барс, Урусов Андрей, Лущан Дмитрий
 * date     :
 * bref     :
 * indent   : 4T
 */

%{
#define YYPARSE_PARAM lexer
#define YYLEX_PARAM lexer
%}

%pure-parser

%token RDO_Resource_type                257
%token RDO_permanent                    258
%token RDO_Parameters                   259
%token RDO_integer                              260
%token RDO_real                                 261
%token RDO_End                          262
%token RDO_temporary                    263
%token RDO_IDENTIF                              264
%token RDO_INT_CONST                    265
%token RDO_REAL_CONST                   266
%token RDO_such_as                              267
%token RDO_dblpoint                             268
%token RDO_Resources                    269
%token RDO_trace                                270
%token RDO_no_trace                             271
%token RDO_IDENTIF_COLON                272
%token RDO_Constant                             273
%token RDO_Body                                 274
%token RDO_Function                             275
%token RDO_Type                                 276
%token RDO_algorithmic                  277
%token RDO_table                                278
%token RDO_list                                 279
%token RDO_Exist                                281
%token RDO_Not_Exist                    282
%token RDO_For_All                              283
%token RDO_Not_For_All                  284
%token RDO_neq                                  285
%token RDO_leq                                  286
%token RDO_geq                                  287
%token RDO_NoCheck                              288
%token RDO_Calculate_if                         289
%token RDO_or                                   290
%token RDO_and                                  291
%token RDO_Sequence                             292
%token RDO_uniform                              293
%token RDO_exponential                  294
%token RDO_normal                               295
%token RDO_by_hist                              296
%token RDO_enumerative                  297

%token RDO_Pattern                              298
%token RDO_operation                    299
%token RDO_irregular_event              300
%token RDO_rule                                 301
%token RDO_keyboard                             302
%token RDO_Relevant_resources           303
%token RDO_Keep                                 304
%token RDO_Create                               305
%token RDO_Erase                                306
%token RDO_NonExist                             307
%token RDO_IDENTIF_NoChange             308
%token RDO_Time                                 309
```

```
%token RDO_Choice                                310
%token RDO_from                                          311
%token RDO_first                                 312
%token RDO_Convert_begin                313
%token RDO_Convert_end                  314
%token RDO_Convert_rule                          315
%token RDO_Convert_event                316
%token RDO_with_max                              317
%token RDO_with_min                              318
%token RDO_set                                   319
%token RDO_IDENTIF_NoChange_NoChange 320
%token RDO_Operations                   321

%token RDO_Results                               322
%token RDO_watch_par                    323
%token RDO_watch_state                  324
%token RDO_watch_quant                  325
%token RDO_watch_value                  326
%token RDO_get_value                    327

%token RDO_Model_name                   328
%token RDO_Resource_file                329
%token RDO_OprIev_file                  330
%token RDO_Frame_file                   331
%token RDO_Statistic_file               332
%token RDO_Results_file                          333
%token RDO_Trace_file                   334
%token RDO_Show_mode                    335
%token RDO_Frame_number                          336
%token RDO_Show_rate                    337
%token RDO_Run_StartTime                338
%token RDO_Trace_StartTime              339
%token RDO_Trace_EndTime                340
%token RDO_Terminate_if                          341
%token RDO_Break_point                  342
%token RDO_Seed                                          343
%token RDO_NoShow                                342
%token RDO_Monitor                               345
%token RDO_Animation                    346
%token RDO_NoChange                              347

%token RDO_Decision_point               348
%token RDO_search                                349
%token RDO_trace_stat                   350
%token RDO_trace_tops                   351
%token RDO_trace_all                    352
%token RDO_Condition                    353
%token RDO_Term_condition               354
%token RDO_Evaluate_by                  355
%token RDO_Compare_tops                          356
%token RDO_NO                                    357
%token RDO_YES                                   358
%token RDO_Activities                   359
%token RDO_value_before                          360
%token RDO_value_after                  361
%token RDO_some                                          362
%token RDO_Process                               363
%token RDO_SEIZE                                 364
%token RDO_GENERATE                              365
%token RDO_TERMINATE                    366
%token RDO_ADVANCE                               367
%token RDO_RELEASE                               368
%token RDO_if                                    369
%token RDO_result                                370
%token RDO_CF                                    371
%token RDO_Priority                              372
%token RDO_prior                                 373
%token RDO_Parent                                374
%token RDO_PlusEqual                    375
%token RDO_MinusEqual                   376
%token RDO_MultiplyEqual                377
%token RDO_DivideEqual                  378
%token RDO_array                                 379
%token RDO_event                                 380
```

```
%token RDO_Planning                                 381
%token RDO_else                                             382

%token RDO_Frame                                    400
%token RDO_Show_if                                  401
%token RDO_Back_picture                             402
%token RDO_Show                                             403
%token RDO_frm_cell                                 404
%token RDO_text                                             405
%token RDO_bitmap                                   406
%token RDO_s_bmp                                    407
%token RDO_rect                                             408
%token RDO_r_rect                                   409
%token RDO_line                                             410
%token RDO_ellipse                                  411
%token RDO_triang                                   412
%token RDO_active                                   413
%token RDO_ruler                                    414
%token RDO_space                                    415
%token RDO_color_transparent            416
%token RDO_color_last                           417
%token RDO_color_white                          418
%token RDO_color_black                          419
%token RDO_color_red                            420
%token RDO_color_green                          421
%token RDO_color_blue                           422
%token RDO_color_cyan                           423
%token RDO_color_magenta                        424
%token RDO_color_yellow                                 425
%token RDO_color_gray                           426

%token RDO_IDENTIF_RELRES                       427
%token RDO_typedef                                  428
%token RDO_enum                                             429

%token RDO_STRING_CONST                                 430
%token RDO_STRING_CONST_BAD                     431
%token RDO_IDENTIF_BAD                          432
%token RDO_Select                                   433
%token RDO_Size                                             434
%token RDO_Empty                                    435
%token RDO_not                                      436
%token RDO_UMINUS                                   437
%token RDO_string                                   438
%token RDO_bool                                             439
%token RDO_BOOL_CONST                           440
%token RDO_Fuzzy                                    441
%token RDO_Fuzzy_Term                           442
%token RDO_eq                                       443
%token RDO_External_Model                       444
%token RDO_QUEUE                                    445
%token RDO_DEPART                                   446
%token RDO_ASSIGN                                   447


%{
// ==================================================================== PCH
#include "rdo_lib/rdo_parser/pch.h"
// ===================================================================== INCLUDES
// ===================================================================== SYNOPSIS
#include "rdo_lib/rdo_parser/rdoparser.h"
#include "rdo_lib/rdo_parser/rdoparser_lexer.h"
#include "rdo_lib/rdo_parser/rdopat.h"
#include "rdo_lib/rdo_parser/rdortp.h"
#include "rdo_lib/rdo_parser/rdofun.h"
#include "rdo_lib/rdo_parser/rdo_type_range.h"
#include "rdo_lib/rdo_runtime/rdotrace.h"
#include "rdo_lib/rdo_runtime/calc_event_plan.h"
// ===============================================================================

#define PARSER  LEXER->parser()
#define RUNTIME PARSER->runtime()

#define P_RDOVALUE(A) reinterpret_cast<PTR(RDOValue)>(A)
```

```
#define P_ARITHM(A)   reinterpret_cast<PTR(RDOFUNArithm)>(A)
#define P_LOGIC(A)    reinterpret_cast<PTR(RDOFUNLogic)>(A)

#define RDOVALUE(A)   (*P_RDOVALUE(A))
#define ARITHM(A)     (*P_ARITHM(A))
#define LOGIC(A)      (*P_LOGIC(A))

OPEN_RDO_PARSER_NAMESPACE
%}

%left RDO_or
%left RDO_and
%left '+' '-'
%left '*' '/'
%left RDO_not
%left RDO_UMINUS

%%

pat_main
        : /* empty */
        | pat_main pat_pattern
        | error
        {
                PARSER->error().error(@1, _T("Неизвестная ошибка"));
        }
        ;

pat_header
        : RDO_Pattern RDO_IDENTIF_COLON RDO_operation pat_trace
        {
                PTR(RDOValue) name = P_RDOVALUE($2);
                $$ = (int)new RDOPatternOperation(PARSER, name->src_info(), $4 != 0);
        }
        | RDO_Pattern RDO_IDENTIF_COLON RDO_irregular_event pat_trace
        {
                PTR(RDOValue) name = P_RDOVALUE($2);
                $$ = (int)new RDOPatternIrregEvent(PARSER, name->src_info(), $4 != 0);
        }
        | RDO_Pattern RDO_IDENTIF_COLON RDO_event pat_trace
        {
                PTR(RDOValue) name = P_RDOVALUE($2);
                $$ = (int)new RDOPatternEvent(PARSER, name->src_info(), $4 != 0);
        }
        | RDO_Pattern RDO_IDENTIF_COLON RDO_rule pat_trace
        {
                PTR(RDOValue) name = P_RDOVALUE($2);
                $$ = (int)new RDOPatternRule(PARSER, name->src_info(), $4 != 0);
        }
        | RDO_Pattern RDO_IDENTIF_COLON RDO_keyboard pat_trace
        {
                PTR(RDOValue) name = P_RDOVALUE($2);
                $$ = (int)new RDOPatternKeyboard(PARSER, name->src_info(), $4 != 0);
        }
        | RDO_Pattern error
        {
                PARSER->error().error(@2, _T("Ожидается имя образца"));
        }
        | RDO_Pattern RDO_IDENTIF_COLON error
        {
                PARSER->error().error(@2, @3, _T("Ожидается тип образца"));
        }
        ;

pat_trace
        : /* empty */  { $$ = 0; }
        | RDO_trace           { $$ = 1; }
        | RDO_no_trace { $$ = 0; }
        ;

pat_params_begin
        : pat_header RDO_Parameters { $$ = $1; }
        ;
```

```
pat_params
	: pat_params_begin RDO_IDENTIF_COLON param_type
	{
		PTR(RDOPATPattern)       pattern    = reinterpret_cast<PTR(RDOPATPattern)>($1);
		PTR(RDOValue)            param_name = P_RDOVALUE($2);
		LPRDOTypeParam           param_type = PARSER->stack().pop<RDOTypeParam>($3);
		PTR(RDOFUNFunctionParam) param      = new RDOFUNFunctionParam(pattern, param_name-
>src_info(), param_type);
		pattern->add(param);
	}
	| pat_params RDO_IDENTIF_COLON param_type
	{
		PTR(RDOPATPattern)       pattern    = reinterpret_cast<PTR(RDOPATPattern)>($1);
		PTR(RDOValue)            param_name = P_RDOVALUE($2);
		LPRDOTypeParam           param_type = PARSER->stack().pop<RDOTypeParam>($3);
		PTR(RDOFUNFunctionParam) param      = new RDOFUNFunctionParam(pattern, param_name-
>src_info(), param_type);
		pattern->add(param);
	}
	| pat_params_begin error
	{
		if (@1.last_line != @2.last_line)
		{
			PARSER->error().error(@2, _T("Ожидается имя параметра образца"));
		}
		else
		{
			PARSER->error().error(@2, rdo::format(_T("Ожидается имя параметра образца,
найдено: %s"), LEXER->YYText()));
		}
	}
	| pat_params_begin RDO_IDENTIF error
	{
		if (@2.last_line != @3.last_line)
		{
			PARSER->error().error(@2, @3, _T("Ожидается двоеточие"));
		}
		else
		{
			PARSER->error().error(@2, @3, rdo::format(_T("Ожидается двоеточие, найдено:
%s"), LEXER->YYText()));
		}
	}
	| pat_params_begin RDO_IDENTIF_COLON error
	{
		if (@2.last_line != @3.last_line)
		{
			PARSER->error().error(@2, @3, _T("Ожидается тип параметра образца"));
		}
		else
		{
			PARSER->error().error(@2, @3, rdo::format(_T("Ожидается тип параметра
образца, найдено: %s"), LEXER->YYText()));
		}
	}
	| pat_params error
	{
		if (@1.last_line != @2.last_line)
		{
			PARSER->error().error(@2, _T("Ожидается имя параметра образца"));
		}
		else
		{
			PARSER->error().error(@2, rdo::format(_T("Ожидается имя параметра образца,
найдено: %s"), LEXER->YYText()));
		}
	}
	| pat_params RDO_IDENTIF error
	{
		if (@2.last_line != @3.last_line)
		{
			PARSER->error().error(@2, @3, _T("Ожидается двоеточие"));
		}
		else
```

```
		{
			PARSER->error().error(@2, @3, rdo::format(_T("Ожидается двоеточие, найдено: %s"), LEXER->YYText()));
		}
	}
	| pat_params RDO_IDENTIF_COLON error
	{
		if (@2.last_line != @3.last_line)
		{
			PARSER->error().error(@2, @3, _T("Ожидается тип параметра образца"));
		}
		else
		{
			PARSER->error().error(@2, @3, rdo::format(_T("Ожидается тип параметра образца, найдено: %s"), LEXER->YYText()));
		}
	}
	;

pat_params_end
	: pat_params RDO_Relevant_resources
	{
		$$ = $1;
	}
	| pat_header RDO_Relevant_resources
	{
		$$ = $1;
	}
	| pat_header error
	{
		PARSER->error().error(@2, _T("Ожидается ключевое слово $Relevant_resources"));
	}
	;

pat_rel_res
	: pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF pat_conv pat_conv
	{
		// проверено для ie,event,rule,opr,key
		PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		switch (pattern->getType())
		{
			case RDOPATPattern::PT_Operation:
			case RDOPATPattern::PT_Keyboard :
			{
				PTR(RDOValue) rel_name  = P_RDOVALUE($2);
				PTR(RDOValue) type_name = P_RDOVALUE($3);
				static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name->src_info(), type_name->src_info(), (rdoRuntime::RDOResource::ConvertStatus)$4, (rdoRuntime::RDOResource::ConvertStatus)$5, @4, @5);
				break;
			}
			case RDOPATPattern::PT_IE:
			{
				PARSER->error().error(@5, _T("У нерегулярного события нет события конца, а значит и второго статуса конвертора"));
				break;
			}
			case RDOPATPattern::PT_Event:
			{
				PARSER->error().error(@5, _T("У события нет события конца, а значит и второго статуса конвертора"));
				break;
			}
			case RDOPATPattern::PT_Rule:
			{
				PARSER->error().error(@5, _T("У продукционного правила нет события конца, а значит и второго статуса конвертора"));
				break;
			}
		}
	}
	| pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF pat_conv pat_conv
	{
		// проверено для ie,event,rule,opr,key
```

```
		PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		switch (pattern->getType())
		{
			case RDOPATPattern::PT_Operation:
			case RDOPATPattern::PT_Keyboard :
			{
				PTR(RDOValue) rel_name  = P_RDOVALUE($2);
				PTR(RDOValue) type_name = P_RDOVALUE($3);
				static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name->src_info(), type_name->src_info(), (rdoRuntime::RDOResource::ConvertStatus)$4, (rdoRuntime::RDOResource::ConvertStatus)$5, @4, @5);
				break;
			}
			case RDOPATPattern::PT_IE:
			{
				PARSER->error().error(@5, _T("У нерегулярного события нет события конца, а значит и второго статуса конвертора"));
				break;
			}
			case RDOPATPattern::PT_Event:
			{
				PARSER->error().error(@5, _T("У события нет события конца, а значит и второго статуса конвертора"));
				break;
			}
			case RDOPATPattern::PT_Rule:
			{
				PARSER->error().error(@5, _T("У продукционного правила нет события конца, а значит и второго статуса конвертора"));
				break;
			}
		}
	}
	| pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF pat_conv
	{
		// проверено для ie,event,rule,opr,key
		PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		switch (pattern->getType())
		{
			case RDOPATPattern::PT_Operation:
			case RDOPATPattern::PT_Keyboard :
			{
				PARSER->error().error(@4, rdo::format(_T("Помимо статуса конвертора начала (%s), ожидается статус конвертора конца, потому что у операции есть событие конца"), RDOPATPattern::StatusToStr((rdoRuntime::RDOResource::ConvertStatus)$4).c_str()));
				break;
			}
			case RDOPATPattern::PT_IE   :
			case RDOPATPattern::PT_Event:
			case RDOPATPattern::PT_Rule :
			{
				PTR(RDOValue) rel_name  = P_RDOVALUE($2);
				PTR(RDOValue) type_name = P_RDOVALUE($3);
				pattern->addRelRes(rel_name->src_info(), type_name->src_info(), (rdoRuntime::RDOResource::ConvertStatus)$4, @4);
				break;
			}
		}
	}
	| pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF pat_conv
	{
		// проверено для ie,event,rule,opr,key
		PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		switch (pattern->getType())
		{
			case RDOPATPattern::PT_Operation:
			case RDOPATPattern::PT_Keyboard :
			{
				PARSER->error().error(@4, rdo::format(_T("Помимо статуса конвертора начала (%s), ожидается статус конвертора конца, потому что у операции есть событие конца"), RDOPATPattern::StatusToStr((rdoRuntime::RDOResource::ConvertStatus)$4).c_str()));
				break;
			}
			case RDOPATPattern::PT_IE   :
```

```
                        case RDOPATPattern::PT_Event:
                        case RDOPATPattern::PT_Rule :
                        {
                                PTR(RDOValue) rel_name  = P_RDOVALUE($2);
                                PTR(RDOValue) type_name = P_RDOVALUE($3);
                                pattern->addRelRes(rel_name->src_info(), type_name->src_info(),
(rdoRuntime::RDOResource::ConvertStatus)$4, @4);
                                break;
                        }
                }
        }
        | pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF_NoChange pat_conv
        {
                // проверено для ie,event,rule,opr,key
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                switch (pattern->getType())
                {
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PTR(RDOValue) rel_name  = P_RDOVALUE($2);
                                PTR(RDOValue) type_name = P_RDOVALUE($3);
                                YYLTYPE convertor_pos = @3;
                                convertor_pos.first_line   = convertor_pos.last_line;
                                convertor_pos.first_column = convertor_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name-
>src_info(), type_name->src_info(), rdoRuntime::RDOResource::CS_NoChange,
(rdoRuntime::RDOResource::ConvertStatus)$4, convertor_pos, @4);
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
                                PARSER->error().error(@4, _T("У нерегулярного события нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@4, _T("У события нет события конца, а значит
и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Rule:
                        {
                                PARSER->error().error(@4, _T("У продукционного правила нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                }
        }
        | pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF_NoChange pat_conv
        {
                // проверено для ie,event,rule,opr,key
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                switch (pattern->getType())
                {
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PTR(RDOValue) rel_name  = P_RDOVALUE($2);
                                PTR(RDOValue) type_name = P_RDOVALUE($3);
                                YYLTYPE convertor_pos = @3;
                                convertor_pos.first_line   = convertor_pos.last_line;
                                convertor_pos.first_column = convertor_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name-
>src_info(), type_name->src_info(), rdoRuntime::RDOResource::CS_NoChange,
(rdoRuntime::RDOResource::ConvertStatus)$4, convertor_pos, @4);
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
```

```
                                PARSER->error().error(@4, _T("У нерегулярного события нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@4, _T("У события нет события конца, а значит
и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Rule:
                        {
                                PARSER->error().error(@4, _T("У продукционного правила нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                }
        }
        | pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF_NoChange_NoChange
        {
                // проверено для ie,event,rule,opr,key
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                switch (pattern->getType())
                {
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PTR(RDOValue) rel_name  = P_RDOVALUE($2);
                                PTR(RDOValue) type_name = P_RDOVALUE($3);
                                YYLTYPE convertor_begin_pos = @3;
                                tstring str = LEXER->YYText();
                                rdo::toLower(str);
                                tstring::size_type first_nochange = str.find(_T("nochange"));
                                int i = 0;
                                while (true)
                                {
                                        if (str[i] == '\n')
                                        {
                                                convertor_begin_pos.first_line++;
                                                convertor_begin_pos.first_column = 0;
                                        }
                                        else if (str[i] != '\r')
                                        {
                                                convertor_begin_pos.first_column++;
                                        }
                                        i++;
                                        if (i == first_nochange)
                                                break;
                                }
                                convertor_begin_pos.last_line   = convertor_begin_pos.first_line;
                                convertor_begin_pos.last_column = convertor_begin_pos.first_column
+ RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                YYLTYPE convertor_end_pos = @3;
                                convertor_end_pos.first_line   = convertor_end_pos.last_line;
                                convertor_end_pos.first_column = convertor_end_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name-
>src_info(), type_name->src_info(), rdoRuntime::RDOResource::CS_NoChange,
rdoRuntime::RDOResource::CS_NoChange, convertor_begin_pos, convertor_end_pos);
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
                                PARSER->error().error(@3, _T("У нерегулярного события нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@3, _T("У события нет события конца, а значит
и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Rule:
```

```
                        {
                                PARSER->error().error(@3, _T("У продукционного правила нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                }
        }
        | pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF_NoChange_NoChange
        {
                // проверено для ie,event,rule,opr,key
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                switch (pattern->getType())
                {
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PTR(RDOValue) rel_name  = P_RDOVALUE($2);
                                PTR(RDOValue) type_name = P_RDOVALUE($3);
                                YYLTYPE convertor_begin_pos = @3;
                                tstring str = LEXER->YYText();
                                rdo::toLower(str);
                                tstring::size_type first_nochange = str.find(_T("nochange"));
                                int i = 0;
                                while (true)
                                {
                                        if (str[i] == '\n')
                                        {
                                                convertor_begin_pos.first_line++;
                                                convertor_begin_pos.first_column = 0;
                                        }
                                        else if (str[i] != '\r')
                                        {
                                                convertor_begin_pos.first_column++;
                                        }
                                        i++;
                                        if (i == first_nochange)
                                                break;
                                }
                                convertor_begin_pos.last_line   = convertor_begin_pos.first_line;
                                convertor_begin_pos.last_column = convertor_begin_pos.first_column
+ RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                YYLTYPE convertor_end_pos = @3;
                                convertor_end_pos.first_line   = convertor_end_pos.last_line;
                                convertor_end_pos.first_column = convertor_end_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name-
>src_info(), type_name->src_info(), rdoRuntime::RDOResource::CS_NoChange,
rdoRuntime::RDOResource::CS_NoChange, convertor_begin_pos, convertor_end_pos);
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
                                PARSER->error().error(@3, _T("У нерегулярного события нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@3, _T("У события нет события конца, а значит
и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Rule:
                        {
                                PARSER->error().error(@3, _T("У продукционного правила нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                }
        }
        | pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF_NoChange
        {
                // проверено для ie,rule,opr,key
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
```

```
		switch (pattern->getType())
		{
			case RDOPATPattern::PT_Operation:
			case RDOPATPattern::PT_Keyboard :
			{
				PARSER->error().error(@3, rdo::format(_T("Помимо статуса конвертора
начала (%s), ожидается статус конвертора конца, потому что у операции есть событие конца"),
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).c_str()));
				break;
			}
			case RDOPATPattern::PT_IE   :
			case RDOPATPattern::PT_Event:
			case RDOPATPattern::PT_Rule :
			{
				PTR(RDOValue) rel_name  = P_RDOVALUE($2);
				PTR(RDOValue) type_name = P_RDOVALUE($3);
				YYLTYPE convertor_pos = @3;
				convertor_pos.first_line   = convertor_pos.last_line;
				convertor_pos.first_column = convertor_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
				pattern->addRelRes(rel_name->src_info(), type_name->src_info(),
rdoRuntime::RDOResource::CS_NoChange, convertor_pos);
				break;
			}
		}
	}
	| pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF_NoChange
	{
		// проверено для ie,event,rule,opr,key
		PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		switch (pattern->getType())
		{
			case RDOPATPattern::PT_Operation:
			case RDOPATPattern::PT_Keyboard :
			{
				PARSER->error().error(@3, rdo::format(_T("Помимо статуса конвертора
начала (%s), ожидается статус конвертора конца, потому что у операции есть событие конца"),
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).c_str()));
				break;
			}
			case RDOPATPattern::PT_IE   :
			case RDOPATPattern::PT_Event:
			case RDOPATPattern::PT_Rule :
			{
				PTR(RDOValue) rel_name  = P_RDOVALUE($2);
				PTR(RDOValue) type_name = P_RDOVALUE($3);
				YYLTYPE convertor_pos = @3;
				convertor_pos.first_line   = convertor_pos.last_line;
				convertor_pos.first_column = convertor_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
				pattern->addRelRes(rel_name->src_info(), type_name->src_info(),
rdoRuntime::RDOResource::CS_NoChange, convertor_pos);
				break;
			}
		}
	}
	| pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF RDO_IDENTIF_NoChange
	{
		// проверено для ie,rule,opr,key
		PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		switch (pattern->getType())
		{
			case RDOPATPattern::PT_Operation:
			case RDOPATPattern::PT_Keyboard :
			{
				PTR(RDOValue) rel_name  = P_RDOVALUE($2);
				PTR(RDOValue) type_name = P_RDOVALUE($3);
				tstring convert_begin = RDOVALUE($4)->getIdentificator();
				YYLTYPE convertor_begin_pos = @4;
				convertor_begin_pos.last_line   = convertor_begin_pos.first_line;
				convertor_begin_pos.last_column = convertor_begin_pos.first_column
+ convert_begin.length();
				YYLTYPE convertor_end_pos = @4;
				convertor_end_pos.first_line   = convertor_end_pos.last_line;
```

```
                                convertor_end_pos.first_column = convertor_end_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name-
>src_info(), type_name->src_info(), pattern->StrToStatus(convert_begin, convertor_begin_pos),
rdoRuntime::RDOResource::CS_NoChange, convertor_begin_pos, convertor_end_pos);
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
                                PARSER->error().error(@4, _T("У нерегулярного события нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@4, _T("У события нет события конца, а значит
и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Rule:
                        {
                                PARSER->error().error(@4, _T("У продукционного правила нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                }
        }
        | pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF RDO_IDENTIF_NoChange
        {
                // проверено для ie,event,rule,opr,key
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                switch (pattern->getType())
                {
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PTR(RDOValue) rel_name  = P_RDOVALUE($2);
                                PTR(RDOValue) type_name = P_RDOVALUE($3);
                                tstring convert_begin = RDOVALUE($4)->getIdentificator();
                                YYLTYPE convertor_begin_pos = @4;
                                convertor_begin_pos.last_line   = convertor_begin_pos.first_line;
                                convertor_begin_pos.last_column = convertor_begin_pos.first_column
+ convert_begin.length();
                                YYLTYPE convertor_end_pos = @4;
                                convertor_end_pos.first_line   = convertor_end_pos.last_line;
                                convertor_end_pos.first_column = convertor_end_pos.last_column -
RDOPATPattern::StatusToStr(rdoRuntime::RDOResource::CS_NoChange).length();
                                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelRes(rel_name-
>src_info(), type_name->src_info(), pattern->StrToStatus(convert_begin, convertor_begin_pos),
rdoRuntime::RDOResource::CS_NoChange, convertor_begin_pos, convertor_end_pos);
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
                                PARSER->error().error(@4, _T("У нерегулярного события нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@4, _T("У события нет события конца, а значит
и второго статуса конвертора"));
                                break;
                        }
                        case RDOPATPattern::PT_Rule:
                        {
                                PARSER->error().error(@4, _T("У продукционного правила нет события
конца, а значит и второго статуса конвертора"));
                                break;
                        }
                }
        }
        | pat_params_end error
        {
```

```
			PARSER->error().error(@2, _T("Ошибка в описании релевантных ресурсов"));
		}
		| pat_rel_res error
		{
			PARSER->error().error(@2, _T("Ошибка в описании релевантных ресурсов"));
		}
		| pat_params_end RDO_IDENTIF_COLON error
		{
			PARSER->error().error(@2, @3, _T("Ожидается описатель (имя типа или ресурса)"));
		}
		| pat_rel_res RDO_IDENTIF_COLON error
		{
			PARSER->error().error(@2, @3, _T("Ожидается описатель (имя типа или ресурса)"));
		}
		| pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF error
		{
			if (PARSER->getLastPATPattern()->isHaveConvertEnd())
			{
				PARSER->error().error(@3, @4, _T("Ожидается статус конвертора начала"));
			}
			else
			{
				PARSER->error().error(@3, @4, _T("Ожидается статус конвертора"));
			}
		}
		| pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF error
		{
			if (PARSER->getLastPATPattern()->isHaveConvertEnd())
			{
				PARSER->error().error(@3, @4, _T("Ожидается статус конвертора начала"));
			}
			else
			{
				PARSER->error().error(@3, @4, _T("Ожидается статус конвертора"));
			}
		}
		| pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF pat_conv error
		{
			switch (PARSER->getLastPATPattern()->getType())
			{
				case RDOPATPattern::PT_Rule:
				{
					PARSER->error().error(@5, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
					break;
				}
				case RDOPATPattern::PT_Event:
				{
					PARSER->error().error(@5, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
					break;
				}
				case RDOPATPattern::PT_IE:
				{
					PARSER->error().error(@5, _T("Ожидается способ выбора
(first/with_min/with_max) или $Time"));
					break;
				}
				case RDOPATPattern::PT_Operation:
				case RDOPATPattern::PT_Keyboard :
				{
					PARSER->error().error(@4, @5, rdo::format(_T("Ожидается статус
конвертора конца, найдено: %s"), LEXER->YYText()));
					break;
				}
			}
		}
		| pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF pat_conv error
		{
			switch (PARSER->getLastPATPattern()->getType())
			{
				case RDOPATPattern::PT_Rule:
				{
```

```
                                PARSER->error().error(@5, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@5, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
                                PARSER->error().error(@5, _T("Ожидается способ выбора
(first/with_min/with_max) или $Time"));
                                break;
                        }
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PARSER->error().error(@4, @5, rdo::format(_T("Ожидается статус
конвертора конца, найдено: %s"), LEXER->YYText()));
                                break;
                        }
                }
        }
        | pat_params_end RDO_IDENTIF_COLON RDO_IDENTIF_NoChange error
        {
                switch (PARSER->getLastPATPattern()->getType())
                {
                        case RDOPATPattern::PT_Rule:
                        {
                                PARSER->error().error(@4, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@4, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
                                PARSER->error().error(@4, _T("Ожидается способ выбора
(first/with_min/with_max) или $Time"));
                                break;
                        }
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PARSER->error().error(@3, @4, rdo::format(_T("Ожидается статус
конвертора конца, найдено: %s"), LEXER->YYText()));
                                break;
                        }
                }
        }
        | pat_rel_res RDO_IDENTIF_COLON RDO_IDENTIF_NoChange error
        {
                switch (PARSER->getLastPATPattern()->getType())
                {
                        case RDOPATPattern::PT_Rule:
                        {
                                PARSER->error().error(@4, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
                                break;
                        }
                        case RDOPATPattern::PT_Event:
                        {
                                PARSER->error().error(@4, _T("Ожидается способ выбора
(first/with_min/with_max) или $Body"));
                                break;
                        }
                        case RDOPATPattern::PT_IE:
                        {
```

```
                        PARSER->error().error(@4, _T("Ожидается способ выбора
(first/with_min/with_max) или $Time"));
                        break;
                }
                case RDOPATPattern::PT_Operation:
                case RDOPATPattern::PT_Keyboard :
                {
                        PARSER->error().error(@3, @4, rdo::format(_T("Ожидается статус
конвертора конца, найдено: %s"), LEXER->YYText()));
                        break;
                }
        }
    }
    ;

pat_conv
    : RDO_Keep                  { $$ = rdoRuntime::RDOResource::CS_Keep;     }
    | RDO_Create          { $$ = rdoRuntime::RDOResource::CS_Create;   }
    | RDO_Erase                 { $$ = rdoRuntime::RDOResource::CS_Erase;    }
    | RDO_NonExist        { $$ = rdoRuntime::RDOResource::CS_NonExist; }
    ;

pat_common_choice
    : pat_rel_res
    | pat_rel_res RDO_first
    {
        PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
        if (pattern->getType() == RDOPATPattern::PT_IE || pattern->getType() ==
RDOPATPattern::PT_Event)
        {
                PARSER->error().error(@2, _T("В событиях не используется способ выбора
релевантных ресурсов"));
        }
        else
        {
                pattern->setCommonChoiceFirst();
        }
    }
    | pat_rel_res RDO_with_min fun_arithm
    {
        PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
        if (pattern->getType() == RDOPATPattern::PT_IE || pattern->getType() ==
RDOPATPattern::PT_Event)
        {
                PARSER->error().error(@2, _T("В событиях не используется способ выбора
релевантных ресурсов"));
        }
        else
        {
                PTR(RDOFUNArithm) arithm = P_ARITHM($3);
                arithm->setSrcPos (@2, @3);
                arithm->setSrcText(_T("with_min ") + arithm->src_text());
                pattern->setCommonChoiceWithMin(arithm);
        }
    }
    | pat_rel_res RDO_with_max fun_arithm
    {
        PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
        if (pattern->getType() == RDOPATPattern::PT_IE || pattern->getType() ==
RDOPATPattern::PT_Event)
        {
                PARSER->error().error(@2, _T("В событиях не используется способ выбора
релевантных ресурсов"));
        }
        else
        {
                PTR(RDOFUNArithm) arithm = P_ARITHM($3);
                arithm->setSrcPos (@2, @3);
                arithm->setSrcText(_T("with_max ") + arithm->src_text());
                pattern->setCommonChoiceWithMax(arithm);
        }
    }
    | pat_rel_res RDO_with_min error
    {
```

```
			PARSER->error().error(@3, _T("Ошибка в арифметическом выражении"));
		}
		| pat_rel_res RDO_with_max error
		{
			PARSER->error().error(@3, _T("Ошибка в арифметическом выражении"));
		}
		;

pat_time
		: pat_common_choice RDO_Body
		{
			PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
			switch (pattern->getType())
			{
				case RDOPATPattern::PT_IE       :
				case RDOPATPattern::PT_Operation:
				case RDOPATPattern::PT_Keyboard :
				{
					PARSER->error().error(@2, _T("Перед $Body пропущено ключевое слово
$Time"));
					break;
				}
			}
		}
		| pat_common_choice RDO_Time '=' fun_arithm RDO_Body
		{
			PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
			switch (pattern->getType())
			{
				case RDOPATPattern::PT_Event:
				{
					PARSER->error().error(@2, _T("Поле $Time не используется в
событии"));
					break;
				}
				case RDOPATPattern::PT_Rule:
				{
					PARSER->error().error(@2, _T("Поле $Time не используется в
продукционном правиле"));
					break;
				}
			}
			PTR(RDOFUNArithm) arithm = P_ARITHM($4);
			arithm->setSrcPos (@2, @4);
			arithm->setSrcText(_T("$Time = ") + arithm->src_text());
			pattern->setTime(arithm);
		}
		| pat_common_choice RDO_Time '=' fun_arithm error
		{
			PARSER->error().error(@4, @5, _T("Ожидается ключевое слово $Body"));
		}
		| pat_common_choice RDO_Time '=' error
		{
			PARSER->error().error(@4, _T("Ошибка в арифметическом выражении"));
		}
		| pat_common_choice RDO_Time error
		{
			PARSER->error().error(@2, @3, _T("После ключевого слова $Time ожидается знак
равенства"));
		}
		| pat_common_choice error
		{
			PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
			switch (pattern->getType())
			{
				case RDOPATPattern::PT_Rule:
				{
					PARSER->error().error(@2, rdo::format(_T("Ожидается $Body, найдено:
%s"), LEXER->YYText()));
					break;
				}
				case RDOPATPattern::PT_Event:
				{
```

```
                                PARSER->error().error(@2, rdo::format(_T("Ожидается $Body, найдено:
%s"), LEXER->YYText()));
                                break;
                        }
                        case RDOPATPattern::PT_IE       :
                        case RDOPATPattern::PT_Operation:
                        case RDOPATPattern::PT_Keyboard :
                        {
                                PARSER->error().error(@2, rdo::format(_T("Ожидается $Time, найдено:
%s"), LEXER->YYText()));
                                break;
                        }
                }
        }
        ;

pat_body
        : pat_time RDO_IDENTIF_RELRES
        {
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                tstring            name    = RDOVALUE($2)->getIdentificator();
                pattern->addRelResBody(RDOParserSrcInfo(@2, name));
        }
        | pat_convert RDO_IDENTIF_RELRES
        {
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                tstring            name    = RDOVALUE($2)->getIdentificator();
                pattern->addRelResBody(RDOParserSrcInfo(@2, name));
        }
        | pat_time error
        {
                tstring str(LEXER->YYText());
                PARSER->error().error(@2, rdo::format(_T("Неизвестный релевантный ресурс: %s"),
str.c_str()));
        }
        | pat_convert error
        {
                tstring str(LEXER->YYText());
                PARSER->error().error(@2, rdo::format(_T("Неизвестный релевантный ресурс: %s"),
str.c_str()));
        }
        ;

pat_res_usage
        : pat_body pat_choice pat_order
        {
                PTR(RDOPATChoiceFrom) choice_from = reinterpret_cast<PTR(RDOPATChoiceFrom)>($2);
                choice_from->setSrcPos(@2);
                PTR(RDOPATChoiceOrder) choice_order =
reinterpret_cast<PTR(RDOPATChoiceOrder)>($3);
                choice_order->setSrcPos(@3);
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                pattern->addRelResUsage(choice_from, choice_order);
        }
        ;

pat_choice
        : /* empty */
        {
                PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::choiceEmpty;
                $$ = (int) new RDOPATChoiceFrom(PARSER->getLastPATPattern()->m_pCurrRelRes,
RDOParserSrcInfo(_T("Choice NoCheck")), RDOPATChoiceFrom::ch_empty);
        }
        | pat_choice_nocheck
        {
                $$ = (int) new RDOPATChoiceFrom(PARSER->getLastPATPattern()->m_pCurrRelRes,
RDOParserSrcInfo(_T("Choice NoCheck")), RDOPATChoiceFrom::ch_nocheck);
        }
        | pat_choice_from fun_logic
        {
                PTR(RDOFUNLogic) logic = P_LOGIC($2);
                $$ = (int) new RDOPATChoiceFrom(PARSER->getLastPATPattern()->m_pCurrRelRes,
RDOParserSrcInfo(_T("Choice from ") + logic->src_text()), RDOPATChoiceFrom::ch_from, logic);
```

```
	}
	| pat_choice_from error
	{
		PARSER->error().error(@2, _T("Ошибка в логическом выражении"));
	}
	;

pat_choice_nocheck
	: RDO_Choice RDO_NoCheck
	{
		PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::choiceNoCheck;
	}
	;

pat_choice_from
	: RDO_Choice RDO_from
	{
		PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::choiceFrom;
	}
	;

pat_order
	: /* empty */
	{
		PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::choiceOrderEmpty;
		$$ = (int) new RDOPATChoiceOrder(PARSER->getLastPATPattern()->m_pCurrRelRes,
RDOParserSrcInfo(), rdoRuntime::RDOSelectResourceCalc::order_empty);
	}
	| pat_choice_first
	{
		$$ = (int) new RDOPATChoiceOrder(PARSER->getLastPATPattern()->m_pCurrRelRes,
RDOParserSrcInfo(_T("first")), rdoRuntime::RDOSelectResourceCalc::order_first);
	}
	| pat_choice_with_min fun_arithm
	{
		PTR(RDOFUNArithm) arithm = P_ARITHM($2);
		$$ = (int) new RDOPATChoiceOrder(PARSER->getLastPATPattern()->m_pCurrRelRes,
RDOParserSrcInfo(_T("with_min ") + arithm->src_text()),
rdoRuntime::RDOSelectResourceCalc::order_with_min, arithm);
	}
	| pat_choice_with_max fun_arithm
	{
		PTR(RDOFUNArithm) arithm = P_ARITHM($2);
		$$ = (int) new RDOPATChoiceOrder(PARSER->getLastPATPattern()->m_pCurrRelRes,
RDOParserSrcInfo(_T("with_max ") + arithm->src_text()),
rdoRuntime::RDOSelectResourceCalc::order_with_max, arithm);
	}
	| pat_choice_with_min error
	{
		PARSER->error().error(@2, _T("Ошибка в арифметическом выражении"));
	}
	| pat_choice_with_max error
	{
		PARSER->error().error(@2, _T("Ошибка в арифметическом выражении"));
	}
	;

pat_choice_first
	: RDO_first
	{
		PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::choiceOrderFirst;
	}
	;

pat_choice_with_min
	: RDO_with_min
	{
		PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::choiceOrderWithMin;
	}
```

```
	;

pat_choice_with_max
	: RDO_with_max
	{
		PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::choiceOrderWithMax;
	}
	;

pat_convert
	: pat_res_usage
	{
		PTR(RDOPATPattern)	pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		PTR(RDORelevantResource) rel_res = pattern->m_pCurrRelRes;
		tstring str;
		if (rel_res->m_pChoiceOrder->m_type !=
rdoRuntime::RDOSelectResourceCalc::order_empty)
		{
			str = _T("Сразу после ключевого слова ") + rel_res->m_pChoiceOrder-
>asString();
		}
		else if (rel_res->m_pChoiceFrom->m_type != RDOPATChoiceFrom::ch_empty)
		{
			str = _T("Сразу после условия выбора");
		}
		else
		{
			str = _T("Сразу после имени");
		}
		if (rel_res->m_statusBegin != rdoRuntime::RDOResource::CS_NoChange && rel_res-
>m_statusBegin != rdoRuntime::RDOResource::CS_Erase && rel_res->m_statusBegin !=
rdoRuntime::RDOResource::CS_NonExist)
		{
			switch (pattern->getType())
			{
				case RDOPATPattern::PT_IE:
				case RDOPATPattern::PT_Event:
				{
					PARSER->error().error(@1, rdo::format(_T("%s ожидается
ключевое слово Convert_event для релевантного ресурса '%s', т.к. его статус '%s', но найдено:
%s"), str.c_str(), rel_res->name().c_str(), RDOPATPattern::StatusToStr(rel_res-
>m_statusBegin).c_str(), LEXER->YYText()));
					break;
				}
				case RDOPATPattern::PT_Rule:
				{
					PARSER->error().error(@1, rdo::format(_T("%s ожидается
ключевое слово Convert_rule для релевантного ресурса '%s', т.к. его статус '%s', но найдено:
%s"), str.c_str(), rel_res->name().c_str(), RDOPATPattern::StatusToStr(rel_res-
>m_statusBegin).c_str(), LEXER->YYText()));
					break;
				}
				case RDOPATPattern::PT_Operation:
				case RDOPATPattern::PT_Keyboard :
				{
					PARSER->error().error(@1, rdo::format(_T("%s ожидается
ключевое слово Convert_begin для релевантного ресурса '%s', т.к. его статус '%s', но найдено:
%s"), str.c_str(), rel_res->name().c_str(), RDOPATPattern::StatusToStr(rel_res-
>m_statusBegin).c_str(), LEXER->YYText()));
					break;
				}
			}
		}
		if (rel_res->m_statusEnd != rdoRuntime::RDOResource::CS_NoChange && rel_res-
>m_statusEnd != rdoRuntime::RDOResource::CS_Erase && rel_res->m_statusEnd !=
rdoRuntime::RDOResource::CS_NonExist)
		{
			switch (pattern->getType())
			{
				case RDOPATPattern::PT_IE:
				case RDOPATPattern::PT_Event:
				case RDOPATPattern::PT_Rule:
				{
```

```
                                        PARSER->error().error(@1, _T("Внутренняя ошибка"));
                                        break;
                                }
                                case RDOPATPattern::PT_Operation:
                                case RDOPATPattern::PT_Keyboard :
                                {
                                        PARSER->error().error(@1, rdo::format(_T("%s ожидается
ключевое слово Convert_end для релевантного ресурса '%s', т.к. его статус '%s', но найдено: %s"),
str.c_str(), rel_res->name().c_str(), RDOPATPattern::StatusToStr(rel_res->m_statusBegin).c_str(),
LEXER->YYText()));
                                        break;
                                }
                        }
                }
        }
        | pat_res_usage convert_begin pat_trace pat_convert_cmd
        {
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                if (pattern->getType() != RDOPATPattern::PT_Operation && pattern->getType() !=
RDOPATPattern::PT_Keyboard)
                {
                        tstring type = _T("");
                        switch (pattern->getType())
                        {
                                case RDOPATPattern::PT_IE:
                                {
                                        type = _T("нерегулярном событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Event:
                                {
                                        type = _T("событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Rule:
                                {
                                        type = _T("продукционном правиле");
                                        break;
                                }
                        }
                        PARSER->error().error(@2, rdo::format(_T("Ключевое слово Convert_begin
может быть использовано в обыкновенной или клавиатурной операции, но не в %s '%s'"),
type.c_str(), pattern->name().c_str()));
                }
                LPConvertCmdList pCmdList = PARSER->stack().pop<ConvertCmdList>($4);
                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelResConvertBeginEnd($3 != 0,
pCmdList, false, NULL, @2, @2, @3, @3);
        }
        | pat_res_usage convert_end pat_trace pat_convert_cmd
        {
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                if (pattern->getType() != RDOPATPattern::PT_Operation && pattern->getType() !=
RDOPATPattern::PT_Keyboard)
                {
                        tstring type = _T("");
                        switch (pattern->getType())
                        {
                                case RDOPATPattern::PT_IE:
                                {
                                        type = _T("нерегулярном событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Event:
                                {
                                        type = _T("событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Rule:
                                {
                                        type = _T("продукционном правиле");
                                        break;
                                }
                        }
```

```
                    PARSER->error().error(@2, rdo::format(_T("Ключевое слово Convert_end может
быть использовано в обыкновенной и клавиатурной операции, но не в %s '%s'"), type.c_str(),
pattern->name().c_str()));
                }
                LPConvertCmdList pCmdList = PARSER->stack().pop<ConvertCmdList>($4);
                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelResConvertBeginEnd(false,
NULL, $3 != 0, pCmdList, @2, @2, @3, @3);
        }
        | pat_res_usage convert_begin pat_trace pat_convert_cmd convert_end pat_trace
pat_convert_cmd
        {
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                if (pattern->getType() != RDOPATPattern::PT_Operation && pattern->getType() !=
RDOPATPattern::PT_Keyboard)
                {
                        tstring type = _T("");
                        switch (pattern->getType())
                        {
                                case RDOPATPattern::PT_IE:
                                {
                                        type = _T("нерегулярном событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Event:
                                {
                                        type = _T("событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Rule:
                                {
                                        type = _T("продукционном правиле");
                                        break;
                                }
                        }
                        PARSER->error().error(@2, rdo::format(_T("Ключевые слова Convert_begin и
Convert_end могут быть использованы в обыкновенной и клавиатурной операции, но не в %s '%s'"),
type.c_str(), pattern->name().c_str()));
                }
                LPConvertCmdList pCmdListBegin = PARSER->stack().pop<ConvertCmdList>($4);
                LPConvertCmdList pCmdListEnd   = PARSER->stack().pop<ConvertCmdList>($7);
                static_cast<PTR(RDOPatternOperation)>(pattern)->addRelResConvertBeginEnd($3 != 0,
pCmdListBegin, $6 != 0, pCmdListEnd, @2, @5, @3, @6);
        }
        | pat_res_usage convert_rule pat_trace pat_convert_cmd
        {
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                if (pattern->getType() != RDOPATPattern::PT_Rule)
                {
                        tstring type = _T("");
                        switch (pattern->getType())
                        {
                                case RDOPATPattern::PT_IE:
                                {
                                        type = _T("нерегулярном событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Event:
                                {
                                        type = _T("событии");
                                        break;
                                }
                                case RDOPATPattern::PT_Operation:
                                {
                                        type = _T("операции");
                                        break;
                                }
                                case RDOPATPattern::PT_Keyboard :
                                {
                                        type = _T("клавиатурной операции");
                                        break;
                                }
                        }
```

```
                        PARSER->error().error(@2, rdo::format(_T("Ключевое слово Convert_rule может
быть использовано в продукционном правиле, но не в %s '%s'"), type.c_str(), pattern-
>name().c_str()));
                }
                LPConvertCmdList pCmdList = PARSER->stack().pop<ConvertCmdList>($4);
                ASSERT(pattern->m_pCurrRelRes);
                pattern->addRelResConvert($3 != 0, pCmdList, @2, @3, pattern->m_pCurrRelRes-
>m_statusBegin);
        }
        | pat_res_usage convert_event pat_trace pat_convert_cmd
        {
                PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
                if (pattern->getType() != RDOPATPattern::PT_IE && pattern->getType() !=
RDOPATPattern::PT_Event)
                {
                        tstring type = _T("");
                        switch (pattern->getType())
                        {
                                case RDOPATPattern::PT_Rule     :
                                {
                                        type = _T("продукционном правиле");
                                        break;
                                }
                                case RDOPATPattern::PT_Operation:
                                {
                                        type = _T("операции");
                                        break;
                                }
                                case RDOPATPattern::PT_Keyboard :
                                {
                                        type = _T("клавиатурной операции");
                                        break;
                                }
                        }
                        PARSER->error().error(@2, rdo::format(_T("Ключевое слово Convert_event
может быть использовано в событии или в нерегулярном событии, но не в %s '%s'"), type.c_str(),
pattern->name().c_str()));
                }
                LPConvertCmdList pCmdList = PARSER->stack().pop<ConvertCmdList>($4);
                ASSERT(pattern->m_pCurrRelRes);
                pattern->addRelResConvert($3 != 0, pCmdList, @2, @3, pattern->m_pCurrRelRes-
>m_statusBegin);
        }
        ;

convert_rule
        : RDO_Convert_rule
        {
                PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::convertBegin;
        }
        ;

convert_event
        : RDO_Convert_event
        {
                PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::convertBegin;
        }
        ;

convert_begin
        : RDO_Convert_begin
        {
                PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::convertBegin;
        }
        ;

convert_end
        :       RDO_Convert_end
        {
                PARSER->getLastPATPattern()->m_pCurrRelRes->m_currentState =
RDORelevantResource::convertEnd;
```

```
	}
	;

pat_convert_cmd
	: /* empty */
	{
		LPConvertCmdList pCmdList = rdo::Factory<ConvertCmdList>::create();
		PTR(RDORelevantResource) pRelRes = PARSER->getLastPATPattern()->m_pCurrRelRes;
		ASSERT(pRelRes);
		pRelRes->getParamSetList().reset();
		$$ = PARSER->stack().push(pCmdList);
	}
	| pat_convert_cmd statement
	{
		LPConvertCmdList         pCmdList    = PARSER->stack().pop<ConvertCmdList>($1);
		PTR(rdoRuntime::RDOCalc) pCalc       =
reinterpret_cast<PTR(rdoRuntime::RDOCalc)>($2);
		pCmdList->insertCommand(pCalc);
		$$ = PARSER->stack().push(pCmdList);
	}
	;

statement
	: empty_statement
	| nochange_statement
	| equal_statement
	| planning_statement
	| if_statement
	| '{' statement_list '}'
	{
		$$ = $2;
	}
	;

statement_list
	: /* empty */
	{
		PTR(rdoRuntime::RDOCalcList) pCalcList = new rdoRuntime::RDOCalcList(RUNTIME);
		ASSERT(pCalcList);

		$$ = reinterpret_cast<int>(pCalcList);
	}
	| statement_list statement
	{
		PTR(rdoRuntime::RDOCalcList) pCalcList =
reinterpret_cast<PTR(rdoRuntime::RDOCalcList)>($1);
		ASSERT(pCalcList);

		PTR(rdoRuntime::RDOCalc    ) pCalc     = reinterpret_cast<PTR(rdoRuntime::RDOCalc
)>($2);
		ASSERT(pCalc);

		pCalcList->addCalc(pCalc);

		$$ = reinterpret_cast<int>(pCalcList);
	}
	;

empty_statement
	: ';'
	{
		PTR(rdoRuntime::RDOCalcNoChange) pCalc = new rdoRuntime::RDOCalcNoChange(RUNTIME);

		$$ = reinterpret_cast<int>(pCalc);
	}
	| error ';'
	{
		PARSER->error().error(@1, _T("Ошибка в инструкции"));
	}
	;

nochange_statement
	: RDO_IDENTIF_NoChange ';'
	{
```

```
		PTR(rdoRuntime::RDOCalcNoChange) pCalc = new rdoRuntime::RDOCalcNoChange(RUNTIME);

		$$ = reinterpret_cast<int>(pCalc);
	}
	| RDO_IDENTIF_NoChange error
	{
		PARSER->error().error(@2, _T("Не найден символ окончания инструкции - точка с
запятой"));
	}
	;

equal_statement
	: RDO_IDENTIF param_equal_type fun_arithm ';'
	{
		tstring                  paramName   = RDOVALUE($1)->getIdentificator();
		rdoRuntime::EqualType    equalType   = static_cast<rdoRuntime::EqualType>($2);
		PTR(RDOFUNArithm)        rightArithm = P_ARITHM($3);
		PTR(RDORelevantResource) pRelRes     = PARSER->getLastPATPattern()->m_pCurrRelRes;
		ASSERT(pRelRes);
		LPRDORTPParam param = pRelRes->getType()->findRTPParam(paramName);
		if (!param)
		{
			PARSER->error().error(@1, rdo::format(_T("Неизвестный параметр: %s"),
paramName.c_str()));
		}
		PTR(rdoRuntime::RDOCalc) pCalcRight = rightArithm->createCalc(param-
>getParamType().get());
		PTR(rdoRuntime::RDOCalc) pCalc      = NULL;
		switch (equalType)
		{
			case rdoRuntime::ET_NOCHANGE:
			{
				break;
			}
			case rdoRuntime::ET_EQUAL:
			{
				pCalc = new rdoRuntime::RDOSetRelParamCalc<rdoRuntime::ET_EQUAL
>(PARSER->runtime(), pRelRes->m_relResID, pRelRes->getType()->getRTPParamNumber(paramName),
pCalcRight);
				pRelRes->getParamSetList().insert(param);
				break;
			}
			case rdoRuntime::ET_PLUS:
			{
				pCalc = new rdoRuntime::RDOSetRelParamCalc<rdoRuntime::ET_PLUS
>(PARSER->runtime(), pRelRes->m_relResID, pRelRes->getType()->getRTPParamNumber(paramName),
pCalcRight);
				break;
			}
			case rdoRuntime::ET_MINUS:
			{
				pCalc = new rdoRuntime::RDOSetRelParamCalc<rdoRuntime::ET_MINUS
>(PARSER->runtime(), pRelRes->m_relResID, pRelRes->getType()->getRTPParamNumber(paramName),
pCalcRight);
				break;
			}
			case rdoRuntime::ET_MULTIPLY:
			{
				pCalc = new
rdoRuntime::RDOSetRelParamCalc<rdoRuntime::ET_MULTIPLY>(PARSER->runtime(), pRelRes->m_relResID,
pRelRes->getType()->getRTPParamNumber(paramName), pCalcRight);
				break;
			}
			case rdoRuntime::ET_DIVIDE:
			{
				pCalc = new rdoRuntime::RDOSetRelParamCalc<rdoRuntime::ET_DIVIDE
>(PARSER->runtime(), pRelRes->m_relResID, pRelRes->getType()->getRTPParamNumber(paramName),
pCalcRight);
				break;
			}
			default:
			{
				NEVER_REACH_HERE;
			}
```

```
                }
                ASSERT(pCalc);
                //! Проверка на диапазон
                //! TODO: проверить работоспособность
                if (dynamic_cast<PTR(RDOTypeIntRange)>(param->getParamType().get()))
                {
                        LPRDOTypeIntRange pRange = param->getParamType()-
>type().cast<RDOTypeIntRange>();
                        pCalc = new rdoRuntime::RDOSetRelParamDiapCalc(PARSER->runtime(), pRelRes-
>m_relResID, pRelRes->getType()->getRTPParamNumber(paramName), pRange->range()->getMin().value(),
pRange->range()->getMax().value(), pCalc);
                }
                else if (dynamic_cast<PTR(RDOTypeRealRange)>(param->getParamType().get()))
                {
                        LPRDOTypeRealRange pRange = param->getParamType()-
>type().cast<RDOTypeRealRange>();
                        pCalc = new rdoRuntime::RDOSetRelParamDiapCalc(PARSER->runtime(), pRelRes-
>m_relResID, pRelRes->getType()->getRTPParamNumber(paramName), pRange->range()->getMin().value(),
pRange->range()->getMax().value(), pCalc);
                }
                tstring oprStr;
                switch (equalType)
                {
                        case rdoRuntime::ET_EQUAL:
                        {
                                oprStr = _T("=");
                                break;
                        }
                        case rdoRuntime::ET_PLUS:
                        {
                                oprStr = _T("+=");
                                break;
                        }
                        case rdoRuntime::ET_MINUS:
                        {
                                oprStr = _T("-=");
                                break;
                        }
                        case rdoRuntime::ET_MULTIPLY:
                        {
                                oprStr = _T("*=");
                                break;
                        }
                        case rdoRuntime::ET_DIVIDE:
                        {
                                oprStr = _T("/=");
                                break;
                        }
                        default:
                        {
                                oprStr = _T("");
                                break;
                        }
                }
                pCalc->setSrcText(rdo::format(_T("%s %s %s"), paramName.c_str(), oprStr.c_str(),
pCalcRight->src_text().c_str()));
                pCalc->setSrcPos (@1.first_line, @1.first_column, @3.last_line, @3.last_column);

                $$ = reinterpret_cast<int>(pCalc);
        }
        | RDO_IDENTIF param_equal_type error
        {
                PARSER->error().error(@3, _T("Ошибка в арифметическом выражении"));
        }
        | RDO_IDENTIF param_equal_type fun_arithm error
        {
                PARSER->error().error(@4, _T("Не найден символ окончания инструкции - точка с
запятой"));
        }
        | RDO_IDENTIF error fun_arithm
        {
                PARSER->error().error(@2, _T("Ошибка в операторе присваивания"));
        }
        ;
```

```
planning_statement
	: RDO_IDENTIF '.' RDO_Planning '(' fun_arithm ')' ';'
	{
		tstring           eventName   = RDOVALUE($1)->getIdentificator();
		PTR(RDOFUNArithm) pTimeArithm = P_ARITHM($5);
		LPRDOEvent        pEvent      = PARSER->findEvent(eventName);
		if (!pEvent)
		{
			PARSER->error().error(@1, rdo::format(_T("Попытка запланировать неизвестное
событие: %s"), eventName.c_str()));
		}

		PTR(rdoRuntime::RDOCalc) pCalcTime = pTimeArithm->createCalc(NULL);
		ASSERT(pCalcTime);

		PTR(rdoRuntime::RDOCalcEventPlan) pCalc = new
rdoRuntime::RDOCalcEventPlan(RUNTIME, pCalcTime);
		ASSERT(pCalc);
		pEvent->attachCalc(pCalc);

		$$ = reinterpret_cast<int>(pCalc);
	}
	| RDO_IDENTIF '.' RDO_Planning '(' fun_arithm ')' error
	{
		PARSER->error().error(@7, _T("Не найден символ окончания инструкции - точка с
запятой"));
	}
	| RDO_IDENTIF '.' RDO_Planning '(' error
	{
		PARSER->error().error(@5, _T("Ошибка в арифметическом выражении"));
	}
	| RDO_IDENTIF '.' RDO_Planning error
	{
		PARSER->error().error(@4, _T("Ожидается открывающая скобка"));
	}
	| RDO_IDENTIF '.' RDO_Planning '(' fun_arithm error
	{
		PARSER->error().error(@6, _T("Ожидается закрывающая скобка"));
	}
	;

if_statement
	: RDO_if '(' fun_logic ')' statement
	{
		PTR(RDOFUNLogic) pCondition = P_LOGIC($3);
		ASSERT(pCondition);

		PTR(rdoRuntime::RDOCalc) pConditionCalc = pCondition->getCalc();
		ASSERT(pConditionCalc);

		PTR(rdoRuntime::RDOCalc) pStatementCalc =
reinterpret_cast<PTR(rdoRuntime::RDOCalc)>($5);
		ASSERT(pStatementCalc);

		PTR(rdoRuntime::RDOCalcIf) pCalc = new rdoRuntime::RDOCalcIf(RUNTIME,
pConditionCalc, pStatementCalc);
		ASSERT(pCalc);

		$$ = reinterpret_cast<int>(pCalc);
	}
	| RDO_if '(' fun_logic ')' statement RDO_else statement
	{
		PTR(RDOFUNLogic) pCondition = P_LOGIC($3);
		ASSERT(pCondition);

		PTR(rdoRuntime::RDOCalc) pConditionCalc = pCondition->getCalc();
		ASSERT(pConditionCalc);

		PTR(rdoRuntime::RDOCalc) pIfStatementCalc =
reinterpret_cast<PTR(rdoRuntime::RDOCalc)>($5);
		ASSERT(pIfStatementCalc);
```

```
		PTR(rdoRuntime::RDOCalc) pElseStatementCalc =
reinterpret_cast<PTR(rdoRuntime::RDOCalc)>($7);
		ASSERT(pElseStatementCalc);

		PTR(rdoRuntime::RDOCalcIfElse) pCalc = new rdoRuntime::RDOCalcIfElse(RUNTIME,
pConditionCalc, pIfStatementCalc, pElseStatementCalc);
		ASSERT(pCalc);

		$$ = reinterpret_cast<int>(pCalc);
	}
	| RDO_if error fun_logic
	{
		PARSER->error().error(@2, _T("Ожидается открывающая скобка"));
	}
	| RDO_if '(' fun_logic error
	{
		PARSER->error().error(@4, _T("Ожидается закрывающая скобка"));
	}
	;

param_equal_type
	: RDO_set
	{
		$$ = rdoRuntime::ET_EQUAL;
	}
	| '='
	{
		$$ = rdoRuntime::ET_EQUAL;
	}
	| RDO_PlusEqual
	{
		$$ = rdoRuntime::ET_PLUS;
	}
	| RDO_MinusEqual
	{
		$$ = rdoRuntime::ET_MINUS;
	}
	| RDO_MultiplyEqual
	{
		$$ = rdoRuntime::ET_MULTIPLY;
	}
	| RDO_DivideEqual
	{
		$$ = rdoRuntime::ET_DIVIDE;
	}
	;

pat_pattern
	: pat_convert RDO_End
	{
		PTR(RDOPATPattern) pattern = reinterpret_cast<PTR(RDOPATPattern)>($1);
		pattern->end();
	}
	;

// --------------------------------------------------------------------------------
// ---------- Описание типа параметра
// --------------------------------------------------------------------------------
param_type
	: RDO_integer param_type_range param_value_default
	{
		LPRDOTypeRangeRange pRange = PARSER->stack().pop<RDOTypeRangeRange>($2);
		LPRDOTypeParam pType;
		if (pRange)
		{
			if (pRange->getMin().typeID() != rdoRuntime::RDOType::t_int ||
				pRange->getMax().typeID() != rdoRuntime::RDOType::t_int)
			{
				PARSER->error().error(@2, _T("Диапазон целого типа должен быть
целочисленным"));
			}
			LPRDOTypeIntRange pIntRange =
rdo::Factory<RDOTypeIntRange>::create(pRange);
```

```
                        pType = rdo::Factory<RDOTypeParam>::create(pIntRange, RDOVALUE($3),
RDOParserSrcInfo(@1, @3));
                }
                else
                {
                        pType = rdo::Factory<RDOTypeParam>::create(g_int, RDOVALUE($3),
RDOParserSrcInfo(@1, @3));
                }
                $$ = PARSER->stack().push(pType);
        }
        | RDO_real param_type_range param_value_default
        {
                LPRDOTypeRangeRange pRange = PARSER->stack().pop<RDOTypeRangeRange>($2);
                LPRDOTypeParam pType;
                if (pRange)
                {
                        LPRDOTypeRealRange pRealRange =
rdo::Factory<RDOTypeRealRange>::create(pRange);
                        pType = rdo::Factory<RDOTypeParam>::create(pRealRange, RDOVALUE($3),
RDOParserSrcInfo(@1, @3));
                }
                else
                {
                        pType = rdo::Factory<RDOTypeParam>::create(g_real, RDOVALUE($3),
RDOParserSrcInfo(@1, @3));
                }
                $$ = PARSER->stack().push(pType);
        }
        | RDO_string param_value_default
        {
                LPRDOTypeParam pType = rdo::Factory<RDOTypeParam>::create(g_string, RDOVALUE($2),
RDOParserSrcInfo(@1, @2));
                $$ = PARSER->stack().push(pType);
        }
        | RDO_bool param_value_default
        {
                LPRDOTypeParam pType = rdo::Factory<RDOTypeParam>::create(g_bool, RDOVALUE($2),
RDOParserSrcInfo(@1, @2));
                $$ = PARSER->stack().push(pType);
        }
        | param_type_enum param_value_default
        {
                LEXER->enumReset();
                LPRDOEnumType pEnum = PARSER->stack().pop<RDOEnumType>($1);
                LPRDOTypeParam pType = rdo::Factory<RDOTypeParam>::create(pEnum, RDOVALUE($2),
RDOParserSrcInfo(@1, @2));
                $$ = PARSER->stack().push(pType);
        }
        | param_type_such_as param_value_default
        {
                LPRDOTypeParam pTypeSuchAs = PARSER->stack().pop<RDOTypeParam>($1);
                ASSERT(pTypeSuchAs);
                LPRDOTypeParam pType = rdo::Factory<RDOTypeParam>::create(pTypeSuchAs->type(),
RDOVALUE($2), RDOParserSrcInfo(@1, @2));
                $$ = PARSER->stack().push(pType);
        }
        ;

param_type_range
        : /* empty */
        {
                $$ = PARSER->stack().push<RDOTypeRangeRange>(LPRDOTypeRangeRange());
        }
        | '[' RDO_INT_CONST RDO_dblpoint RDO_INT_CONST ']'
        {
                LPRDOTypeRangeRange pRange = rdo::Factory<RDOTypeRangeRange>::create(RDOVALUE($2),
RDOVALUE($4), RDOParserSrcInfo(@1, @5));
                pRange->checkRange();
                $$ = PARSER->stack().push(pRange);
        }
        | '[' RDO_REAL_CONST RDO_dblpoint RDO_REAL_CONST ']'
        {
                LPRDOTypeRangeRange pRange = rdo::Factory<RDOTypeRangeRange>::create(RDOVALUE($2),
RDOVALUE($4), RDOParserSrcInfo(@1, @5));
```

```
                pRange->checkRange();
                $$ = PARSER->stack().push(pRange);
        }
        | '[' RDO_REAL_CONST RDO_dblpoint RDO_INT_CONST ']'
        {
                LPRDOTypeRangeRange pRange = rdo::Factory<RDOTypeRangeRange>::create(RDOVALUE($2),
RDOVALUE($4), RDOParserSrcInfo(@1, @5));
                pRange->checkRange();
                $$ = PARSER->stack().push(pRange);
        }
        | '[' RDO_INT_CONST RDO_dblpoint RDO_REAL_CONST ']'
        {
                LPRDOTypeRangeRange pRange = rdo::Factory<RDOTypeRangeRange>::create(RDOVALUE($2),
RDOVALUE($4), RDOParserSrcInfo(@1, @5));
                pRange->checkRange();
                $$ = PARSER->stack().push(pRange);
        }
        | '[' RDO_REAL_CONST RDO_dblpoint RDO_REAL_CONST error
        {
                PARSER->error().error(@4, _T("Диапазон задан неверно"));
        }
        | '[' RDO_REAL_CONST RDO_dblpoint RDO_INT_CONST error
        {
                PARSER->error().error(@4, _T("Диапазон задан неверно"));
        }
        | '[' RDO_INT_CONST RDO_dblpoint RDO_REAL_CONST error
        {
                PARSER->error().error(@4, _T("Диапазон задан неверно"));
        }
        | '[' RDO_INT_CONST RDO_dblpoint RDO_INT_CONST error
        {
                PARSER->error().error(@4, _T("Диапазон задан неверно"));
        }
        | '[' RDO_REAL_CONST RDO_dblpoint error
        {
                PARSER->error().error(@4, _T("Диапазон задан неверно"));
        }
        | '[' RDO_INT_CONST RDO_dblpoint error
        {
                PARSER->error().error(@4, _T("Диапазон задан неверно"));
        }
        | '[' error
        {
                PARSER->error().error(@2, _T("Диапазон задан неверно"));
        }
        ;

param_type_enum
        : '(' param_type_enum_list ')'
        {
                LPRDOEnumType pEnum = PARSER->stack().pop<RDOEnumType>($2);
                $$ = PARSER->stack().push(pEnum);
        }
        | '(' param_type_enum_list error
        {
                PARSER->error().error(@2, _T("Перечисление должно заканчиваться скобкой"));
        }
        ;

param_type_enum_list
        : RDO_IDENTIF
        {
                LPRDOEnumType pEnum = rdo::Factory<RDOEnumType>::create();
                pEnum->add(RDOVALUE($1));
                LEXER->enumBegin();
                $$ = PARSER->stack().push(pEnum);
        }
        | param_type_enum_list ',' RDO_IDENTIF
        {
                if (!LEXER->enumEmpty())
                {
                        LPRDOEnumType pEnum = PARSER->stack().pop<RDOEnumType>($1);
                        pEnum->add(RDOVALUE($3));
                        $$ = PARSER->stack().push(pEnum);
```

```
		}
		else
		{
			PARSER->error().error(@3, _T("Ошибка в описании значений перечислимого
типа"));
		}
	}
	| param_type_enum_list RDO_IDENTIF
	{
		if (!LEXER->enumEmpty())
		{
			LPRDOEnumType pEnum = PARSER->stack().pop<RDOEnumType>($1);
			pEnum->add(RDOVALUE($2));
			$$ = PARSER->stack().push(pEnum);
			PARSER->error().warning(@1, rdo::format(_T("Пропущена запятая перед: %s"),
RDOVALUE($2)->getIdentificator().c_str()));
		}
		else
		{
			PARSER->error().error(@2, _T("Ошибка в описании значений перечислимого
типа"));
		}
	}
	| param_type_enum_list ',' RDO_INT_CONST
	{
		PARSER->error().error(@3, _T("Значение перечислимого типа не может быть цифрой"));
	}
	| param_type_enum_list ',' RDO_REAL_CONST
	{
		PARSER->error().error(@3, _T("Значение перечислимого типа не может быть цифрой"));
	}
	| param_type_enum_list RDO_INT_CONST
	{
		PARSER->error().error(@2, _T("Значение перечислимого типа не может быть цифрой"));
	}
	| param_type_enum_list RDO_REAL_CONST
	{
		PARSER->error().error(@2, _T("Значение перечислимого типа не может быть цифрой"));
	}
	| RDO_INT_CONST
	{
		PARSER->error().error(@1, _T("Значение перечислимого типа не может начинаться с
цифры"));
	}
	| RDO_REAL_CONST
	{
		PARSER->error().error(@1, _T("Значение перечислимого типа не может начинаться с
цифры"));
	}
	;

param_type_such_as
	: RDO_such_as RDO_IDENTIF '.' RDO_IDENTIF
	{
		tstring type  = RDOVALUE($2)->getIdentificator();
		tstring param = RDOVALUE($4)->getIdentificator();
		LPRDORTPResType pRTP = PARSER->findRTPResType(type);
		if (!pRTP)
		{
			PARSER->error().error(@2, rdo::format(_T("Ссылка на неизвестный тип
ресурса: %s"), type.c_str()));
		}
		LPRDORTPParam pParam = pRTP->findRTPParam(param);
		if (!pParam)
		{
			PARSER->error().error(@4, rdo::format(_T("Ссылка на неизвестный параметр
ресурса: %s.%s"), type.c_str(), param.c_str()));
		}
		$$ = PARSER->stack().push(pParam->getParamType());
	}
	| RDO_such_as RDO_IDENTIF
	{
		tstring constName = RDOVALUE($2)->getIdentificator();
		CPTR(RDOFUNConstant) const cons = PARSER->findFUNConstant(constName);
```

```
                if (!cons)
                {
                        PARSER->error().error(@2, rdo::format(_T("Ссылка на несуществующую
константу: %s"), constName.c_str()));
                }
                $$ = PARSER->stack().push(cons->getType());
        }
        | RDO_such_as RDO_IDENTIF '.' error
        {
                tstring type = RDOVALUE($2)->getIdentificator();
                LPRDORTPResType const rt = PARSER->findRTPResType(type);
                if (!rt)
                {
                        PARSER->error().error(@2, rdo::format(_T("Ссылка на неизвестный тип
ресурса: %s"), type.c_str()));
                }
                else
                {
                        PARSER->error().error(@4, _T("Ошибка при указании параметра"));
                }
        }
        | RDO_such_as error
        {
                PARSER->error().error(@2, _T("После ключевого слова such_as необходимо указать тип
и параметер ресурса для ссылки"));
        }
        ;

param_value_default
        : /* empty */
        {
                $$ = (int)PARSER->addValue(new rdoParse::RDOValue());
        }
        | '=' RDO_INT_CONST
        {
                $$ = $2;
        }
        | '=' RDO_REAL_CONST
        {
                $$ = $2;
        }
        | '=' RDO_STRING_CONST
        {
                $$ = $2;
        }
        | '=' RDO_IDENTIF
        {
                $$ = $2;
        }
        | '=' RDO_BOOL_CONST
        {
                $$ = $2;
        }
        | '=' error
        {
                RDOParserSrcInfo src_info(@1, @2, true);
                if (src_info.src_pos().point())
                {
                        PARSER->error().error(src_info, _T("Не указано значение по-умолчанию"));
                }
                else
                {
                        PARSER->error().error(src_info, _T("Неверное значение по-умолчанию "));
                }
        }
        ;

// --------------------------------------------------------------------------
// ---------- Логические выражения
// --------------------------------------------------------------------------
fun_logic_eq
        : '='    { $$ = RDO_eq; }
        | RDO_eq { $$ = RDO_eq; }
        ;
```

```
fun_logic
	: fun_arithm  fun_logic_eq  fun_arithm   { $$ = (int)(ARITHM($1) == ARITHM($3));  }
	| fun_arithm  RDO_neq       fun_arithm   { $$ = (int)(ARITHM($1) != ARITHM($3));  }
	| fun_arithm  '<'           fun_arithm   { $$ = (int)(ARITHM($1) <  ARITHM($3));  }
	| fun_arithm  '>'           fun_arithm   { $$ = (int)(ARITHM($1) >  ARITHM($3));  }
	| fun_arithm  RDO_leq       fun_arithm   { $$ = (int)(ARITHM($1) <= ARITHM($3));  }
	| fun_arithm  RDO_geq       fun_arithm   { $$ = (int)(ARITHM($1) >= ARITHM($3));  }
	| fun_logic   RDO_and       fun_logic    { $$ = (int)(LOGIC($1) && LOGIC($3));    }
	| fun_logic   RDO_or        fun_logic    { $$ = (int)(LOGIC($1) || LOGIC($3));    }
	| fun_arithm                             { $$ = (int)new RDOFUNLogic(ARITHM($1)); }
	| fun_group
	| fun_select_logic
	| '[' fun_logic ']'
	{
		PTR(RDOFUNLogic) logic = P_LOGIC($2);
		logic->setSrcPos (@1, @3);
		logic->setSrcText(_T("[") + logic->src_text() + _T("]"));
		$$ = $2;
	}
	| '(' fun_logic ')'
	{
		PTR(RDOFUNLogic) logic = P_LOGIC($2);
		logic->setSrcPos (@1, @3);
		logic->setSrcText(_T("(") + logic->src_text() + _T(")"));
		$$ = $2;
	}
	| RDO_not fun_logic
	{
		PTR(RDOFUNLogic) logic = P_LOGIC($2);
		PTR(RDOFUNLogic) logic_not = logic->operator_not();
		logic_not->setSrcPos (@1, @2);
		logic_not->setSrcText(_T("not ") + logic->src_text());
		$$ = (int)logic_not;
	}
	| '[' fun_logic error
	{
		PARSER->error().error(@2, _T("Ожидается закрывающаяся скобка"));
	}
	| '(' fun_logic error
	{
		PARSER->error().error(@2, _T("Ожидается закрывающаяся скобка"));
	}
	;

// --------------------------------------------------------------------------------
// ---------- Арифметические выражения
// --------------------------------------------------------------------------------
fun_arithm
	: RDO_INT_CONST                      { $$ = (int)new RDOFUNArithm(PARSER, RDOVALUE($1)); }
	| RDO_REAL_CONST                     { $$ = (int)new RDOFUNArithm(PARSER, RDOVALUE($1)); }
	| RDO_BOOL_CONST                     { $$ = (int)new RDOFUNArithm(PARSER, RDOVALUE($1)); }
	| RDO_STRING_CONST                   { $$ = (int)new RDOFUNArithm(PARSER, RDOVALUE($1)); }
	| RDO_IDENTIF                        { $$ = (int)new RDOFUNArithm(PARSER, RDOVALUE($1)); }
	| RDO_IDENTIF '.' RDO_IDENTIF        { $$ = (int)new RDOFUNArithm(PARSER, RDOVALUE($1),
RDOVALUE($3)); }
	| RDO_IDENTIF_RELRES '.' RDO_IDENTIF { $$ = (int)new RDOFUNArithm(PARSER, RDOVALUE($1),
RDOVALUE($3)); }
	| fun_arithm '+' fun_arithm              { $$ = (int)(ARITHM($1) + ARITHM($3)); }
	| fun_arithm '-' fun_arithm              { $$ = (int)(ARITHM($1) - ARITHM($3)); }
	| fun_arithm '*' fun_arithm              { $$ = (int)(ARITHM($1) * ARITHM($3)); }
	| fun_arithm '/' fun_arithm              { $$ = (int)(ARITHM($1) / ARITHM($3)); }
	| fun_arithm_func_call
	| fun_select_arithm
	| '(' fun_arithm ')'
	{
		PTR(RDOFUNArithm) arithm = P_ARITHM($2);
		arithm->setSrcPos (@1, @3);
		arithm->setSrcText(_T("(") + arithm->src_text() + _T(")"));
		$$ = $2;
	}
	| '-' fun_arithm %prec RDO_UMINUS
	{
		RDOParserSrcInfo info;
```

```
            info.setSrcPos (@1, @2);
            info.setSrcText(_T("-") + ARITHM($2).src_text());
            $$ = (int)new RDOFUNArithm(PARSER, RDOValue(ARITHM($2).type(), info), new
rdoRuntime::RDOCalcUMinus(RUNTIME, ARITHM($2).createCalc()));
        }
        ;

// --------------------------------------------------------------------------
// ---------- Функции и последовательности
// --------------------------------------------------------------------------
fun_arithm_func_call
        : RDO_IDENTIF '(' ')'
        {
            PTR(RDOFUNParams) fun = new RDOFUNParams(PARSER);
            tstring fun_name = RDOVALUE($1)->getIdentificator();
            fun->funseq_name.setSrcInfo(RDOParserSrcInfo(@1, fun_name));
            fun->setSrcPos (@1, @3);
            fun->setSrcText(fun_name + _T("()"));
            PTR(RDOFUNArithm) arithm = fun->createCall(fun_name);
            $$ = (int)arithm;
        }
        | RDO_IDENTIF '(' fun_arithm_func_call_pars ')'
        {
            PTR(RDOFUNParams) fun      = reinterpret_cast<PTR(RDOFUNParams)>($3);
            tstring           fun_name = RDOVALUE($1)->getIdentificator();
            fun->funseq_name.setSrcInfo(RDOParserSrcInfo(@1, fun_name));
            fun->setSrcPos (@1, @4);
            fun->setSrcText(fun_name + _T("(") + fun->src_text() + _T(")"));
            PTR(RDOFUNArithm) arithm = fun->createCall(fun_name);
            $$ = (int)arithm;
        }
        | RDO_IDENTIF '(' error
        {
            PARSER->error().error(@3, _T("Ошибка в параметрах функции"));
        }
        ;

fun_arithm_func_call_pars
        : fun_arithm
        {
            PTR(RDOFUNParams) fun    = new RDOFUNParams(PARSER);
            PTR(RDOFUNArithm) arithm = P_ARITHM($1);
            fun->setSrcText  (arithm->src_text());
            fun->addParameter(arithm);
            $$ = (int)fun;
        }
        | fun_arithm_func_call_pars ',' fun_arithm
        {
            PTR(RDOFUNParams) fun    = reinterpret_cast<PTR(RDOFUNParams)>($1);
            PTR(RDOFUNArithm) arithm = P_ARITHM($3);
            fun->setSrcText  (fun->src_text() + _T(", ") + arithm->src_text());
            fun->addParameter(arithm);
            $$ = (int)fun;
        }
        | fun_arithm_func_call_pars error
        {
            PARSER->error().error(@2, _T("Ошибка в арифметическом выражении"));
        }
        | fun_arithm_func_call_pars ',' error
        {
            PARSER->error().error(@3, _T("Ошибка в арифметическом выражении"));
        }
        ;

// --------------------------------------------------------------------------
// ---------- Групповые выражения
// --------------------------------------------------------------------------
fun_group_keyword
        : RDO_Exist                 { $$ = RDOFUNGroupLogic::fgt_exist;     }
        | RDO_Not_Exist             { $$ = RDOFUNGroupLogic::fgt_notexist;  }
        | RDO_For_All         { $$ = RDOFUNGroupLogic::fgt_forall;    }
        | RDO_Not_For_All     { $$ = RDOFUNGroupLogic::fgt_notforall; }
        ;
```

```
fun_group_header
	: fun_group_keyword '(' RDO_IDENTIF_COLON
	{
		PTR(RDOValue) type_name = P_RDOVALUE($3);
		$$ = (int)(new RDOFUNGroupLogic(PARSER, (RDOFUNGroupLogic::FunGroupType)$1,
type_name->src_info()));
	}
	| fun_group_keyword '(' error
	{
		PARSER->error().error(@3, _T("Ожидается имя типа"));
	}
	| fun_group_keyword error
	{
		PARSER->error().error(@1, _T("После имени функции ожидается октрывающаяся
скобка"));
	}
	;

fun_group
	: fun_group_header fun_logic ')'
	{
		PTR(RDOFUNGroupLogic) groupfun = reinterpret_cast<PTR(RDOFUNGroupLogic)>($1);
		groupfun->setSrcPos(@1, @3);
		$$ = (int)groupfun->createFunLogic(P_LOGIC($2));
	}
	| fun_group_header RDO_NoCheck ')'
	{
		PTR(RDOFUNGroupLogic) groupfun = reinterpret_cast<PTR(RDOFUNGroupLogic)>($1);
		groupfun->setSrcPos(@1, @3);
		PTR(RDOFUNLogic) trueLogic = new RDOFUNLogic(groupfun, new
rdoRuntime::RDOCalcConst(RUNTIME, 1));
		trueLogic->setSrcPos (@2);
		trueLogic->setSrcText(_T("NoCheck"));
		$$ = (int)groupfun->createFunLogic(trueLogic);
	}
	| fun_group_header fun_logic error
	{
		PARSER->error().error(@2, _T("Ожидается закрывающаяся скобка"));
	}
	| fun_group_header RDO_NoCheck error
	{
		PARSER->error().error(@2, _T("Ожидается закрывающаяся скобка"));
	}
	| fun_group_header error
	{
		PARSER->error().error(@1, @2, _T("Ошибка в логическом выражении"));
	}
	;

// --------------------------------------------------------------------------
// ---------- Select
// --------------------------------------------------------------------------
fun_select_header
	: RDO_Select '(' RDO_IDENTIF_COLON
	{
		PTR(RDOValue) type_name  = P_RDOVALUE($3);
		PTR(RDOFUNSelect) select = new RDOFUNSelect(PARSER, type_name->src_info());
		select->setSrcText(_T("Select(") + type_name->value().getIdentificator() + _T(":
"));
		$$ = (int)select;
	}
	| RDO_Select '(' error
	{
		PARSER->error().error(@3, _T("Ожидается имя типа"));
	}
	| RDO_Select error
	{
		PARSER->error().error(@1, _T("Ожидается октрывающаяся скобка"));
	}
	;

fun_select_body
	: fun_select_header fun_logic ')'
	{
```

```
                PTR(RDOFUNSelect) select = reinterpret_cast<PTR(RDOFUNSelect)>($1);
                PTR(RDOFUNLogic)  flogic = P_LOGIC($2);
                select->setSrcText(select->src_text() + flogic->src_text() + _T(")"));
                select->initSelect(flogic);
        }
        | fun_select_header RDO_NoCheck ')'
        {
                PTR(RDOFUNSelect) select = reinterpret_cast<PTR(RDOFUNSelect)>($1);
                RDOParserSrcInfo logic_info(@2, _T("NoCheck"));
                select->setSrcText(select->src_text() + logic_info.src_text() + _T(")"));
                PTR(rdoRuntime::RDOCalcConst) calc_nocheck = new rdoRuntime::RDOCalcConst(RUNTIME,
1);
                PTR(RDOFUNLogic)              flogic       = new RDOFUNLogic(select, calc_nocheck,
true);
                flogic->setSrcInfo(logic_info);
                select->initSelect(flogic);
        }
        | fun_select_header fun_logic error
        {
                PARSER->error().error(@2, _T("Ожидается закрывающаяся скобка"));
        }
        | fun_select_header RDO_NoCheck error
        {
                PARSER->error().error(@2, _T("Ожидается закрывающаяся скобка"));
        }
        | fun_select_header error
        {
                PARSER->error().error(@1, @2, _T("Ошибка в логическом выражении"));
        }
        ;

fun_select_keyword
        : RDO_Exist                  { $$ = RDOFUNGroupLogic::fgt_exist;     }
        | RDO_Not_Exist              { $$ = RDOFUNGroupLogic::fgt_notexist;  }
        | RDO_For_All         { $$ = RDOFUNGroupLogic::fgt_forall;    }
        | RDO_Not_For_All     { $$ = RDOFUNGroupLogic::fgt_notforall; }
        ;

fun_select_logic
        : fun_select_body '.' fun_select_keyword '(' fun_logic ')'
        {
                PTR(RDOFUNSelect) select = reinterpret_cast<PTR(RDOFUNSelect)>($1);
                select->setSrcPos(@1, @6);
                PTR(RDOFUNLogic) logic = select-
>createFunSelectGroup((RDOFUNGroupLogic::FunGroupType)$3, P_LOGIC($5));
                $$ = (int)logic;
        }
        | fun_select_body '.' fun_select_keyword '(' error
        {
                PARSER->error().error(@4, @5, _T("Ошибка в логическом выражении"));
        }
        | fun_select_body '.' fun_select_keyword error
        {
                PARSER->error().error(@3, _T("Ожидается октрывающаяся скобка"));
        }
        | fun_select_body '.' RDO_Empty '(' ')'
        {
                PTR(RDOFUNSelect) select = reinterpret_cast<PTR(RDOFUNSelect)>($1);
                select->setSrcPos(@1, @5);
                RDOParserSrcInfo empty_info(@3, @5, _T("Empty()"));
                PTR(RDOFUNLogic) logic = select->createFunSelectEmpty(empty_info);
                $$ = (int)logic;
        }
        | fun_select_body '.' RDO_Empty '(' error
        {
                PARSER->error().error(@4, _T("Ожидается закрывающаяся скобка"));
        }
        | fun_select_body '.' RDO_Empty error
        {
                PARSER->error().error(@3, _T("Ожидается октрывающаяся скобка"));
        }
        | fun_select_body '.' error
        {
                PARSER->error().error(@2, @3, _T("Ожидается метод списка ресурсов"));
```

```
        }
        | fun_select_body error
        {
                PARSER->error().error(@1, _T("Ожидается '.' (точка) для вызова метода списка
ресурсов"));
        }
        ;

fun_select_arithm
        : fun_select_body '.' RDO_Size '(' ')'
        {
                PTR(RDOFUNSelect) select = reinterpret_cast<PTR(RDOFUNSelect)>($1);
                select->setSrcPos(@1, @5);
                RDOParserSrcInfo size_info(@3, @5, _T("Size()"));
                PTR(RDOFUNArithm) arithm = select->createFunSelectSize(size_info);
                $$ = (int)arithm;
        }
        | fun_select_body '.' RDO_Size error
        {
                PARSER->error().error(@3, _T("Ожидается октрывающаяся скобка"));
        }
        | fun_select_body '.' RDO_Size '(' error
        {
                PARSER->error().error(@4, _T("Ожидается закрывающаяся скобка"));
        }
        ;

%%

CLOSE_RDO_PARSER_NAMESPACE
```

Приложение 2. Код имитационной модели работы почтового отделения связи на языке РДО.

Postoffice.rtp (Типы ресурсов):

```
$Resource_type Парикмахерские: permanent
$Parameters
	состояние_парикмахера : ( Свободен, Занят )
	число_ожидающих       : integer
	количество_обслуженных: integer
$End
```

Postoffice.rss (Ресурсы):

```
$Resources
	Парикмахерская: Парикмахерские trace Свободен 0 0
$End
```

Postoffice.fun (Константы, последовательности и функции):

```
$Sequence Время_между_приходами : real
$Type = exponential 2345
$End

$Sequence Длительность_обслуживания : real
$Type = uniform 879433216
$End
```

Postoffice.pat (Образцы):

```
$Pattern Событие_прихода_клиента: event trace
$Relevant_resources
	_Парикмахерская: Парикмахерская Keep
$Body
_Парикмахерская
	Convert_event
		Событие_прихода_клиента.Planning(Time_now + Время_между_приходами(30));
		if (состояние_парикмахера == Занят)
		{
			число_ожидающих += 1;
		}
		else
		{
			состояние_парикмахера = Занят;
			Событие_окончания_обслуживания_клиента.planning(Time_now +
Длительность_обслуживания( 20, 40 ));
		}
$End

$Pattern Событие_окончания_обслуживания_клиента: event trace
$Relevant_resources
	_Парикмахерская: Парикмахерская Keep
$Body
_Парикмахерская
	Convert_event
		количество_обслуженных += 1;
		if (число_ожидающих>0)
		{
			число_ожидающих -= 1;
			Событие_окончания_обслуживания_клиента.planning(Time_now +
Длительность_обслуживания( 20, 40 ));
```

```
                }
                else
                {
                        состояние_парикмахера = Свободен;
                }
$End
```

Postoffice.smr (Прогон):

```
Model_name     = mymodel2

Resource_file  = mymodel2
Statistic_file = mymodel2
Results_file   = mymodel2
Trace_file     = mymodel2
Show_mode      = NoShow
Show_rate      = 3600.0

Событие_прихода_клиента.Planning(Время_между_приходами(30))

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

Postoffice.pmd (Показатели):

```
$Results
        Занятость_парикмахера : watch_state Парикмахерская.состояние_парикмахера = Занят
        Длина_очереди         : watch_par   Парикмахерская.число_ожидающих
        Всего_обслужено       : get_value   Парикмахерская.количество_обслуженных
        Пропускная_способность: get_value   Парикмахерская.количество_обслуженных / Time_now * 60
        Длительность_работы   : get_value   Time_now / 60
$End
```