Оглавление

Перечень сокращений

ИМ – Имитационное Моделирование

СДС – Сложная Дискретная Система

LALR – Look-Ahead Left to Right Rightmost derivation

1. Введение

Имитационное моделирование (ИМ) на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, морские порты, аэропорты, комплексы перекачки нефти и газа, ирригационные системы, программное обеспечение сложных систем управления, вычислительные сети и многие другие. Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

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

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

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

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском государственном техническом университете (МГТУ им.Н.Э. Баумана) на кафедре "Компьютерные системы автоматизации производства". Причинами ее проведения и создания РДО явились требования универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей, моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в реальном

масштабе времени) и ряд других, сформировавшихся у разработчиков при выполнении работ, связанных с системным анализом и организационным управлением сложными системами различной природы.

2. Предпроектное исследование

2.1. Основные положения языка РДО

Основные положения системы РДО могут быть сформулированы следующим образом[1]:

- Все элементы СДС представлены как ресурсы, описываемые некоторыми параметрами. Ресурсы могут быть разбиты на несколько типов; каждый ресурс определенного типа описывается одними и теми же 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 состояние ресурсов; действия ограничены во времени двумя событиями: событиями начала и событиями конца.

- Нерегулярные события описывают изменения 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 состояния ресурсов в начале и в конце соответствующего действия.

- Множество ресурсов R и множество операций O образуют модель СДС.

2.2. Описание ресурсов в языке РДО

Ресурсы определяют начальное состояние глобальной базы данных модели и описываются в отдельном объекте (с расширением .rss).

Объект ресурсов имеет следующий формат:

```
$Resources

     <описание_ресурса> { <описание_ресурса> }

$End
```

Описание каждого ресурса имеет следующий формат:

```
<имя_ресурса>:<имя_типа_ресурса> [trace/no_trace]
<начальные_значения_параметров>);
```

Имя типа ресурса - это имя одного из типов ресурсов, описанных в объекте типов.

Имя ресурса - это простое имя. Имена должны быть различными для всех ресурсов и не должны совпадать с предопределенными и ранее использованными именами.

Начальные значения параметров ресурса задают в позиционном соответствии с порядком следования параметров в описании типа. Значения задают целой или вещественной численной константой либо именем значения в соответствии с типом параметра. Для тех параметров, у которых при описании типа указано значение по умолчанию, вместо начального значения можно

указать символ "*". В этом случае параметр примет значение по умолчанию. Если для параметра задан диапазон возможных значений, то проверяется соответствие начального значения этому диапазону.

Для того, чтобы использовать неопределенное значение параметра, необходимо указать символ "#". В этом случае параметр будет задан как неопределенный (т.е его значение не является проинициализированным), и с ним нельзя будет работать до тех пор, пока он не будет явно проинициализирован.

Признак трассировки - это один из допустимых признаков трассировки (подробнее смотри описание трассировки).

Пример текущего синтаксиса описания ресурсов в языке РДО:

```
$Resources
     Ресурс_1 : Тип_1 0 5.25 100 0.0 Занят 20 10. Свободен Погрузка
Занят
     Ресурс_2 : Тип_1 trace * 5.25 * 0.0 * 10 10. * * *
     Ресурс_3 : Тип_1 no_trace * 5.25 * 0.0 * 10 10. * * *
$End
```

2.3. Синтаксический анализ[3]

Анализ исходной программы разбивает её на составные части и накладывает на них грамматическую структуру. Анализ исходный программы разбивается на две фазы: лексический и синтаксический анализ. Лексический анализатор читает поток символов, составляющих исходную программу, и группирует эти символы в значащие последовательности, называемые лексемами. Для каждой лексемы анализатор строит выходной токен, состоящий из имени токена и значения его атрибута. Этот токен передается на следующую фазу – синтаксический анализ, которая также называется разбором (parsing). Синтаксический анализатор использует первые компоненты токенов, полученные при лексическом анализе для создания древовидного промежуточного представления, которое описывает грамматическую структуру потока токенов.

Синтаксис языка программирования описывает корректный вид его программы, а семантика определяет смысл написанной на нем программы. Иерархическая структура множества конструкций языка программирования, т.е. его синтаксис естественным образом описывается грамматикой. Синтаксис большинства языков программирования описывается контекстно-свободной грамматикой или BNF (Backus-Naur Form – Форма Бэкуса-Наура). Контекстно-свободная грамматика имеет 4 компонента:

- множество терминальных символов - элементарных символов языка, определяемых грамматикой, т.е. токенов;
- множество нетерминальных символов – синтаксических переменных;
- множество продукций, каждая из которых состоит из нетерминала, называемого заголовком или левой частью продукции, стрелки и последовательности терминалов или нетерминалов, называемых телом или правой частью продукции. Продукции определяют один из возможных видов конструкции языка;
- стартовый (начальный) символ – специально указанный нетерминальный символ.

Грамматика выводит (порождает) строки, начиная со стартового символа и неоднократно замещая нетерминалы телом продукции этих нетерминалов. Строки токенов, порождаемые из стартового символа, образуют язык, определяемый грамматикой.

Таким образом, синтаксический анализ представляет собой выяснение для полученной строки терминалов способа её вывода из стартового символа грамматики.

Существует три основных типа синтаксических анализаторов: универсальные, восходящие и нисходящие. Наиболее эффективные нисходящие и восходящие методы работают только с определенными подклассами контекстно-свободных грамматик, однако некоторые из этих подклассов достаточно выразительны для описания большинства синтаксических конструкций языков программирования.

Грамматика языка РДО принадлежит подклассу LALR(1) грамматик - подклассу грамматик, синтаксический анализ которых может быть проведен LALR(1)-методами анализа. Это методы восходящего синтаксического анализа, в которых чтение входного потока производится слева направо, и которым достаточно одного предпросмотренного символа входного потока для принятия решения. LALR(1)-анализаторы работают по следующему принципу[6]:

Парсер (анализатор) состоит из конечного автомата со стеком и способен запоминать следующий входной символ (символ предпросмотра). Текущее состояние автомата всегда находится на вершине стека. Автомат способен осуществлять 4 действия: перенос (shift), свертка (reduce), принятие (accept) и ошибка (error). На основании своего текущего состояния парсер определяет, нужен ли ему символ предпросмотра и, если нужен, считывает его. Далее на основании своего текущего состояния и, если необходимо, символа предпросмотра, парсер совершает одно из возможных действий:

- В случае переноса, парсер кладет состояние, определяемое предпросмотренным символом, в стек, и оно становится текущим.

- Свертка осуществляется, когда парсер способен применить одно из грамматических правил и заменить правую часть продукции левой (если этому не противоречит символ предпросмотра). Для этого парсер снимает определенное количество состояний со стека (обычно их число равно числу символов в правой части продукции) и, кладет новое состояние, определяемое символом левой части продукции и открытым теперь состоянием стека, в стек, и оно становится текущим.

- Принятие означает, что парсер успешно закончил работу. Это происходит в том случае, если вся входная строка была прочитана и она порождается грамматикой языка.

- В состояние ошибка парсер переходит в том случае, когда не может продолжать работу, следуя грамматическим правилам.

В системе РДО для создания синтаксического анализатора использует генератор синтаксических анализаторов Bison, который преобразует описание контекстно-свободной LALR(1) грамматики в программу на языке C++ для разбора этой грамматики. Синтаксический анализатор, генерируемый Bison, работает по принципу «перенос/свертка» («shift/reduce»), описанному выше.

3. Формирование ТЗ

3.1. Введение

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

3.2. Общие сведения

Основание для разработки: задание на курсовой проект.

Заказчик: Кафедра «Компьютерные системы автоматизации производства» МГТУ им. Н.Э. Баумана

Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Богачев П.А.

Наименование темы разработки: «Разработка нового синтаксиса описания ресурсов в языке РДО»

3.3. Назначение разработки

Разработать и внедрить в текущую версию RAO-studio новый синтаксис описания ресурсов в языке РДО.

3.4. Требования к программе или программному изделию

3.4.1. Требования к функциональным характеристикам

- Модели в новом синтаксисе описания ресурсов должны компилироваться и запускаться в текущей версии RAO-studio;
- Должен быть разработан конвертер моделей для автоматизированного перевода ресурсов на новый синтаксис. Все сконвертированные модели должны компилироваться, запускаться и проходить тесты;
- Модели для автоматических тестов должны быть переведены на новый синтаксис. Модели должны проходить все тесты в новой версии RAO-studio;
- Документация по языку РДО должна соответствовать новому синтаксису ресурсов.

3.4.2. Требования к надежности

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

3.4.3. Условия эксплуатации

- Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости.

- Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

3.4.4. Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 256 Мб;
- объем жесткого диска не менее 20 Гб;
- микропроцессор с тактовой частотой не менее 400 МГц;
- монитор с разрешением от 800*600 и выше.

3.4.5. Требования к информационной и программной совместимости

Система должна работать под управлением следующих ОС: Windows 2000, Windows XP, Windows Vista, Windows 7.

3.4.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

3.4.7. Требования к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

3.5. Требования к программной документации

Программная документация должна быть выполнена в формате HTML и включена в состав документации RAO-studio. Разработанная документация должна стилистически соответствовать существующей документации RAO-studio.

3.6. Стадии и этапы разработки

Плановый срок начала разработки – 10 сентября 2013г.

Плановый срок окончания разработки – 13 декабря 2013г.

Этапы разработки:

- Концептуальный этап проектирования системы;
- Технический этап проектирования системы;
- Рабочий этап проектирования системы.

3.7. Порядок контроля и приемки

Контроль и приемка работоспособности системы осуществляются системой автоматического тестирования.

4. Концептуальный этап проектирования системы

Разработка формата нового синтаксиса ресурсов осуществлялась на основании моделей существующих объектно-ориентированных языков программирования. Было принято решение реализовать синтаксис описания ресурса в стиле вызова конструктора, т.е. создания объекта, а указание признака трассировки отделить от создания ресурса и реализовать его в стиле вызова метода для данного ресурса. В качестве моделей были выбраны следующие языки программирования: Java, Python, Go.

Язык Java был выбран как один из наиболее широко используемых на данный момент объектно-ориентированных языков программирования. Язык Python – как современный, активно развивающийся и также широко используемый. Язык Go – как новый и перспективный.

Были рассмотрены примеры создания объекта и вызова метода на этих языках и проведен анализ положительных и отрицательных сторон для всех примеров.

4.1. Рассмотрение синтаксиса языка Java

Синтаксис создания объектов и вызова методов на языке Java имеет следующий вид:

```
Class_1 object_1 = new Class_1(0, 5.25, param3_1, param4_1);
Class_1 object_2 = new Class_1(0, 5.00, param3_2, param4_2);
object_2.Method_1();
```

Положительные стороны:

- Подобный синтаксис является классическим, и, так как язык Java широко распространен, понятен большинству современных программистов;
- Тип ресурса указан в явном виде, что уменьшает вероятность ошибки в программе, так как проверка соответствия типа ресурса будет проводиться на этапе компиляции.

Отрицательные стороны:

- Синтаксис языка Java довольно громоздкий, что замедляет написание программ и ухудшает их внешний вид;
- Имеются ключевые слова (в данном случае “new”), без которых можно обойтись.

4.2. Рассмотрение синтаксиса языка Python

Синтаксис создания объектов и вызова методов на языке Python имеет следующий вид:

```
object_1 = Class_1(0, 5.25, param3_1, param4_1)
object_2 = Class_1(0, 5.00, param3_2, param4_2)
object_2.Method_1()
```

Положительные стороны:

- Язык Python также широко распространен и потому подобный синтаксис понятен большинству современных программистов;
- Синтаксис компактен и легко читается.

Отрицательные стороны:

- Язык Python использует пробельные символы для разделения программных блоков и выражений и потому требует строгого соблюдения стиля форматирования программы.

4.3. Рассмотрение синтаксиса языка Go

Синтаксис создания объектов и вызова методов на языке Go имеет следующий вид:

```
obj_1 := Struct_1{0, 5.25, param3_1, param4_1}
obj_2 := Struct_1{0, 5.00, param3_2, param4_2}
obj_2.func_1()
```

Положительные стороны:

- Язык Go имеет компактный и легко читаемый синтаксис

Отрицательные стороны:

- Подобный синтаксис может быть непривычен для программистов, не работавших до этого с языком Go.

4.4. Описание разработанного синтаксиса языка РДО

На основании анализа выбранных моделей был сделан следующий вывод: новый синтаксис языка РДО должен иметь вид, приближенный к синтаксису создания объектов и вызова методов на языке Python, однако разделение выражений будет осуществлено не пробельными символами, а точками с запятыми.

Ниже приведен пример того, как будет выглядеть новый синтаксис:

```
$Resources
     Ресурс_1 = Тип_1(0, 5.25, 100, 0.0, Занят, 20, 10., Свободен,
Погрузка, Занят);
     Ресурс_2 = Тип_1(*, 5.25, *, 0.0, *, 10, 10., *, *, *);
     Ресурс_2.trace();
     Ресурс_3 = Тип_1(*, 5.25, *, 0.0, *, 10, 10., *, *, *);
     Ресурс_3.no_trace();
$End
```

5. Технический этап проектирования системы

5.1. Формальное описание нового синтаксиса

Таким образом, исходя из сформированного на этапе концептуального проектирования примера, новый синтаксис должен иметь следующий формат.

Объект ресурсов имеет следующий формат:

```
$Resources
      <описание_ресурса> [ <вызов_метода_трассировки> ]
      { <описание_ресурса> [ <вызов_метода_трассировки> ] }
$End
```

описание_ресурса

Описание каждого ресурса имеет следующий формат:

```
<имя_ресурса> = <имя_типа_ресурса>(<начальные_значения_параметров>);
```

имя_типа_ресурса

Имя типа ресурса - это имя одного из типов ресурсов, описанных в объекте типов.

имя_ресурса

Имя ресурса - это простое имя. Имена должны быть различными для всех ресурсов и не должны совпадать с предопределенными и ранее использованными именами.

начальные_значения_параметров

Начальные значения параметров ресурса задают в позиционном соответствии с порядком следования параметров в описании типа и разделяются запятыми. Значения задают целой или вещественной численной константой либо именем значения в соответствии с типом параметра. Для тех параметров, у которых при описании типа указано значение по умолчанию, вместо начального значения можно указать символ "*". В этом случае параметр примет значение по умолчанию. Если для параметра задан диапазон возможных значений, то проверяется соответствие начального значения этому диапазону.

Для того, чтобы использовать неопределенное значение параметра, необходимо указать символ "#". В этом случае параметр будет задан как неопределенный (т.е его значение не является проинициализированным), и с ним нельзя будет работать до тех пор, пока он не будет явно проинициализирован.

вызов_метода_трассировки

Вызов метода трассировки может отсутствовать. В этом случае трассировка для ресурса по умолчанию отключена.

Вызов метода трассировки имеет следующий формат:

```
<имя_ресурса>.<признак_трассировки>();
```

имя_ресурса

Имя ресурса - это имя одного из уже описанных ресурсов, для которого вызывается метод.

признак_трассировки

Признак трассировки - это один из допустимых признаков трассировки (подробнее смотри описание трассировки).

Формальное описание нового синтаксиса было добавлено в документацию по языку РДО. Синтаксические диаграммы, описывающие новый синтаксис приведены на листе «Синтаксические диаграммы грамматики ресурсов».

5.2. Разработка грамматики, описывающей новый синтаксис

Поскольку синтаксический анализатор грамматики языка РДО формируется Bison, разработанная грамматика должна относиться к подклассу LALR(1)-грамматик. На данном этапе игнорируются обработка ошибок и семантические правила.

Ниже приведена разработанная грамматика в синтаксисе Bison:

```
rss_main
      : RDO_Resources rss_resource_list RDO_End
      ;

rss_resource_list
      : /* empty */
      | rss_resource_list rss_resource ';'
      ;

rss_resource
      : rss_res_init '(' rss_opt_value_list ')'
      | RDO_IDENTIF '.' rss_trace '(' ')'
      ;

rss_res_init
      : RDO_IDENTIF '=' RDO_IDENTIF
      ;

rss_trace
      : RDO_trace
      | RDO_no_trace
      ;

rss_opt_value_list
      : /* empty */
      | rss_value_list
      ;

rss_value_list
      : rss_value
      | rss_value_list ',' rss_value
      ;
```

Данная грамматика порождает все возможные строки корректного описания ресурсов в новом синтаксисе. Конечный автомат, построенный Bison на основании данной грамматики, приведен на листе «Диаграмма состояний синтаксического анализатора». Данный автомат принимает любую грамматически корректную строку, то есть любое корректное описание ресурсов. Дерево разбора, которое строит автомат при разборе входной строки, приведено на листе «Дерево разбора» для следующего описания ресурсов:

```
$Resources
     Resource_1 = Type_1(Free, 0, 0);
     Resource_1.trace();
$End
```

Примечание:

В языке РДО существует большое количество возможных типов параметров ресурса, и потому лексический анализатор формирует большое количество токенов (терминалов) для всех возможных типов параметров. Нетерминальный символ «rss_value» имеет следующие продукции:

```
rss_value
     : '*'
     | '#'
     | RDO_INT_CONST
     | RDO_REAL_CONST
     | RDO_BOOL_CONST
     | RDO_STRING_CONST
     | RDO_IDENTIF
     | param_array_value
     ;
```

Для упрощения графического изображения конечного автомата все возможные токены параметров ресурса были заменены на псевдотокен RDO_CONST_VALUE.

5.3. Разработка алгоритма работы конвертера

Конвертер в языке РДО работает по следующему принципу: он проводит синтаксический анализ программы в старом синтаксисе языка РДО. Текст программы представляет как некоторый документ, в который конвертер вносит изменения. Конвертер формирует обновления (update), которые представляют собой правила, по которым должен быть изменен исходный документ.

Работа обновлений основана на следующих принципах:

- Все обновления применяются только к исходному документу;
- При применении нескольких обновлений в одном и том же месте документа, они "суммируются";
- Обновления применяются в порядке их создания в конвертере.

Для преобразования вкладки .RSS программы в новый синтаксис конвертер должен выполнить следующие действия:

- Заменить знак двоеточия (":") после имени ресурса на знак равенства ("=");

- Заключить список параметров ресурса в скобки;
- Разделить параметры ресурса запятыми;
- Поставить в конце выражения для описания ресурса точку с запятой (";");
- Вынести задание признака трассировки в отдельную конструкцию;

Алгоритм работы конвертера приведен на листе «Диаграмма деятельности конвертера». Все действия по изменению исходного документа совершаются в момент свертки по тому или иному грамматическому правилу.

6. Рабочий этап проектирования системы

6.1. Реализация новой грамматики

Для реализации нового синтаксиса были внесены изменения в файлы грамматики генератора синтаксических анализаторов Bison. К разработанной в пункте 5.2 грамматике были добавлены семантические правила и дополнительные продукции для обработки ошибок.

```
rss_main
     : RDO_Resources rss_resource_list RDO_End
     | RDO_Resources rss_resource_list
     {
          PARSER->error().error(@2, "После описания всех ресурсов
                                ожидается ключевое слово $End");
     }
     ;

rss_resource_list
     : /* empty */
     | rss_resource_list rss_resource ';'
     | rss_resource_list rss_resource error
     {
          PARSER->error().error(@2, rdo::format("Пропущена ';'"));
     }
     ;

rss_resource
     : rss_res_init '(' rss_opt_value_list ')'
     {
          LPRDORSSResource pResource =
               PARSER->stack().pop<RDORSSResource>($1);
          ASSERT(pResource);

          if (!pResource->defined())
          {
               PARSER->error().error(@3, rdo::format(
                    "Заданы не все параметры ресурса: '%s'",
                    pResource->name().c_str()));
          }
          pResource->end();
     }
     | RDO_IDENTIF '.' rss_trace '(' ')'
     {
          LPRDOValue pName = PARSER->stack().pop<RDOValue>($1);
          ASSERT(pName);
          LPRDORSSResource pResource = PARSER->findRSSResource(
               pName->value().getIdentificator());
          if (!pResource)
```

```
            {
                  PARSER->error().error(@1, rdo::format(
                        "Ресурс '%s' не существует",
                        pName->value().getIdentificator().c_str()));
            }
            pResource->setTrace($3 != 0);
      }
      | error
      {
            PARSER->error().error(@1, rdo::format(
                  "Синтаксическая ошибка"));
      }
      ;

rss_res_init
      : RDO_IDENTIF '=' RDO_IDENTIF
      {
            LPRDOValue pName = PARSER->stack().pop<RDOValue>($1);
            LPRDOValue pType = PARSER->stack().pop<RDOValue>($3);
            ASSERT(pName);
            ASSERT(pType);

            LPRDORTPResType pResType = PARSER->findRTPResType(
                  pType->value().getIdentificator());
            if (!pResType)
            {
                  PARSER->error().error(@3, rdo::format(
                        "Неизвестный тип ресурса: %s",
                        pType->value().getIdentificator().c_str()));
            }
            LPRDORSSResource pResourceExist = PARSER->findRSSResource(
                  pName->value().getIdentificator());
            if (pResourceExist)
            {
                  PARSER->error().push_only(@1, rdo::format(
                        "Ресурс '%s' уже существует",
                        pName->value().getIdentificator().c_str()));
                  PARSER->error().push_only(
                        pResourceExist->src_info(),
                        "См. первое определение");
                  PARSER->error().push_done();
            }
            LPRDORTPResType pNameExist = PARSER->findRTPResType(
                        pName->value().getIdentificator());
            if (pNameExist)
            {
```

```
                PARSER->error().push_only(@1, rdo::format(
                    "Недопустимое имя ресурса: '%s'.
                    Данное имя уже зарезервировано ",
                    pName->value().getIdentificator().c_str()));
                PARSER->error().push_only(
                    pNameExist->src_info(),
                    "См. первое определение");
                PARSER->error().push_done();
            }
            LPRDORSSResource pResource = pResType->createRes(
                    PARSER,
                    pName->src_info());
            $$ = PARSER->stack().push(pResource);
        }
        ;

rss_trace
        : RDO_trace    {$$ = 1;}
        | RDO_no_trace {$$ = 0;}
        ;

rss_opt_value_list
        : /* empty */
        | rss_value_list
        ;

rss_value_list
        : rss_value
        | rss_value_list ',' rss_value
        ;
```

Список синтаксических ошибок, наличие которых проверяется в процессе синтаксического анализа:

- Отсутствие точки с запятой в конце корректного описания ресурса.
- Отсутствие ключевого слова `$End` после описания всех ресурсов

В случае любой другой синтаксической ошибки пользователю выдается сообщение «Синтаксическая ошибка» с выделением строки, в которой была совершена ошибка.

Список семантических ошибок, наличие которых проверяется в процессе синтаксического анализа:

- Задание недостаточного количества параметров ресурса.
- Повторное создание ресурса.
- Создание ресурса неопределенного в закладке RTP типа.

- Создание ресурса с именем, которое уже было использовано при объявлении типа ресурса в закладке RTP.

6.2. Реализация конвертера

Для конвертации описания ресурсов моделей в новый синтаксис, в файлы грамматики Bison конвертера были внесены следующие изменения (обработка ошибок и семантические действия, не являющиеся важными для конвертера опущены):

```
rss_main
	: /* empty */
	| rss_resources_begin rss_resources rss_resources_end
	;

rss_resources
	: /* empty */
	| rss_resources rss_res_descr
	;

rss_res_descr
	: rss_res_type rss_trace rss_values
	{
		LPRDORSSResource pResource =
			CONVERTER->stack().pop<RDORSSResource>($1);
		ASSERT(pResource);
		if (!pResource->defined())
		{
			CONVERTER->error().error(@3, rdo::format(
				"Заданы не все параметры ресурса: %s",
				pResource->name().c_str()));
		}
		pResource->setTrace($2 == 1);

		if ($2 != 2)
		{
			LPDocUpdate pTraceDelete =
				rdo::Factory<UpdateDelete>::create(
					@2.m_first_seek,
					@2.m_last_seek
			);
			ASSERT(pTraceDelete);
			CONVERTER->insertDocUpdate(pTraceDelete);
		}

		LPDocUpdate pLeftParenInsert =
			rdo::Factory<UpdateInsert>::create(
				@3.m_first_seek,
				"("
```

```
            );
            ASSERT(pLeftParenInsert);
            CONVERTER->insertDocUpdate(pLeftParenInsert);

            LPDocUpdate pRightParenSemicolonInsert =
                  rdo::Factory<UpdateInsert>::create(
                        @3.m_last_seek,
                        ");"
            );
            ASSERT(pRightParenSemicolonInsert);
            CONVERTER->insertDocUpdate(pRightParenSemicolonInsert);

            switch ($2)
            {
            case 0:
            {
                  LPDocUpdate pAddTrace =
                        rdo::Factory<UpdateInsert>::create(
                              @3.m_last_seek,
                              "\n\t" + pResource->name() + ".no_trace();"
                  );
                  ASSERT(pAddTrace);
                  CONVERTER->insertDocUpdate(pAddTrace);
                  break;
            }
            case 1:
            {
                  LPDocUpdate pAddTrace =
                        rdo::Factory<UpdateInsert>::create(
                              @3.m_last_seek,
                              "\n\t" + pResource->name() + ".trace();"
                  );
                  ASSERT(pAddTrace);
                  CONVERTER->insertDocUpdate(pAddTrace);
                  break;
            }
            default:
                  break;
            }
      }
      ;

rss_res_type
      : RDO_IDENTIF_COLON RDO_IDENTIF
      {
            LPRDOValue pName = CONVERTER->stack().pop<RDOValue>($1);
```

```
            LPRDOValue pType = CONVERTER->stack().pop<RDOValue>($2);
            LPRDORTPResType pResType =
                  CONVERTER->findRTPResType(
                        pType->value().getIdentificator());
            if (!pResType)
            {
                  CONVERTER->error().error(@2, rdo::format(
                        "Неизвестный тип ресурса: %s",
                        pType->value().getIdentificator().c_str()));
            }
            LPRDORSSResource pResourceExist =
                  CONVERTER->findRSSResource(
                        pName->value().getIdentificator());
            if (pResourceExist)
            {
                  CONVERTER->error().push_only(
                        pName->src_info(),
                        rdo::format(
                              "Ресурс '%s' уже существует",
                              pName->value().getIdentificator().c_str()));
                  CONVERTER->error().push_only(
                        pResourceExist->src_info(),
                        "См. первое определение");
                  CONVERTER->error().push_done();
            }
            LPRDORSSResource pResource =
                  rdo::Factory<RDORSSResource>::create(
                        CONVERTER, pName->src_info(), pResType);

            LPDocUpdate pColonReplace =
                  rdo::Factory<UpdateReplace>::create(
                        @1.m_last_seek - 1,
                        @1.m_last_seek,
                        " ="
            );
            ASSERT(pColonReplace);
            CONVERTER->insertDocUpdate(pColonReplace);

            $$ = CONVERTER->stack().push(pResource);
      }
      ;

rss_trace
      : /* empty */  {$$ = 2;}
      | RDO_trace    {$$ = 1;}
      | RDO_no_trace {$$ = 0;}
```

```
      ;

rss_values
      : /* empty */
      {
            $$ = 1;
      }
      | rss_values rss_value
      {
            if ($1 != 1)
            {
                  LPDocUpdate pCommaInsert =
                        rdo::Factory<UpdateInsert>::create(
                              @1.m_last_seek,
                              ","
                  );
                  ASSERT(pCommaInsert);
                  CONVERTER->insertDocUpdate(pCommaInsert);
            }
            $$ = 0;
      }
      ;
```

В ходе работы конвертера к исходному документу применяются следующие обновления:

- Обновление `pColonReplace` заменяет двоеточие между именем и типом ресурса на знак равенства.
- Обновление `pCommaInsert` вставляет запятую после предыдущего параметра, если он существует. Проверка на существование предыдущего параметра производится с помощью семантических значений токенов Bison.
- Обновление `pTraceDelete` удаляет описание признака трассировки после типа ресурса в том случае, если признак трассировки задан явно. Проверка на явное задание признака трассировки осуществляется с помощью семантических значений токенов Bison.
- Обновления `pLeftParenInsert` и `pRightParenSemicolonInsert` вставляют соответственно открывающую скобку перед списком параметров ресурса и закрывающую скобку и точку с запятой после списка параметров ресурса.
- Обновление `pAddTrace` добавляет выражение для задания трассировки ресурса после списка параметров ресурса. Это обновление должно создаваться после обновления `pRightParenSemicolonInsert` (см. 5.3). Проверка того, какой именно признак трассировки стоит задать осуществляется с помощью семантических значений токенов Bison.

7. Апробирование разработанной системы в модельных условиях

7.1. Апробирование работы конвертера

Для тестирования работы конвертера использовались шаблонные модели РДО, а также модели системы автоматического тестирования. Контрольной моделью служила модель Heidel, код вкладки RSS которой до конвертирования приведен в приложении 1. Результат конвертации приведен в приложении 2. Сконвертированная модель прошла как ручные, так и автоматические тесты.

7.2. Апробирование новой грамматики

Тестирование правильности реализации новой грамматики проводилось системой автоматического тестирования Jenkins. Система проверяет не только компилируемость и запускаемость моделей, но и сравнивает результаты их работы с эталонными. Все модели системы прошли тестирование.

8. Заключение

В рамках данного курсового проекты были получены следующие результаты:

- Проведено предпроектное исследование системы имитационного моделирования РДО.
- На этапе концептуального проектирования системы был выбран новый синтаксис ресурсов на основе синтаксиса современных языков программирования.
- На этапе технического проектирования было сформулировано описание нового синтаксиса ресурсов, которое представлено в виде синтаксических диаграмм, была разработана LALR(1)-грамматика нового синтаксиса, а также разработан алгоритм работы конвертера, позволяющего перевести уже существующие модели на новый синтаксис ресурсов.
- На этапе рабочего проектирования была написана часть программного кода файла грамматики Bison, отвечающая за описание ресурсов языка РДО.
- Было проведено ручное тестирование моделей в новом синтаксисе, а также ручное тестирование работы конвертера на моделях в старом синтаксисе. После чего модели для автоматического тестирования были переведены на новый синтаксис и стало возможным проведение автоматического тестирования. Результаты проведения тестирования позволяют сделать вывод об правильности работы нового синтаксического анализатора и конвертера.
- Все внесенные в систему изменения отражены в справочной информациипо системе РДО, что позволяет пользователям оперативно получить справку по всем изменениям в системе.

Список используемых источников

1. **Емельянов В.В., Ясиновский С.И.** Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО. - М.: "Анвик", 1998. - 427 с., ил. 136.

2. **Б. Страуструп.** Язык программирования C++. Специальное издание / пер. с англ. – М.: ООО «»Бином-Пресс, 2006. – 1104 с.: ил.

3. **Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д.** Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. - М. : ООО "И.Д.Вильямс", 2008. - 1184 с. : ил. - Парал. тит. англ.

4. **DeRemer, F.** Practical Translators for LR(k) Languages, Ph. D. thesis, MIT, Cambridge, MA, 1969.

5. **Charles Donnelly, Richard Stallman.** Bison. The YACC-compatible Parser Generator. [http://dinosaur.compilertools.net/bison/]

6. **C. Johnson.** Yacc: Yet Another Compiler-Compiler. [http://dinosaur.compilertools.net/yacc/]

Список использованного программного обеспечения

1. RAO-Studio;
2. ArgoUml v0.34;
3. Graphviz 2.34;
4. Inkscape 0.48.4;
5. Microsoft® Office Word 2007;
6. Microsoft® Visual Studio 2010 SP1.

Приложение 1 – Вкладка RSS модели «Heidel» в старом синтаксисе

```
$Resources
  Транспортер_1 : Транспортер  trace  Тр_1  5.0  0  1200  *  *  *  *  *  *  *  *  *
  Транспортер_2 : Транспортер  trace  Тр_2  3.0  0  1200  *  *  *  *  *  *  *  *  *
  Транспортер_3 : Транспортер  trace  Тр_3  5.0  0  1200  *  вперед  *  *  *  *  *  *
*
  Конвейер_1    : Конвейер  /*trace*/  Кн_1   8.3  25  0  2175  *  *  *  *
  Конвейер_2    : Конвейер  /*trace*/  Кн_2   8.3  25  0  2700  *  *  *  *
  Конвейер_3    : Конвейер  /*trace*/  Кн_3   8.3  25  0  1100  *  *  *  *
  Конвейер_4    : Конвейер  /*trace*/  Кн_4   8.3  25  0  1100  *  *  *  *
  Конвейер_5    : Конвейер  /*trace*/  Кн_5   8.3  25  0  5800  *  *  *  *
  Склад_1     : Склад      Ск_1    500  правая   2.5  *  *  *  *  *
  Склад_2     : Склад      Ск_2    500  левая    2.5  *  *  *  *  *
  Склад_3     : Склад      Ск_3    900  правая   3.0  *  *  *  *  *
  Склад_4     : Склад      Ск_4    900  левая    3.0  *  *  *  *  *
  Склад_5     : Склад      Ск_5   1400  правая   4.0  *  *  *  *  *
  Склад_6     : Склад      Ск_6   1400  левая    4.0  *  *  *  *  *
  Склад_7     : Склад      Ск_7   1900  правая   4.0  *  *  *  *  *
  Склад_8     : Склад      Ск_8   1900  левая    4.0  *  *  *  *  *
  Склад_9     : Склад      Ск_9   2500  правая   5.0  *  *  *  *  *
  Склад_10    : Склад      Ск_10  2500  левая    5.0  *  *  *  *  *
  Склад_11    : Склад      Ск_11  3100  правая   5.0  *  *  *  *  *
  Склад_12    : Склад      Ск_12  3100  левая    5.0  *  *  *  *  *
  Склад_13    : Склад      Ск_13  3700  правая   5.0  *  *  *  *  *
  Склад_14    : Склад      Ск_14  3700  левая    5.0  *  *  *  *  *
  Склад_15    : Склад      Ск_15  4400  правая   6.0  *  *  *  *  *
  Склад_16    : Склад      Ск_16  4400  левая    6.0  *  *  *  *  *
  Склад_17    : Склад      Ск_17  5100  правая   6.0  *  *  *  *  *
  Склад_18    : Склад      Ск_18  5100  левая    6.0  *  *  *  *  *
  Склад_19    : Склад      Ск_19  5800  правая   6.0  *  *  *  *  *
  Склад_20    : Склад      Ск_20  5800  левая    6.0  *  *  *  *  *
  Склад_21    : Склад      Ск_21  1600  правая   8.0  *  *  *  *  *
  Склад_22    : Склад      Ск_22  1600  левая    8.0  *  *  *  *  *
  Склад_23    : Склад      Ск_23  2700  правая  10.0  *  *  *  *  *
  Склад_24    : Склад      Ск_24  2700  левая   10.0  *  *  *  *  *
  Склад_25    : Склад      Ск_25  3900  правая  10.0  *  *  *  *  *
  Склад_26    : Склад      Ск_26  3900  левая   10.0  *  *  *  *  *
  Склад_27    : Склад      Ск_27  5300  правая  12.0  *  *  *  *  *
  Склад_28    : Склад      Ск_28  5300  левая   12.0  *  *  *  *  *
  Склад_29    : Склад      Ск_29   000  левая   12.0  *  *  *  *  *
  Склад_30    : Склад      Ск_30   000  левая   12.0  *  *  *  *  *
  Распилочная_машина_1      : Распилочная_машина  trace  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *  *
  Позиция_измерения_1 : Позиция_измерения  *  *  *  *  *

  Заказ_1       : Позиция_заказа trace  1  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_2       : Позиция_заказа trace  2  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_3       : Позиция_заказа trace  3  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_4       : Позиция_заказа trace  4  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_5       : Позиция_заказа trace  5  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
```

```
  Заказ_6       : Позиция_заказа trace  6  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_7       : Позиция_заказа trace  7  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_8       : Позиция_заказа trace  8  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_9       : Позиция_заказа trace  9  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_10      : Позиция_заказа trace 10  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_11      : Позиция_заказа trace 11  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_12      : Позиция_заказа trace 12  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_13      : Позиция_заказа trace 13  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_14      : Позиция_заказа trace 14  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_15      : Позиция_заказа trace 15  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *
  Заказ_16      : Позиция_заказа trace 16  1  *  0  0.0  *  _60x60  50  50  10  0.0
сосна  *  1  *  0  *  *  *

  Бревно_показать_1 : Бревно_показать trace  *  *  *  *  *
  Бревно_показать_2 : Бревно_показать trace  *  *  *  *  *
  Бревно_показать_3 : Бревно_показать trace  *  *  *  *  *
  Бревно_показать_4 : Бревно_показать trace  *  *  *  *  *
  Бревно_показать_5 : Бревно_показать trace  *  *  *  *  *
  Бревно_показать_6 : Бревно_показать trace  *  *  *  *  *
  Бревно_показать_7 : Бревно_показать trace  *  *  *  *  *

  Результат_раскроя_1   : Результат_раскроя trace  1  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *
  Результат_раскроя_2   : Результат_раскроя trace  2  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *
  Результат_раскроя_3   : Результат_раскроя trace  3  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *
  Результат_раскроя_4   : Результат_раскроя trace  4  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *
  Результат_раскроя_5   : Результат_раскроя trace  5  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *
  Результат_раскроя_6   : Результат_раскроя trace  6  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *
  Результат_раскроя_7   : Результат_раскроя trace  7  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *
$End
```

Приложение 2 – Вкладка RSS модели «Heidel» после конвертирования

```
$Resources
  Транспортер_1  = Транспортер  (  Тр_1,  5.0,  0,  1200,  *,  *,  *,  *,  *,  *,  *,
*,  *);
       Транспортер_1.trace();
  Транспортер_2  = Транспортер  (  Тр_2,  3.0,  0,  1200,  *,  *,  *,  *,  *,  *,  *,
*,  *);
       Транспортер_2.trace();
  Транспортер_3  = Транспортер  (  Тр_3,  5.0,  0,  1200,  *,  вперед,  *,  *,  *,  *,
*,  *,  *);
       Транспортер_3.trace();
  Конвейер_1     = Конвейер(  /*trace*/  Кн_1,   8.3,  25,  0,  2175,  *,  *,  *,  *);
  Конвейер_2     = Конвейер(  /*trace*/  Кн_2,   8.3,  25,  0,  2700,  *,  *,  *,  *);
  Конвейер_3     = Конвейер(  /*trace*/  Кн_3,   8.3,  25,  0,  1100,  *,  *,  *,  *);
  Конвейер_4     = Конвейер(  /*trace*/  Кн_4,   8.3,  25,  0,  1100,  *,  *,  *,  *);
  Конвейер_5     = Конвейер(  /*trace*/  Кн_5,   8.3,  25,  0,  5800,  *,  *,  *,  *);
  Склад_1      = Склад(      Ск_1,    500,  правая,   2.5,  *,  *,  *,  *,  *);
  Склад_2      = Склад(      Ск_2,    500,  левая,    2.5,  *,  *,  *,  *,  *);
  Склад_3      = Склад(      Ск_3,    900,  правая,   3.0,  *,  *,  *,  *,  *);
  Склад_4      = Склад(      Ск_4,    900,  левая,    3.0,  *,  *,  *,  *,  *);
  Склад_5      = Склад(      Ск_5,   1400,  правая,   4.0,  *,  *,  *,  *,  *);
  Склад_6      = Склад(      Ск_6,   1400,  левая,    4.0,  *,  *,  *,  *,  *);
  Склад_7      = Склад(      Ск_7,   1900,  правая,   4.0,  *,  *,  *,  *,  *);
  Склад_8      = Склад(      Ск_8,   1900,  левая,    4.0,  *,  *,  *,  *,  *);
  Склад_9      = Склад(      Ск_9,   2500,  правая,   5.0,  *,  *,  *,  *,  *);
  Склад_10     = Склад(      Ск_10,  2500,  левая,    5.0,  *,  *,  *,  *,  *);
  Склад_11     = Склад(      Ск_11,  3100,  правая,   5.0,  *,  *,  *,  *,  *);
  Склад_12     = Склад(      Ск_12,  3100,  левая,    5.0,  *,  *,  *,  *,  *);
  Склад_13     = Склад(      Ск_13,  3700,  правая,   5.0,  *,  *,  *,  *,  *);
  Склад_14     = Склад(      Ск_14,  3700,  левая,    5.0,  *,  *,  *,  *,  *);
  Склад_15     = Склад(      Ск_15,  4400,  правая,   6.0,  *,  *,  *,  *,  *);
  Склад_16     = Склад(      Ск_16,  4400,  левая,    6.0,  *,  *,  *,  *,  *);
  Склад_17     = Склад(      Ск_17,  5100,  правая,   6.0,  *,  *,  *,  *,  *);
  Склад_18     = Склад(      Ск_18,  5100,  левая,    6.0,  *,  *,  *,  *,  *);
  Склад_19     = Склад(      Ск_19,  5800,  правая,   6.0,  *,  *,  *,  *,  *);
  Склад_20     = Склад(      Ск_20,  5800,  левая,    6.0,  *,  *,  *,  *,  *);
  Склад_21     = Склад(      Ск_21,  1600,  правая,   8.0,  *,  *,  *,  *,  *);
  Склад_22     = Склад(      Ск_22,  1600,  левая,    8.0,  *,  *,  *,  *,  *);
  Склад_23     = Склад(      Ск_23,  2700,  правая,  10.0,  *,  *,  *,  *,  *);
  Склад_24     = Склад(      Ск_24,  2700,  левая,   10.0,  *,  *,  *,  *,  *);
  Склад_25     = Склад(      Ск_25,  3900,  правая,  10.0,  *,  *,  *,  *,  *);
  Склад_26     = Склад(      Ск_26,  3900,  левая,   10.0,  *,  *,  *,  *,  *);
  Склад_27     = Склад(      Ск_27,  5300,  правая,  12.0,  *,  *,  *,  *,  *);
  Склад_28     = Склад(      Ск_28,  5300,  левая,   12.0,  *,  *,  *,  *,  *);
  Склад_29     = Склад(      Ск_29,   000,  левая,   12.0,  *,  *,  *,  *,  *);
  Склад_30     = Склад(      Ск_30,   000,  левая,   12.0,  *,  *,  *,  *,  *);
  Распилочная_машина_1       = Распилочная_машина  (  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *,  *,  *,  *,  *,  *,  *);
       Распилочная_машина_1.trace();
  Позиция_измерения_1  = Позиция_измерения(  *,  *,  *,  *,  *);

  Заказ_1        = Позиция_заказа (  1,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
       Заказ_1.trace();
  Заказ_2        = Позиция_заказа (  2,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
```

```
      Заказ_2.trace();
  Заказ_3        = Позиция_заказа (  3,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_3.trace();
  Заказ_4        = Позиция_заказа (  4,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_4.trace();
  Заказ_5        = Позиция_заказа (  5,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_5.trace();
  Заказ_6        = Позиция_заказа (  6,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_6.trace();
  Заказ_7        = Позиция_заказа (  7,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_7.trace();
  Заказ_8        = Позиция_заказа (  8,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_8.trace();
  Заказ_9        = Позиция_заказа (  9,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_9.trace();
  Заказ_10       = Позиция_заказа ( 10,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_10.trace();
  Заказ_11       = Позиция_заказа ( 11,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_11.trace();
  Заказ_12       = Позиция_заказа ( 12,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_12.trace();
  Заказ_13       = Позиция_заказа ( 13,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_13.trace();
  Заказ_14       = Позиция_заказа ( 14,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_14.trace();
  Заказ_15       = Позиция_заказа ( 15,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_15.trace();
  Заказ_16       = Позиция_заказа ( 16,  1,  *,  0,  0.0,  *,  _60x60,  50,  50,  10,
0.0,  сосна,  *,  1,  *,  0,  *,  *,  *);
      Заказ_16.trace();

  Бревно_показать_1  = Бревно_показать (  *,  *,  *,  *,  *);
      Бревно_показать_1.trace();
  Бревно_показать_2  = Бревно_показать (  *,  *,  *,  *,  *);
      Бревно_показать_2.trace();
  Бревно_показать_3  = Бревно_показать (  *,  *,  *,  *,  *);
      Бревно_показать_3.trace();
  Бревно_показать_4  = Бревно_показать (  *,  *,  *,  *,  *);
      Бревно_показать_4.trace();
  Бревно_показать_5  = Бревно_показать (  *,  *,  *,  *,  *);
      Бревно_показать_5.trace();
  Бревно_показать_6  = Бревно_показать (  *,  *,  *,  *,  *);
      Бревно_показать_6.trace();
  Бревно_показать_7  = Бревно_показать (  *,  *,  *,  *,  *);
```

```
      Бревно_показать_7.trace();

  Результат_раскроя_1    = Результат_раскроя (  1,  *,  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *);
      Результат_раскроя_1.trace();
  Результат_раскроя_2    = Результат_раскроя (  2,  *,  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *);
      Результат_раскроя_2.trace();
  Результат_раскроя_3    = Результат_раскроя (  3,  *,  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *);
      Результат_раскроя_3.trace();
  Результат_раскроя_4    = Результат_раскроя (  4,  *,  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *);
      Результат_раскроя_4.trace();
  Результат_раскроя_5    = Результат_раскроя (  5,  *,  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *);
      Результат_раскроя_5.trace();
  Результат_раскроя_6    = Результат_раскроя (  6,  *,  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *);
      Результат_раскроя_6.trace();
  Результат_раскроя_7    = Результат_раскроя (  7,  *,  *,  *,  *,  *,  *,  *,  *,  *,
*,  *,  *,  *,  *,  *,  *,  *,  *,  *);
      Результат_раскроя_7.trace();
$End
```