Оглавление

Перечень сокращений

ИМ – Имитационное Моделирование

СДС – Сложная Дискретная Система

Терминология

Парсер – компонент rdo_parser системы.

Рантайм – компонент rdo_runtime системы.

Калк – асбстрактный вычислитель системы. Калки формируются в процессе компиляции модели, и выполняются в процессе её работы. Скомпилированная модель состоит из последовательности калков, так же как и скомпилированная программа состоит из ассемблеровских команд.

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 отоношению 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. Типы данных в языке РДО

Тип данных определяет множество значений, которые может принимать та или иная переменная, и те операции, которые можно к ним применять. Концепцию типов данных широко используют в современных языках программирования, с ее помощью можно абстрагироваться от физического представления данных, повысить наглядность и надежность программы. С каждой встречающейся в программе константой, функцией, последовательностью, параметром ресурса, параметром функции, параметром образца должен быть сопоставлен один и только один тип.

В данной версии РДО-языка определены следующие типы данных:

- целый тип - `integer`;

- вещественный тип - `real`;

- логический тип - `bool`;

- строковый тип – `string`;

- перечислимый тип;

- ссылка на один из выше определенных типов - `such_as`.

Для целых и вещественных типов возможно задание диапазона допустимых значений. Диапазон указывают за зарезервированным словом integer или real в квадратных скобках. Границы

диапазона представляют собой численные константы вещественного или целого типа, их разделяют двумя точками.

Примеры:

- `integer [1..100]`
- `real [0.0..50.7]`

Если диапазон указан, то при присвоении значения проверяется нахождение значения объекта в диапазоне допустимых, и в случае выхода за границы выдается собщение об ошибке как на этапе компиляции модели, так и во время прогона.

Перечислимые типы задают указанием всех возможных значений. Имена всех возможных значений перечислимого типа указывают в круглых скобках через запятую. Максимальное количество имен значений равно 256, значение перечислимого типа хранится как байт. Имя значения перечислимого типа - это простое имя.

Примеры:

- `(Свободен, Занят, Погрузка, Разгрузка)`
- `(Первый, Второй, Третий, Седьмой)`

Если несколько различных объектов программы имеют одинаковый тип, то нет необходимости повторять описание типа. Вместо этого можно воспользоваться ссылкой на ранее описанный тип. Ссылка имеет следующий формат:

```
such_as <имя_ранее_описанного_объекта>
```

Ссылки возможны на типы ранее описанных констант и параметров ресурсов, представленных в объекте типов ресурсов. Допустимы цепочные ссылки, т.е. ссылки на объект, тип которого также описан ссылкой.

Примеры:

- `wood_kind  : such_as a_trunk.wood_kind`
- `quality    : such_as a_trunk.quality`
- `diameter_b : such_as a_trunk.diameter_a`
- `diameter_e : such_as diameter_b`

В языке используют следующие соглашения о соответствии типов. Целый тип всегда соответствует целому, а вещественный - вещественному независимо от диапазона допустимых значений, если он задан. Перечислимые типы считаются соответствующими только если один из них описан ссылкой на тип другого либо если они оба описаны ссылкой на тип третьего.

2.3. 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> [ <вызов_метода_трассировки> ]
```

```
{ <описание_ресурса> [ <вызов_метода_трассировки> ] }

$End
```

Описание каждого ресурса имеет следующий формат:

```
<имя_ресурса> = <имя_типа_ресурса>(<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.

Примеры синтаксиса создания ресурсов:

```
$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
```

2.4. Основные компоненты системы rdo_studio

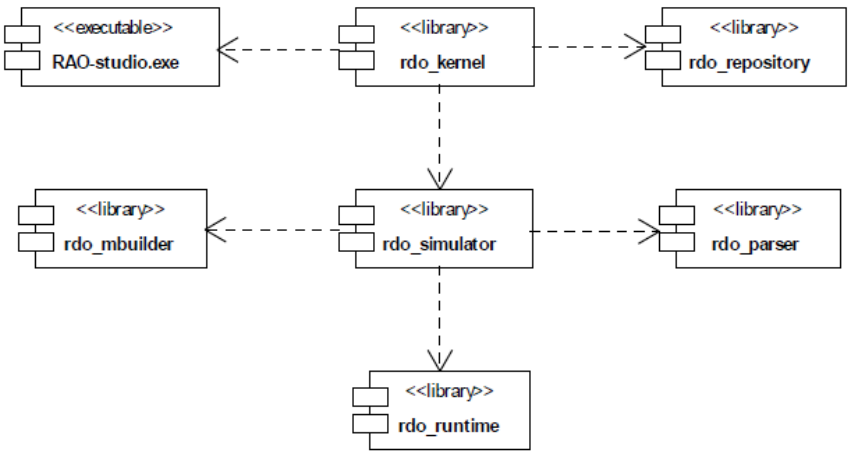

Рис. 1. Упрощенная диаграмма компонентов системы

Основные компоненты системы приведены на Рис. 1.

- **rdo_kernel** реализует ядровые функции системы.
- **RAO-studio.exe** реализует графический интерфейс пользователя.
- **rdo_repository** реализует управление потоками данных внутри системы и отвечает за хранение и получение информации о модели.
- **rdo_mbuilder** реализует функционал, используемый для программного управления типами ресурсов и ресурсами модели.
- **rdo_simulator** управляет процессом моделирования на всех его этапах. Он осуществляет координацию и управление компонентами rdo_runtime и rdo_parser.
- **rdo_parser** *(парсер)* производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО.
- **rdo_runtime** *(рантайм)* отвечает за непосредственное выполнение модели, управление базой данных и базой знаний.

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: «Вложенные ресурсы в языке РДО»

3.3. Назначение разработки

Разработать и внедрить в текущую версию RAO-studio возможность работы с вложенными ресурсами.

3.4. Требования к программе или программному изделию

3.4.1. Требования к функциональным характеристикам

- Должна быть реализована возможность полноценно работать с вложенными ресурсами, а именно, создавать их и работать с их параметрами;
- Допустимый уровень вложенности должен быть неограниченным;
- Должна быть написана тестовая модель для проверки нового функционала. Модель должна быть внедрена в систему автоматического тестирования;
- Документация по языку РДО должна быть обновлена в соответствии с нововведениями.

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 7, Ubuntu 14.04.

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 февраля 2014г.

Плановый срок окончания разработки – 31 мая 2014г.

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

На концептуальном этапе проектирования системы изначально необходимо было определить ключевую терминологию нового функционала. Далее было необходимо разработать ключевые требования и ограничения вложенных ресурсов, синтаксис их создания и синтаксис обращения к их параметрам.

4.1. Ключевая терминология

Ресурс, задаваемый в качестве параметра другого ресурса называется **вложенным**. Ресурс, по отношению к которому данный ресурс является вложенным, называется **родительским**.

4.2. Особенности работы с вложенными ресурсами

Основным принципом вложенных ресурсов является то, что они не являются самостоятельными сущностями, а существуют нераздельно от родительского ресурса. Из этого следуют следующие правила:

- Создание и удаление вложенных ресурсов осуществляется только совместно с созданием их родителя;
- Вложенный ресурс не может быть релевантным, т.к. в этом случае он становится независимым от родительского ресурса;
- Вложенный ресурс не имеет имени, т.к. обращение к нему осуществляется только как к параметру родительского ресурса;
- Вложенный ресурс может, в свою очередь, иметь другие вложенные ресурсы в качестве параметров. Допустиый уровень вложенности неограничен.

4.3. Синтаксис создания вложенных ресурсов

Т.к. вложенные ресурсы не могут создаваться отдельно от родительского, их создание осуществляется как определение значения соответствующего параметра родительского ресурса. В этом случае вместо начального значения параметра приводится конструктор вложенного ресурса с начальными значениями его параметров. Если вложенный ресурс тоже имеет ресурсы в качестве параметров, их создание описывается аналогично.

Ниже приведен пример создания ресурсов с параметрами различного уровня вложенности:

```
Ресурс_1 = Тип_1(*, 5.25, *, 0.0, *, 10, 10., *, *, *);
Ресурс_2 = Тип_2(0, 5.25);
Ресурс_3 = Тип_3(1, 0.0, Тип_2(0, 0.0), *);
Ресурс_4 = Тип_4(
      Тип_3(0, 1.0, Тип_2(0, 1.0), 5),
      Тип_2(0, 0.0)
);
```

4.4. Синтаксис обращения к параметрам произвольного уровня вложенности

В исходной версии языка РДО обращение к параметрам ресурса осуществлялось через оператор "точка" (".''). Для доступа к параметру произвольного уровня вложенности используется этот же оператор. В таком случае, оператор вызывается последовательно столько раз, сколько необходимо для перехода к параметру стандартного типа.

Пример:

```
$Results
      Показатель_1 : watch_state Ресурс_1.Параметр_2 == 0
      Показатель_2 : get_value Ресурс_3.Параметр_3.Параметр_1
      Показатель_3 : watch_par Ресурс_4.Параметр_1.Параметр_1
      Показатель_4 : watch_par
            Ресурс_4.Параметр_1.Параметр_4.Параметр_1
      Показатель_5 : get_value
            Ресурс_1.Параметр_2 + Ресурс_3.Параметр_3.Параметр_1
$End
```

5. Технический этап проектирования системы

5.1. Создание вложенных ресурсов

Для реализации описанных выше правил было необходимо изменить формальную грамматику вкладки RSS. Исходя из того, что допустимый уровень вложенности ресурсов должен быть неограниченным и того, что для создания вложенного ресурса в качестве параметра, указывается его конструктор, было принято решение выделить описание констуктора ресурса в отдельный нетерминальный символ `rss_constructor_call`, и добавить грамматическое правило, позволяющее использовать этот нетерминал на месте параметра ресурса.

Таким образом, измененная грамматика создания ресурса выглядит следующим образом (конструкции для трассировки опущены):

```
rss_resource
      : RDO_IDENTIF =  rss_constructor_call
      ;

rss_constructor_call
      : rss_constructor_name '(' rss_opt_value_list ')'
      ;

rss_constructor_name
      : RDO_IDENTIF
      ;

rss_opt_value_list
      : /* empty */
      | rss_value_list
      ;

rss_value_list
      : rss_value
      | rss_value_list ',' rss_value
      ;

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

Требуемая рекурсия обеспечивается за счет нового правила `rss_value : rss_constructor_call`.

5.2. Обращение к параметрам произвольной вложенности

При попытке совершить некоторое действие над некоторым параметром, определенным идентификатором, поиск разрешенных в данном месте модели идентификаторов производится через систему контекстов. Эта система во многом аналогична системе пространств имен, которая используемой во многих языках программировани. Контексты хранятся в стеке и поиск производится начиная с верхнего контекста в стеке. Если указанное действие над данным идентификатором не определено ни в одном из контекстов в стеке, пользователь получает сообщение об ошибке «неизвестный идентификатор». Поиск также возможно проводить и в контексте, не находящемся в стеке контекстов, в этом случае поиск производится только в данном контексте и ни в каком другом. Диаграмма классов, имеющих непосредственное отношение к системе контекстов приведена на листе «Диаграмма классов».

Было принято решение производить переход к параметрам произвольной вложенности по их идентификаторам, используя систему контекстов. Поиск первого параметра производится в контекстах стека контекстов начиная с верхнего. Если над идентификатором можно выполнить корректное действие, выполняется переключение на контест соответствующего объекта. При этом формируются необходимые для совершаемого действия калки. Для всех последующих параметров, обращение к которым производится через оператор «точка», поиск производится в найденном на предыдущем шаге контексте.

Чтобы обеспечить работу с неограниченной вложенностью необходимо рекурсивное грамматическое правило:

```
param_full_name
      : RDO_IDENTIF
      | param_full_name '.' RDO_IDENTIF
      ;
```

Новый универсальный нетерминал используется для большинства правил с использованием идентификаторов, которые необходимо искать в контексте. Чтобы избежать грамматических конфликтов, `param_full_name` так же используется в правилах, где не ожидается вложенных параметров, т.к. он работает и с одиночным идентификатором. Примеры правил с использованием `param_full_name` приведены ниже:

- `fun_arithm : param_full_name`
- `pmd_result : RDO_IDENTIF_COLON pmd_trace RDO_watch_par`
- `set_statement : param_full_name set_increment_or_decrement_type`
- `set_statement : param_full_name set_operation_type fun_arithm`
- `param_full_name '.' RDO_Size`
- `stopping_statement : param_full_name '.' RDO_Stopping '(' ')' ';'`

Таким образом обеспечивается возможность обращаться к параметрам произвольной вложенности в любом месте модели, где была возможность обращаться к обычным параметрам.

6. Рабочий этап проектирования системы

6.1. Создание вложенных ресурсов

Фактическое создание ресурса происходит в момент выполнения грамматического правила `rss_constructor_name : RDO_IDENTIF`, т.е. в момент описания имени типа ресурса. Далее в ресурс последовательно добавляются параметры с помощью метода `RDORSSResource::addParam(parser::RDOValue)`. Если в качестве параметра передается ресурс, то необходимо создать `RDOValue` на основании этого ресурса и его типа.

Для этого в грамматическом правиле `rss_value : rss_constructor_call` выполняется следующий код:

```
LPRDORSSResource pResource = PARSER->stack().pop<RDORSSResource>($1);
pResource->setIsNested(true);
rdo::runtime::RDOValue runtimeValue(
        pResource->getType(),
        pResource
);
LPTypeInfo pResourceTypeInfo = Factory<TypeInfo>::create(
        pResource->getType(),
        pResource->getType()->src_info()
);
LPRDOValue pValue = Factory<parser::RDOValue>::create(
        runtimeValue,
        pResource->src_info(),
        pResourceTypeInfo
);

PARSER->context()->cast<RDORSSResource>()->addParam(pValue);
```

В методе `addParam()` необходимо создать корректный параметр ресурса на основании переданной `RDOValue`. Созданные параметры хранятся в ресурсе парсера как умные указатели на `Expression`. `Expression` создается на основании калка (объекта, формирующего элементарную вычислительную операцию в рантайме) и типа возвращаемого значения. В случае параметра стандартного типа в параметр ресурса записывается калк `RDOCalcConst`, хранящий константное значение данного параметра, а в случае параметра типа ресурса - калк `RDOCalcCreateResource`, создающий ресурс рантайма и возвращающий `runtime::RDOValue` от этого ресурса и его типа. Проверка на то, является ли параметр вложенным ресурсом осуществляется на основании типа переданной `parser::RDOValue`.

Код, осуществляющий данную проверку и формирующий параметр ресурса, приведен ниже:

```
if (pAddParamValue->value().type().object_dynamic_cast
      <rdo::runtime::RDOResourceTypeList>()
)
{
      LPRDORSSResource pResourceValue =
            pAddParamValue->value().getPointerByType<RDORTPResType>();
      ASSERT(pResourceValue);

      pResourceValue->setSrcInfo(pParam->src_info());
      pResourceValue->setSrcText(
            std::to_string(getID()) + '_' + (*m_currParam)->name());

      rdo::runtime::LPRDOCalc pCalc = pResourceValue->createCalc();

      pAddParam = rdo::Factory<Expression>::create(
```

```
                  rdo::Factory<TypeInfo>::create(pAddParamValue->typeInfo()),
                  pCalc,
                  pAddParamValue->src_info()
      );
}
      else
            pAddParam = rdo::Factory<Expression>::create(
                  rdo::Factory<TypeInfo>::create(pAddParamValue->typeInfo()),
                  rdo::Factory<rdo::runtime::RDOCalcConst>::create(
                        pAddParamValue->value().clone()),
                        pAddParamValue->src_info()
            );
```

Здесь `pAddParamValue` это переданный в метод параметр после проверки на совпадение его типа с ожидаемым:

```
pAddParamValue = (*m_currParam)->getTypeInfo()->value_cast(pParam);
```

После формирования `Expression` параметр добавляется в список параметров ресурса:

```
m_paramList.push_back(Param(pAddParam));
m_currParam++;
```

Диаграмма деятельности, описывающая процесс создания вложенных ресурсов как параметров другого ресурса, приведена на листе "Диаграмма деятельности создания вложенных ресурсов".

6.2. Создание ресурсов рантайма

Вложенные ресурсы, как и обычные, регистрируются в базе данных модели при их описании во вкладке `RSS`. Создание ресурсов рантайма на основе загеристрированных ресурсов вкладки `RSS` осуществляется при запуске модели. При этом выполняются сформированные калки `RDOCreateResourceCalc`. Однако для вложенных ресурсов эти калки должны выполняться при определении параметров ресурса, родительского по отношению к ним. Чтобы не происходило повторного создания, ресурсам парсера был добавлен атрибут `bool isNested`, значение которого проверяется перед созданием ресурсов, описанных во вкладке `RSS`:

```
if (!rss->getIsNested())
{
      const std::vector<rdo::runtime::LPRDOCalc> calcList =
            rss->createCalcList();
      for (const rdo::runtime::LPRDOCalc& calc: calcList)
      {
            runtime()->addInitCalc(calc);
      }
}
```

Создание вложенных ресурсов осуществляется при выполнении калка `RDOCreateResourceCalc` родительского ресурса. В процессе создания родительского ресурса предварительно выполняются калки для подсчета начальных значений его параметров. В случае параметра типа ресурса выполняется калк создания этого ресурса `RDOCreateResourceCalc`. Таким образом выполняется рекурсивное создание всех вложенных ресурсов в данном родительском. При этом все вложенные ресурсы создаются перед созданием их родителя.

В процессе компиляции вкладки `RSS` модели ресурсы создаются в обратном порядке: сначала родительский, а затем вложенные. Поэтому необходимо было изменить схему задания ID ресурсам рантайма. В исходной версии РДО ID ресурсам рантайма задавались по порядку, что

обеспечивало соответствие ID ресурсов парсера и рантайма потому, что калки для создания ресурсов рантайма выполнялись в том же порядке, в каком создавались ресурсы парсера. В случае наличия вложенных ресурсов это неверно, и потому соответствие ID ресурсов обеспечивается явным образом. Если при формировании калка был задан ID ресурса парсера, то в конструктор ресурса рантайма передается именно он. В противном случае выполняется алгоритм поиска наименьшего незанятого на данный момент ID, и в конструктор ресурса рантайма передается найденное значение.

Код, выполняемый в калке `RDOCreateResourceCalc` приведен ниже:

```
RDOValue RDOCalcCreateResource::doCalc(const LPRDORuntime& pRuntime)
{
	const LPRDOResourceTypeList& resourceType =
		pRuntime->getResType(m_resourceTypeID);
	std::vector<RDOValue> paramValueList;

	for (const auto& calc : m_paramCalcList)
	{
		paramValueList.push_back(calc->calcValue(pRuntime));
	}

	const std::size_t resourceID = m_resourceID.is_initialized()
		? m_resourceID.get()
		: pRuntime->getResourceId();
	pRuntime->registerResourceId(resourceID);

	LPRDOResource pResource =
		resourceType.interface_cast<IResourceType>()->createRes(
			pRuntime,
			resourceID,
			paramValueList,
			m_traceFlag,
			m_permanentFlag
	);
	ASSERT(pResource);

	if (m_relResID != std::size_t(~0))
	{
		pRuntime->getCurrentActivity()->setRelRes(
			m_relResID, pResource->getTraceID()
		);
	}

	LPRDOType pType = resourceType;
	ASSERT(pType);
	return RDOValue(pType, pResource);
}
```

Диаграмма деятельности, описывающая процесс создания ресурсов рантайма, приведена на листе "Диаграмма деятельности создания ресурсов рантайма".

В случае, если родительский ресурс является временным, необходимо также удалить все вложенные в него ресурсы, при его удалении. Удаление вложенных ресурсов, также как и создание, выполняется рекурсивно. При этом удаление начинается с ресурсов наибольшей вложенности, а родительский ресурс удаляется последним. Для реализации рекурсивного удаления в ресурс рантайма был добавлен метод `RDOResource::onDestroy()`, который проверяет типа параметров ресуса и, если параметром является вложенный ресурс, рекурсивно вызывает метод `onDestroy()` для него.

Код метода `RDOResource::onDestroy()` приведен ниже:

```
void RDOResource::onDestroy(
      const LPRDORuntime& pRuntime,
      const LPRDOEraseResRelCalc& pCalc)
{
      for (auto& param: m_paramList)
      {
            if(param.type().object_dynamic_cast<RDOResourceTypeList>())
            {
                  LPRDOResource pNestedResource =
                        param.getPointerByType<RDOResourceTypeList>();
                  pNestedResource->onDestroy(pRuntime, pCalc);
            }
      }
      pRuntime->onEraseRes(getTraceID(), pCalc);
}
```

6.3. Работа с параметрами произвольной вложенности

При работе с правилами, в левой части которых имеется `param_full_name`, в качестве одного из параметров поиска в контексте используется ключевая строка `METHOD_OPERATOR_DOT`. Использование этой строки означает, что ожидается переключение на некоторый внутренний для данного контекст. Для последнего контекста выполняется поиск по ключевой строке `METHOD_GET` или `METHOD_SET` при выполнении соответствующего грамматического правила. Для этого в описанных выше правилах выполняется следующий код:

```
param_full_name
      : RDO_IDENTIF
      {
            LPRDOValue pName = PARSER->stack().pop<RDOValue>($1);
            Context::Params params;
            params[Context::Params::IDENTIFIER] =
                  pName->value().getIdentificator();
            Context::FindResult result =
                  PARSER->context()->find(
                        Context::METHOD_OPERATOR_DOT,
                        params,
                        pName->src_info()
                  );
            Context::LPFindResult pResult =
                  rdo::Factory<Context::FindResult>::create(result);
            $$ = PARSER->stack().push(pResult);
      }
      | param_full_name '.' RDO_IDENTIF
      {
            Context::LPFindResult pParentResult =
                  PARSER->stack().pop<Context::FindResult>($1);
            ASSERT(pParentResult->getSwitchContext());
            LPContext pParentContext =
                  pParentResult->getSwitchContext().context;
            LPRDOValue pName = PARSER->stack().pop<RDOValue>($3);

            Context::Params params =
                  pParentResult->getSwitchContext().params;
            params[Context::Params::IDENTIFIER] =
                  pName->value().getIdentificator();
            Context::FindResult result =
```

```
                pParentContext->find(
                      Context::METHOD_OPERATOR_DOT,
                      params,
                      pName->src_info()
                );
          Context::LPFindResult pResult =
                rdo::Factory<Context::FindResult>::create(result);

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

Помимо этого, необходимо было изменить код методов `onFindContext()` ключевых контекстов таким образом, чтобы для всех допустимых объектов, идентификатор которых может встретиться в данном контексте, было реализовано корректное переключение на его контекст при поиске по строке `METHOD_OPERATOR_DOT`.

Для параметров ресурса переключение производится на контекст типа параметра до тех пор, пока поиск не дойдет до параметра стандартного типа. В этом случае параметр является конечным, и переключение производится на контекст самого параметра. Информация о том, какому ресурсу принадлежит параметр, передается через структуру `Context::Params`. Для одновременной передачи через стек парсера и контекста, и структуры `Context::Params`, используется объект класса `FindResult`.

Фрагмент кода метода `onFindContext()` типа ресурса для ключевой строки `METHOD_OPERATOR_DOT` приведен ниже:

```
if (method == Context::METHOD_OPERATOR_DOT)
{
      const std::string paramName = params.identifier();

      const std::size_t parNumb = getRTPParamNumber(paramName);
      if (parNumb == RDORTPResType::UNDEFINED_PARAM)
            return FindResult();

      LPRDOParam pParam = findRTPParam(paramName);
      ASSERT(pParam);

      Context::Params params_;
      params_[RDORSSResource::GET_RESOURCE] =
            params.get<LPExpression>(RDORSSResource::GET_RESOURCE);
      params_[RDOParam::CONTEXT_PARAM_PARAM_ID] = parNumb;
      params_[Context::Params::IDENTIFIER] = paramName;

      LPRDORTPResType pParamType =
      pParam->getTypeInfo()->itype().object_dynamic_cast<RDORTPResType>();

      if (!pParamType)
            return FindResult(SwitchContext(pParam, params_));

      Context::FindResult result = pParam->find(
            Context::METHOD_GET, params_, srcInfo);
      LPExpression pNestedResource = result.getCreateExpression()();

      Context::Params params__;
      params__[RDORSSResource::GET_RESOURCE] = pNestedResource;

      return FindResult(SwitchContext(pParamType, params__));
}
```

Диаграмма деятельности, описывающая схему переключения контекстов в процессе работы с правилом `param_full_name` приведена на листе «Диаграмма деятельности переключения контекстов».

7. Апробирование разработанной системы в модельных условиях

7.1. Тестовая модель

Для тестирование корректности работы системы с вложенными ресурсами была написана тестовая модель. Смысловое содержание модели таково: в некотором магазине продаются ноутбуки. Клиенты приходят в магазин и подбирают себе ноутбук по параметрам его комплектующих: процессора, жесткого диска, оперативной памяти и т.п. Помимо этого ноутбуки имеют и собственные параметры: вес, размер экрана, цена и др. Клиент имеет в своем воображении некоторый идеальный ноутбук, который ему нужен, со своими характеристиками, некоторые из которых клиенту могут быть важны, а другие могут быть неважны. Если в магазине имеется ноутбук, параметры которого не хуже чем те, которые требуются клиенту, а также если у него достаточно денег, клиент его покупает. В противном случае клиент уходит.

Комплектующие ноутбука представляют собой вложенные ресурсы. Идеальные параметры ноутбука для клиента также организованы с использованием вложенных ресурсов. Таким образом в модели тестируются:

- Создание вложенных ресурсов во вкладке `RSS`;
- Создание временных вложенных ресурсов при создании их родителя;
- Удаление временных вложенных ресурсов при удалении их родителя;
- Использование вложенных ресурсов в различных типах образцов: `operation`, `rule` и `event`.
- Подбор релевантных ресурсов по параметрам вложенных ресурсов;
- Сбор различных видов показателей (`get_value`, `watch_par` и `watch_state`) по параметрам вложенных ресурсов.

Текст тестовой модели приведен в приложении.

7.2. Система автоматического тестирования

Для проверки корректности работы системы rdo_studio в автоматическом режиме используется система автоматического тестирования Jenkins. Система проверяет компилируемость и выполняемость моделей, сравнивает их результаты и файлы трассировки с эталонными.

Написанная для тестирования вложенных ресурсов модель была внедрена в систему автоматического тестирования. Модель успешно проходит автоматические тесты.

8. Заключение

В рамках данного курсового проекты были получены следующие результаты:

- Проведено предпроектное исследование системы имитационного моделирования РДО.
- На этапе концептуального проектирования системы были сформулированы ключевые особенности работы с вложенными ресурсами, разработан синтаксис их создания и работы с их параметрами.
- На этапе технического проектирования были разработаны алгоритмы создания вложенных ресурсов на этапе компиляции, создания ресурсов рантайма, имеющих вложенные ресурсы, и алгоритм работы с параметрами произвольной вложенности с использованием системы контекстов.
- На этапе рабочего проектирования был написан программный код, реализующих разработанные алгоритмы, а также проведен вспомогательный рефакторинг необходимых элементов программы.
- Было написана тестовая модель, использующая вложенные ресурсы. В модели ресурсы, имеющие вложенные в качестве параметров, создаются, удаляются и отбираются в паттернах по параметрам различного уровня вложенности; собираются показатели, зависящие от параметров различного уровня вложенности. Модель была внедрена в систему автоматического тестирования.
- 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. **Charles Donnelly, Richard Stallman.** Bison. The YACC-compatible Parser Generator. [http://dinosaur.compilertools.net/bison/]

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

Список использованного программного обеспечения

1. RAO-Studio;
2. ArgoUml v0.34;
3. UMLet 12.2;
4. Inkscape 0.48.4;
5. Microsoft® Office Word 2007;
6. NetBeans IDE 8.0.
7. Microsoft® Visual Studio 2012 SP1.

Приложение – Код модели для тестирования работы вложенных ресурсов

Вкладка RTP (типы ресурсов):

```
$Resource_type CPU_Type: permanent
$Parameters
       number_of_cores:  integer = 1
       clock_rate: integer = 0
       rating: real = 0
$End

$Resource_type RAM_Type: permanent
$Parameters
       memory_size: integer = 0
       clock_rate: integer = 0
       rating: real = 0
$End

$Resource_type Graphic_Card_Type: permanent
$Parameters
       memory_size: integer = 0
       memory_clock: integer = 0
       rating: real = 0
$End

$Resource_type HDD_Type: permanent
$Parameters
       memory_size: integer = 0
       rotational_frequency: integer = 0
       rating: real = 0
$End

$Resource_type Laptop: permanent
$Parameters
       CPU: CPU_Type
       RAM: RAM_Type
       Graphic_Card: Graphic_Card_Type
       HDD: HDD_Type
       screen_size: real
       weight: real
       price: integer = 0
       amount: integer = 0
       sold: integer = 0
       rating: real = 0
$End

$Resource_type Clients: temporary
$Parameters
       desired_Laptop : Laptop
       money_available: real = 0
       status: (Ready, Done)
$End
```

Вкладка RSS (ресурсы):

```
$Resources
```

```
	Laptop_1 = Laptop(
		CPU_Type(4, 3700, *),
		RAM_Type(4096, 1600, *),
		Graphic_Card_Type(2048, 2600, *),
		HDD_Type(500, 7200, *),
		15.6,
		2.1,
		1499,
		50,
		*,
		0
	);
	Laptop_2 = Laptop(
		CPU_Type(2, 2700, *),
		RAM_Type(2048, 1333, *),
		Graphic_Card_Type(1024, 1200, *),
		HDD_Type(500, 5400, *),
		17.3,
		1.5,
		799,
		50,
		*,
		0
	);
	Laptop_3 = Laptop(
		CPU_Type(2, 2300, *),
		RAM_Type(1024, 700, *),
		Graphic_Card_Type(512, 600, *),
		HDD_Type(320, 5400, *),
		15.6,
		1.8,
		499,
		50,
		*,
		0
	);
$End
```

Вкладка EVN (события):

```
$Pattern Client_coming : event trace
$Relevant_resources
	_Client: Clients Create
$Body
_Client:
	Convert_event
		Client_coming.planning( time_now + normal_sequence(20,5));
		money_available = Money_Available();
		desired_Laptop.weight = Choose_weight();
		desired_Laptop.screen_size = Choose_Screen_Size();
		desired_Laptop.RAM.memory_size = Choose_RAM_size();
		desired_Laptop.RAM.clock_rate = Choose_RAM_clock_rate();
		desired_Laptop.Graphic_Card.memory_size = Choose_GC_memory_size();
		desired_Laptop.Graphic_Card.memory_clock = Choose_GC_clock_rate();
		desired_Laptop.CPU.number_of_cores = Choose_CPU_number_of_cores();
		desired_Laptop.CPU.clock_rate = Choose_CPU_clock_rate();
```

```
            desired_Laptop.HDD.memory_size = Choose_HDD_memory_size();
            desired_Laptop.HDD.rotational_frequency =
Choose_HDD_rotational_frequency();
            status = Ready;
$End
```

Вкладка PAT (образцы):

```
$Pattern Choosing_laptop_pattern : operation trace
$Relevant_resources
      _Client : Clients Keep Erase
      _Chosen_laptop : Laptop Keep NoChange
$Time = exponential_sequence(20)
$Body
_Client:
      Choice from status == Ready
      Convert_begin
            status = Done;
_Chosen_laptop:
      Choice from _Client.money_available >= price
            and _Client.desired_Laptop.weight >= weight
            and _Client.desired_Laptop.screen_size <= screen_size
            and _lient.desired_Laptop.RAM.memory_size <= RAM.memory_size
            and _Client.desired_Laptop.RAM.clock_rate <= RAM.clock_rate
            and _Client.desired_Laptop.Graphic_Card.memory_size <=
                  Graphic_Card.memory_size
            and _Client.desired_Laptop.Graphic_Card.memory_clock <=
                  Graphic_Card.memory_clock
            and _Client.desired_Laptop.CPU.clock_rate <= CPU.clock_rate
            and _Client.desired_Laptop.CPU.number_of_cores <= CPU.number_of_cores
            and _Client.desired_Laptop.HDD.memory_size <= HDD.memory_size
            and _Client.desired_Laptop.HDD.rotational_frequency <=
                  HDD.rotational_frequency
            and amount > 0
            with_min price
      Convert_begin
            amount--;
            sold++;
            HDD.rating = Compute_HDD_rating(_Chosen_laptop, _Client);
            CPU.rating = Compute_CPU_rating(_Chosen_laptop, _Client);
            RAM.rating = Compute_RAM_rating(_Chosen_laptop, _Client);
            Graphic_Card.rating =
                  Compute_Graphic_Card_rating(_Chosen_laptop, _Client);
            rating = Compute_total_rating(_Chosen_laptop, _Client);
$End

$Pattern Client_leaving_pattern : rule trace
$Relevant_resources
      _Client: Clients Erase
$Body
_Client:
      Choice from status == Ready
$End
```

Вкладка DPT (точки принятия решений):

```
$Decision_point client_choosing: some
$Condition NoCheck
$Activities
	Choosing_laptop: Choosing_laptop_pattern
	Client_leaving: Client_leaving_pattern
$End
```

Вкладка FUN (функции):

```
$Sequence exponential_sequence : real
$Type = exponential 12345678
$End

$Sequence uniform_sequence : real
$Type = uniform 12345678
$End

$Sequence normal_sequence : real
$Type = normal 12345678
$End

$Function ifRequired : integer
$Type = algorithmic
$Parameters
$Body
	if (uniform_sequence(0,1) < 0.7)
		return 0;
	else
		return 1;
$End

$Function Money_Available : real
$Type = algorithmic
$Parameters
$Body
	return uniform_sequence(300, 2500);
$End

$Function Choose_weight: real
$Type = algorithmic
$Parameters
$Body
	return uniform_sequence(1, 5);
$End

$Function Choose_Screen_Size: real
$Type = algorithmic
$Parameters
$Body
	return uniform_sequence(8, 19);
$End

$Function Choose_RAM_size : real
$Type = algorithmic
$Parameters
$Body
```

```
	return ifRequired() * uniform_sequence(0, 4096);
$End

$Function Choose_RAM_clock_rate : real
$Type = algorithmic
$Parameters
$Body
	return ifRequired() * uniform_sequence(0, 1600);
$End

$Function Choose_GC_memory_size : real
$Type = algorithmic
$Parameters
$Body
	return ifRequired() * uniform_sequence(0, 2048);
$End

$Function Choose_GC_clock_rate : real
$Type = algorithmic
$Parameters
$Body
	return ifRequired() * ifRequired() * uniform_sequence(0, 2400);
$End

$Function Choose_CPU_clock_rate : real
$Type = algorithmic
$Parameters
$Body
	return ifRequired() * ifRequired() * uniform_sequence(0,3700);
$End

$Function Choose_CPU_number_of_cores : real
$Type = algorithmic
$Parameters
$Body
	return ifRequired() * ifRequired() * uniform_sequence(0,4);
$End

$Function Choose_HDD_memory_size : real
$Type = algorithmic
$Parameters
$Body
	return ifRequired() * uniform_sequence(0,500);
$End

$Function Choose_HDD_rotational_frequency : real
$Type = algorithmic
$Parameters
$Body
	return ifRequired() * uniform_sequence(0,7200);
$End

$Function Compute_HDD_rating : real
$Type = algorithmic
$Parameters
_Laptop : Laptop
_Client : Clients
```

```
$Body
	real ms = _Laptop.HDD.memory_size / (_Laptop.HDD.memory_size +
_Client.desired_Laptop.HDD.memory_size);
	real rf = _Laptop.HDD.rotational_frequency / (_Laptop.HDD.rotational_frequency +
_Client.desired_Laptop.HDD.rotational_frequency);
	return (ms + rf) / 2 * 5;
$End

$Function Compute_CPU_rating : real
$Type = algorithmic
$Parameters
_Laptop : Laptop
_Client : Clients
$Body
	real full_laptop_cl = _Laptop.CPU.number_of_cores * _Laptop.CPU.clock_rate;
	real full_desired_cl = _Client.desired_Laptop.CPU.number_of_cores *
_Client.desired_Laptop.CPU.clock_rate;
	return  full_laptop_cl / ( full_laptop_cl + full_desired_cl) * 5;
$End

$Function Compute_RAM_rating : real
$Type = algorithmic
$Parameters
_Laptop : Laptop
_Client : Clients
$Body
	real cl = _Laptop.RAM.clock_rate / (_Laptop.RAM.clock_rate +
_Client.desired_Laptop.RAM.clock_rate);
	real ms = _Laptop.RAM.memory_size / (_Laptop.RAM.memory_size +
_Client.desired_Laptop.RAM.memory_size);
	return (cl + ms) / 2 * 5;
$End

$Function  Compute_Graphic_Card_rating : real
$Type = algorithmic
$Parameters
_Laptop : Laptop
_Client : Clients
$Body
	real ms = _Laptop.Graphic_Card.memory_size / (_Laptop.Graphic_Card.memory_size +
_Client.desired_Laptop.Graphic_Card.memory_size);
	real mc = _Laptop.Graphic_Card.memory_clock / (_Laptop.Graphic_Card.memory_clock
+ _Client.desired_Laptop.Graphic_Card.memory_clock);
	return (ms + mc) / 2 * 5;
$End

$Function Compute_total_rating : real
$Type = algorithmic
$Parameters
_Laptop : Laptop
_Client : Clients
$Body
	return (_Laptop.CPU.rating +
			_Laptop.RAM.rating +
			_Laptop.Graphic_Card.rating +
			_Laptop.HDD.rating ) /4;
$End
```

Вкладка SMR (объект прогона):

```
Show_mode      = NoShow
Show_rate      = 3600.0

Client_coming.planning( time_now + normal_sequence(20,5) )

Terminate_if Time_now >= 3000
```

Вкладка PMD (сбор показателей):

```
$Results
       Rating_Laptop_1 : watch_par Laptop_1.rating
       Rating_Laptop_2 : watch_par Laptop_2.rating
       Rating_Laptop_3 : watch_par Laptop_3.rating
       HDD_Rating_Laptop_1 : watch_par Laptop_1.HDD.rating
       RAM_Rating_Laptop_2 : watch_par Laptop_2.RAM.rating
       CPU_Rating_Laptop_3 : watch_par Laptop_3.CPU.rating
       HDD_1_2_comparison : watch_state Laptop_1.HDD.rating > Laptop_2.HDD.rating
       CPU_2_3_comparison : watch_state Laptop_2.CPU.rating > Laptop_3.CPU.rating
       RAM_3_1_comparison : watch_state Laptop_3.RAM.rating > Laptop_1.RAM.rating
       Sold_Laptop_1 : get_value Laptop_1.sold
       Sold_Laptop_2 : get_value Laptop_2.sold
       Sold_Laptop_3 : get_value Laptop_3.sold
       Money_got_from_Laptop_1 : get_value Laptop_1.sold * Laptop_1.price
       Money_got_from_Laptop_2 : get_value Laptop_2.sold * Laptop_2.price
       Money_got_from_Laptop_3 : get_value Laptop_3.sold * Laptop_3.price
       Total_money_got : get_value Laptop_1.sold * Laptop_1.price +
             Laptop_2.sold * Laptop_2.price + Laptop_3.sold * Laptop_3.price
$End
```

Результаты моделирования:

```
Results_file   = nested.pmv    2014-Jun-02 00:45:11.317702
Run_file       = nested.smr
Model_name     = nested
Resource_file  = nested.rss

$Changes

$Status = NORMAL_TERMINATION
$Result_values         0         3025.14      1.007
  EventCount           217       0.0717322    215
  OperRuleCheckCounter 678       0.224122     673
  AExpCalcCounter      6830      2.25775      6782
  BExpCalcCounter      17602     5.81857      17479

Rating_Laptop_1                  Тип:   par    Посл.знач.:  4.73566      Ср.знач.:
       4.011605     Мин.знач.:   0      Макс.знач.:  5      Числ.наб.:   13
       Стд.откл.:   1.466860     К.вар.%:     53.636319    Медиана:     4.442646
Rating_Laptop_2                  Тип:   par    Посл.знач.:  4.69716      Ср.знач.:
       4.679063     Мин.знач.:   0      Макс.знач.:  5      Числ.наб.:   25
       Стд.откл.:   0.667855     К.вар.%:     9.532462     Медиана:     4.730019
```

```
Rating_Laptop_3                 Тип:   par    Посл.знач.:  4.95809      Ср.знач.:
      4.738119     Мин.знач.:   0      Макс.знач.:  5      Числ.наб.:   23
      Стд.откл.:   0.791084     К.вар.%:     13.208055    Медиана:     4.890141
HDD_Rating_Laptop_1             Тип:   par    Посл.знач.:  5      Ср.знач.:
      3.900568     Мин.знач.:   0      Макс.знач.:  5      Числ.наб.:   12
      Стд.откл.:   1.523913     К.вар.%:     59.537749    Медиана:     3.978257
RAM_Rating_Laptop_2             Тип:   par    Посл.знач.:  5      Ср.знач.:
      4.537144     Мин.знач.:   0      Макс.знач.:  5      Числ.наб.:   19
      Стд.откл.:   0.838443     К.вар.%:     15.494051    Медиана:     3.916874
CPU_Rating_Laptop_3             Тип:   par    Посл.знач.:  5      Ср.знач.:
      4.872463     Мин.знач.:   0      Макс.знач.:  5      Числ.наб.:   2
      Стд.откл.:   0.788300     К.вар.%:     12.753663    Медиана:     0.000000
HDD_1_2_comparison              Тип:   state  Посл.знач.:  TRUE   % соотв.:
      0.272913     Мин.длит.:   51.937607    Макс.длит.:  373.591302   Числ.наб.:   6
CPU_2_3_comparison              Тип:   state  Посл.знач.:  FALSE  % соотв.:
      0.007631     Мин.длит.:   23.085205    Макс.длит.:  23.085205    Числ.наб.:   1
RAM_3_1_comparison              Тип:   state  Посл.знач.:  FALSE  % соотв.:
      0.587354     Мин.длит.:   59.869935    Макс.длит.:  1600.699158  Числ.наб.:   3
Sold_Laptop_1                   Тип:   get_value    Значение:    12
Sold_Laptop_2                   Тип:   get_value    Значение:    24
Sold_Laptop_3                   Тип:   get_value    Значение:    28
Money_got_from_Laptop_1         Тип:   get_value    Значение:    17988
Money_got_from_Laptop_2         Тип:   get_value    Значение:    19176
Money_got_from_Laptop_3         Тип:   get_value    Значение:    13972
Total_money_got                 Тип:   get_value    Значение:    51136
```