Оглавление

[1. Перечень сокращений 7](#_Toc359407633)

[2. Введение 8](#_Toc359407634)

[3. Предпроектное исследование 10](#_Toc359407635)

[3.1. Основные положения языка РДО 10](#_Toc359407636)

[3.2. Взаимодействие с ресурсами в среде РДО 12](#_Toc359407637)

[3.3. Производительность РДО при работе с ресурсами 12](#_Toc359407638)

[3.4. Постановка задачи 13](#_Toc359407639)

[4. Формирование ТЗ 14](#_Toc359407640)

[4.1. Основания для разработки 14](#_Toc359407641)

[4.2. Общие сведения. 14](#_Toc359407642)

[4.3. Назначение и цели развития системы 14](#_Toc359407643)

[4.4. Характеристики объекта автоматизации 14](#_Toc359407644)

[4.5. Требования к системе 14](#_Toc359407645)

[4.5.1. Требования к функциональным характеристикам 14](#_Toc359407646)

[4.5.2. Требования к надежности 14](#_Toc359407647)

[4.5.3. Условия эксплуатации 14](#_Toc359407648)

[4.5.4. Требования к составу и параметрам технических средств 15](#_Toc359407649)

[4.5.5. Требования к информационной и программной совместимости 15](#_Toc359407650)

[4.5.6. Требования к маркировке и упаковке 15](#_Toc359407651)

[4.5.7. Требования к транспортированию и хранению 15](#_Toc359407652)

[4.5.8. Порядок контроля и приемки 15](#_Toc359407653)

[5. Концептуальный этап проектирования системы 16](#_Toc359407654)

[5.1. Диаграмма компонентов 16](#_Toc359407655)

[5.2. Новая структура хранения ресурсов 17](#_Toc359407656)

[6. Технический этап проектирования системы 19](#_Toc359407657)

[6.1. Разработка архитектуры компонента rdo\_runtime 19](#_Toc359407658)

[6.2. Оптимизации при работе с ресурсами 19](#_Toc359407659)

[7. Рабочий этап проектирования системы 20](#_Toc359407660)

[7.1. Изменения в пространстве имен rdo::runtime 20](#_Toc359407661)

[7.1.1. Объявление класса RDOResourceTypeList 20](#_Toc359407662)

[7.1.2. Изменение шаблонного класса RDOResourceTypeBase<T> 21](#_Toc359407663)

[7.1.3. Изменения в интерфейсе IResourceType 22](#_Toc359407664)

[7.1.4. Изменение класса RDORuntime 23](#_Toc359407665)

[7.2. Оптимизации при работе с ресурсами 24](#_Toc359407666)

[7.2.1. Изменения в вычислителе RDOSelectResourceByTypeCalc 24](#_Toc359407667)

[7.2.2. Изменения в вычислителе RDOFunCalcForAll 25](#_Toc359407668)

[7.2.3. Изменения в классе RDOPMDWatchQuant 26](#_Toc359407669)

[8. Методика тестирования производительности 28](#_Toc359407670)

[8.1. Имитационная модель для тестирования производительности 28](#_Toc359407671)

[8.2. Методика проведения экспериментов 29](#_Toc359407672)

[8.3. Таблица с результатами экспериментов 30](#_Toc359407673)

[8.4. Анализ результатов тестирования 31](#_Toc359407674)

[9. Влияние внесенных изменений на результаты моделирования 33](#_Toc359407675)

[10. Заключение 35](#_Toc359407676)

[11. Список используемых источников 36](#_Toc359407677)

[12. Приложение А. Код имитационной модели 37](#_Toc359407678)

[13. Приложение Б. Листинг классов типа ресурса](#_Toc358034477) 40

1. **Перечень сокращений**

РП – Рабочий Проект

ТЗ – Техническое Задание

ТП – Технический Проект

ИМ – Имитационное Моделирование, Имитационная Модель

СДС – Сложная Дискретная Система

ЭВМ - Электронная Вычислительная Машина

ОЗУ - Оперативное Запоминающее Устройство

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.

Интеллектуальное имитационное моделирование, 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.

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.

Разработка интеллектуальной среды имитационного моделирования РДО – «Ресурсы, Действия, Операции», выполнена в Московском Государственном Техническом Университете им. Н.Э. Баумана на кафедре "Компьютерные системы автоматизации производства". Причинами создания РДО явились требования к универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей, а также моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в реальном масштабе времени). Таким образом, среда РДО стала решением указанных выше проблем ИМ и обеспечила исследователя и проектировщика новыми возможностями.

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.

1. **Предпроектное исследование**
   1. **Основные положения языка РДО**

В основе системы РДО лежат следующие положения:

* Все элементы сложной дискретной системы (СДС) представлены как ресурсы, описываемые некоторыми параметрами.
* Состояние ресурса определяется вектором значений всех его параметров; состояние СДС – значением всех параметров всех ресурсов.
* Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояния ресурсов; действия ограничены во времени двумя событиями: событиями начала и конца.
* Нерегулярные события описывают изменение состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.
* Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения ресурсов в начале и конце соответствующего действия.

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

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

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

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

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

Объектами исходных данных являются:

* [типы ресурсов](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_base/rdo_base_res_types.htm) (с расширением .rtp);
* [ресурсы](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_base/rdo_base_resources.htm) (с расширением .rss);
* [события](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_pat_type/rdo_pat_type_event.htm) (с расширением .evn);
* [образцы активностей](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_pat.htm) (с расширением .pat);
* [точки принятия решений](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_dpt.htm) и [процессы обслуживания](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_proc.htm) (с расширением .dpt);
* [константы, функции и последовательности](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_const_fun_sec.htm) (с расширением .fun);
* [кадры анимации](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_frm.htm) (с расширением .frm);
* [требуемая статистика](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_pmd.htm) (с расширением .pmd);
* [прогон](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_smr.htm) (с расширением .smr);
* [процессы обслуживания](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_obj/rdo_obj_proc.htm) (с расширением .prc).

Объекты, создаваемые РДО-имитатором при выполнении прогона:

* [результаты](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_res/rdo_res_pmv.htm) (с расширением .pmv);
* [трассировка](http://rdo.rk9.bmstu.ru/help/help/rdo_lang_rus/html/rdo_res/rdo_res_trc.htm) (с расширением .trc). [2]
  1. **Взаимодействие с ресурсами в среде РДО**

В рамках задачи, поставленной в данной бакалаврской работе, рассматривается влияние структуры имитационной модели, а именно составляющих ее *ресурсов* на производительность, то есть на время, затрачиваемое на моделирование.

В среде РДО во время прогона имитационной модели ресурсы могут создаваться и удаляться по желанию разработчика модели, при этом их количество ничем ограничено.

*Примечание:* *в распространяемой через сайт кафедры РК9 пробной версии системы РДО присутствует ограничение на общее количество ресурсов в системе, равное 200. Исследование производительности, проводимое в рамках данной бакалаврской работы, подразумевает версию среды РДО, лишенную этого ограничения в силу необходимости использования большого количества ресурсов.*

Взаимодействие с ресурсами модели может осуществляться различными способами: через продукционные правила, путем отображения параметров в кадрах анимации и т.д. В моделях, описывающих сложные системы, таких взаимодействий может быть большое количество (в том числе из-за необходимости моделирования большого количества ресурсов), а скорость имитационного моделирования может быть весьма критичной.

* 1. **Производительность РДО при работе с ресурсами**

В результате изучения архитектуры системы, проводимого в ходе предпроектного исследования, был выявлен ее недостаток, способный при определенных условиях серьезно замедлить процесс моделирования, и также усложняющий процесс ее разработки. Этот недостаток вызван тем, что в среде РДО создаваемые в имитационной модели ресурсы хранятся общим списком, а каждая операция, связанная с доступом к определенному типу ресурсов, приводит к их перебору, и проверке каждого ресурса на соответствие типа запрашиваемому. В случае нахождения релевантных запросу ресурсов в конце списка такое поведение может серьезным образом замедлить процесс моделирования. На рис. 1 схематично изображена подобная структура списка ресурсов (в качестве релевантных ресурсов выступают прямоугольники с цифрой *2*).

**1**

**1**

**N** штук

**1**

**…**

**2**

**M** штук

**2**

**…**

Рис. 1. Пример «неудачной» структуры списка ресурсов

При такой структуре списка каждый запрос будет вызывать перебор ресурсов, причем использование необходимых ресурсов будет предваряться большим количеством проверок типа.

* 1. **Постановка задачи**

Таким образом, можно сделать вывод, что для решения задачи увеличения производительности имитационных моделей с большим количеством ресурсов разных типов необходимо изменить архитектуру системы таким образом, чтобы избежать лишних проверок типа и сразу использовать список релевантных ресурсов.

1. **Формирование ТЗ**
   1. **Основания для разработки**

Задание на дипломный проект.

* 1. **Общие сведения.**

Увеличить производительность в системе имитационного моделирования РДО за счет перепроектирования архитектуры системы.

* 1. **Назначение и цели развития системы**

Основная цель работы – увеличить производительность в системе имитационного моделирования РДО за счет перепроектирования архитектуры системы, а также исследовать влияние внесенных изменений на скорость моделирования.

* 1. **Характеристики объекта автоматизации**

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

* 1. **Требования к системе**
     1. **Требования к функциональным характеристикам**
* увеличение производительности имитационного моделирования;
* неизменность результатов моделирования после внесения изменений в систему.
  + 1. **Требования к надежности**

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

* + 1. **Условия эксплуатации**

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

* + 1. **Требования к составу и параметрам технических средств**

Программный продукт должен работать на компьютерах со следующими характеристиками:

* объем ОЗУ не менее 1 Гб;
* объем жесткого диска не менее 20 Гб;
* микропроцессор с тактовой частотой не менее 1 ГГц;
* монитор с разрешением от 800\*600 и выше;
  + 1. **Требования к информационной и программной совместимости**

Данная система должна работать под управлением операционных систем Windows XP, Windows 7.

* + 1. **Требования к маркировке и упаковке**

Не предъявляются.

* + 1. **Требования к транспортированию и хранению**

Не предъявляются.

* + 1. **Порядок контроля и приемки**

Контроль скорости моделирования проводится на тестовой имитационной модели.

1. **Концептуальный этап проектирования системы**
   1. **Диаграмма компонентов**

**N:\RDOKURSACH\Диаграммаразвертывания.emf**

Рис. 2. Упрощенная диаграмма компонентов

Базовый функционал представленных на диаграмме компонентов (Рис. 2):

**rdo\_kernel** реализует ядровые функции системы. Не изменяется при разработке системы.

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

**rdo\_repository** реализует управление потоками данных внутри системы и отвечает за хранение и получение информации о модели. Не изменяется при разработке системы.

**rdo\_mbuilder** реализует функционал, используемый для программного управления типами ресурсов и ресурсами модели. Не изменяется при разработке системы.

**rdo\_simulator** управляет процессом моделирования на всех его этапах. Он осуществляет координацию и управление компонентами rdo\_runtime и rdo\_parser. Не изменяется при разработке системы.

**rdo\_parser** производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО. Не изменяется при разработке системы.

**rdo\_runtime** отвечает за непосредственное выполнение модели, управление базой данных и базой знаний. Модернизируется при разработке системы.

Объекты компонента **rdo\_runtime** инициализируются при разборе исходного текста модели компонентом **rdo\_parser**.

В данной работе все модификации могут быть произведены лишь над компонентом **rdo\_runtime**. Такой подход позволит уменьшить количество ошибок и упростить конечную архитектуру системы.

* 1. **Новая структура хранения ресурсов**

В системе РДО на текущий момент уже существуют объекты типа ресурса (шаблонный класс RDOResourceTypeBase<T>), реализующие фабричный метод (порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса). Таким образом, эти объекты занимаются созданием экземпляров ресурсов во время моделирования, что наводит на мысль о том, что именно они могут быть носителями списка создаваемых ими же ресурсов.

Общий список ресурсов хранится в объекте, отвечающем за прогон модели – RDORuntime. Также этот объект предоставляет методы, позволяющие получить указатели на начало и конец общего списка ресурсов.

Таким образом, логичным преобразованием архитектуры системы станет вариант, в котором объект типа ресурса будет хранить список создаваемых им ресурсов, а в объекте моделирования RDORuntime вместо общего списка ресурсов появится список их типов и методы их добавления и доступа к ним.

В объекте типа ресурса необходимо реализовать методы получения указателей на начало и конец списка релевантных ресурсов.

Все методы в компонентах РДО, использующие механизм доступа к списку ресурсов, должны быть модифицированы с учетом изменившейся структуры получения доступа к указателям на начало и конец списка ресурсов. Также, в каждом из этих методов необходимо убрать проверку на соответствие типа за ее ненадобностью.

1. **Технический этап проектирования системы**
   1. **Разработка архитектуры компонента rdo\_runtime**

В компоненте необходимо создать подкласс шаблонного класса RDOResourceTypeBase<T>, обладающий списком ресурсов, а также реализовать в нем методы res\_begin() и res\_end(), возвращающие указатели на начало и конец списка ресурсов данного типа, и метод eraseRes(…), необходимый для реализации удаления временных ресурсов.

Данный класс унаследует все базовые классы шаблонного типа, в том числе интерфейс *IResourceType*, содержащий чисто виртуальные функции по работе с объектом типа ресурса, в том числе и те, которые необходимо реализовать в новом подклассе.

Функцию создания ресурсов createRes(…) в шаблонном классе необходимо модифицировать так, чтобы при ее вызове создаваемый ресурс добавлялся в список типа.

В объекте моделирования RDORuntime необходимо заменить общий список ресурсов m\_resourceListByTime списком типов ресурсов m\_resourceTypeList, и реализовать методы регистрации (добавления) типа ресурса и доступа к определенному типу по его идентификатору.

* 1. **Оптимизации при работе с ресурсами**

Изменения, связанные с удалением проверок типа, должны коснуться классов RDOSelectResourceByTypeCalc (вычислитель, обрабатывающий выбор ресурсов при применении продукционных правил), RDOFunCalcForAll (вычислитель, использующийся при групповых операциях с ресурсами), RDOPMDWatchQuant (класс, занимающийся трассировкой количества временных ресурсов).

1. **Рабочий этап проектирования системы** 
   1. **Изменения в пространстве имен rdo::runtime**
      1. **Объявление класса RDOResourceTypeList**

class RDOResourceTypeList

: public RDOType

, public IResourceType

, public RDORuntimeObject

, public RDOTraceableObject

{

friend class rdo::Factory<RDOResourceTypeList>;

public:

ResCIterator res\_begin() const;

ResCIterator res\_end () const;

void eraseRes(CREF(rdo::runtime::LPRDOResource) pResource);

protected:

RDOResourceTypeList(ruint number,

CREF(rdo::runtime::LPRDORuntime) pRuntime);

virtual ~RDOResourceTypeList();

typedef std::list<rdo::runtime::LPRDOResource> ResourceList;

ResourceList m\_resourceList;

};

Данный класс обладает методами res\_begin() и res\_end(), возвращающие указатели на начало и конец списка ресурсов данного типа, а также методом eraseRes(…) для удаления временных ресурсов.

В конструкторе данного класса происходит формирование умного указателя на тип ресурса и его дальнейшая передача в метод добавления типов ресурсов объекта RDORuntime:

RDOResourceTypeList::RDOResourceTypeList(ruint number, CREF(rdo::runtime::LPRDORuntime) pRuntime)

: RDOType (t\_pointer)

, RDOTraceableObject(false, number, rdo::toString(number + 1))

{

rdo::intrusive\_ptr<RDOResourceTypeList> pThis(this);

ASSERT(pThis);

pRuntime->addResType(pThis);

}

Листинг методов res\_begin(), res\_end() и eraseRes(…) не приводится из-за их тривиальности.

* + 1. **Изменение шаблонного класса RDOResourceTypeBase<T>**

Данный класс был переименован для лучшей читаемости кода, список его базовых классов был заменен единственным классом RDOResourceTypeList.

template <class T>

class RDOResourceTypeListT: public RDOResourceTypeList

{

DECLARE\_FACTORY(RDOResourceTypeListT<T>);

public:

typedef T value\_type;

private:

//! Конструктор

//! \param number - Целочисленный идентификатор

RDOResourceTypeListT(ruint number,

rdo::runtime::LPRDORuntime pRuntime);

virtual ~RDOResourceTypeListT();

DECLARE\_IResourceType;

};

Строчка DECLARE\_IResourceType является макроопределением и означает реализацию этим классом чисто виртуальной функции создания ресурса:

template <class T>

inline LPRDOResource RDOResourceTypeListT<T>::createRes(CREF(LPRDORuntime) pRuntime, ruint resID, CREF(std::vector<RDOValue>) paramsCalcs, rbool traceFlag, rbool permanentFlag)

{

rdo::intrusive\_ptr<RDOResourceTypeListT<T> > pThis(this);

ASSERT(pThis);

LPIResourceType pIResType = pThis.template interface\_cast<IResourceType>();

ASSERT(pIResType);

rdo::intrusive\_ptr<T> pResource = rdo::Factory<T>::create(pRuntime, paramsCalcs, pIResType, resID, this->getTraceID(), traceFlag, permanentFlag);

ASSERT(pResource);

m\_resourceList.push\_back(pResource);

pRuntime->insertNewResource(pResource);

return pResource;

}

Строка m\_resourceList.push\_back(pResource) обозначает добавление ресурса в список его типа. Перед ней происходит вызов фабрики для создания объекта ресурса.

* + 1. **Изменения в интерфейсе IResourceType**

Изменения в интерфейсе касаются введения новых чисто виртуальных функций по получению указателей на начало и конец списка ресурсов:

OBJECT\_INTERFACE(IResourceType)

{

DECLARE\_FACTORY(IResourceType);

public:

virtual rdo::runtime::LPRDOResource createRes(CREF(LPRDORuntime) pRuntime, ruint resID, CREF(std::vector<RDOValue>) paramsCalcs, rbool traceFlag, rbool permanentFlag) = 0;

virtual void eraseRes(CREF(rdo::runtime::LPRDOResource) pResource) = 0;

typedef std::list<rdo::runtime::LPRDOResource> ResList;

typedef ResList::const\_iterator ResCIterator;

virtual ResCIterator res\_begin() const = 0;

virtual ResCIterator res\_end() const = 0;

typedef RDOResource value\_type;

protected:

IResourceType() {}

virtual ~IResourceType() {}

};

* + 1. **Изменение класса RDORuntime**

Атрибут класса, хранящий список типов ресурсов:

...

typedef std::vector<LPRDOResource> ResourceListByID;

typedef std::vector<LPRDOResourceTypeList> ResourceTypeList;

...

ResourceTypeList m\_resourceTypeList;

Функция регистрации типа ресурса:

inline void RDORuntime::addResType(CREF(LPRDOResourceTypeList) pResType)

{

ASSERT(pResType);

ASSERT(m\_resourceTypeList.size() == pResType->getTraceID() - 1);

m\_resourceTypeList.push\_back(pResType);

}

В данном методе есть конструкция ASSERT (проверка на истинность), которая выдает ошибку при попытке добавить тип ресурса с неверным идентификатором (идентификатор добавляемого типа должен быть на 1 больше предыдушего).

Функция доступа к типу реурса:

inline CREF(LPRDOResourceTypeList) RDORuntime::getResType

(ruint number) const

{

return m\_resourceTypeList[number - 1];

}

* 1. **Оптимизации при работе с ресурсами**
     1. **Изменения в вычислителе RDOSelectResourceByTypeCalc**

RDOSelectResourceByTypeCalc - вычислитель, обрабатывающий подбор ресурсов по заданным параметрам при применении продукционных правил во время моделирования.

В исходной версии системы в коде вычислителя проверка типа ресурса производилась посредством функции checkType(…):

...

RDORuntime::ResCIterator end = pRuntime->res\_end();

for (RDORuntime::ResCIterator it = pRuntime->res\_begin();

it != end; it++)

{

if (\*it && (\*it)->checkType(m\_resTypeID))

{

ResourceID resID = (\*it)->getTraceID();

...

В измененной версии системы этот же код в функции выглядит следующим образом:

...

RDORuntime::ResCIterator end =

pRuntime->getResType(m\_resTypeID)->res\_end();

for (RDORuntime::ResCIterator it = pRuntime

->getResType(m\_resTypeID)->res\_begin(); it != end; it++)

{

ResourceID resID = (\*it)->getTraceID();

...

* + 1. **Изменения в вычислителе RDOFunCalcForAll**

AS IS:

...

RDORuntime::ResCIterator end = pRuntime->res\_end();

for (RDORuntime::ResCIterator it = pRuntime->res\_begin();

it != end && res; it++)

{

if (\*it == LPRDOResource(NULL))

continue;

if (!(\*it)->checkType(m\_nResType))

continue;

...

TO BE:

...

RDORuntime::ResCIterator end =

pRuntime->getResType(m\_nResType)->res\_end();

for (RDORuntime::ResCIterator it = pRuntime

->getResType(m\_nResType)->res\_begin(); it != end && res; it++)

{

if (\*it == LPRDOResource(NULL))

continue;

...

* + 1. **Изменения в классе RDOPMDWatchQuant**

AS IS:

ruint RDOPMDWatchQuant::calcCurrentQuant(CREF(LPRDORuntime) pRuntime) const

{

ruint newQuant = 0;

for (RDORuntime::ResCIterator it =

pRuntime->res\_begin(); it != pRuntime->res\_end(); ++it)

{

if (\*it == 0)

continue;

if (!(\*it)->checkType(m\_rtpID))

continue;

pRuntime->pushGroupFunc(\*it);

if (m\_pLogicCalc->calcValue(pRuntime).getAsBool())

{

newQuant++;

}

pRuntime->popGroupFunc();

}

return newQuant;

}

TO BE:

ruint RDOPMDWatchQuant::calcCurrentQuant(CREF(LPRDORuntime) pRuntime) const

{

ruint newQuant = 0;

RDORuntime::ResCIterator end = pRuntime->getResType(m\_rtpID)

->res\_end();

for (RDORuntime::ResCIterator it = pRuntime

->getResType(m\_rtpID)->res\_begin(); it != end; ++it)

{

if (\*it == 0)

continue;

pRuntime->pushGroupFunc(\*it);

if (m\_pLogicCalc->calcValue(pRuntime).getAsBool())

{

newQuant++;

}

pRuntime->popGroupFunc();

}

return newQuant;

}

1. **Методика тестирования производительности**
   1. **Имитационная модель для тестирования производительности**

Для тестирования производительности была написана синтетическая имитационная модель, состоящая из ресурсов 2х типов (Рис. 3).

**1**

**1**

**1**

**1**

**1**

**\*** **N**

**2**

**2**

**2**

**\*** **M**

Рис. 3. Ресурсы имитационной модели

1й тип – пустой ресурс (**N** штук), 2й тип – ресурс со счетчиком (**M** штук).

В модели описано одно продукционное правило, последовательно увеличивающее счетчик каждого ресурса типа 2 на единицу с **0** до **R**. Таким образом, в модели производится **M\*R** операций над ресурсами второго типа.

Ресурсы, в свою очередь, могут добавляться в систему в различной последовательности (Рис. 4):

Тип **1** перед типом **2** (Тип 2 в конце)

**1**

**1**

**N**

**1**

**…**

**2**

**M**

**2**

**…**

**1**

**1**

**1**

**…**

**2**

**2**

**…**

**N**

**M**

Тип **2** перед

типом **1**

(Тип 2 в начале)

Рис. 4. Два варианта размещения ресурсов

а) Добавление всех ресурсов 1го типа, затем добавление ресурсов типа 2

б) Добавление ресурсов второго типа перед ресурсами первого

* 1. **Методика проведения экспериментов**

Структура имитационной модели позволяет проводить эксперименты для четырех различных вариантов (2 варианта размещения ресурсов \* архитектура до и после внесения изменений):

* Исходная архитектура системы (“AS IS”), ресурсы типа 2 в начале
* Исходная архитектура системы (“AS IS”), ресурсы типа 2 в конце
* Конечная архитектура системы (“TO BE”), ресурсы типа 2 в начале
* Конечная архитектура системы (“TO BE”), ресурсы типа 2 в конце

Для тестирования производительности проводится три эксперимента:

1. Зависимость времени моделирования от количества ресурсов 1го типа:

N = 5 .. 1000; M = 5; R = 2000

1. Зависимость относительного прироста производительности системы (после внесения в нее изменений) от количества полезных ресурсов (типа 2):

N = 250; M = 5 .. 50; R = 1000

1. Зависимость относительного прироста производительности системы (после внесения в нее изменений) от количества операций над ресурсами:

N = 1000; M = 10; R = 100 .. 10000

Для каждого значения изменяемого параметра в каждом эксперименте записывается время моделирования в миллисекундах (среднее между **тремя** прогонами).

* 1. **Таблица с результатами экспериментов**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **M** | **N** | **R** | **M\*R** |  | AS IS (Тип 2 в конце) | TO BE (Тип 2 в конце) | AS IS (Тип 2 в начале) | TO BE (Тип 2 в начале) |
| время моделирования, мс | | | | | | | | |
| Эксперимент №1 | | | | | | | | |
| 5 | 5 | 2000 | 10000 |  | 2102 | 2095 | 2108 | 2092 |
| 5 | 10 | 2000 | 10000 |  | 2116 | 2101 | 2098 | 2106 |
| 5 | 25 | 2000 | 10000 |  | 2156 | 2109 | 2112 | 2115 |
| 5 | 50 | 2000 | 10000 |  | 2168 | 2127 | 2120 | 2112 |
| 5 | 100 | 2000 | 10000 |  | 2194 | 2120 | 2122 | 2124 |
| 5 | 250 | 2000 | 10000 |  | 2350 | 2122 | 2119 | 2129 |
| 5 | 500 | 2000 | 10000 |  | 2549 | 2125 | 2187 | 2143 |
| 5 | 1000 | 2000 | 10000 |  | 2965 | 2264 | 2266 | 2173 |
|  |  |  |  |  |  |  |  |  |
| Эксперимент №2 | | | | | | | | |
| 5 | 250 | 1000 | 5000 |  | 1199 | 1084 | 1087 | 1085 |
| 10 | 250 | 1000 | 10000 |  | 2341 | 2134 | 2144 | 2124 |
| 15 | 250 | 1000 | 15000 |  | 3522 | 3224 | 3220 | 3213 |
| 20 | 250 | 1000 | 20000 |  | 4682 | 4321 | 4339 | 4299 |
| 25 | 250 | 1000 | 25000 |  | 5933 | 5454 | 5454 | 5451 |
| 30 | 250 | 1000 | 30000 |  | 7179 | 6663 | 6644 | 6605 |
| 35 | 250 | 1000 | 35000 |  | 8427 | 7841 | 7892 | 7880 |
| 40 | 250 | 1000 | 40000 |  | 9716 | 9086 | 9117 | 9107 |
| 50 | 250 | 1000 | 50000 |  | 12369 | 11663 | 11680 | 11656 |
|  |  |  |  |  |  |  |  |  |
| Эксперимент №3 | | | | | | | | |
| 10 | 1000 | 100 | 1000 |  | 319 | 254 | 264 | 261 |
| 10 | 1000 | 250 | 2500 |  | 729 | 575 | 573 | 571 |
| 10 | 1000 | 500 | 5000 |  | 1373 | 1097 | 1109 | 1109 |
| 10 | 1000 | 1000 | 10000 |  | 2693 | 2129 | 2138 | 2120 |
| 10 | 1000 | 2500 | 25000 |  | 6770 | 5427 | 5466 | 5433 |
| 10 | 1000 | 5000 | 50000 |  | 13336 | 10650 | 10871 | 10720 |
| 10 | 1000 | 10000 | 100000 |  | 26893 | 21124 | 21437 | 21102 |

* 1. **Анализ результатов тестирования**

**Эксперимент 1:**

Для «быстрой» схемы размещения ресурсов в модели (тип 2 в начале) можно сделать вывод, что производительность системы после внесения в нее изменений находится на том же уровне, что и до. Если используется схема размещения ресурсов, в которой тип 2 находится в конце списка, то в исходной системе наблюдается серьезное падение производительности при большом количестве ресурсов. В системе с измененной архитектурой размещение ресурсов не оказывает влияния на время выполнения имитационной модели, производительность находится на уровне предыдущих вариантов размещением ресурсов типа 2 в начале.

**Эксперимент 2:**

С увеличением количества ресурсов типа 2 (размещение в конце) относительный прирост производительности в системе с измененной архитектурой по отношению к исходной системе имеет тенденцию к уменьшению, при размещении ресурсов типа 2 в начале производительность практически не отличается.

**Вывод 1:**

С учетом статистической погрешности в 1-2%, по результатам экспериментов 1 и 2 можно сделать следующие выводы:

При «неудачной» схеме размещения ресурсов в системе с исходной архитектурой может наблюдаться значительное падение производительности, зависящее от соотношения количества релевантных ресурсов к ресурсам остальных типов (большее количество нерелевантных ресурсов означает большее падение производительности, т.е. большее время моделирования).

**Эксперимент 3:**

С увеличением количества операций над релевантными ресурсами (тип 2 в конце) прирост производительности в системе с измененной архитектурой по отношению к исходной системе не изменяется и находится примерно на одном уровне. При размещении ресурсов типа 2 в начале производительность практически не отличается.

**Вывод 1:**

Прирост производительности в измененной системе не зависит от количества операций над ресурсами, а лишь от количества самих ресурсов в модели.

1. **Влияние внесенных изменений на результаты моделирования**

Одним из необходимых условий при внесении изменений в такой продукт, как среда имитационного моделирования, является неизменность результатов работы системы относительно этих изменений при одинаковых условиях.

В данном случае для такой проверки была использована модель простой многоканальной СМО с временными ресурсами – модель парикмахерской. В этой модели присутствуют все операции, так или иначе затронутые внесенными в систему изменениями (добавление и удаление ресурсов, применение продукционных правил).

Критерием неизменности результатов стали:

* Результаты работы имитационной модели, собираемые средой РДО по завершении процесса моделирования
* Файл трассировки (отслеживания значений переменных в модели в процессе ее выполнения)

Результаты, выдаваемые системой до и после внесения в нее изменений, были сравнены визуально и оказались полностью идентичными.

Занятость\_парикмахера\_1 Тип: state Посл.знач.: FALSE % соотв.: 0.128582 Мин.длит.: 20.021139 Макс.длит.: 22.963191 Числ.наб.: 30

Занятость\_парикмахера\_2 Тип: state Посл.знач.: TRUE % соотв.: 0.609160 Мин.длит.: 29.277054 Макс.длит.: 69.203516 Числ.наб.: 62

Занятость\_парикмахера\_3 Тип: state Посл.знач.: FALSE % соотв.: 0.577632 Мин.длит.: 35.046200 Макс.длит.: 59.993740 Числ.наб.: 61

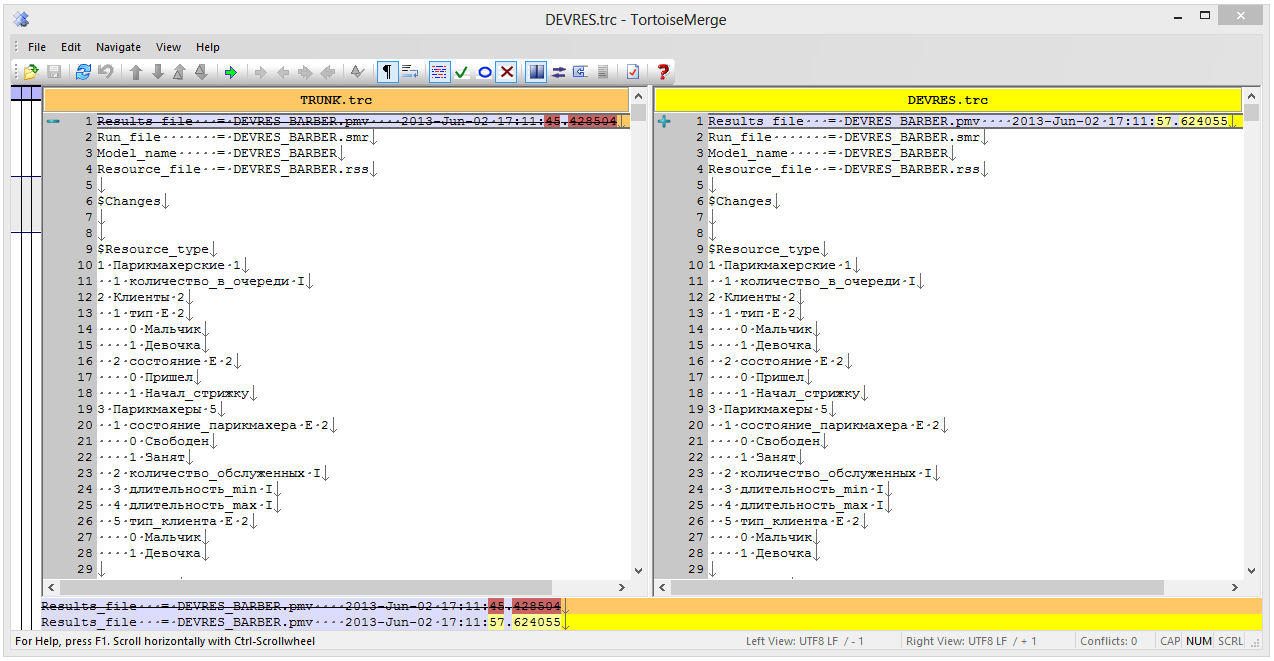
Обслужено\_парикмахером\_1 Тип: **get\_value** Значение: 30

Обслужено\_парикмахером\_2 Тип: **get\_value** Значение: 61

Обслужено\_парикмахером\_3 Тип: **get\_value** Значение: 61

Длина\_очереди Тип: par Посл.знач.: 0 Ср.знач.: 0.585730 Мин.знач.: 0 Макс.знач.: 5 Числ.наб.: 307 Стд.откл.: 1.045320 К.вар.%: 186.552399 Медиана: 0.000052

Для сравнения файлов трассировки была использована утилита TortoiseMerge, входящая в комплект TortoiseSVN, клиента для системы контроля версий Subversion. Снимок экрана представлен на рис. 5.



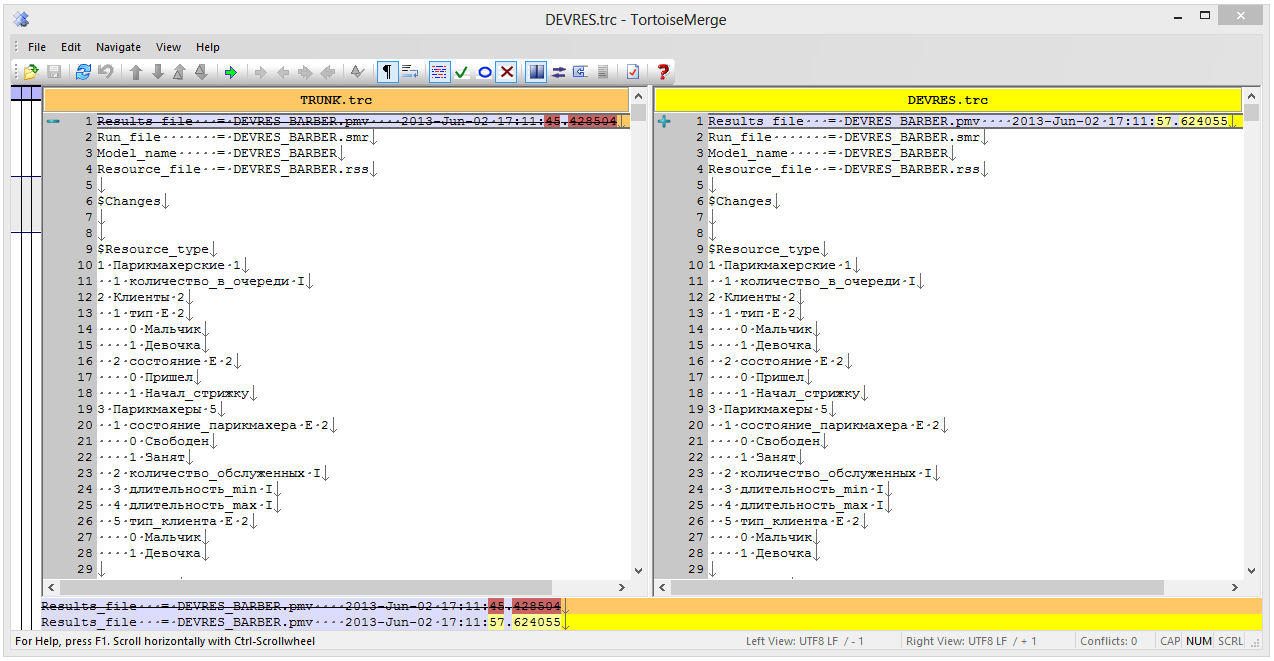


Рис. 5. TortoiseMerge – сравнение файлов трассировки

По рисунку видно, что единственным отличием этих файлов друг от друга является время проведения моделирования.

Таким образом, можно сделать вывод, что внесенные в систему изменения *не оказали влияния* на ее поведение и результаты работы имитационных моделей.

1. **Заключение**

В рамках данной бакалаврской работы были получены следующие результаты:

* 1. Проведено предпроектное исследование системы имитационного моделирования РДО. Определены основные направления, по которым производилось проектирование системы.
  2. На этапе концептуального проектирования системы выделены подзадачи, необходимые для выполнения основного задания. Составлено дерево целей.
  3. На этапе технического проектирования проработана итоговая архитектура системы и изображена с помощью диаграммы классов нотации UML.
  4. На этапе рабочего проектирования написан программный код для реализации спроектированных ранее алгоритмов работы и архитектуры компонента rdo\_runtime системы РДО. Проведены отладка и тестирование функционала системы, в ходе которых исправлялись найденные ошибки.
  5. Для анализа производительности была написана имитационная модель, получены результаты экспериментов и произведен их анализ.
  6. На примере простой имитационной модели была проверена неизменность результатов работы системы относительно внесенных в нее изменений.

1. **С****писок используемых источников**
2. Справка по языку РДО [<http://rdo.rk9.bmstu.ru/help/>];
3. Справка по RAO-studio [<http://rdo.rk9.bmstu.ru/help/>];
4. Емельянов В. В., Ясиновский С. И. Имитационное моделирование систем: Учеб. Пособие – М.: Издательство МГТУ им. Н. Э. Баумана, 2009. – 584 с.: ил. (Информатика в техническом университете);
5. Мартин Р. Чистый код. Создание, анализ и рефакторинг / пер. с анг. Е. Матвеев – CПб.: Питер, 2010. – 464 стр.;
6. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78;
7. Б. Страуструп Язык программирования С++. Специальное издание / пер. с англ. – М.: ООО «»Бином-Пресс, 2006. – 1104 с.: ил.;
8. Леоненков. Самоучитель по UML [http://khpi-iip.mipk.kharkiv.edu/library/case/leon/index.html];
9. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-90. Условные обозначения и правила выполнения.

**Список использованного программного обеспечения**

* + - 1. RAO-Studio;
      2. Autodesk AutoCAD 2012;
      3. Microsoft Office Word 2010;
      4. Microsoft Office PowerPoint 2010;
      5. Microsoft Office Visio 2010;
      6. Microsoft Visual Studio 2008.
      7. Inkscape 0.48.2;
      8. TortoiseSVN;

1. **Приложение А. Код имитационной модели**

**Вкладка RTP**

$Resource\_type Тип1 : temporary

$Parameters

номер : integer

$End

$Resource\_type Тип2 : temporary

$Parameters

номер : integer

параметр : integer = 0

занят : bool = false

$End

$Resource\_type Системы: permanent

$Parameters

счетчик : integer

счетчик2 : integer

счетчик3 : integer

$End

**Вкладка RSS**

**$Resources**

Система: Системы 0 0 0

**$End**

**Вкладка EVN**

**$Pattern** Создание\_Тип1 : **event**

**$Relevant\_resources**

\_Система : Система **Keep**

\_Тип1 : Тип1 **Create**

**$Body**

\_Система

**Convert\_event**

счетчик++;

**if** (\_Система.счетчик < 250){

Создание\_Тип1.**planning**( **time\_now** + 1);

}

\_Тип1

**Convert\_event**

номер = \_Система.счетчик;

**$End**

**$Pattern** Создание\_Тип2 : **event**

**$Relevant\_resources**

\_Система : Система **Keep**

\_Тип2 : Тип2 **Create**

**$Body**

\_Система

**Convert\_event**

счетчик2++;

**if** (\_Система.счетчик2 < 10){

Создание\_Тип2.**planning**( **time\_now** + 1);

}

\_Тип2

**Convert\_event**

номер = \_Система.счетчик2;

параметр = 0;

**$End**

**Вкладка PAT**

**$Pattern** Обработка\_Тип2 : **operation**

**$Relevant\_resources**

\_Тип2 : Тип2 **Keep** **Keep**

\_Система : Система **NoChange** **Keep**

**$Time** = 1

**$Body**

\_Тип2

**Choice** **from** \_Тип2.параметр < 100 **and** \_Тип2.занят == **false**

**Convert\_begin**

занят = **true**;

**Convert\_end**

занят = **false**;

параметр++;

\_Система

**Choice** **from** \_Система.счетчик2 == 10 **and** \_Система.счетчик == 250

**Convert\_end**

**if** (\_Тип2.параметр == 100)

счетчик3++;

**$End**

**Вкладка DPT**

**$Decision\_point** model: **some**

**$Condition** **NoCheck**

**$Activities**

ОТ2: Обработка\_Тип2

**$End**

**Вкладка SMR**

**Show\_mode** = **NoShow**

**Show\_rate** = 3600.0

Создание\_Тип1.**planning**( **time\_now** + 1 )

Создание\_Тип2.**planning**( **time\_now** + 251 )

**Terminate\_if** Система.счетчик3 == 10

1. **Приложение Б. Листинг классов типа ресурса**

**rdo\_res\_type\_i.h**

#ifndef \_LIB\_RUNTIME\_RES\_TYPE\_I\_H\_

#define \_LIB\_RUNTIME\_RES\_TYPE\_I\_H\_

// -------------------------------------------- INCLUDES

// -------------------------------------------- SYNOPSIS

#include "utils/smart\_ptr/interface\_ptr.h"

#include "simulator/runtime/rdo\_value.h"

// -----------------------------------------------------

OPEN\_RDO\_RUNTIME\_NAMESPACE

PREDECLARE\_POINTER(RDORuntime );

PREDECLARE\_POINTER(RDOResource);

/\*!

\interface IResourceType

\brief Предоставляет фабричный метод createRes()

\*/

OBJECT\_INTERFACE(IResourceType)

{

DECLARE\_FACTORY(IResourceType);

public:

virtual rdo::runtime::LPRDOResource createRes(CREF(LPRDORuntime) pRuntime, ruint resID, CREF(std::vector<RDOValue>) paramsCalcs, rbool traceFlag, rbool permanentFlag) = 0;

virtual void eraseRes(CREF(rdo::runtime::LPRDOResource) pResource) = 0;

typedef std::list<rdo::runtime::LPRDOResource> ResList;

typedef ResList::const\_iterator ResCIterator;

virtual ResCIterator res\_begin() const = 0;

virtual ResCIterator res\_end() const = 0;

typedef RDOResource value\_type;

protected:

IResourceType() {}

virtual ~IResourceType() {}

};

#define DECLARE\_IResourceType \

rdo::runtime::LPRDOResource createRes(CREF(LPRDORuntime) pRuntime, ruint resID, CREF(std::vector<RDOValue>) paramsCalcs, rbool traceFlag, rbool permanentFlag);

CLOSE\_RDO\_RUNTIME\_NAMESPACE

#endif // \_LIB\_RUNTIME\_RES\_TYPE\_I\_H\_

**rdo\_res\_type.h**

#ifndef \_LIB\_RUNTIME\_RES\_TYPE\_H\_

#define \_LIB\_RUNTIME\_RES\_TYPE\_H\_

// --------------------------------------------- INCLUDES

// --------------------------------------------- SYNOPSIS

#include "simulator/runtime/rdotrace.h"

#include "simulator/runtime/rdo\_res\_type\_i.h"

#include "simulator/runtime/rdo\_type.h"

// ------------------------------------------------------

OPEN\_RDO\_RUNTIME\_NAMESPACE

class RDOResourceTypeList

: public RDOType

, public IResourceType

, public RDORuntimeObject

, public RDOTraceableObject

{

friend class rdo::Factory<RDOResourceTypeList>;

public:

ResCIterator res\_begin() const;

ResCIterator res\_end () const;

void eraseRes(CREF(rdo::runtime::LPRDOResource) pResource);

protected:

RDOResourceTypeList(ruint number, CREF(rdo::runtime::LPRDORuntime) pRuntime);

virtual ~RDOResourceTypeList();

typedef std::list<rdo::runtime::LPRDOResource> ResourceList;

ResourceList m\_resourceList;

};

//! Описывает РДО-тип ресурса (RTP), который суть фабрика для РДО-ресурсов

//! tparam T - ресурс, который будет создаваться данной фабрикой

template <class T>

class RDOResourceTypeListT: public RDOResourceTypeList

{

DECLARE\_FACTORY(RDOResourceTypeListT<T>);

public:

typedef T value\_type;

private:

//! Конструктор

//! \param number - Целочисленный идентификатор

RDOResourceTypeListT(ruint number, rdo::runtime::LPRDORuntime pRuntime);

virtual ~RDOResourceTypeListT();

DECLARE\_IResourceType;

};

typedef rdo::intrusive\_ptr<RDOResourceTypeList> LPRDOResourceTypeList;

//! Тип ресурсов для создания обычных ресурсов РДО

//! \details Создает ресурсы, которые могут быть релевантны активностям и

//! событиям, но не могут использоваться в процессах

typedef RDOResourceTypeListT<RDOResource> RDOResourceType;

typedef rdo::intrusive\_ptr<RDOResourceType> LPRDOResourceType;

CLOSE\_RDO\_RUNTIME\_NAMESPACE

#include "simulator/runtime/rdo\_res\_type.inl"

#endif // \_LIB\_RUNTIME\_RES\_TYPE\_H\_

**rdo\_res\_type.inl**

// ---------------------------------------------- INCLUDES

// ---------------------------------------------- SYNOPSIS

#include "utils/smart\_ptr/intrusive\_ptr.h"

#include "simulator/runtime/rdo\_runtime.h"

// -------------------------------------------------------

OPEN\_RDO\_RUNTIME\_NAMESPACE

template <class T>

inline RDOResourceTypeListT<T>::RDOResourceTypeListT(ruint number, rdo::runtime::LPRDORuntime pRuntime)

: RDOResourceTypeList(number, pRuntime)

{}

template <class T>

inline RDOResourceTypeListT<T>::~RDOResourceTypeListT()

{}

template <class T>

inline LPRDOResource RDOResourceTypeListT<T>::createRes(CREF(LPRDORuntime) pRuntime, ruint resID, CREF(std::vector<RDOValue>) paramsCalcs, rbool traceFlag, rbool permanentFlag)

{

rdo::intrusive\_ptr<RDOResourceTypeListT<T> > pThis(this);

ASSERT(pThis);

LPIResourceType pIResType = pThis.template interface\_cast<IResourceType>();

ASSERT(pIResType);

rdo::intrusive\_ptr<T> pResource = rdo::Factory<T>::create(pRuntime, paramsCalcs, pIResType, resID, this->getTraceID(), traceFlag, permanentFlag);

ASSERT(pResource);

m\_resourceList.push\_back(pResource);

pRuntime->insertNewResource(pResource);

return pResource;

}

CLOSE\_RDO\_RUNTIME\_NAMESPACE

**rdo\_res\_type.cpp**

// ------------------------------------------------- PCH

#include "simulator/runtime/pch/stdpch.h"

// -------------------------------------------- INCLUDES

// -------------------------------------------- SYNOPSIS

#include "simulator/runtime/rdo\_runtime.h"

// -----------------------------------------------------

OPEN\_RDO\_RUNTIME\_NAMESPACE

RDOResourceTypeList::RDOResourceTypeList(ruint number, CREF(rdo::runtime::LPRDORuntime) pRuntime)

: RDOType (t\_pointer)

, RDOTraceableObject(false, number, rdo::toString(number + 1))

{

rdo::intrusive\_ptr<RDOResourceTypeList> pThis(this);

ASSERT(pThis);

pRuntime->addResType(pThis);

}

RDOResourceTypeList::~RDOResourceTypeList()

{}

void RDOResourceTypeList::eraseRes(CREF(rdo::runtime::LPRDOResource) pResource)

{

m\_resourceList.remove(pResource);

}

IResourceType::ResCIterator RDOResourceTypeList::res\_begin() const

{

return m\_resourceList.begin();

}

IResourceType::ResCIterator RDOResourceTypeList::res\_end() const

{

return m\_resourceList.end();

}

CLOSE\_RDO\_RUNTIME\_NAMESPACE