



**«Московский государственный технический университет
имени Н.Э. Баумана»**

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____

КАФЕДРА _____

РАСЧЁТНО - ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему:

Студент _____ (Подпись, дата) _____ (И.О.Фамилия)

Руководитель курсового проекта _____ (Подпись, дата) _____ (И.О.Фамилия)

Москва, 20__

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

(И.О.Фамилия)

« ____ » _____ 20 ____ г.

З А Д А Н И Е на выполнение курсового проекта

по дисциплине _____

(Тема курсового проекта)

Студент _____
(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к ____ нед., 50% к ____ нед., 75% к ____ нед., 100% к ____ нед.

1. Техническое задание

2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на ____ листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) _____

Дата выдачи задания « ____ » _____ 20 ____ г.

Руководитель курсового проекта

(Подпись, дата)

(И.О.Фамилия)

Студент

(Подпись, дата)

(И.О.Фамилия)

Примечание:

1. Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

Оглавление

Введение	3
1. Предпроектное исследование	4
1.1. Исходное состояние программной части проекта РДО	4
1.2. Основные положения языка РДО	5
1.3. Постановка задачи	6
2. Разработка технического задания на систему	7
2.1. Основания для разработки	7
2.2. Назначение разработки	7
2.3. Характеристики объекта автоматизации	7
2.4. Требования к программе или программному изделию	7
2.4.1. Требования к системе	7
2.4.2. Требования к надежности	7
2.4.3. Условия эксплуатации	7
2.4.4. Требования к составу и параметрам технических средств	7
2.4.5. Требования к информационной и программной совместимости	8
2.4.6. Требования к маркировке и упаковке	8
2.4.7. Требования к транспортированию и хранению	8
2.5. Требования к программной документации	8
2.6. Стадии и этапы разработки	8
2.7. Порядок контроля и приемки	8
3. Концептуальный этап проектирования	9
3.1. Диаграмма пакетов RDO	9
3.2. Разработка системы сборки	10
3.3. Разработка системы тестирования	10
4. Техническое проектирование	11
4.1. Проектирование RDO-Studio-Console	11

5. Рабочее проектирование	13
5.1. Разработка диаграммы состояний RDO-Studio-Console	13
5.2. Разработка диаграммы состояний системы тестирования приложения RDO-Studio-Console	14
6. Результаты	15
6.1. Обновленное состояние программной части проекта РДО	15
6.2. Система автоматизированного тестирования	18
6.3. Сравнение результатов моделирования RAO-Studio Windows и RDO-Studio-Console GNU/Linux	19
Заключение.....	20
Список использованных источников.....	21
Приложение А. Исходный код системы тестирования приложения RDO Studio Console	22

Введение

Развитие технологий Open Source меняет наш мир к лучшему, многие государственные и коммерческие организации внедряют открытое ПО, предпочитая операционную систему GNU/Linux вместо Windows, офисный пакет Open Office вместо MS Office. Эта операционная поддерживается множеством фанатов и профессионалов из таких компаний как IBM, Intel, Novell, Red Hat, Oracle и многими другими. Open Source – это новый этап развития отрасли IT.

До сегодняшнего дня моделирование на языке РДО было возможно только в операционной Windows или в режиме эмуляции в других операционных системах. Для того чтобы систему моделирования РДО можно было компилировать и запускать имитационные модели, я решил разработать консольную версию программы, которая могла бы быть скомпилирована из исходных кодов РДО и выполнять моделирование в операционной системе GNU/Linux.

1. Предпроектное исследование.

1.1. Исходное состояние программной части проекта РДО

Система дискретного имитационного моделирования РДО написана на языке программирования C++, система сборки проекта базируется на файлах .sln, .project для Microsoft Visual Studio. Синтаксис языка РДО описан в файлах “*.l” – входные данные для flex: генератора лексических анализаторов и “*.y” – входные данные для bison: генератора синтаксических анализаторов по данному описанию грамматики. Существующий графический интерфейс системы написан с использованием WinAPI: набор базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows корпорации «Майкрософт».

Проект РДО состоит из следующих библиотек:

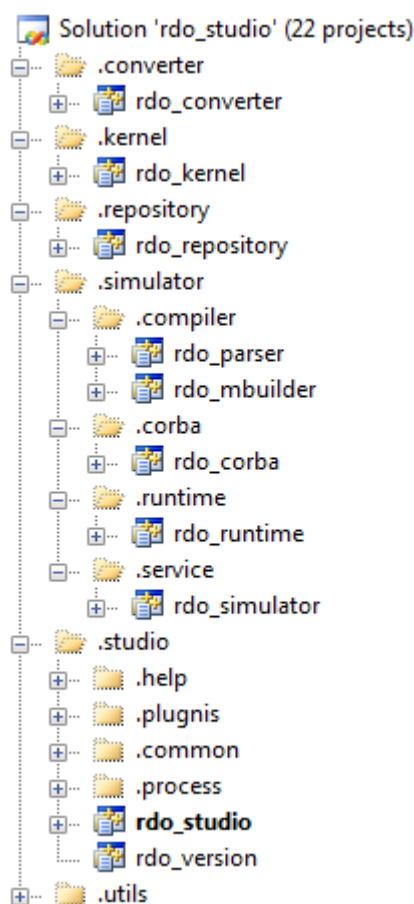


Рис 1.1. Дерево проекта rdo_studio в IDE Microsoft Visual Studio Windows

1. rdo_utils – библиотека вспомогательных функций, для работы с файлами, временем, умными указателями.
2. rdo_kernel – ядро системы моделирования, отвечает за обмен сообщениями между компонентами системы.
3. rdo_repository – подсистема для открытия, закрытия моделей, сохранения результатов моделирования.
4. rdo_parser – компилятор современного синтаксиса РДО.
5. rdo_mbuilder – компилятор графических моделей.
6. rdo_convertor – преобразователь старых моделей в новые.
7. rdo_runtime – симулятор, виртуальная машина системы моделирования.
8. rdo_simulator – обобщенная система для моделирования : объединяет работу Parser, Runtime.
9. rdo_studio – графическая версия системы моделирования РДО.

1.2. Основные положения языка РДО.

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

- Все элементы сложной дискретной системы (СДС) представлены как ресурсы, описываемые некоторыми параметрами.
- Состояние ресурса определяется вектором значений всех его параметров; состояние СДС – значением всех параметров всех ресурсов.
- Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояния ресурсов; действия ограничены во времени двумя событиями: событиями начала и конца.
- Нерегулярные события описывают изменение состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.
- Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения ресурсов в начале и конце соответствующего действия.
- При выполнении работ, связанных с созданием и использованием ИМ в среде РДО, пользователь оперирует следующими основными понятиями:
 - **Модель** - совокупность объектов РДО-языка, описывающих какой-то реальный объект, собираемые в процессе имитации показатели, кадры анимации и графические элементы, используемые при анимации, результаты трассировки.
 - **Прогон** - это единая неделимая точка имитационного эксперимента. Он характеризуется совокупностью объектов, представляющих собой исходные данные и результаты, полученные при запуске имитатора с этими исходными данными.

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

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

Объектами исходных данных являются:

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

Объекты, создаваемые РДО-имитатором при выполнении прогона:

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

1.3. Постановка задачи

Требуется разработать систему сборки, которая может быть использована в GNU/Linux для существующих библиотек; исправить ошибки компиляции в существующих библиотеках, связанные с отличиями компилятора MSVC и GCC; заменить несуществующие в GNU/Linux методы, которые используются в существующих библиотеках РДО, на аналоги; разработать консольную версию программы; разработать дополнительные тесты для перевода системы под GNU/Linux.

2. Разработка технического задания на систему

2.1. Основания для разработки

Задание на курсовой проект.

2.2. Назначение разработки

Основная цель данного курсового проекта – разработать консольную версию РДО для операционной системы Linux (т.е. написать консольное кроссплатформенное приложение для запуска РДО-моделей работоспособное под Linux, и портировать существующие программные библиотеки необходимые для работы этого приложения)

2.3. Характеристики объекта автоматизации.

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

2.4. Требования к программе или программному изделию

2.4.1. Требования к системе.

Консольная версия системы имитационного моделирования RDO-Studio должна:

1. компилироваться
2. исполнять модели в формате .rdox
3. выводить трассировку .trc и результат .pmv в соответствующие файлы

2.4.2 Требования к надежности

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

2.4.3 Условия эксплуатации

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

2.4.4 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 не менее 15” с разрешением от 800*600 и выше;

2.4.5 Требования к информационной и программной совместимости

Данная система должна работать под управлением операционных систем Windows 2000, Windows XP, Windows Vista, Windows 7 и различных дистрибутивах GNU/Linux.

2.4.6 Требования к маркировке и упаковке

Не предъявляются.

2.4.7 Требования к транспортированию и хранению

Не предъявляются.

2.5. Требования к программной документации

Необходимо написать справку для консольной версии системы RDO-Studio.

2.6. Стадии и этапы разработки

- предпроектное исследование.
- концептуальный этап проектирования.
- технический этап проектирования.
- рабочий этап проектирования.

2.7. Порядок контроля и приемки

Контроль и приемка работоспособности консольной версии должны осуществляться путем сравнения результатов моделирования и трассировки полученных консольной версией программы в Linux и RAO-Studio под Windows.

3. Концептуальный этап проектирования

3.1. Диаграмма пакетов RDO

Для определения последовательности перевода библиотек под GNU/Linux была построена диаграмма пакетов.

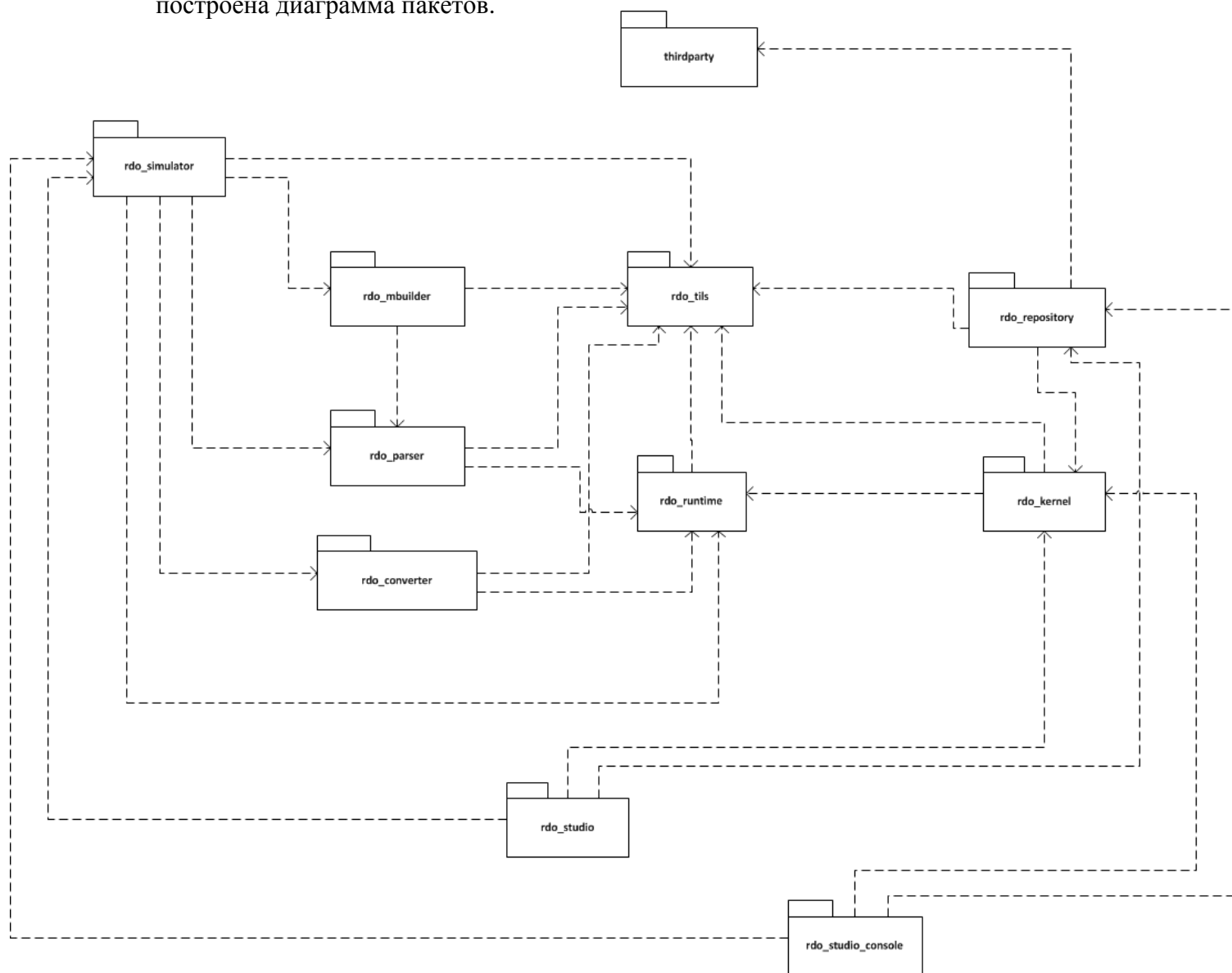


Рис 3.1. Диаграмма пакетов RDO.

Из диаграммы пакетов можно определить последовательность портирования библиотек:

1. rdo_utils – независимый компонент системы
2. rdo_runtime – зависит от rdo_utils
3. rdo_kernel – зависит от rdo_utils и rdo_runtime
4. rdo_repository – зависит от rdo_kernel и rdo_utils

5. `rdo_parser` – зависит от `rdo_runtime` и `rdo_utils`
6. `rdo_mbuilder` – зависит от `rdo_utils` и `rdo_parser`
7. `rdo_converter` – зависит от `rdo_runtime` и `rdo_utils`
8. `rdo_simulator` – зависит `rdo_parser`, `rdo_mbuilder` и `rdo_utils`

Далее после перевода всех библиотек и успешного прохождения их тестов необходимо разработать консольную версию `rdo_studio_console`. Консольное приложение будет зависеть от `rdo_simulator`, `rdo_kernel` и `rdo_repository`.

3.2. Разработка системы сборки

Для сборки системы под GNU/Linux также необходимо переработать систему сборки исходного кода. Из существующих систем кроссплатформенной сборки: `sconf`, `qmake`, `CMake` - была выбрана `CMake`, как наиболее прогрессивная система.

`CMake` (от англ. *cross platform make*) — это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода. `CMake` не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из файлов `CMakeLists.txt`:

- `Makefile` в системах Unix для сборки с помощью `make`;
- файлы `projects/workspaces (.dsp/.dsw)` в Windows для сборки с помощью Visual C++;
- проекты XCode в Mac OS X

Система сборки `CMake` позволяет собирать проект под разными операционными системами, этот факт позволяет разрабатывать и тестировать систему сборки в Windows, а после переносить в GNU/Linux.

3.3. Разработка системы тестирования

Для успешного перевода системы моделирования РДО и возможности тестирования ее компонентов необходимо разработать систему тестирования. В качестве Фреймворка системы тестирования был выбран `BOOST_UNIT_TEST_FRAMEWORK`, по следующим причинам:

- Проект РДО зависит от библиотеки `BOOST`, в которую входит этот Фреймворк.
- `BOOST_UNIT_TEST_FRAMEWORK` позволяет создавать автотесты.

В качестве системы сборки и запуска тестов был выбран `CTest` – компонент `CMake`.

Требуется разработать или доработать тесты для библиотек: `rdo_utils`, `rdo_runtime` и тест для приложения `rdo_stuido_console`.

4. Техническое проектирование

4.1. Проектирование RDO-Studio-Console

Изучив приложение RAO-Studio, был выделен набор классов необходимый для написания консольной версии РДО. Для работы с системой моделирования необходимо разработать класс, который бы обменивался сообщениями с ядром система – RDOKernel.

Разрабатываемый класс RDOStudioConsoleController необходимо протестировать от RDOThread. Была разработана диаграмма классов.

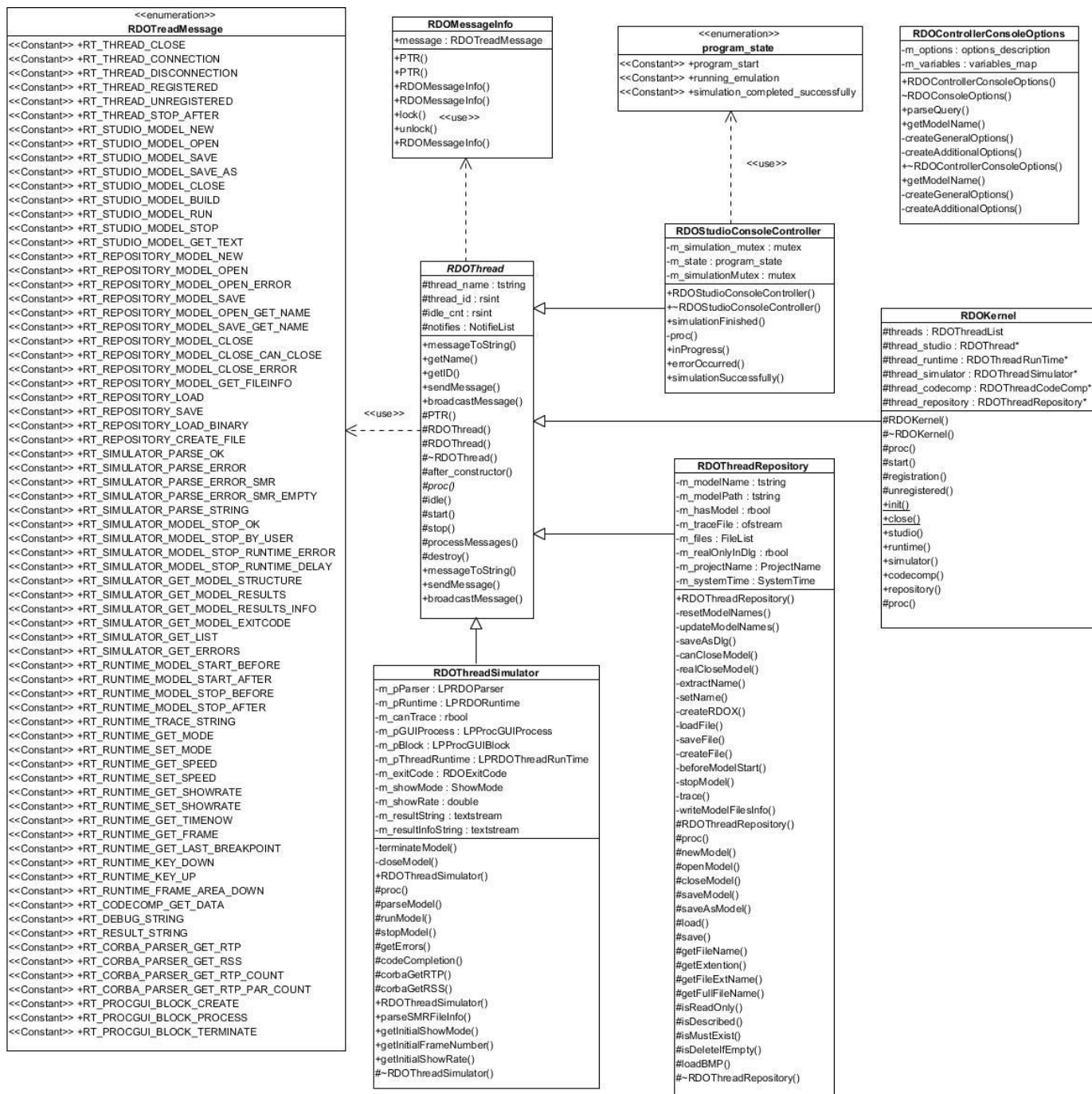


Рис 4.1. Диаграмма классов RDO-Studio-Console

Приложение RDO-Studio-Console должно содержать классы:

- RDOKernel – класс ядра системы моделирования
- RDOThreadSimulator – класс системы моделирования
- RDOThreadRepository – класс системы открытия, сохранения моделей и результатов
- RDOStudioConsoleController – клиентский класс управления системой моделирования
- RDOControllerConsoleOptions – вспомогательный класс обработки параметров командной строки

К консольной версии программы предъявляются требования:

1. Исполнение моделей в формате RDOX
2. Сохранение результатов и трассировки после моделирования на диск
3. Возможность запросить версию программы
4. Возможность запросить версию поддерживаемых форматов моделей
5. Возможность запросить справку о программе

5. Рабочее проектирование

5.1. Разработка диаграммы состояний RDO-Studio-Console

Диаграммы состояний (state machine diagrams) – это известная технология описания поведения системы. В том или ином виде диаграммы состояний существуют с 1960 года, и на заре объектно-ориентированного программирования они применялись для представления поведения системы.

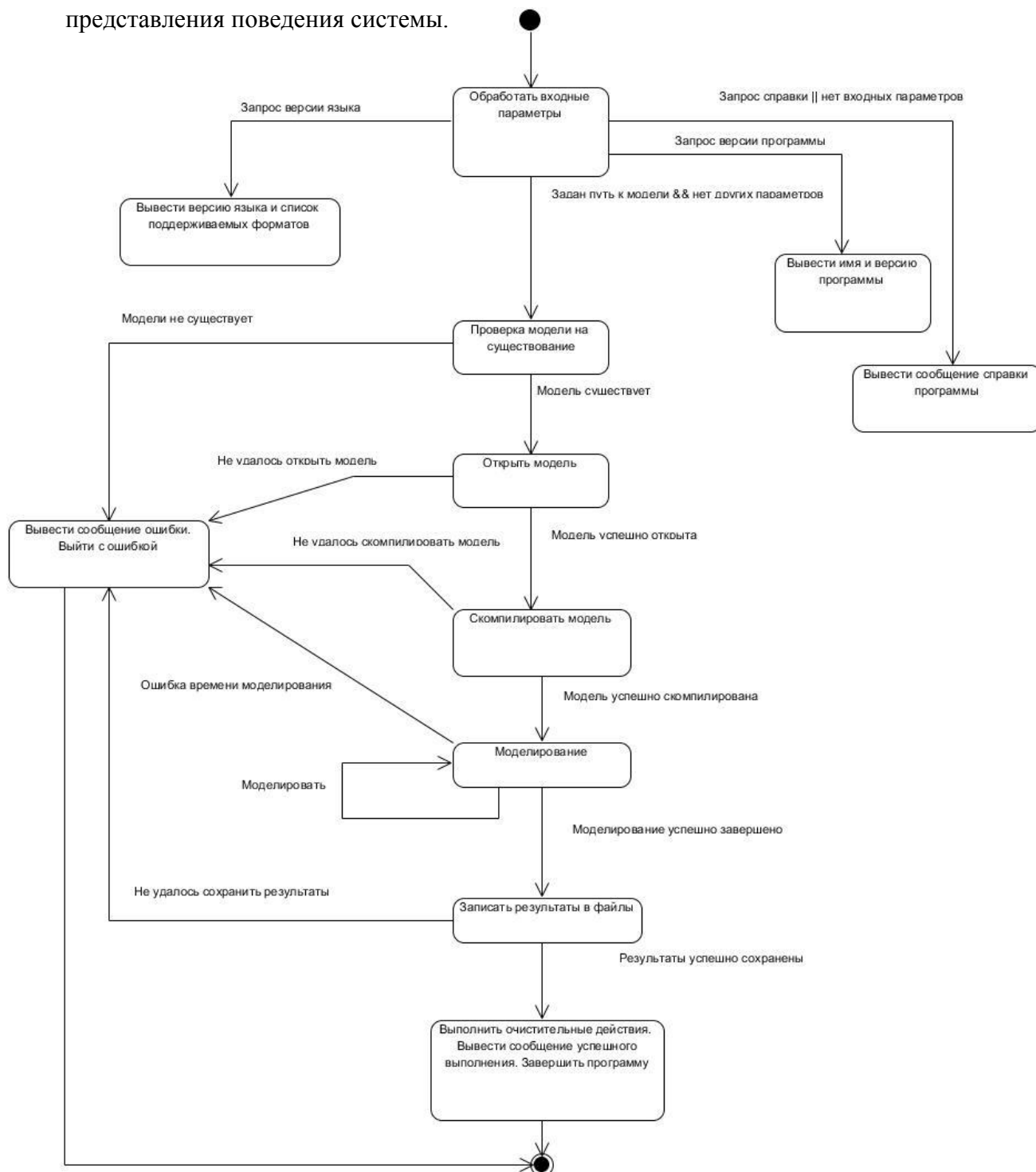


Рис 5.1. Диаграмма состояний RDO-Studio-Console

Диаграмма позволяет понять последовательность выполнения действий, происходящих в процессе работы программы. В ходе работы программа проверяет наличие модели, выполняет запуск модели на компиляцию, исполняет модель и обрабатывает ошибки времени выполнения.

5.2. Разработка диаграммы состояний системы тестирования приложения RDO-Studio-Console

Для тестирования консольной версии системы моделирования РДО был разработан автотест, который выполняет запуск системы моделирования на тестовых моделях, если моделирование прошло успешно программа сравнивает результаты моделирования и эталоны, полученные в RAO-Studio. Если эталоны и полученные файлы результатов и трассировки идентичны, то программа работает верно и тест считается пройденным, иначе тест считается проваленным, также программа проверяет произошло ли моделирование.

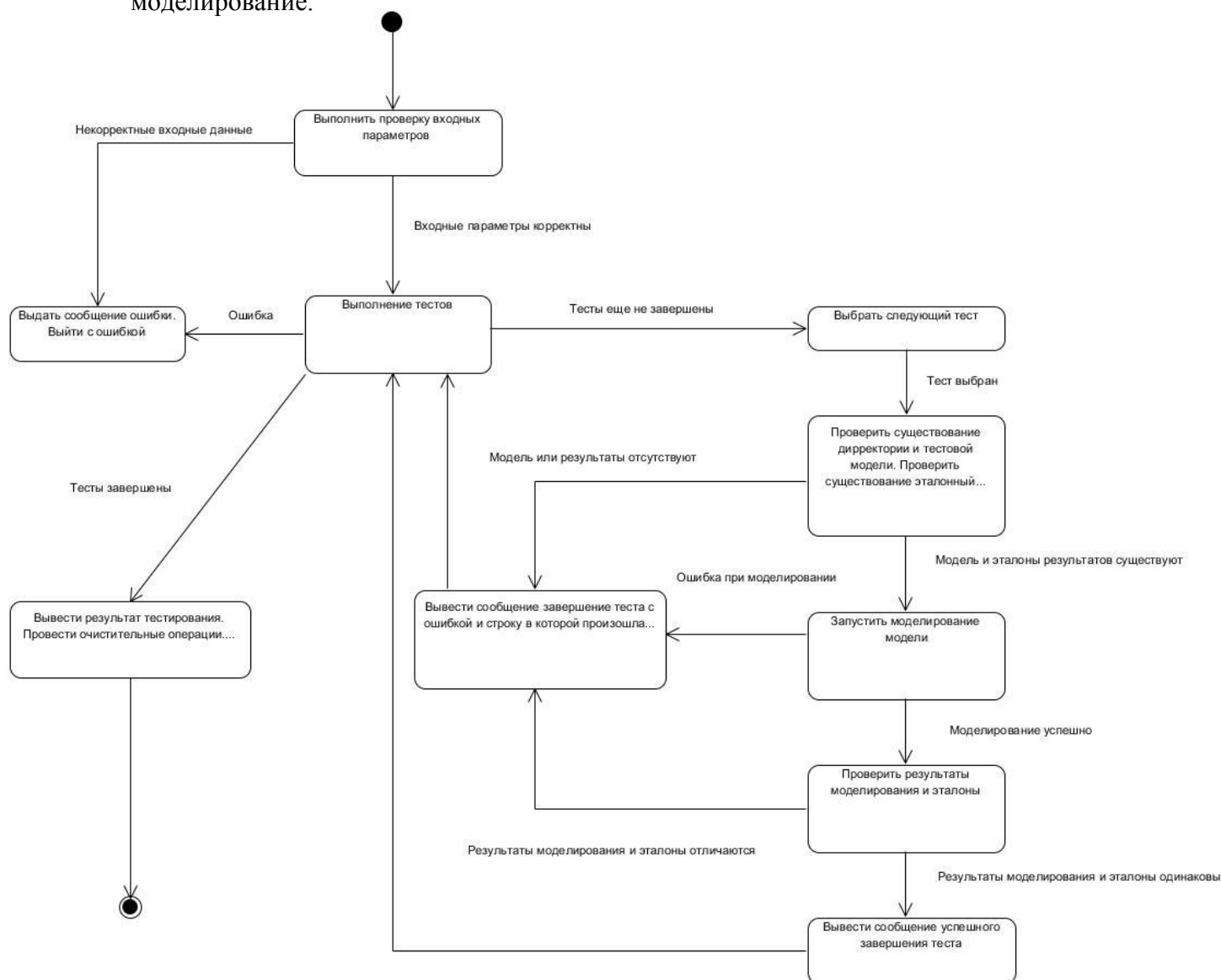


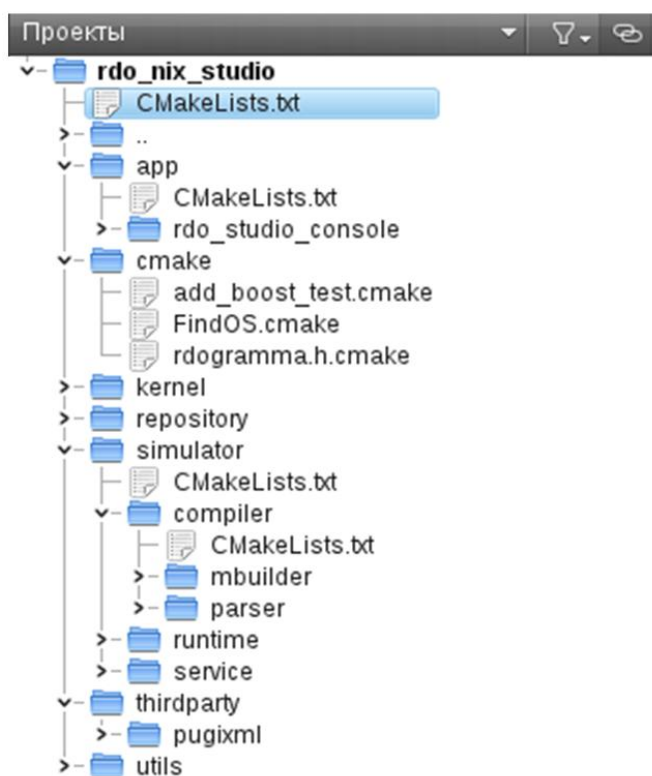
Рис 5.2. Диаграмма состояний системы тестирования RDO-Studio-Console

6. Результаты

6.1. Обновленное состояние программной части проекта РДО

При портировании системы дискретного имитационного моделирования РДО были проведены следующие работы:

1. Была разработана система сборки проекта с использованием CMake. **CMake** - это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода. CMake не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt.



1. rdo_utils – библиотека успешно портирована.
2. rdo_kernel – библиотека успешно портирована.
3. rdo_repository – библиотека успешно портирована.
4. rdo_parser – библиотека успешно портирована.
5. rdo_mbuilder – библиотека успешно портирована, не используется.
6. rdo_convertor – библиотека не портировалась.
7. rdo_runtime – библиотека успешно портирована.
8. rdo_simulator – библиотека успешно портирована.
9. rdo_studio_console – консольная версия системы, разработана.

Рис 6.1 Дерево проекта rdo_nix_studio в IDE Qt Creator GNU/Linux

Пример успешного создания генерации файлов сборки с помощью CMake:

```
-----  
Create RDO  
Operating system is Windows  
CREATE RDO_UTILS LIBRARY  
CREATE test_rdo_utils  
CREATE test_rdo_utils_smart_prt  
CREATE test_rdo_utils_interface  
CREATE test_rdo_animation
```

```

CREATE RDO_KERNEL LIBRARY
CREATE RDO_REPOSITORY LIBRARY
CREATE CONVERTERS
CREATE CONVERTER SMR2RDOX LIBRARY
CREATE RDO_RUNTIME LIBRARY
CREATE test_rdo_runtime_logic
CREATE test_rdo_runtime_array
CREATE test_rdo_runtime_matrix
CREATE test_rdo_runtime_value
CREATE test_rdo_runtime_fuzzy
CREATE test_rdo_sequences
CREATE RDO_PARSER LIBRARY
CREATE RDO_MBUILDER LIBRARY
CREATE RDO_SIMULATOR LIBRARY
CREATE RDO_STUDIO_CONSOLE EXECUTABLE
CREATE test_rdo_studio_console
Configuring done
Generating done

```

-
2. Были разрешены проблемы компиляции при сборке проекта с помощью GNU GCC: большая часть ошибок компиляции была связана с нестандартными конструкциями при описании шаблонов, которые не заявлены в стандарте, но поддерживаются компилятором MSVC.
 3. В существующих библиотеках были переработаны функции для работы с файлами и временем, которые отсутствуют в GNU/Linux. Они были заменены на существующие аналоги.
 4. Разработана консольная кроссплатформенная версия системы дискретного имитационного моделирования РДО.

```

evgeny@linux-d017:~/repository/rdo/build> ./rdo --help

rdo console studio v0.3.5 Linux( http://rdo.rk9.bmstu.ru ):

General options:
  -i [ --input ] arg      : path to model
  -v [ --version ]       : display program version
  -h [ --help ]          : display help message
  -l [ --language ]      : display language version of rdo

Compatibility options (skipped in console version):
  --autorun              : autostart program
  --autoexit             : autoexit program
  --dont_close_if_error  : don't close if error
  --pluginstart          : start plugin
  --pluginautoexit       : plugin auto exit

evgeny@linux-d017:~/repository/rdo/build> █

```

Рис 6.2. Пример вызова справки у программы RDO-Studio-Console

```
evgeny@linux-d017:~/repository/rdo/build> ./rdo --language  
rdo language v2.0 ( supported rdox )  
evgeny@linux-d017:~/repository/rdo/build> █
```

Рис 6.3. Пример вызова информации о версии языка у программы RDO-Studio-Console

```
evgeny@linux-d017:~/repository/rdo/build> ./rdo --version  
rdo console studio v0.3.5  
evgeny@linux-d017:~/repository/rdo/build> █
```

Рис 6.4. Пример вызова информации о версии у программы RDO-Studio-Console

```
evgeny@linux-d017:~/repository/rdo/build> ./rdo -i ../app/rdo_studio_console/test/array/array.rdox  
simulation finished successfully  
evgeny@linux-d017:~/repository/rdo/build> █
```

Рис 6.5. Пример запуска RDO-Studio-Console и успешного выполнения моделирования

5. Обновлена система тестов, разработана система автотестов с использованием CMake / CTest в качестве системы сборки, запуска автотестов и Boost Unit Test Framework – система классов для автоматического тестирования на C++.

6.2. Система автоматизированного тестирования

Для проверки успешного перевода различных компонентов были модифицированы существующие тесты или разработаны новые. Все тесты стали автоматическими – могут быть выполнены без человека компьютером. В результате тесты выдает результат его выполнения: успешен или завершился с ошибкой и данные об ошибке. В качестве системы автоматического тестирования была выбрана CTest – подсистема интегрированная в CMake для создания автотестов.

```
evgeny@linux-d017:~/repository/rdo/build> make test
Running tests...
Test project /home/evgeny/repository/rdo/build
  Start 1: test_rdo_utils
1/11 Test #1: test_rdo_utils ..... Passed    0.04 sec
  Start 2: test_rdo_utils_smart_prt
2/11 Test #2: test_rdo_utils_smart_prt ..... Passed    0.01 sec
  Start 3: test_rdo_utils_interface
3/11 Test #3: test_rdo_utils_interface ..... Passed    0.01 sec
  Start 4: test_rdo_animation
4/11 Test #4: test_rdo_animation ..... Passed    0.01 sec
  Start 5: test_rdo_runtime_logic
5/11 Test #5: test_rdo_runtime_logic ..... Passed    0.01 sec
  Start 6: test_rdo_runtime_array
6/11 Test #6: test_rdo_runtime_array ..... Passed    0.01 sec
  Start 7: test_rdo_runtime_matrix
7/11 Test #7: test_rdo_runtime_matrix ..... Passed    0.01 sec
  Start 8: test_rdo_runtime_value
8/11 Test #8: test_rdo_runtime_value ..... Passed    0.01 sec
  Start 9: test_rdo_runtime_fuzzy
9/11 Test #9: test_rdo_runtime_fuzzy ..... Passed    0.01 sec
  Start 10: test_rdo_sequences
10/11 Test #10: test_rdo_sequences ..... Passed    0.02 sec
  Start 11: test_rdo_studio_console
11/11 Test #11: test_rdo_studio_console ..... Passed   52.20 sec

100% tests passed, 0 tests failed out of 11

Total Test time (real) = 53.17 sec
evgeny@linux-d017:~/repository/rdo/build> █
```

Рис 6.5. Пример успешного выполнения автотестов в консоли GNU/Linux.

Для запуска автотестов в GNU/Linux в консоли в каталоге сборки необходимо выполнить команду **make test**, для запуска в Visual Studio используется соответствующая цель запуска **RUN_TESTS**.

6.3. Сравнение результатов моделирования RAO-Studio Windows и RDO-Studio-Console GNU/Linux

В качестве результатов прогона имитационной модели РДО выдает два файла:

6. .trc – файл трассировки модели
7. .pmv – файл результатов моделирования

Разработанная консольная версия системы RDO-Studio-Console также как и графическая среда моделирования выдает файлы результатов.



 array	08.12.2011 19:39	Файл "PMV"	1 КБ
 array	08.12.2011 19:39	Файл "TRC"	2 КБ

Рис 6.6. Файлы результатов полученные в Windows с использованием RAO-Studio.

```
-rw-r--r-- 1 evgeny users 371 Дек 8 20:29 array.pmv  
-rw-r--r-- 1 evgeny users 96 Дек 4 15:08 array.rdox  
-rw-r--r-- 1 evgeny users 69 Дек 4 15:08 array.rss  
-rw-r--r-- 1 evgeny users 159 Дек 4 15:08 array.rtp  
-rw-r--r-- 1 evgeny users 1117 Дек 8 20:29 array.trc
```

Рис 6.7. Файлы результатов полученные в GNU/Linux с использованием RDO-Studio-Console.

Для проверки правильности работы консольной версии системы в GNU/Linux был разработан тест, который для заданных имитационных моделей выполняет запуск консольной версии системы и сравнивает полученные файлы трассировки и результатов с эталонными файлами полученными под Windows в RAO-Studio. Исходный код автотеста приведен в приложении А.

В качестве тестовых моделей были выбраны модели:

1. array
2. create_res
3. fms_event
4. fms_planing

Результаты моделирования в консольной и графической версии системы совпадают.

Заключение

В результате проведенной работы была разработана кроссплатформенная консольная версия системы моделирования RDO-Studio-Console способная исполнять модели моделирования в формате .rdox в операционных системах семейства Windows NT и GNU/Linux, система моделирования выполняет тестовые модели “Heidel” и “FmsPlaning”. Разработана система сборки исходного кода для GNU/Linux на основе CMake, переведены существующие библиотеки. Разработана система автоматического тестирования.

Список использованных источников

1. Бьерн Страуструп. Язык моделирования C++. Специальное издание. Пер. с англ. – М.: ООО «Бином-пресс», 2007 г. – 1104 с.
2. Солтер Н.А., Клепер С.Дж. C++ для профессионалов. М.: Диалектика, 2006 г. , пер. с 2005, 907 с.
3. Фаулер М. Основы UML, 3-е издание. : СПб: Символ-Плюс, 2002 г. 185 с.
4. Ларман, Крэг. Применение UML и шаблонов проектирования. 2-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2004. — 624 с.
5. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78.
6. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-90. Условные обозначения и правила выполнения.

Приложение А. Исходный код системы тестирования приложения

RDO Studio Console

Файл app\rdo_studio_console\test\main.cpp

```

/ * !
  \copyright (c) RDO-Team, 2011
  \file      main.cpp
  \author    Пройдаков Евгений (lord.tiran@gmail.com)
  \date      10.05.2009
  \brief     Тест приложения rdo_studio_console
  \indent    4T
*/

// ----- PCH
// ----- INCLUDES
#include <list>
#include <fstream>
#include <boost/filesystem.hpp>
namespace fs = boost::filesystem;

#include <stdlib.h>

#define BOOST_TEST_MODULE RDOStudioConsoleTest
#include <boost/test/included/unit_test.hpp>
// ----- SYNOPSIS
#include "utils/rdocommon.h"
// -----

typedef std::list<tstring> file_data_list;

const tstring RDO_STUDIO_CONSOLE_TEST_PATH_STRING =
RDO_STUDIO_CONSOLE_TEST_PATH;
const tstring RDO_STUDIO_CONSOLE_APP_STRING = RDO_STUDIO_CONSOLE_APP;

void read_trace(CREF(tstring) file, REF(file_data_list) list)
{
    std::fstream stream(file.c_str(), std::ios::in);
    if (!stream.is_open()) {
        BOOST_ERROR("Can't open file " + file);
    }
    tstring temp_string;
    bool key = false;
    while(true) {
        std::getline(stream, temp_string);
        if (stream.fail())
            break;

        if (temp_string.find("DPS_MM") != -1)
            break;

        if (!key)
            key = temp_string.find("$Changes") == -1 ? false : true;

        if (key)
            list.push_back(temp_string);
    }
}
```



```

void compare_trace(CREF(tstring) etalon_trace, CREF(tstring) trace)
{
    file_data_list etalon_trace_list;
    file_data_list trace_list;

    read_trace(etalon_trace, etalon_trace_list);
    read_trace(trace, trace_list);

    BOOST_CHECK(etalon_trace_list.size() == trace_list.size());
    BOOST_CHECK(etalon_trace_list == trace_list);
}

void read_result(CREF(tstring) file, REF(file_data_list) list)
{
    std::fstream stream(file.c_str(), std::ios::in);
    if (!stream.is_open()) {
        BOOST_ERROR("Can't open file " + file);
    }
    tstring temp_string;
    bool key = false;
    while(true) {
        std::getline(stream, temp_string);
        if (stream.fail())
            break;

        if (key)
            list.push_back(temp_string);

        if (!key)
            key = temp_string.find("$BExpCalcCounter") == -1 ? false :
true;
    }
}

void compare_result(CREF(tstring) etalon_result, CREF(tstring) result)
{
    file_data_list etalon_result_list;
    file_data_list result_list;

    read_result(etalon_result, etalon_result_list);
    read_result(result, result_list);

    BOOST_CHECK(etalon_result_list.size() == result_list.size());
    BOOST_CHECK(etalon_result_list == result_list);
}

void test_model(CREF(tstring) path, CREF(tstring) model_name)
{
    tstring dir = path;
    if(dir[dir.size() - 1] != '/')
        dir += '/';
    fs::path directory = dir.c_str();
    tstring file = dir + model_name + ".rdox";

    if (!fs::exists(directory)) {
        tstring message("file \"" + file + "\" does not exist");
        std::cerr << message.c_str() << std::endl;
        BOOST_REQUIRE(false);
    }
    if(!fs::exists(file)) {
        tstring message("directory \"" + dir + "\" does not exist");
        std::cerr << message.c_str() << std::endl;
    }
}

```

```

        BOOST_REQUIRE(false);
    }
    tstring etalon_mark("_etalon");
    tstring etalon_trace = dir + model_name + etalon_mark + ".trc";
    tstring etalon_result = dir + model_name + etalon_mark + ".pmv";

    BOOST_REQUIRE(fs::exists(etalon_trace));
    BOOST_REQUIRE(fs::exists(etalon_result));

    tstring simulation_trace = dir + model_name + ".trc";
    tstring simulation_result = dir + model_name + ".pmv";

    boost::filesystem::remove(simulation_trace);
    boost::filesystem::remove(simulation_result);

    tstring command(RDO_STUDIO_CONSOLE_APP_STRING + tstring(" -i ") + file);
    system(command.c_str());

    BOOST_REQUIRE(fs::exists(simulation_trace));
    BOOST_REQUIRE(fs::exists(simulation_result));

    compare_trace(etalon_trace, simulation_trace);
    compare_result(etalon_result, simulation_result);

    boost::filesystem::remove(simulation_trace);
    boost::filesystem::remove(simulation_result);
}

BOOST_AUTO_TEST_SUITE(RDOSTudioConsoleTest)

BOOST_AUTO_TEST_CASE(RDOSTudioConsoleTestCheckInputData)
{
    if(RDO_STUDIO_CONSOLE_APP_STRING == "NULL")
    {
        BOOST_ERROR("Invalid input data");
        exit(1);
    }
}

BOOST_AUTO_TEST_CASE(RDOSTudioConsoleModelSimpleQS)
{
    tstring string("simple_qs");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}

BOOST_AUTO_TEST_CASE(RDOSTudioConsoleModelMultichannelQS)
{
    tstring string("multichannel_qs");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}

BOOST_AUTO_TEST_CASE(RDOSTudioConsoleModelArray)
{
    tstring string("array");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}

BOOST_AUTO_TEST_CASE(RDOSTudioConsoleModelCreateRes)
{
    tstring string("create_res");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}

```

```

BOOST_AUTO_TEST_CASE(RDOStudioConsoleModelFmsEvent)
{
    tstring string("fms_event");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}

BOOST_AUTO_TEST_CASE(RDOStudioConsoleModelFmsPlaning)
{
    tstring string("fms_planing");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}

BOOST_AUTO_TEST_CASE(RDOStudioConsoleModelHeidel)
{
    tstring string("heidel");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}
/*
BOOST_AUTO_TEST_CASE(RDOStudioConsoleModelEventQS)
{
    tstring string("event_qs");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}

BOOST_AUTO_TEST_CASE(RDOStudioConsoleModelPoliclinic)
{
    tstring string("policlinic");
    test_model(RDO_STUDIO_CONSOLE_TEST_PATH_STRING + string, string);
}
*/
BOOST_AUTO_TEST_SUITE_END() // RDOStudioConsoleTest

```

Файл app\rdo_studio_console\test\CMakeLists.txt

```
#=====
# Copyright (c) 2011 Evgeny Proydakov <lord.tiran@gmail.com>
#=====
# Cmake for RDO STUDIO CONSOLE TEST
#=====

SET(MAIN_FILE
    main.cpp)

SET(ARRAY_MODEL
    array/array_etalon.pmv
    array/array_etalon.trc
    array/array.dpt
    array/array.frm
    array/array.fun
    array/array.opr
    array/array.pat
    array/array.rdox
    array/array.rss
    array/array.rtp)

SET(CREATE_RES_MODEL
    create_res/create_res_etalon.pmv
    create_res/create_res_etalon.trc
    create_res/create_res.prc
    create_res/create_res.rdox
    create_res/create_res.rss
    create_res/create_res.rtp)

SET(EVENT_QS_MODEL
    event_qs/event_qs_etalon.pmv
    event_qs/event_qs_etalon.trc
    event_qs/event_qs.fun
    event_qs/event_qs.pmd
    event_qs/event_qs.rdox
    event_qs/event_qs.rss
    event_qs/event_qs.rtp)

SET(FMS_EVENT_MODEL
    fms_event/fms_event_etalon.pmv
    fms_event/fms_event_etalon.trc
    fms_event/fms_event.evn
    fms_event/fms_event.fun
    fms_event/fms_event.pmd
    fms_event/fms_event.rdox
    fms_event/fms_event.rss
    fms_event/fms_event.rtp
    fms_event/fms_event.smr)

SET(FMS_PLANING_MODEL
    fms_planing/fms_planing.dpt
    fms_planing/fms_planing_etalon.pmv
    fms_planing/fms_planing_etalon.trc
    fms_planing/fms_planing.evn
    fms_planing/fms_planing.frm
    fms_planing/fms_planing.fun
    fms_planing/fms_planing.pat
    fms_planing/fms_planing.pmd
```

```

fms_planing/fms_planing.prc
fms_planing/fms_planing.rdox
fms_planing/fms_planing.rss
fms_planing/fms_planing.rtp
fms_planing/fms_planing.smr)

SET(POLICLINIC_MODEL
  policlinic/policlinic.dpt
  policlinic/policlinic_etalon.pmv
  policlinic/policlinic_etalon.trc
  policlinic/policlinic.evn
  policlinic/policlinic.fun
  policlinic/policlinic.pat
  policlinic/policlinic.pmd
  policlinic/policlinic.rdox
  policlinic/policlinic.rss
  policlinic/policlinic.rtp)

SET(SIMPLE_QS_MODEL
  simple_qs/simple_qs.dpt
  simple_qs/simple_qs_etalon.pmv
  simple_qs/simple_qs_etalon.trc
  simple_qs/simple_qs.evn
  simple_qs/simple_qs.fun
  simple_qs/simple_qs.pat
  simple_qs/simple_qs.pmd
  simple_qs/simple_qs.rdox
  simple_qs/simple_qs.rss
  simple_qs/simple_qs.rtp
  simple_qs/simple_qs.smr)

SET(MULTICHANNEL_QS_MODEL
  multichannel_qs/multichannel_qs.dpt
  multichannel_qs/multichannel_qs_etalon.pmv
  multichannel_qs/multichannel_qs_etalon.trc
  multichannel_qs/multichannel_qs.evn
  multichannel_qs/multichannel_qs.fun
  multichannel_qs/multichannel_qs.pat
  multichannel_qs/multichannel_qs.pmd
  multichannel_qs/multichannel_qs.rdox
  multichannel_qs/multichannel_qs.rss
  multichannel_qs/multichannel_qs.rtp
  multichannel_qs/multichannel_qs.smr)

SET(HEIDEL_MODEL
  heidel/Cut1.bmp
  heidel/Cut2.bmp
  heidel/Cut3.bmp
  heidel/Cut4.bmp
  heidel/Cut6.bmp
  heidel/Cut8.bmp
  heidel/Cut9.bmp
  heidel/Cut_m.bmp
  heidel/F1.bmp
  heidel/Fr1111.bmp
  heidel/Fr3.bmp
  heidel/Fr4.bmp
  heidel/Fr5.bmp
  heidel/heidel.dpt
  heidel/heidel_etalon.pmv
  heidel/heidel_etalon.trc
  heidel/heidel.evn

```

```

    heidel/heidel.frm
    heidel/heidel.fun
    heidel/heidel.pat
    heidel/heidel.pmd
    heidel/heidel.rdox
    heidel/heidel.rss
    heidel/heidel.rtp
    heidel/heidel.smr)

SET(TEST_MODEL_FILES
    ${ARRAY_MODEL}
    ${CREATE_RES_MODEL}
    ${EVENT_QS_MODEL}
    ${FMS_EVENT_MODEL}
    ${FMS_PLANING_MODEL}
    ${POLICLINIC_MODEL}
    ${SIMPLE_QS_MODEL}
    ${MULTICHANNEL_QS_MODEL}
    ${HEIDEL_MODEL})

IF(MSVC)
    SET(RDO_STUDIO_CONSOLE_APP "-DRDO_STUDIO_CONSOLE_APP
    ${EXECUTABLE_OUTPUT_PATH}/${CMAKE_BUILD_TYPE}/rdo.exe")
    SET(RDO_STUDIO_CONSOLE_TEST_PATH "-DRDO_STUDIO_CONSOLE_TEST_PATH
    ${CMAKE_CURRENT_SOURCE_DIR}/")
    #    CONFIGURE_FILE(${CMAKE_CURRENT_SOURCE_DIR}/config.msvc.h.cmake
    ${CMAKE_CURRENT_SOURCE_DIR}/config.h)
ELSE()
    SET(RDO_STUDIO_CONSOLE_APP "-DRDO_STUDIO_CONSOLE_APP
    ${EXECUTABLE_OUTPUT_PATH}/rdo")
    SET(RDO_STUDIO_CONSOLE_TEST_PATH "-DRDO_STUDIO_CONSOLE_TEST_PATH
    ${CMAKE_CURRENT_SOURCE_DIR}/")
    #    CONFIGURE_FILE(${CMAKE_CURRENT_SOURCE_DIR}/config.gcc.h.cmake
    ${CMAKE_CURRENT_SOURCE_DIR}/config.h)
ENDIF(MSVC)

SET(CONFIG_FILES)
#    config.h)

SET(RDO_STUDIO_CONSOLE_APP ${EXECUTABLE_OUTPUT_PATH}/rdo)
SET(RDO_STUDIO_CONSOLE_TEST_PATH ${CMAKE_CURRENT_SOURCE_DIR})

ADD_BOOST_TEST(test_rdo_studio_console ${MAIN_FILE} ${TEST_MODEL_FILES}
    ${CONFIG_FILES})

ADD_DEPENDENCIES(test_rdo_studio_console rdo)
ADD_DEPENDENCIES(test_rdo_studio_console rdo_utils)

TARGET_LINK_LIBRARIES(test_rdo_studio_console rdo_utils)

#SET_TARGET_PROPERTIES(test_rdo_studio_console PROPERTIES COMPILE_FLAGS
    ${RDO_STUDIO_CONSOLE_APP})
#SET_TARGET_PROPERTIES(test_rdo_studio_console PROPERTIES COMPILE_FLAGS
    ${RDO_STUDIO_CONSOLE_TEST_PATH})

IF(MSVC)
    ADD_DEFINITIONS(-
    DRDO_STUDIO_CONSOLE_APP="${EXECUTABLE_OUTPUT_PATH}/${CMAKE_BUILD_TYPE}/rdo.ex
    e")
ELSE()
    ADD_DEFINITIONS(-DRDO_STUDIO_CONSOLE_APP="${EXECUTABLE_OUTPUT_PATH}/rdo")

```

```

ENDIF(MSVC)

ADD_DEFINITIONS(-
DRDO_STUDIO_CONSOLE_TEST_PATH="${CMAKE_CURRENT_SOURCE_DIR}/")

IF(MSVC) # lotions for windows #

SOURCE_GROUP(".array_model" FILES
    ${ARRAY_MODEL})

SOURCE_GROUP(".create_res_model" FILES
    ${CREATE_RES_MODEL})

SOURCE_GROUP(".event_qs_model" FILES
    ${EVENT_QS_MODEL})

SOURCE_GROUP(".fms_event_model" FILES
    ${FMS_EVENT_MODEL})

SOURCE_GROUP(".fms_planing_model" FILES
    ${FMS_PLANING_MODEL})

SOURCE_GROUP(".policlinic_model" FILES
    ${POLICLINIC_MODEL})

SOURCE_GROUP(".simple_qs_model" FILES
    ${SIMPLE_QS_MODEL})

SOURCE_GROUP(".multichannel_qs" FILES
    ${MULTICHANNEL_QS_MODEL})

SOURCE_GROUP(".heidel" FILES
    ${HEIDEL_MODEL})

ENDIF(MSVC)

```