Пакет программ

реализации стандарта распределенного моделирования (PCPM)

(в соответствии с международным стандартом IEEE-1516-2010 (HLA))

RRTI, версия 2.0

Руководство пользователя

Содержание

1 Введение

1.1 Об этом руководстве

Данное руководство пользователя содержит информацию по установке, конфигурированию и использованию пакета программ RRTI-2 поддержки стандарта HLA. HLA (*High Level Architecture*) – это международный стандарт IEEE 1516 для организации взаимодействия компонентов в распределенных системах моделирования. Аббревиатура RTI (*Run-Time Infrastructure*) в этом стандарте используется для обозначения пакета программ, обеспечивающего поддержку стандарта HLA для разработки и выполнения приложений. Аббревиатура RRTI обозначает российскую RTI (*Russian RTI)*.

Руководство предназначено для:

- разработчиков программных моделей и систем моделирования, совместимых с HLA и желающих использовать RRTI как пакет реализации HLA;
- инженерно-технического персонала, ответственного за установку, конфигурирование и сопровождение таких систем моделирования.

Во введении представлен ознакомительный обзор RRTI.

Глава 2 содержит общую информацию о компонентах RRTI, их назначении и функциях.

В главе 3 описан процесс установки RRTI на вычислительные средства пользователя.

В главе 4 представлены файлы конфигурации и параметры конфигурации компонентов RRTI, которые могут быть использованы системным администратором или пользователем для настройки функциональности и производительности RRTI.

Глава 5 описывает файлы журнала и сообщения компонентов RRTI. Эти сообщения могут использоваться при необходимости уточнить детали функционирования RRTI или при поиске источника проблем.

Глава 6 представляет собой руководство оператора по работе с центральным компонентом RRTI.

Глава 7 посвящена вопросам разработки приложений (федератов), использующих RRTI.

1.2 Введение в HLA и RRTI

RRTI предназначена для организации взаимодействия между компонентами моделирующих систем и интеграции моделирующих систем в единые моделирующие комплексы в соответствии с международными стандартами для архитектуры высокого уровня HLA (High Level Architecture).

Архитектура высокого уровня HLA представляет собой концепцию обеспечения интероперабельности (способности к взаимодействию) для различных компонентов тренажерных и информационно-моделирующих систем. Ключевым компонентом HLA является Спецификация Интерфейса (Interface Specification); она определяет стандартные сервисы, которые координировано используются отдельными компонентами моделирующих систем (федератами) в процессе распределенного моделирования. При этом часть работы по обеспечению взаимодействия возлагается на приложение (обращение к стандартизованным сервисам и выполнение определенных правил), а другая часть – на программную инфраструктуру времени исполнения (RTI).

Таким образом, если HLA – это архитектура, то RTI – это программная среда, функционирующая между приложениями и операционной системой и обеспечивающая выполнение сервисов HLA и согласованную работу федератов в составе федерации. RRTI-2 представляет собой реализацию RTI для версии стандарта HLA IEEE 1516-2010.

В терминах HLA распределенная моделирующая система рассматривается как федерация, в составе которой согласованно выполняются ее компоненты – федераты. RRTI обеспечивает выполнение федерации:

- на отдельном вычислительном средстве;
- в локальной вычислительной сети (LAN);
- в глобальной вычислительной сети (WAN).

Таблица 1 содержит основные функции, которые обеспечиваются RRTI в соответствии с требованиями архитектуры HLA.

Таблица 1 – Основные функции RRTI

Группа функций	Описание функций
Управление федерацией	Создание/удаление федерации. Управление участием федератов в федерации. Добавление модулей к объектной модели федерации. Управление точками синхронизации. Сохранение/восстановление состояния федерации. Контроль целостности федерации и отслеживание сбоев
Управление декларациями	Публикация атрибутов. Подписка на атрибуты. Публикация взаимодействий (событий). Подписка на взаимодействия (события)
Управление объектами	Создание и обнаружение экземпляров объектов. Обновление значений атрибутов. Отправка и получение взаимодействий. Управление типами транспортировки данных. Поддержка фильтрации данных на основе частоты подписки. Поддержка фильтрации данных на основе регионов
Управление владением	Вступление во владение атрибутами. Отказ от владения атрибутами. Передача владения атрибутами между федератами. Поиск новых владельцев для атрибутов, потерявших владельца
Управление логическим временем	Согласованное продвижение логического времени федерата в составе федерации. Согласованная доставка сообщений, упорядоченных по времени
Управление распространением данных	Ассоциация регионов обновлений с атрибутами. Подписка на атрибуты с указанием регионов подписки. Подписка на взаимодействия с указанием регионов подписки. Распространение данных с фильтрацией по регионам

RRTI полностью соответствует международным стандартам группы IEEE 1516-2010 (HLA evolved). Поддерживается программный интерфейс для языка C++, а также WSDL-интерфейс веб-сервисов, обеспечивающий работу веб-федератов.

Кроме того, поддерживается программный интерфейс первой версии RRTI-1 для языка C++, основанный на международных стандартах группы IEEE 1516-2000 с некоторыми ограничениями и дополнениями (в частности, для старого интерфейса HLA отсутствует реализация MOM).

Таблица 2 содержит характеристики масштабируемости RRTI.

Таблица 2 – характеристики масштабируемости RRTI

Характеристика	Максимальное количество
Хостов, на которых выполняются компоненты RTI	200
Одновременно выполняющихся федераций	10
Федератов в федерации	1000
Федератов, выполняющихся на одном хосте	255
Классов объектов в FOM	1000
Классов взаимодействий в FOM	1000
Экземпляров объектов, созданных одним федератом	65534
Атрибутов объектных классов в FOM	1000000
Атрибутов в одном объектном классе	65534
Параметров взаимодействий в FOM	10000
Размерностей областей фильтрации данных в FOM	100

2 Компоненты RRTI

Пакет программ RRTI включает в себя несколько компонентов, три из которых являются основными и требуются при любой конфигурации RRTI:

- центральный компонент - Central RTI Component (CRC);
- локальный компонент - Local RTI Component (LRC);
- локальный серверный компонент - Local Server RTI Component (LSRC).

2.1 Центральный компонент RRTI (CRC)

Центральный компонент RRTI (CRC) осуществляет централизованное управление федерациями, обеспечивает согласованную работу федератов в составе федерации и арбитраж конфликтующих запросов федератов к RRTI. Кроме того, для каждой группы сервисов RRTI существует такая часть функциональности, которая в целях эффективности реализована в CRC.

Центральный компонент представляет собой самостоятельное приложение. Для работы программного комплекса, использующего RRTI, требуется запуск одного экземпляра CRC. Пользователями CRC являются другие компоненты RRTI.

2.2 Локальный компонент RRTI (LRC)

Локальный компонент RRTI (LRC) обеспечивает взаимодействие федерата с RRTI в целом, осуществляет локальное управление федератом и созданными им объектами и содержит локальную часть реализации сервисов HLA.

Локальный компонент не является самостоятельным приложением, а представляет собой библиотеку динамической компоновки. Каждое приложение, использующее RRTI, должно быть скомпоновано с библиотекой LRC. Часть ресурсов LRC является общей для всех локальных приложений, однако основные ресурсы выделяются в монопольное использование каждому создаваемому приложениями федерату. В частности, каждый федерат должен запросить у LRC собственную копию объекта RTI Ambassador, через которую осуществляются запросы к RRTI, и предоставить собственную копию объекта Federate Ambassador для обратных вызовов федерата из RRTI. В этом смысле можно говорить, что каждый федерат имеет свою копию LRC.

2.3 Локальный серверный компонент RRTI (LSRC)

Локальный серверный компонент транспорта RRTI (LSRC) предназначен для изоляции LRC и CRC от сетевого транспорта данных. На каждом хосте вычислительной сети размещается один LSRC, обслуживающий локальные LRC и CRC. Он обеспечивает общий доступ компонентов к сетевым ресурсам.

Локальный серверный компонент запускается RRTI автоматически (без участия оператора) в виде отдельного процесса при появлении первого приложения, использующего RRTI на данном хосте, и завершается при отсутствии таких приложений на данном хосте в течение не более чем трех секунд.

2.4 Компонент-провайдер веб-сервиса (WSPRC)

Компонент-провайдер веб-сервиса (WSPRC) – это опциональный компонент, который предназначен для поддержки веб-федератов, использующих WSDL HLA API. Если поддержка веб-федератов не требуется, этот компонент может отсутствовать.

WSPRC - составной компонент, который включает в себя:

- платформу поддержки веб-сервисов Apache AXIS2/C;
- модуль веб-сервиса, разворачиваемый на платформе AXIS2/C (WSAmbassadorService);
- прокси-федерат, представляющий собой приложение, взаимодействующее с модулем веб-сервиса и выступающее в роли федерата для СПО РСРМ (FederateProxyApp);
- библиотеку, реализующую прикладной программный интерфейс к веб-сервису для языка программирования C++.

Для работы WSPRC используется встроенный http-сервер платформы AXIS2/C.

Модуль веб-сервиса обеспечивает обработку запросов веб-федератов. Для каждого такого федерата модуль запускает отдельный экземпляр прокси-федерата, который взаимодействует с RTI посредством LRC.

2.5 Взаимодействие компонентов RRTI

На рисунке 1 представлена типичная структура распределенной моделирующей системы (РМС), использующей RRTI.

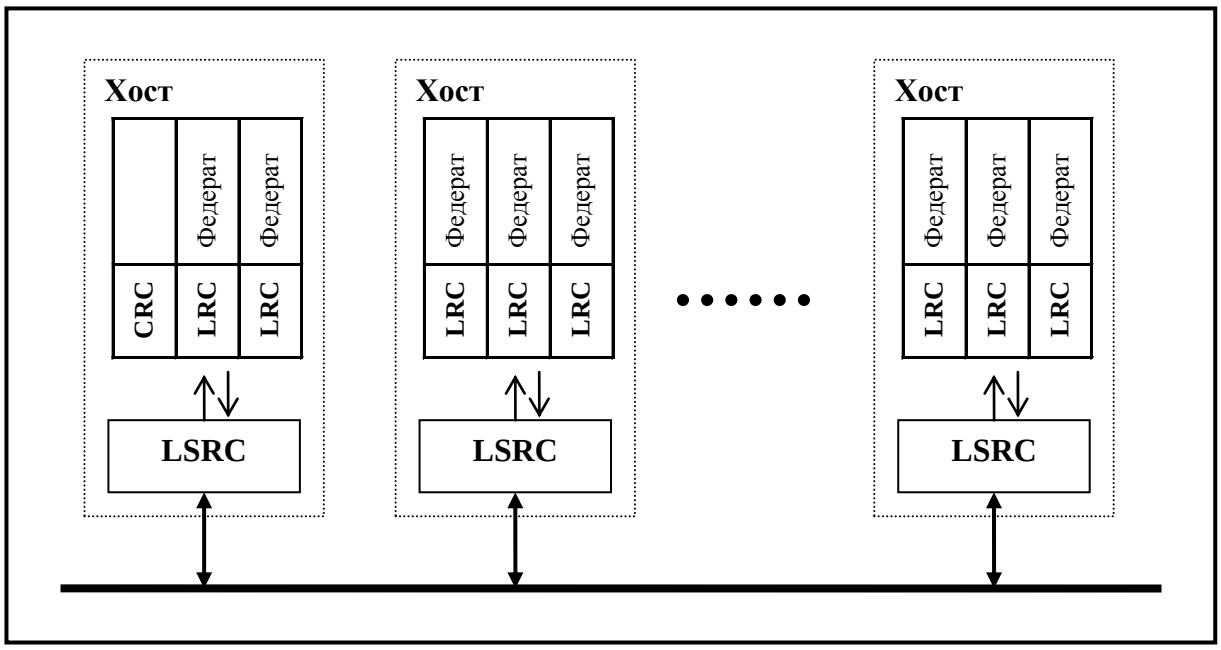

Рисунок 1 – Типовая структура РМС на основе RRTI

Каждый LRC взаимодействует со своим федератом. Кроме того, разные LRC взаимодействуют между собой с помощью сообщений, содержащих служебные или прикладные данные. Каждый LRC также обменивается служебными данными с CRC.

Каждый LSRC взаимодействует с локальными LRC и CRC, а также с LSRC, запущенными на других хостах вычислительной системы.

CRC поддерживает одновременное выполнение нескольких федераций, поэтому на данной схеме различные федераты могут относиться как к одной, так и к разным федерациям.

Рисунок 2 показывает, как меняется типовая структура РМС при наличии веб-федератов.

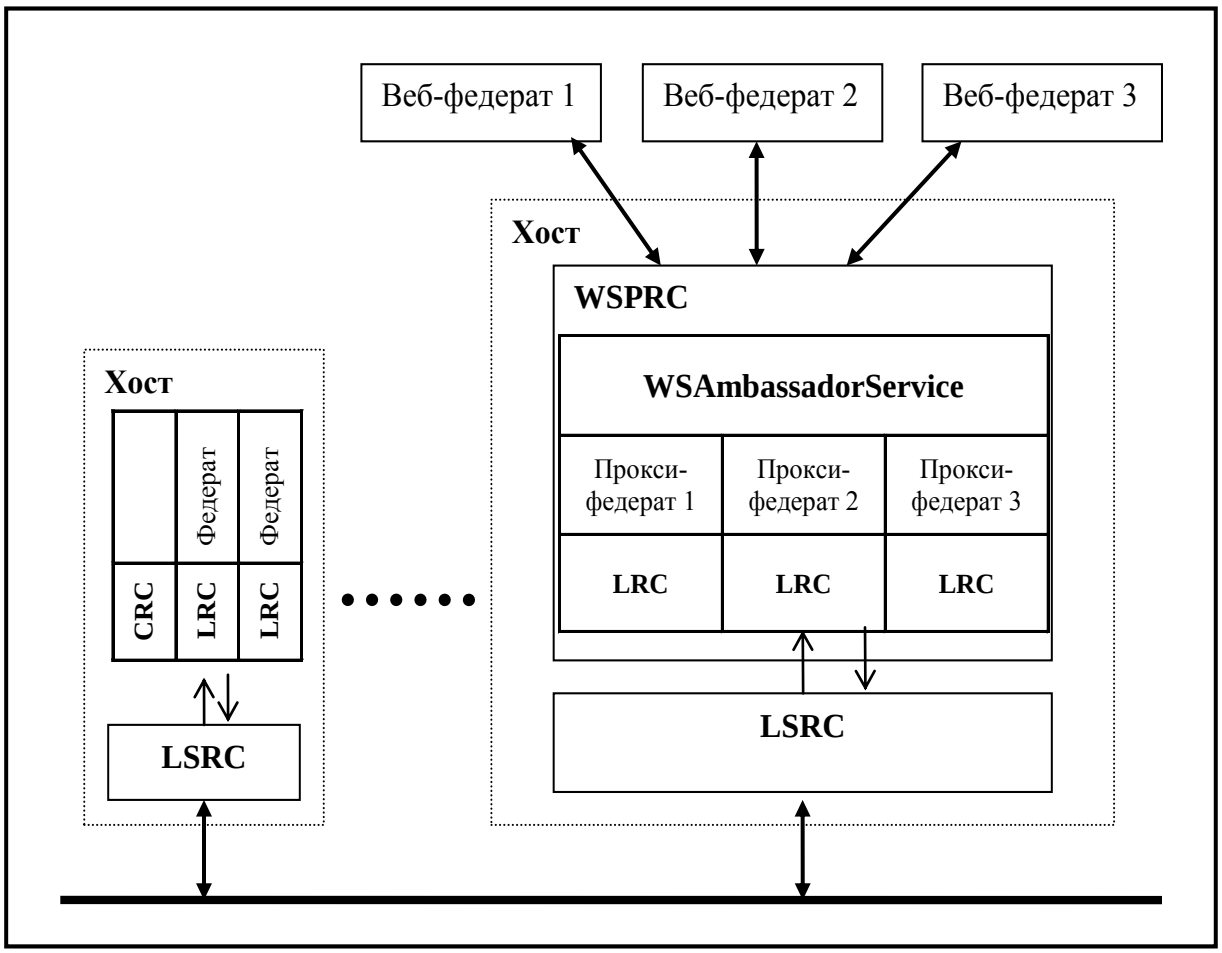

Рисунок 2 – Типовая структура РМС при наличии веб-федератов

3 Установка RRTI на вычислительные средства пользователя

RRTI предназначена для работы на ПЭВМ под управлением одной из следующих операционных систем:

- 32-разрядная версия MS Windows;
- 32-разрядная версия Linux OS;
- 64-разрядная версия Linux OS.

Ниже описывается процесс установки RRTI на вычислительные средства пользователя.

3.1 Установка версии RRTI для MS Windows

3.1.1 Системные требования

Для работы RRTI требуется MS Windows XP Pro с пакетом обновления SP2 или более поздним.

Минимальные требования к техническим средствам:

- процессор – Intel Celeron 2 GHz;
- оперативная память – 512 Mb;
- 350 Mb свободного места на диске.

Рекомендуемые требования к техническим средствам определяются потребностями прикладных задач (федератов), а так же размещением федератов в вычислительной сети (например, предполагается ли использовать одну ПЭВМ для выполнения нескольких федератов).

При размещении Центрального компонента на отдельной ПЭВМ для его работы, как правило, достаточно выполнения минимальных требований к техническим средствам.

Если компьютер, на который устанавливается RRTI, предполагается использовать для разработки федератов, то на нем должна быть установлена среда разработки MS VC++ 8.

3.1.2 Процесс инсталляции

RRTI поставляется пользователям либо в виде файла инсталлятора Windows Installer, либо в виде архива. Процесс инсталляции для каждого из двух вариантов описан ниже.

Перед установкой необходимо удалить предыдущую версию RRTI, если она была установлена.

3.1.2.1 Процесс инсталляции с помощью файла инсталлятора

Процесс установки в целом соответствует типовой установке приложений на платформе MS Windows.

После запуска инсталлятора должно появиться окно с лицензионным соглашением RRTI, с которым пользователь должен согласиться. После этого пользователь может выбрать между установкой по умолчанию и выборочной установкой (Рисунок 3).

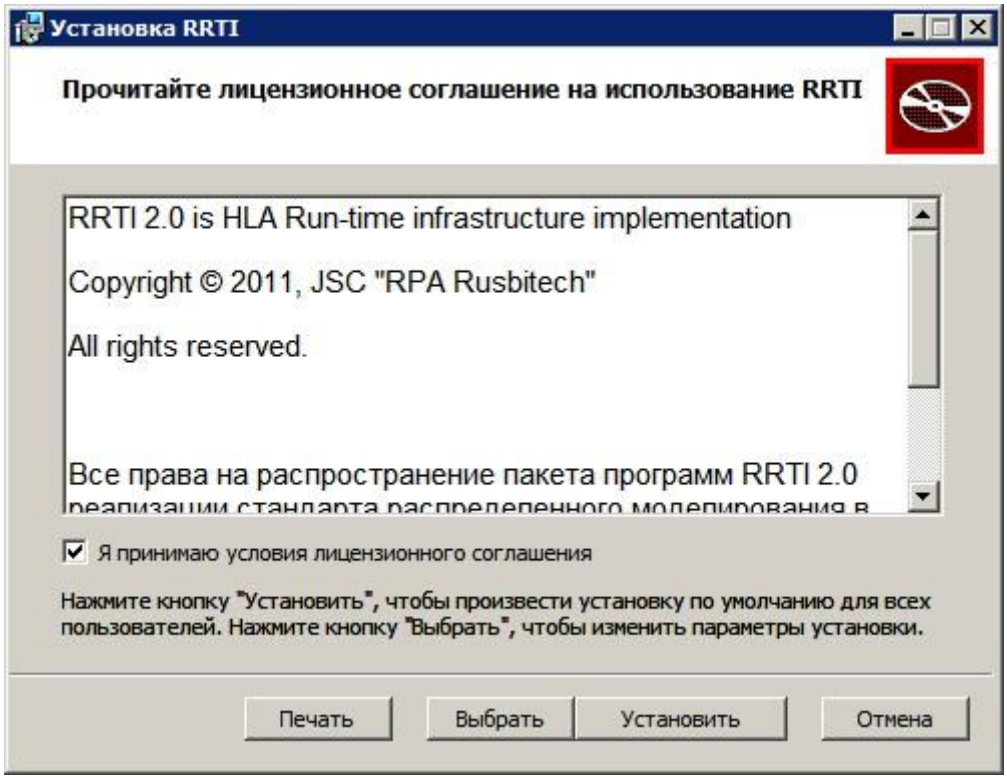

Рисунок 3 – Окно инсталлятора с выбором типа установки

Для установки по умолчанию достаточно активировать кнопку «Установить» и дождаться завершения установки. В этом случае RRTI устанавливается в

типичном объеме (без опционального компонента WSPRC) в каталог RRTI, размещаемый в стандартном для приложений месте файловой системы (обычно это `C:\Program Files\RRTI`).

Если пользователь желает изменить каталог установки, он может активировать кнопку «Выбрать» для выбора места установки и устанавливаемых компонентов. В этом случае появится окно, позволяющее выбрать место установки (Рисунок 4). Пользователь может ввести путь к каталогу вручную, или выбрать его после активации кнопки «Изменить…». После выбора места установки следует активировать кнопку «Далее».

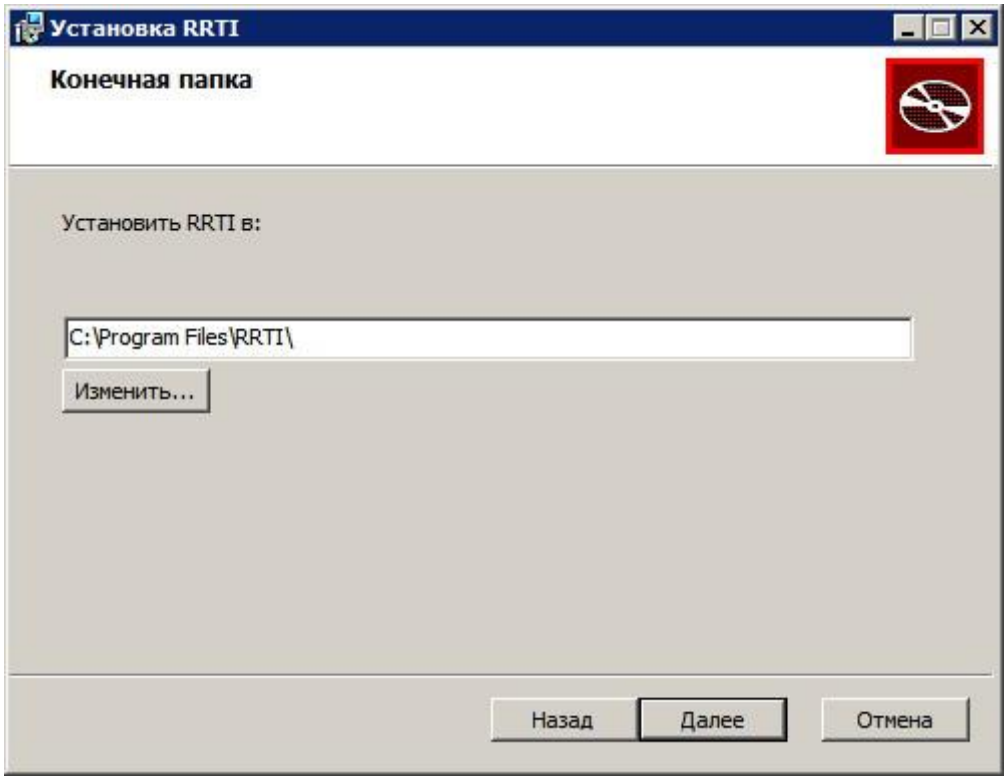

Рисунок 4 – Окно инсталлятора с выбором места установки

Следующее окно (Рисунок 5) позволяет выбрать компоненты, которые требуется установить на данной платформе. Для отмены установки какого-либо компонента требуется активировать меню с иконкой диска и выбрать пункт «Все компоненты данной функции будут недоступны». Завершив выбор, следует активировать кнопку «Установить» и дождаться завершения установки (Рисунок 6).

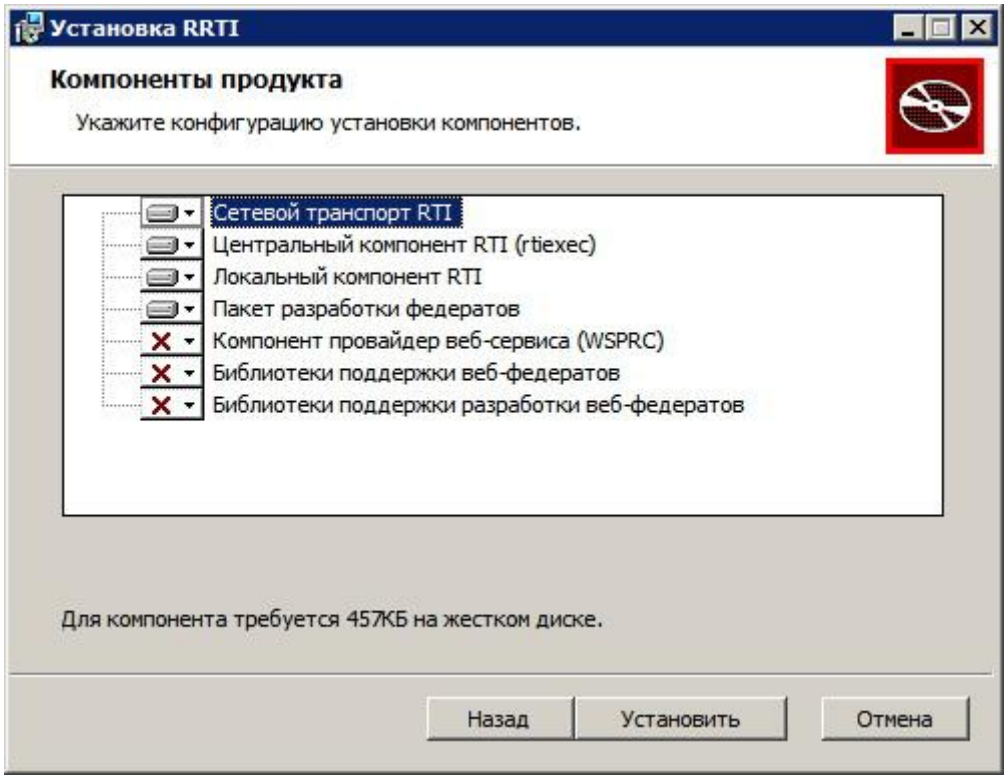

Рисунок 5 – Окно инсталлятора с выбором устанавливаемых компонентов

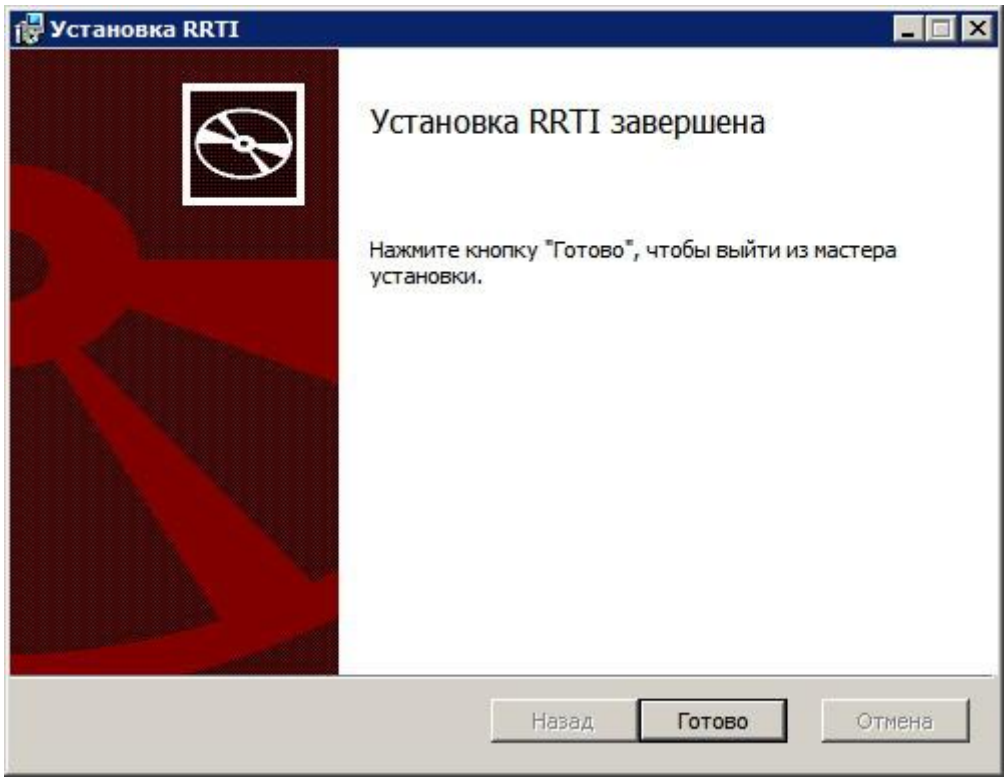

Рисунок 6 – Окно завершения установки

После завершения установки пользователь получает предложение перезагрузить компьютер для вступления новых параметров в силу. Следует согласиться с предложением перед использованием RRTI, поскольку инсталлятор настраивает переменные среды RRTI, новые значения которых будут действовать после перезагрузки.

В результате установки в главном меню MS Windows, открываемом через кнопку «Пуск», появляется группа «RRTI», содержащая ярлыки для установленных компонентов.

3.1.2.2 Процесс инсталляции из архива готового к выполнению ПО

RRTI поставляется в виде файла архива. Процесс инсталляции заключается в выполнении следующих шагов:

- создать директорию, в которой будет установлена RRTI (например, `C:\Runtime\rrti`); запускаемые компоненты RRTI должны иметь права на запись в эту директорию и ее поддиректории;
- распаковать архив в созданную директорию.

Опциональный компонент – провайдер веб-сервиса WSPRC поставляется в виде отдельного архива. Если на целевой платформе требуется развернуть WSPRC, то следует выполнить следующие шаги:

- создать директорию, в которой будут установлены компоненты WSPRC (например, `C:\Runtime\rrti-wsprc`);
- распаковать архив в созданную директорию.

Установка WSPRC требуется только в случае, если предполагается работа с веб-федератами, и только на том вычислительном средстве, которое будет эксплуатироваться в качестве сервера (провайдера веб-сервиса).

3.1.3 Настройка переменных среды RRTI

Инсталлятор автоматически настраивает необходимые переменные среды. Однако, если ПО RRTI устанавливалось из файла архива, то после установки необходимо дополнительно настроить две переменные среды:

- *RRTI_HOME* – должна содержать полный путь к директории, в которую установлена RRTI (например, `C:\Runtime\rrti`);
- *PATH* – должна содержать путь к поддиректории bin директории установки RRTI.

Кроме того, если на компьютер устанавливается опциональный компонент WSPRC, необходимо дополнительно настроить следующие переменные среды:

- *AXIS2C_HOME* – должна содержать полный путь к директории, в которую установлена платформа Apache AXIS2C (например, `C:\Runtime\rrti-wsprc`\axis2c-bin-1.6.0-win32-RRTI2);

- *PATH* – должна содержать пути к каталогам `lib` и `bin` установленной платформы AXIS2C.

Для автоматической настройки этих переменных среды можно воспользоваться скриптами, которые имеются в дистрибутиве.

Для автоматической настройки среды RRTI требуется перейти в подкаталог `bin` каталога, в который установлена RRTI (например, `C:\Runtime\rrti\bin`), и запустить на выполнение скрипт `conf_rti_win.vbs`. Если на целевом компьютере в переменных среды есть настройки для установленной ранее версии RRTI, они будут заменены настройками для устанавливаемой версии. После выполнения настройки скрипт выведет новые значения переменных.

Далее, если на компьютер также устанавливается опциональный компонент WSPRC, требуется перейти в каталог его установки (например, `C:\Runtime\rrti-wsprc`) и запустить на выполнение скрипт `conf_wsprc_win.vbs`. Если на целевом компьютере в переменных среды есть настройки для установленной ранее версии WSPRC, они будут заменены настройками для устанавливаемой версии. После выполнения настройки скрипт выведет новые значения переменных.

При необходимости могут быть вручную дополнительно заданы следующие переменные среды:

- `CRC_CONFIG` – полный путь к файлу конфигурации CRC;

- `LSRC_CONFIG` – полный путь к файлу конфигурации LSRC;

- `LRC_CONFIG` – полный путь к файлу конфигурации LRC.

Полный путь к файлу должен включать в себя имя файла конфигурации. Если эти переменные не заданы, то используются стандартные имена файлов конфигурации: `crc.cfg`, `lsrc.cfg`, `lrc.cfg`.

Поиск файлов конфигурации для компонента RRTI осуществляется в следующем порядке:

- файл со стандартным именем в текущем каталоге программы (только LRC и CRC);
- файл, на который ссылается соответствующая компоненту переменная среды (если значение установлено);
- файл со стандартным именем в каталоге `$(RRTI_HOME)/conf`.

Более подробно работа с файлами конфигурации описана в главе 4.

После установки переменных среды необходимо перегрузить компьютер либо повторно войти в учетную запись пользователя.

3.2 Установка версии RRTI для Linux

3.2.1 Системные требования

Минимальные требования к техническим:

- процессор – Intel Celeron 1 GHz;
- оперативная память – 512 Mb;
- 200 Mb свободного места на диске.

Рекомендуемые требования к техническим средствам определяются потребностями прикладных задач (федератов), а так же размещением федератов в вычислительной сети.

Требования к программным средствам:

- ядро Linux 2.6.17 или старше (совместимого с библиотекой NPTL, поддерживающей стандарт POSIX.1 [IEEE Std 1003.1,2004 Edition] в части потоков процессов и примитивов синхронизации, совместно используемых несколькими процессами);
- локаль UTF8.

Для работы веб-сервиса, обеспечивающего подключение веб-федератов через WSDL-интерфейс, необходимы следующие библиотеки:

- libxml2 версии не ниже 2.7.7;
- iconv версии не ниже 1.9.2;
- zlib версии не ниже 1.2.5.

Для корректного определения компонентами RRTI IP-адреса хоста следует также убедиться, что:

- имя хоста указано в файле конфигурации ОС (обычно это `/etc/sysconfig/network`) и оно отлично от `localhost`;
- в файле /etc/hosts для имени данного хоста указан IP-адрес, соответствующий тому сетевому интерфейсу, который будет использоваться компонентами RRTI для идентификации при обмене данными.

Альтернативный способ выбора сетевого интерфейса для транспорта данных предоставляет конфигурационный параметр Локального серверного компонента RRTI - TRANSPORT.OWN_ADDRESS (см. главу 4). При его корректной установке выполнение предыдущего требования является необязательным.

В связи с большим разнообразием существующих вариантов Linux, работа RRTI не может быть гарантирована для всех этих вариантов. Существует ограниченный список дистрибутивов, для которых возможно получить версию RRTI. Этот список может изменяться, поэтому доступность RRTI для того или иного варианта Linux должна уточняться дополнительно у компании-разработчика RRTI.

3.2.2 Процесс инсталляции

Для инсталляции RRTI на Linux используется архивный файл rrti-X.X.XXXX.tgz, где X.X.XXXX обозначает номер версии и номер релиза центрального компонента. Процесс инсталляции заключается в выполнении следующих шагов:

- создать директорию, в которой будет установлена RRTI (например, `/user/home/name_user/rrti`); запускаемые компоненты RRTI должны иметь права на запись в эту директорию и ее поддиректории;
- скопировать архив rrti-X.X.XXXX.tgz в созданную директорию;
- распаковать скопированный архив в созданной директории, воспользовавшись командой

tar –xzf rrti-X.X.XXXX.tgz

Опциональный компонент – провайдер веб-сервиса WSPRC поставляется в виде отдельного архива rrti-wsprc-X.X.XXXX.tgz, где X.X.XXXX обозначает номер версии и номер релиза. Если на целевой платформе требуется развернуть веб-сервис, то следует выполнить следующие шаги:

- создать директорию, в который будут установлены компоненты WSPRC (например, `/user/home/name_user/rrti-wsprc`);
- скопировать архив `rrti-wsprc-X.X.XXXX.tgz` в созданную директорию;

- распаковать скопированный архив в созданной директории, воспользовавшись командой

tar –xzf rrti-wsprc-X.X.XXXX.tgz

Веб-сервис требуется развернуть только в случае, если предполагается работа с веб-федератами, и только на том вычислительном средстве, которое будет эксплуатироваться в качестве сервера (провайдера веб-сервиса).

3.2.3 Настройка переменных среды RRTI

После распаковки RRTI в целевой каталог, минимально необходимо настроить переменную среды *RRTI_HOME*, которая должна содержать полный путь к директории, в которую установлена RRTI (например, `/user/home/name_user/rrti`). Рекомендуется вставить установку этой переменной в файл, который запускается при логине, например, в файл .bash_profile добавить строку:

export RRTI_HOME=/user/home/name_user/rrti

При необходимости аналогичным путем могут быть дополнительно заданы следующие переменные среды:

- `CRC_CONFIG` – полный путь к файлу конфигурации CRC;
- `LSRC_CONFIG` – полный путь к файлу конфигурации LSRC;
- `LRC_CONFIG` – полный путь к файлу конфигурации LRC.

Полный путь к файлу должен включать в себя имя файла конфигурации. Если эти переменные не заданы, то используются стандартные имена файлов конфигурации: `crc.cfg`, `lsrc.cfg`, `lrc.cfg`.

Поиск файлов конфигурации для компонента RRTI осуществляется в следующем порядке:

- файл со стандартным именем в текущем каталоге программы (только LRC и CRC);
- файл, на который ссылается соответствующая компоненту переменная среды (если значение установлено);
- файл со стандартным (или заданным в приложении) именем в каталоге *$(RRTI_HOME)/conf*.

Работа с файлами конфигурации подробно описана в главе 4.

Настройка переменных среды для работы опционального компонента WSPRC не требуется.

4 Конфигурирование компонентов RRTI

В этой главе описываются параметры конфигурации компонентов RRTI, которые позволяют настроить функции и производительность RRTI.

Следует обратить внимание, что при использовании значений параметров по умолчанию часть функциональности RRTI, которая используется реже, отключена. Причина заключается в том, что если явно объявить о том, что какая-то часть функциональности HLA приложениями не востребована, то производительность RRTI может быть улучшена. В частности, может быть отключена следующая функциональность:

- поддержка MOM;
- поддержка сохранения/восстановления федераций;
- поддержка частот обновления атрибутов;
- поддержка управления владением атрибутами;
- поддержка управления логическим временем;
- поддержка фильтрации по регионам (DDM).

Если отключенная функциональность необходима, требуется изменить значения соответствующих параметров конфигурации. Поддержка MOM конфигурируется в CRC и распространяется на все создаваемые федерации. Прочие указанные функции могут настраиваться в LRC отдельно и независимо для разных федератов.

4.1 Формат конфигурационных файлов RRTI.

Каждый из трех основных компонентов RRTI (CRC, LRC и LSRC) имеет свой конфигурационный файл. После установки RRTI эти файлы с настройками по умолчанию размещены в каталоге `$(RRTI_HOME)/conf`. Пользователи могут скопировать их для использования в иное место или модифицировать сами исходные файлы.

Файл конфигурации представляет собой текстовый файл, каждая значимая строка которого должна иметь вид:

<Имя параметра>=<Значение параметра>.

Начало значимой строки до первого вхождения символа равенства трактуется как имя конфигурационного параметра, а оставшаяся часть строки – как его значение. Пробелы в названии конфигурационного параметра и его значении удаляются. Например, следующие строки трактуются одинаково:

"параметр1=значение"

"параметр1 = значение "

"п а р а м е т р 1 = з н а ч е н и е".

Имя параметра состоит из имени группы и собственного имени параметра, разделенных знаком « . ».

Длина строки не должна превышать 2 КБ. Пустые строки игнорируются. Строка, начинающаяся с символа «#», трактуется как комментарий.

4.2 Конфигурирование центрального компонента RRTI

Для работы RRTI требуется, чтобы в вычислительной сети функционировал один экземпляр CRC. С целью достижения максимальной производительности рекомендуется выделить для CRC отдельный узел вычислительной сети. В этом случае CRC не будет конкурировать за вычислительные ресурсы с LRC и федератами, что позволит ему максимально быстро отвечать на запросы компонентов RRTI.

Исполняемый модуль CRC называется *rtiexec*. Управление его работой возможно с помощью файла конфигурации. CRC осуществляет поиск файла конфигурации в следующей последовательности:

- файл с именем *crc.cfg* в текущем каталоге *rtiexec*;
- файл, на который ссылается переменная среды CRC_CONFIG (если значение установлено);
- файл с именем *crc.cfg* в каталоге $(*RRTI_HOME*)/*conf*.

Для всех параметров Центрального компонента RRTI имя группы – «CRC». Таблица 3 содержит конфигурационные параметры, доступные для управления Центральным компонентом RRTI.

Таблица 3 – Конфигурационные параметры CRC

Имя параметра	Описание	Возможные значения	Значение по умолчанию
CRC.LOG_LEVEL_CONSOLE	Минимальная важность сообщений, выводимых на консоль оператора	НЕ_ВЫВОДИТЬ БЕЗУСЛОВНОЕ ОШИБКА ПРЕДУПРЕЖДЕНИЕ ИНФОРМАЦИОННОЕ КОНФИГУРАЦИЯ ОТЛАДКА ОТЛАДКА_2	ОШИБКА
CRC.LOG_LEVEL_FILE	Минимальная важность сообщений, выводимых в журнал	НЕ_ВЫВОДИТЬ БЕЗУСЛОВНОЕ ОШИБКА ПРЕДУПРЕЖДЕНИЕ ИНФОРМАЦИОННОЕ КОНФИГУРАЦИЯ ОТЛАДКА ОТЛАДКА_2	ОШИБКА
CRC.STATISTICS_LEVEL	Управление выводом статистики обработки сообщений (ресурсоемкое, используется для тестирования)	ПОДРОБНО ИТОГ НЕТ	НЕТ
CRC.USE_HEARTBEATS	Использование маяков для контроля целостности федерации	ДА НЕТ	ДА
CRC. CHECK_INTEGRITY_PERIOD	Цикличность выполнения задачи контроля целостности федерации в миллисекундах	Целое в интервале 2000 – 10000	10000
CRC. DEFAULT_RESIGN_ACTION	Действие по отношению к объектам, принадлежащим федерату, при завершении федерата без вызова resign и явного указания ResignAction. Применяется для старых федератов (формата RRTI1). Для федератов формата Evolved значение игнорируется и берется из FOM	UNCONDITIONALLY_DIVEST_ATTRIBUTES DELETE_OBJECTS CANCEL_PENDING_OWNERSHIP_ACQUISITIONS DELETE_OBJECTS_THEN_DIVEST CANCEL_THEN_DELETE_THEN_DIVEST NO_ACTION	DELETE_OBJECTS
CRC. FEDERATE_INFO_BROADCAST	Рассылка информации об изменении состава федерации всем LRC. Должен быть включен, если требуется поддержка полной функциональности сервисов HLA getFederateId и getFederateName (иначе эти сервисы поддерживаются только для своего федерата)	ДА НЕТ	НЕТ

Окончание таблицы 3

Имя параметра	Описание	Возможные значения	Значение по умолчанию
CRC.SUPPORT_MOM	Поддержка функциональности MOM. Только для федераций Evolved. Для федератов формата RRTI1 игнорируется	ДА НЕТ	НЕТ
CRC. FOM_MODULE_DATA_REQUEST	Поддержка функциональности MOM в части запросов содержимого модулей FOM (HLArequestFOMmoduleDat a). Для включения требуется также включение CRC.SUPPORT_MOM	ДА НЕТ	НЕТ
CRC.SAVE_RESTORE_PATH	Путь для размещения точек сохранения федерации. Если нет значения, сохраняется в подкаталог save установки	Допустимый в файловой системе путь	Пусто
CRC. SAVE_RESTORE_TIMEOUT	Максимально допустимое время в миллисекундах для процессов сохранения и восстановления федерации, по истечении которого операция прерывается. Если задано 0, время считается не ограниченным	Целое в интервале 0 – 600000	60000
CRC. READ_TRANSPORT_QUEUE_ TIMEOUT	Тайм-аут чтения сообщений из очередей транспорта в миллисекундах	Целое в интервале 100 – 10000	1000
CRC. PENDING_FED_DATA_ TIMEOUT	Тайм-аут ожидания данных федерата в миллисекундах	Целое в интервале 1000 – 60000	10000
CRC. TRANSPORT_ADDRESMAP_ TIMEOUT	Тайм-аут некоторых операций доступа к карте адресации транспорта (в миллисекундах). 0 – бесконечно.	Целое в интервале 0 – 1000	0
CRC.TRANSPORT_USE_MLOCK	Использовать ли mlock и аналоги для блокирования в оперативной памяти некоторых страниц	ДА НЕТ	НЕТ
CRC.TRANSPORT_FILTER_ SIGNALS	Использовать ли фильтрацию сигналов	ДА НЕТ	ДА

Описания параметров, управляющих выводом сообщений, приведены в главе 5.

Если параметр `CRC.USE_HEARTBEATS` имеет значение «ДА», то CRC осуществляет контроль целостности федерации, периодически проверяя наличие сообщений-маяков от тех выполняющихся федератов, для которых сконфигурирована отправка маяков. Период проверок устанавливается параметром `CRC.CHECK_INTEGRITY_PERIOD` (значение необходимо устанавливать согласованно с конфигурационным параметром LRC `LRC.HEARTBEAT_INTERVAL`). При отсутствии маяков CRC инициирует исключение сбойного федерата из федерации. Если в режиме отладок федератов разработчики используют отладчики, устанавливая точки останова или выполняя федераты по шагам, то необходимо отключить контроль маяков либо в конфигурации LRC соответствующего федерата, либо для всех федератов сразу, установив для параметра `CRC.USE_HEARTBEATS` значение «`HET`».

Отключение контроля маяков рекомендуется также для тех федератов, которые потребляют все доступные вычислительные ресурсы своего хоста (характерно для некоторых задач синтеза изображений). Задача отправки маяков является низкоприоритетной. При полной загруженности хоста для ее выполнения не остается ресурсов. Как следствие, если контроль маяков не отключить, CRC будет рассматривать такие нагруженные хосты как аварийные, поскольку от них не поступают маяки.

В остальных случаях контроль маяков рекомендуется оставлять включенным. Это позволит контролировать такие ситуации, как зацикливание приложения или перегруженность хостов федератов.

Аварийное завершение федератов также контролируется на транспортном уровне RRTI и может быть обнаружено CRC независимо от использования контроля маяков.

Параметр `CRC.DEFAULT_RESIGN_ACTION` определяет, что происходит с объектами, для которых федерат имеет права на удаление, когда федерат завершается нештатно, т.е. без вызова сервиса resign. Применяется для старых федератов (формата RRTI1). Для федератов формата Evolved значение игнорируется и берется из FOM. Список значений - это стандартные значения `enum ResignAction` из интерфейса HLA.

Параметр `CRC.FEDERATE_INFO_BROADCAST` определяет, будет ли CRC рассылать на LRC информацию о присоединившихся или отключившихся федератах. Должен быть включен, если требуется поддержка полной функциональности сервисов HLA `getFederateId` и `getFederateName` (иначе эти сервисы поддерживаются только для своего федерата). Если полная функциональность этих сервисов не требуется, то отключение рассылки позволяет разгрузить вычислительные и сетевые ресурсы в периоды изменения состава федерации.

Параметр `CRC.SUPPORT_MOM` определяет, должен ли CRC поддерживать функциональность объектной модели управления (MOM). Если хотя бы один федерат в той или иной федерации использует MOM, требуется разрешить поддержку MOM в CRC.

Полная поддержка MOM включает в себя редко используемые запросы состава модулей FOM. Если такие запросы используются, следует дополнительно включить их поддержку с помощью параметра `CRC.FOM_MODULE_DATA_REQUEST`.

Параметр `CRC.SAVE_RESTORE_PATH` позволяет задать путь для хранения точек сохранения федерации. По умолчанию эта информация сохраняется в подкаталог save каталога установки RRTI.

Параметр `CRC.SAVE_RESTORE_TIMEOUT` определяет время, за которое должен завершиться процесс сохранения или восстановления федерации. Если тайм-аут превышен, CRC будет считать, что процесс завершился неудачно и должен быть прерван.

Параметр `CRC.READ_TRANSPORT_QUEUE_TIMEOUT` определяет тайм-аут ожидания сообщения при чтении входной очереди сообщений. Изменять его значение не рекомендуется.

Параметр `CRC.PENDING_FED_DATA_TIMEOUT` определяет время, в течение которого CRC ожидает данные объектной модели федерации от федерата, запросившего создание федерации. В глобальных сетях с большими задержками доставки данных может потребоваться увеличение значения этого параметра, если наблюдаются проблемы при создании федераций федератами.

Значение параметра `CRC.TRANSPORT_ADDRESMAP_TIMEOUT` следует всегда оставлять равным 0 (значение по умолчанию), иначе может пострадать устойчивость работы RRTI под большой нагрузкой.

Значение «ДА» для параметра `CRC.TRANSPORT_USE_MLOCK` может незначительно увеличить быстродействие RRTI, однако при этом в журнал могут начать попадать предупреждения, вызванные ограничениями на объем блокируемой процессом памяти.

Значение «ДА» параметра `CRC.TRANSPORT_FILTER_SIGNALS` ускоряет работу транспорта при очень большом (несколько десятков) количестве LRC, выполняющихся на одном хосте с CRC.

В момент инициализации транспорта помимо файла конфигурации CRC используется и файл конфигурации LSRC, из которого берется значение параметра `TRANSPORT.OWN_ADDRESS` (см. раздел 4.4). При наличии на хосте нескольких сетевых адаптеров значение параметра должно быть равно адресу того адаптера, через который доступны остальные хосты, где выполняются компоненты RRTI. При запуске RRTI локально на хосте с несколькими сетевыми адаптерами может быть использовано значение 127.0.0.1.

4.3 Конфигурирование локального компонента RRTI

Управление работой LRC возможно с помощью файла конфигурации. Разные федераты могут использовать разные настройки LRC.

LRC осуществляет поиск файла конфигурации в следующей последовательности:

- файл с именем *`lrc.cfg`* в текущем каталоге приложения, использующего LRC;
- файл, на который ссылается переменная среды `LRC_CONFIG` (если значение установлено);
- файл с именем *`lrc.cfg`* в каталоге `$(RRTI_HOME)/conf`.

Для параметров Локального компонента RRTI используются группы: «`LRC`» и «`RTI_SERVICES`». Таблица 4 содержит их описание

Таблица 4 – Конфигурационные параметры LRC

Имя параметра	Описание	Возможные значения	Значение по умолчанию
LRC. LOG_LEVEL_CONSOLE	Минимальная важность сообщений, выводимых на консоль оператора	НЕ_ВЫВОДИТЬ БЕЗУСЛОВНОЕ ОШИБКА ПРЕДУПРЕЖДЕНИЕ ИНФОРМАЦИОННОЕ КОНФИГУРАЦИЯ ОТЛАДКА ОТЛАДКА_2	ОШИБКА
LRC.LOG_LEVEL_FILE	Минимальная важность сообщений, выводимых в журнал	НЕ_ВЫВОДИТЬ БЕЗУСЛОВНОЕ ОШИБКА ПРЕДУПРЕЖДЕНИЕ ИНФОРМАЦИОННОЕ КОНФИГУРАЦИЯ ОТЛАДКА ОТЛАДКА_2	ОШИБКА
LRC. STATISTICS_LEVEL	Управление выводом статистики обработки сообщений LRC и федератом (ресурсоемкое, используется для тестирования)	ПОДРОБНО ИТОГ НЕТ	НЕТ
RTI_SERVICES. USE_SAVE_RESTORE	Использование федератом сервисов Save/Restore	ДА НЕТ	НЕТ
RTI_SERVICES. USE_UPDATE_RATES	Использование федератом частот подписки на атрибуты	ДА НЕТ	НЕТ
RTI_SERVICES. USE_OWNERSHIP_MANAGEMENT	Использование федератом сервисов управления владением	ДА НЕТ	НЕТ
RTI_SERVICES. USE_TIME_MANAGEMENT	Использование федератом сервисов управления логическим временем	ДА НЕТ	НЕТ
RTI_SERVICES. USE_DDM	Использование федератом сервисов DDM	ДА НЕТ	НЕТ
LRC.USE_HEARTBEATS	Использование маяков для контроля целостности федерации	ДА НЕТ	ДА
LRC. HEARTBEAT_INTERVAL	Интервал (в миллисекундах), с которым задача контроля целостности LRC отправляет маяки CRC	Целое в интервале 1000 – 5000	5000

Имя параметра	Описание	Возможные значения	Значение по умолчанию
LRC. FOM_SCHEMA_LOCATION	Путь к схеме, определяющей правила для объектной модели. Если пусто, берется из стандартных ресурсов RRTI	Путь размещения схемы в формате, требуемом правилами задания схемы	Пусто

Окончание таблицы 4

Имя параметра	Описание	Возможные значения	Значение по умолчанию
LRC. PENDING_CRC_DATA_ TIMEOUT	Таймаут ожидания ответа CRC в миллисекундах	Целое в интервале 1000 – 60000	10000
LRC.READ_TRANSPORT_ QUEUE_TIMEOUT	Таймаут чтения сообщений из очередей транспорта в миллисекундах	Целое в интервале 100 – 10000	1000
LRC.PRIORITY_SCHEME	Схема управления приоритетами для потоков LRC	ПРИОРИТЕТ_ВХОДЯЩИ Х БЕЗ_ПРИОРИТЕТА	ПРИОРИТЕТ _ВХОДЯЩИХ
LRC.TREAT_RELIABLE_ AS_LOW_LATENCY	Стандартный тип транспортировки RELIABLE рассматривается как LOW_LATENCY_RELIABLE (иначе как BULK_VOLUME_RELIABLE)	ДА НЕТ	ДА
LRC. TREAT_BEST_EFFORT_ AS_BULK_VOLUME	Стандартный тип транспортировки BEST_EFFORT рассматривается как BULK_VOLUME_BEST_EFFORT (иначе как BULK_VOLUME_RELIABLE)	ДА НЕТ	ДА
LRC.TRANSPORT_ ADDRESMAP_TIMEOUT	Тайм-аут некоторых операций доступа к карте адресации транспорта (в миллисекундах). 0 – бесконечно	Целое в интервале 0 – 1000	0
LRC.TRANSPORT_CRC_ WAIT_TIMEOUT	Тайм-аут (в миллисекундах) ожидания локальным транспортом появления CRC в карте адресации	Целое в интервале 1000 - MAXINT	6000
LRC. TRANSPORT_USE_MLOCK	Использовать ли mlock и аналоги для блокирования в оперативной памяти некоторых страниц	ДА НЕТ	НЕТ
LRC. TRANSPORT_FILTER_ SIGNALS	Использовать ли фильтрацию сигналов	ДА НЕТ	ДА

Описания параметров, управляющих выводом сообщений, приведены в главе 5.

Параметры группы RTI_SERVICES определяют, какие группы сервисов могут использоваться федератом. В стандарте HLA есть сервисы, которые используются практически всеми федератами, и сервисы, используемые реже. По умолчанию эти реже используемые сервисы отключены, что позволяет улучшить производительность системы. Если федерату требуются отключенные сервисы,

следует задать значение «ДА» для соответствующей группы в файле конфигурации.

Если параметр LRC.USE_HEARTBEATS имеет значение «ДА», то LRC осуществляет периодическую посылку сообщений маяков CRC, позволяя контролировать целостность федерации. Период посылки маяков устанавливается параметром LRC.HEARTBEAT_INTERVAL (значение необходимо устанавливать согласовано с конфигурационным параметром CRC CRC.CHECK_INTEGRITY_PERIOD). При отсутствии маяков CRC инициирует исключение сбойного федерата из федерации. Если в режиме отладок федератов разработчики используют отладчики, устанавливая точки останова или выполняя федераты по шагам, то необходимо отключить отправку маяков, установив для параметра LRC.USE_HEARTBEATS значение «НЕТ». Для федерата с отключенной отправкой маяков CRC не контролирует их получение.

Отключение отправки маяков рекомендуется так же для тех федератов, которые потребляет все доступные вычислительные ресурсы своего хоста (характерно для некоторых задач синтеза изображений). Задача отправки маяков является низкоприоритетной. При полной загруженности хоста для ее выполнения не остается ресурсов. Как следствие, если контроль маяков не отключить, CRC будет рассматривать такие нагруженные хосты как аварийные, поскольку от них не поступают маяки.

В остальных случаях контроль маяков рекомендуется оставлять включенным. Это позволит контролировать такие ситуации, как зацикливание приложения или перегруженность хостов федератов.

Параметр LRC.FOM_SCHEMA_LOCATION позволяет указать путь к схеме объектной модели федерации. Значение должны задаваться в том же формате, в котором оно задано для параметра schemaLocation стандартного модуля MIM. По умолчанию схема берется из ресурсов RRTI

Параметр LRC.PENDING_CRC_DATA_TIMEOUT определяет время, в течение которого LRC ожидает ответ от CRC при выполнении федератом блокирующих вызов HLA (то есть таких вызовов, которые блокируют текущий поток LRC до получения ответа от CRC). В глобальных сетях с большими задержками доставки данных может потребоваться увеличение значения этого параметра, если у

федератов наблюдаются проблемы с получением ответов от CRC при блокирующих вызовах HLA, таких как создание федерации или присоединение к ней. При таких проблемах в журнале LRC появляются записи вида «Таймаут блокирующего запроса к CRC».

Параметр `LRC.READ_TRANSPORT_QUEUE_TIMEOUT` определяет таймаут ожидания сообщения при чтении входной очереди сообщений.

Параметр `LRC.PRIORITY_SCHEME` позволяет отключать стандартную схему приоритетов потоков LRC. Стандартная схема приоритетов в RRTI предполагает, что федерату требуется обеспечить возможность быстрее реагировать на входящие события и данные. Поэтому потоки чтения входной очереди LRC и поток обработки обратных вызовов федерата (в модели неотложных обратных вызовов) имеют повышенный приоритет. Задача отправки маяков контроля целостности федерации, наоборот, имеет пониженный приоритет.

Такая схема может быть удачной не для всех федератов. Иногда большой поток входящих событий мешает федерату выполнять собственную работу. Стандартная схема приоритетов может быть отключена для выбранных федератов с помощью файла конфигурации LRC, параметр `PRIORITY_SCHEME`. Если выбрано значение параметра `БЕЗ_ПРИОРИТЕТА`, то все потоки LRC выполняются с одинаковым приоритетом, равным приоритету вызывающего федерата.

Параметры `LRC.TREAT_RELIABLE_AS_LOW_LATENCY` и `LRC.TREAT_BEST_EFFORT_AS_BULK_VOLUME` определяют интерпретацию стандартных типов транспортировки данных.

Разница между `LOW_LATENCY` и `BULK_VOLUME` в том, что данные `LOW_LATENCY` отправляются без задержек, а `BULK_VOLUME` собираются транспортом в пакеты. Поэтому `LOW_LATENCY` обеспечивают лучшее время отклика, а `BULK_VOLUME` – лучшую пропускную способность.

Стандартные типы транспортировки HLA интерпретируется по умолчанию так:

- `RELIABLE` = `LOW_LATENCY_RELIABLE;`
- `BEST_EFFORT` = `BULK_VOLUME_BEST_EFFORT.`

Схема интерпретации стандартных типов транспортировки может быть изменена в файле конфигурации LRC с помощью указанных параметров.

Значение параметра `LRC.TRANSPORT_ADDRESMAP_TIMEOUT` следует всегда оставлять равным 0 (значение по умолчанию), иначе может пострадать устойчивость работы RRTI под большой нагрузкой.

Параметр `LRC.TRANSPORT_CRC_WAIT_TIMEOUT` позволяет увеличить величину ожидания CRC в момент инициализации транспорта данных. Это может быть полезно, например, при расположении CRC на удаленном хосте.

Значение «ДА» для параметра `LRC.TRANSPORT_USE_MLOCK` может незначительно увеличить быстродействие RRTI, однако при этом в журнал могут начать попадать предупреждения, вызванные ограничениями на объем блокируемой процессом памяти.

Значение «ДА» параметра `LRC.TRANSPORT_FILTER_SIGNALS` ускоряет работу транспорта при очень большом (несколько десятков) количестве LRC, выполняющихся на одном хосте.

В момент инициализации транспорта данных помимо файла конфигурации LRC используется и файл конфигурации LSRC, из которого берется значение параметра `TRANSPORT.OWN_ADDRESS` (см. раздел 4.4). При наличии на хосте нескольких сетевых адаптеров значение параметра должно быть равно адресу того адаптера, через который доступны остальные хосты, где выполняются компоненты RRTI. При запуске RRTI локально на хосте с несколькими сетевыми адаптерами может быть использовано значение 127.0.0.1.

4.4 Конфигурирование локального серверного компонента RRTI

Для работы RRTI требуется, чтобы на каждом хосте вычислительной сети работал компонент LSRC. Исполняемый модуль LSRC называется *`lsrc`*. Он запускается автоматически при появлении первого компонента RRTI на данном хосте и завершается после того, как завершаются все локальные компоненты RRTI..

Управление работой LSRC возможно с помощью файла конфигурации с именем *`lsrc.cfg`*. LSRC перебирает варианты расположения файла конфигурации в следующей последовательности:

- файл, на который ссылается переменная среды `LSRC_CONFIG` (если значение установлено);

- lsrc.cfg в каталоге $(*RRTI_HOME)/conf*.

Для параметров Локального серверного компонента транспорта используются группы: «`LSRC`», «`CRC`» и «`TRANSPORT`».

Таблица 5 содержит конфигурационные параметры, доступные для управления Локальным серверным компонентом RRTI.

Таблица 5 – Конфигурационные параметры LSRC

Имя параметра	Описание	Возможные значения	Значение по умолчанию
CRC.ADDRESS	IP-адрес хоста, на котором выполняется CRC	Допустимый адрес хоста.	0.0.0.0
LSRC. TCP_PORT_LOW_LATENCY	TCP порт для доставки LOW_LATENCY_RELIABLE сообщений данному LSRC	Целое в интервале 1024 – 65535	4000
LSRC.TCP_PORT_BULK	TCP порт для доставки BULK_VOLUME_RELIABLE сообщений данному LSRC	Целое в интервале 1024 – 65535	4001
LSRC. UDP_PORT_LOW_LATENCY	UDP unicast порт для доставки LOW_LATENCY_BEST_EFFORT сообщений данному LSRC	Целое в интервале 1024 – 65535	4002
LSRC.UDP_PORT_BULK	UDP-порт (multicast и unicast) для доставки BULK_VOLUME_BEST_EFFORT сообщений данному LSRC	Целое в интервале 1024 – 65535	4003
LSRC. MULTICAST_PORT_BULK	UDP multicast порт для доставки BULK_VOLUME_BEST_EFFORT сообщений данному LSRC	Целое в интервале 1024 – 65535	4005
LSRC. USE_MULTICAST	Флаг, определяющий возможность использования множественного вещания для доставки BEST_EFFORT сообщений	ДА НЕТ	НЕТ
LSRC. MULTICAST_BASE_GROUP	Начальная группа для множественного вещания	Допустимый адрес множественного вещания	239.0.0.0
LSRC. MULTICAST_MAX_GROUPS	Максимальное число групп множественного вещания которое может использовать LSRC	Целое в интервале 1 – MAXINT	20
LSRC. MULTICAST_COSTS	Параметр выражает отношение затрат на прием к затратам на отправку для BEST_EFFORT сообщений	Целое в интервале 0 – 90	10
LSRC. MULTICAST_MAX_ GROUPS_IN_SEND	Максимальное число групп множественного вещания, через которые LSRC будет пытаться отправить сообщение для уменьшения затрат на его передачу	Целое в интервале 1 – MAXINT	3

Продолжение таблицы 5

Имя параметра	Описание	Возможные значения	Значение по умолчанию
LSRC.MULTICAST_ MEASUREMENT_DURATION	Длительность измерения статистики использования множественного вещания в миллисекундах	Целое в интервале 1 – MAXINT	1000
LSRC.MULTICAST_SLEEP _ DURATION	Максимальная длительность ожидания между замерами статистики в миллисекундах	Целое в интервале 0 – MAXINT	500
LSRC.UDP_SEND_BUFFSI ZE	Размер буфера отправляемых UDP пакетов, используемый ОС	Целое в интервале 65467 – 500000	100000
LSRC.UDP_RECV_BUFFSI ZE	Размер буфера принимаемых UDP пакетов, используемый ОС	Целое в интервале 65467 – 500000	200000
LSRC.TCP_SEND_BUFFSI ZE	Размер буфера отправляемых TCP пакетов, используемый ОС	Целое в интервале 0 – 500000	200000
LSRC.TCP_RECV_BUFFSI ZE	Размер буфера принимаемых TCP пакетов, используемый ОС	Целое в интервале 0 – 500000	200000
LSRC.UDP_PAYLOAD_SIZ E	Максимальный размер используемых UDP пакетов	Целое в интервале 508 – 65467	65467
LSRC. UDP_REASSEMBLY_TIMEO UT	Максимальное время (в миллисекундах) ожидания всех UDP пакетов, формирующих единое сообщение RTI	Целое в интервале 1 – MAXINT	100
LSRC.UDP_FRAGMENTED_ MESSAGE_POOL_SIZE	Максимальное число одновременно принимаемых LSRC фрагментированных BEST_EFFORT сообщений каждого типа (с учетом деления на multicast, low latency и bulk)	Целое в интервале 1 – MAXINT	6
LSRC.LOG_LEVEL_CONSO LE	Минимальная важность сообщений, выводимых на консоль оператора	НЕ_ВЫВОДИТЬ БЕЗУСЛОВНОЕ ОШИБКА ПРЕДУПРЕЖДЕНИЕ ИНФОРМАЦИОННО Е КОНФИГУРАЦИЯ ОТЛАДКА ОТЛАДКА_2	ОШИБКА

Продолжение таблицы 5

Имя параметра	Описание	Возможные значения	Значение по умолчанию
LSRC.LOG_LEVEL_FILE	Минимальная важность сообщений, выводимых в журнал	НЕ_ВЫВОДИТЬ БЕЗУСЛОВНОЕ ОШИБКА ПРЕДУПРЕЖДЕНИЕ ИНФОРМАЦИОННОЕ КОНФИГУРАЦИЯ ОТЛАДКА ОТЛАДКА_2	ОШИБКА
LSRC. STATISTICS_LEVEL	Сбор статистики о сетевой активности и собственных очередях LSRC	ПОДРОБНО ИТОГ НЕТ	НЕТ
LSRC. CONNECT_TIMEOUT	Тайм-аут для инициирования соединения на сокете (в миллисекундах)	Целое в интервале 1 - MAXINT	1000
LSRC. TRANSPORT_ADDRESMAP_ TIMEOUT	Тайм-аут некоторых операций доступа к карте адресации транспорта (в миллисекундах). 0 – бесконечно	Целое в интервале 0 – 1000	0
LSRC. TRANSPORT_CRC_ WAIT_TIMEOUT	Тайм-аут (в миллисекундах) ожидания локальным транспортом появления CRC в карте адресации	Целое в интервале 1000 - MAXINT	6000
LSRC. TRANSPORT_USE_MLOCK	Использовать ли mlock и аналоги для блокирования в оперативной памяти некоторых страниц	ДА НЕТ	НЕТ
LSRC. TRANSPORT_FILTER_ SIGNALS	Использовать ли фильтрацию сигналов	ДА НЕТ	ДА
LSRC. TCP_SOCKET_TIMEOUT	Тайм-аут записи и чтения в TCP сокет (в миллисекундах)	Целое в интервале 10 - MAXINT	100
LSRC. UDP_SOCKET_TIMEOUT	Тайм-аут записи и чтения в UDP сокет (в миллисекундах)	Целое в интервале 10 - MAXINT	100
LSRC. TCP_SOCKET_ATTEMPTS	Максимальное число попыток записи или чтения сообщения при тайм-аутах (для TCP сокетов). Значение 0 обозначает бесконечное число попыток	Целое в интервале 0 - MAXINT	0

Окончание таблицы 5

Имя параметра	Описание	Возможные значения	Значение по умолчанию
LSRC. TRANSPORT_ATTEMPTS	Максимальное число попыток записи или чтения сообщения при тайм-аутах (для локальных сообщений). Значение 0 обозначает бесконечное число попыток	Целое в интервале 0 - MAXINT	0
TRANSPORT. OWN_ADDRESS	IP адрес, используемый в идентификаторах абонентов на хосте. Позволяет выбрать адрес, соответствующий правильному сетевому интерфейсу	IP адреса сетевых интерфейсов, установленных на хосте, или 0.0.0.0 (автоопределение)	0.0.0.0

Описания параметров, управляющих выводом сообщений, приведены в главе 5.

Для работы RRTI в сети необходимо в файле конфигурации каждого хоста задать значение параметра CRC.ADDRESS: При использовании значения, установленного по умолчанию (0.0.0.0), возможно выполнение компонентов RRTI и федератов только в пределах одного хоста.

Параметр LSRC.MULTICAST_COSTS выражает отношение затрат на прием к затратам на отправку для BEST_EFFORT сообщений. Отношение растет с ростом параметра. Параметр трактуется как угол в градусах. Косинус угла имеет смысл затрат на отправку, а синус – затрат на прием. Таким образом, соотношение затрат на прием к затратам на отправку – это тангенс данного угла. Если параметр равен 0, то затраты на прием отсутствуют. Если 45, то затраты на прием и отправку равны. Если 90, то затраты на отправку отсутствуют и т. д.

Параметры LSRC.MULTICAST_MEASUREMENT_DURATION и LSRC.MULTICAST_SLEEP_DURATION используются в следующем цикле:

1) LSRC собирает статистику в течение MULTICAST_MEASUREMENT_DURATION миллисекунд;

2) LSRC отправляет собранную статистику LSRC, на котором запущен CRC;

3) LSRC ждет от 0 до MULTICAST_SLEEP_DURATION миллисекунд (равномерное распределение);

4) перейти к пункту перечисления 1).

Параметр LSRC.UDP_PAYLOAD_SIZE отвечает за максимальный размер используемых UDP пакетов. Он же является максимальным размером BEST_EFFORT сообщений, которые не будут фрагментированы. Сообщения большего размера будут разбиты LSRC на фрагменты размером не более UDP_PAYLOAD_SIZE.

Значение параметра LSRC.UDP_FRAGMENTED_MESSAGE_POOL_SIZE рекомендуется устанавливать равным количеству используемых LSRC (хостов).

При выполнении федерации в условиях большой нагрузки или в WAN задержки при установлении новых TCP соединений увеличиваются, потому следует соответствующим образом скорректировать значение параметров LSRC.CONNECT_TIMEOUT, LRC.TRANSPORT_CRC_WAIT_TIMEOUT и LSRC.TRANSPORT_CRC_WAIT_TIMEOUT на всех задействованных хостах.

К параметрам LSRC.TRANSPORT_ADDRESMAP_TIMEOUT, LSRC.TRANSPORT_CRC_WAIT_TIMEOUT, LSRC.TRANSPORT_USE_MLOCK, LSRC.TRANSPORT_FILTER_SIGNALS и TRANSPORT.OWN_ADDRESS применимы те же пояснения, что и к аналогичным параметрам LRC (см. раздел 4.3).

5 Файлы журнала компонентов RRTI

RRTI может выводить сообщения системному программисту в журнал и на консоль оператора в соответствии с заданной в конфигурационных файлах степенью детализации сообщений. Данная глава описывает файлы журналов.

5.1 Уровни сообщений в журнал

Компоненты RRTI могут формировать сообщения следующих уровней важности (от более значимых к менее значимым):

- БЕЗУСЛОВНОЕ – это сообщение выводится всегда, независимо от значения конфигурационных параметров;
- ОШИБКА – сообщение информирует о серьезной ошибке, возникшей при выполнении компонента RRTI;
- ПРЕДУПРЕЖДЕНИЕ – сообщение информирует о подозрительной ситуации, не мешающей выполнению компонента RRTI;
- ИНФОРМАЦИОННОЕ – любое сообщение, не имеющее отношения к сбоям или подозрительным ситуациям, например, вывод статистики;
- КОНФИГУРАЦИЯ – используется для сообщений, информирующих о текущей конфигурации СПО РСРМ;
- ОТЛАДКА – используется для вывода отладочной информации;
- ОТЛАДКА_2 – используется для вывода детальной отладочной информации.

5.2 Структура сообщений в журнал

Любое сообщение, выводимое в журнал, имеет следующую структуру:

«ХОСТ ДАТА ВАЖНОСТЬ ИСТОЧНИК СООБЩЕНИЕ»,

где ХОСТ – IP-адрес хоста в формате 999.999.999.999;

ДАТА – дата и время сообщения в формате «ГГГГ.ММ.ДД чч:мм:сс.ссс», где ГГГГ – четырехзначный номер года, ММ – двузначный номер месяца, ДД – двузначный номер дня, чч – двузначный номер часа, мм – двузначный номер минуты, сс – двузначный номер секунды, ссс – трехзначный номер миллисекунды;

ВАЖНОСТЬ – уровень сообщения;

ИСТОЧНИК – компонент RRTI, сформировавший сообщение;

СООБЩЕНИЕ – собственно текст сообщения (возможно, многострочный).

Ниже приведен пример вывода статистической информации в журнал.

Пример.

```
192.168.001.111 2009.09.23 14:07:32.573 ИНФОРМАЦИОННОЕ
InputCommLRC.logStatistics
Статистика приоритетных входящих сообщений
Принято приоритетных сообщений: 3
Минимальное время обработки сообщения: 42 мкс
Максимальное время обработки сообщения: 95 мкс
```

Среднее время обработки приоритетного сообщения: 71 мкс

Когда сообщение дублируется на консоль оператора, то в его начале дополнительно выводится имя журнала, к которому относится сообщение.

5.3 Файл журнала центрального компонента RRTI

Центральный компонент RRTI выводит сообщения системному программисту в файл журнала. Файл журнала создается в текущем каталоге и имеет имя *crc_YYYYMMDD_HHMMSS.mmm.log*, где:

- YYYY – текущий год;
- MM – текущий месяц;
- DD – текущая дата;
- HHMMSS.mmm – часы, минуты, секунды и миллисекунды универсального координированного времени (UTC) на момент создания файла журнала.

Файл журнала позволяет администратору контролировать процесс эксплуатации RRTI, отслеживая диагностические, предупреждающие или информационные записи журнала, в соответствии с конфигурацией уровня вывода в журнал. При каждом запуске CRC создается новый файл журнала, так что их количество со временем растет. В обязанности администратора входит периодически просматривать файлы журнала на предмет наличия сообщений об ошибках и удалять устаревшие файлы журнала.

Состав информации, которая записывается в журнал, зависит от значений конфигурационных параметров CRC (Таблица 3).

Параметр `LOG_LEVEL_FILE (LOG_LEVEL_CONSOLE)` определяет, что в журнал CRC (на консоль оператора) должны выводиться все сообщения, уровень которых соответствует заданному значению или выше.

Параметр `STATISTICS_LEVEL` позволяет запросить подсчет статистики времени обработки поступающих к `CRC` сообщений, а также времени отправки исходящих сообщений. Значение «`ПОДРОБНО`» означает, что каждое поступающее или отправляемое сообщение будет фиксироваться в журнале CRC с подсчетом времени его обработки. Значение «`ИТОГ`» указывает, что только общая статистика (максимальное, минимальное и среднее время обработки сообщений) будет выводиться в журнал при завершении CRC.

Следует иметь в виду, что вывод в журнал и на консоль – это ресурсоемкая операция, поэтому не рекомендуется задавать низкий уровень вывода сообщений и подсчет статистики (особенно уровня «ПОДРОБНО») при реальной работе RRTI. Следует также избегать дублирования сообщений на консоль оператора, когда в этом нет необходимости.

5.4 Файл журнала локального компонента RRTI

Локальный компонент RRTI выводит сообщения системному программисту в файл журнала. Файл журнала создается в текущем каталоге приложения, использующего LRC, с именем *`lrc_YYYYMMDD_HHMMSS.mmm.log`*, где:

- YYYY – текущий год;
- MM – текущий месяц;
- DD – текущая дата;
- HHMMSS.mmm – часы, минуты, секунды и миллисекунды универсального координированного времени (UTC) на момент создания файла журнала.

Файл журнала позволяет администратору контролировать процесс эксплуатации RRTI, отслеживая диагностические, предупреждающие или информационные записи журнала, в соответствии с конфигурацией уровня вывода в журнал. При каждом запуске приложения, использующего LRC, создается новый файл журнала, так что их количество со временем растет. В обязанности администратора входит периодически просматривать файлы журнала на предмет наличия сообщений об ошибках и удалять устаревшие файлы журнала.

Состав информации, которая записывается в журнал, зависит от значений конфигурационных параметров LRC (Таблица 4).

Параметр `LOG_LEVEL_FILE (LOG_LEVEL_CONSOLE)` определяет, что в журнал LRC (на консоль оператора) должны выводиться все сообщения, уровень которых соответствует заданному значению или выше.

Параметр `STATISTICS_LEVEL` позволяет запросить подсчет статистики времени обработки поступающих сообщений в LRC, времени обработки обратных вызовов федератом, а также времени отправки исходящих сообщений. Значение «ПОДРОБНО» означает, что все поступающие или отправляемые сообщение и все обратные вызовы федерата будут фиксироваться в журнале LRC с подсчетом времени их обработки. Значение «ИТОГ» указывает, что только общая статистика (максимальное, минимальное и среднее время обработки сообщений) будет выводиться в журнал при завершении LRC.

Следует иметь в виду, что вывод в журнал и на консоль – это ресурсоемкая операция, поэтому не рекомендуется задавать низкий уровень вывода сообщений и подсчет статистики (особенно уровня «ПОДРОБНО») при реальной работе RRTI. Следует также избегать дублирования сообщений на консоль оператора, когда в этом нет необходимости.

5.5 Файл журнала локального серверного компонента RRTI

Локальный серверный компонент транспорта выводит сообщения системному программисту в файл журнала. Файл журнала создается в текущем каталоге программы, запуск которой привел к старту LSRC, и имеет имя *`lsrc_YYYYMMDD_HHMMSS.mmm.log`*, где:

- YYYY – текущий год;
- MM – текущий месяц;
- DD – текущая дата;
- HHMMSS.mmm – часы, минуты, секунды и миллисекунды универсального координированного времени (UTC) на момент создания файла журнала.

Файл журнала позволяет администратору контролировать процесс эксплуатации RRTI, отслеживая диагностические, предупреждающие или информационные записи журнала, в соответствии с конфигурацией уровня вывода в журнал. При каждом запуске LSRC, создается новый файл журнала, так что их количество со

временем растет. В обязанности администратора входит периодически просматривать файлы журнала на предмет наличия сообщений об ошибках и удалять устаревшие файлы журнала.

Состав информации, которая записывается в журнал, зависит от значений конфигурационных параметров LSRC (Таблица 5).

Параметр `LOG_LEVEL_FILE (LOG_LEVEL_CONSOLE)` определяет, что в журнал LSRC (на консоль оператора) должны выводиться все сообщения, уровень которых соответствует заданному значению или выше.

Следует иметь в виду, что вывод в журнал и на консоль – это ресурсоемкая операция, поэтому не рекомендуется задавать низкий уровень вывода сообщений при реальной работе RRTI. Следует также избегать дублирования сообщений на консоль оператора, когда в этом нет необходимости.

6 Выполнение центрального компонента RRTI

6.1 Запуск CRC на выполнение

Исполняемый файл Центрального компонента RRTI `rtiexec` находится в каталоге `$(RRTI_HOME)/bin`. Приложение не имеет параметров командной строки. Для его запуска на выполнение достаточно перейти в указанный каталог и ввести команду `rtiexec`.

В случае успешного старта CRC выводит на консоль оператора идентификационную информацию:

«Центральный компонент СПО РСРМ-РУС (RRTI), версия X.X.XXXX»,

«Copyright (c) ОАО «НПО РусБИТех»,

«Лицензионные ограничения:

n федератов суммарно для всех выполняющихся федераций»,

где X.X.XXXX обозначает номер версии и номер релиза центрального компонента, а также следующие сообщения:

«Центральный компонент RRTI готов к работе»,

«Для завершения введите: 'exit'».

Дополнительно, в зависимости от установок конфигурационных параметров (см. раздел 4.2), на консоль оператора могут быть продублированы сообщения от CRC системному программисту. Кроме того, запуск CRC приводит к автоматическому старту компонента LSRC, который также может выводить на консоль оператора сообщения системному программисту. Конфигурирование LSRC описано в разделе 4.4.

В текущем каталоге CRC создает журнал своей работы с именем `crc_YYYYMMDD_HHMMSS.mmm.log`, где:

- YYYY – текущий год;
- MM – текущий месяц;
- DD – текущая дата;
- HHMMSS.mmm – часы, минуты, секунды и миллисекунды универсального координированного времени (UTC) на момент создания файла журнала.

Аналогичный журнал `lsrc_YYYYMMDD_HHMMSS.mmm.log` создается компонентом LSRC. Если старт CRC закончился неудачно, на консоль оператора выводятся сообщения об ошибках.

6.2 Команды центрального компонента RRTI

При типичной эксплуатации вмешательство оператора в работу Центрального компонента RRTI не требуется, за исключением процесса завершения его работы, описанного в разделе 6.3 (команда `exit`).

Тем не менее, *rtiexec* поддерживает дополнительные команды оператора, позволяющие упростить работу с приложениями в период их отладки (эти команды не требуются в процессе эксплуатации федераций). Таблица 6 содержит полный перечень команд *rtiexec*.

Таблица 6 – Команды Центрального компонента RRTI

Синтаксис команды	Описание	Результат выполнения
`exit`	Завершение работы CRC	Выход из программы
`fedex`	Вывод списка выполняющихся федераций	Выводится список идентификаторов и имен выполняющихся федераций
`destroy <ид_федерации>`	Удаление пустой федерации	Удаляется федерация, если в ней нет федератов-участников
`list <ид_федерации>`	Вывод списка выполняющихся федератов	Выводится список идентификаторов и имен федератов - участников федерации
`kill <ид_федерации> <ид_федерата>`	Принудительное удаление федерата из федерации	Федерат удаляется из списков участников федерации, его дальнейшие действия будут игнорироваться до нового вступления в федерацию
`help`	Вывод списка поддерживаемых команд	Выводится список команд из данной таблицы с краткими пояснениями

6.3 Завершение работы CRC

Завершение работы CRC должно выполняться оператором только после завершения работы всех федератов. В противном случае функционирование федераций будет нарушено.

Для завершения работы CRC требуется на консоли оператора ввести команду «`exit`».

После ввода команды на консоль оператора выводится сообщение «Завершение работы ...», а после успешного закрытия ресурсов CRC и журнала работы – сообщение «Центральный компонент RRTI остановлен».

Между этими двумя сообщениями, в зависимости от установок конфигурационных параметров, на консоль оператора могут быть продублированы сообщения системному программисту.

После завершения работы CRC завершение работы компонента LSRC может занимать до трех секунд.

В том случае, если выполнение Центрального компонента в результате сбоев не может быть завершено нормально описанным выше способом, оператор может использовать для завершения программы комбинацию клавиш «**Ctrl+C**».

7 Выполнение компонента-провайдера веб-сервиса (WSPRC)

7.1 Запуск веб-сервиса

Для использования WSPRC необходимо запустить на выполнение http-сервер платформы Apache AXIS2/C. Исполняемый файл `axis2_http_server` (`axis2_http_server.exe` в случае ОС Windows) расположен в каталоге `$(AXIS2C_HOME)/bin`. Конфигурирование http-сервера осуществляется при его запуске заданием параметров командной строки. Набор и формат параметров содержится в документации на платформу AXIS2/C и во встроенной справке, которая выводится на экран при запуске сервера с параметром `-h`.

Модуль веб-сервиса загружается http-сервером автоматически при получении первого запроса от веб-федерата. Этот модуль создает по одному локальному прокси-федерату для каждого веб-федерата.

В процессе работы в каталоге `$(AXIS2C_HOME)/logs` создаются файлы журналов платформы и размещенного на ней веб-сервиса.

7.2 Завершение работы веб-сервиса

Веб-сервис останавливается путем прекращения работы http-сервера платформы AXIS2/C. Для завершения работы программы оператор может закрыть окно, в котором отображается консоль сервера или использовать комбинацию клавиш «**Ctrl+C**».

8 Создание федератов на языке C++ для работы с RRTI

8.1 Использование библиотеки LRC в приложении

Библиотека LRC обеспечивает интерфейс RRTI с приложениями на языке C++. Рисунок 6 иллюстрирует способ взаимодействия приложения с библиотекой.

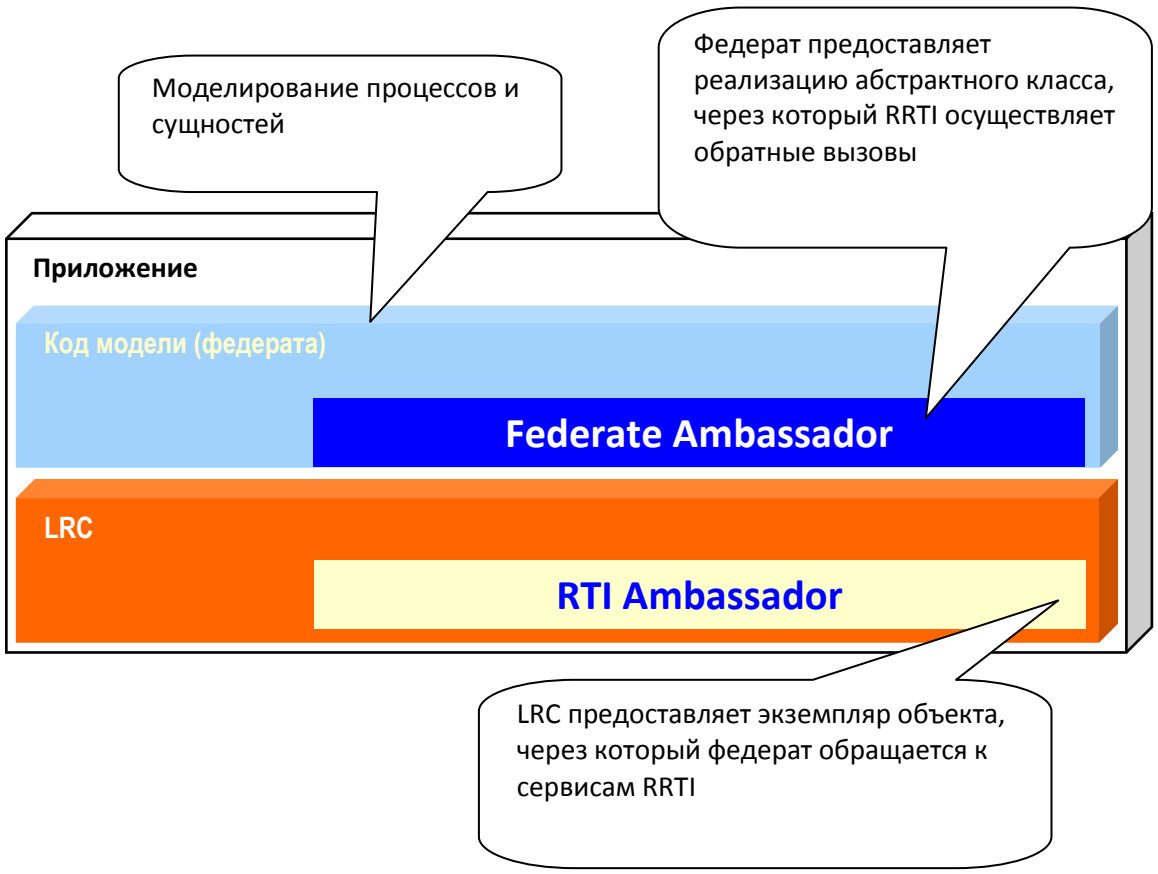

Рисунок 7 – Взаимодействие приложения с библиотекой LRC

Все сервисы (прямые вызовы) RRTI являются методами класса `RTIambassador`. Перед обращениями к этим сервисам каждый федерат должен получить свой экземпляр объекта этого класса, используя фабрику `RTIambassadorFactory`.

Единственный метод фабрики имеет следующий синтаксис:

```
std::auto_ptr<RTIambassador> createRTIambassador()
throw (
```

RTIinternalError)

Метод создает новый экземпляр объекта класса RTIambassador и возвращает автоматический указатель на созданный объект. При вызове данного метода осуществляется инициализация задач и данных LRC для вызвавшего федерата.

LRC осуществляет обратные вызовы федерата через методы класса FederateAmbassador. RRTI содержит пустую реализацию этого класса в файле RTI/NullFederateAmbassador.h. Федерат может создать собственный класс, производный от NullFederateAmbassador, предоставив реализацию тех методов, которые необходимы федерату. Ссылка на экземпляр этого класса (вместе с выбранной моделью обратных вызовов) передается LRC при вызове сервиса connect.

Дальнейшие обратные вызовы федерата осуществляются в зависимости от выбранной модели обратных вызовов.

Модель обратных вызовов определяет, как и когда LRC вызывает эти методы. RRTI поддерживает две модели обратных вызовов HLA:

- модель обратных вызовов по требованию evoked;

- модель неотложных обратных вызовов immediate.

В модели «обратных вызовов по требованию» обратные вызовы возможны исключительно после явной передачи федератом управления коду LRC посредством вызовов evokeCallback или evokeMultipleCallbacks. Обратные вызовы поступают федерату только последовательно в порядке очередности, определяемой внутренней логикой LRC и требованиями по доставке упорядоченных по времени сообщений федерату. Таким образом, приложение, использующее данную модель, может не быть многопоточным, так как весь интерфейс с RTI реализуется в одной нити (потоке) управления. Это не означает, что федерат, скомпонованный с LRC, не является многопоточным процессом, так как LRC может создавать собственные потоки, не взаимодействующие непосредственно с федератом, для реализации внутренней логики работы.

В модели неотложных обратных вызовов LRC вызывает методы федерата немедленно, как только соответствующие данные становятся доступны (то есть, по приходу соответствующих сообщений или возникновению событий, требующих

обратного вызова). Таким образом, вызовы методов федерата осуществляются асинхронно с основным потоком федерата (в отдельном потоке). При этом к федерату предъявляются дополнительные требования безопасности кода с точки зрения его многопоточности и явного учета указанной асинхронности. В частности, разработчик федерата должен позаботиться о защите данных федерата при конкурентном доступе к ним из основного кода и обработчиков обратных вызовов (методов класса `FederateAmbassador`).

Выбор используемой модели обратных вызовов осуществляется независимо каждым федератом с помощью сервиса `connect`. Таким образом, разные члены федерации могут использовать разные режимы работы LRC.

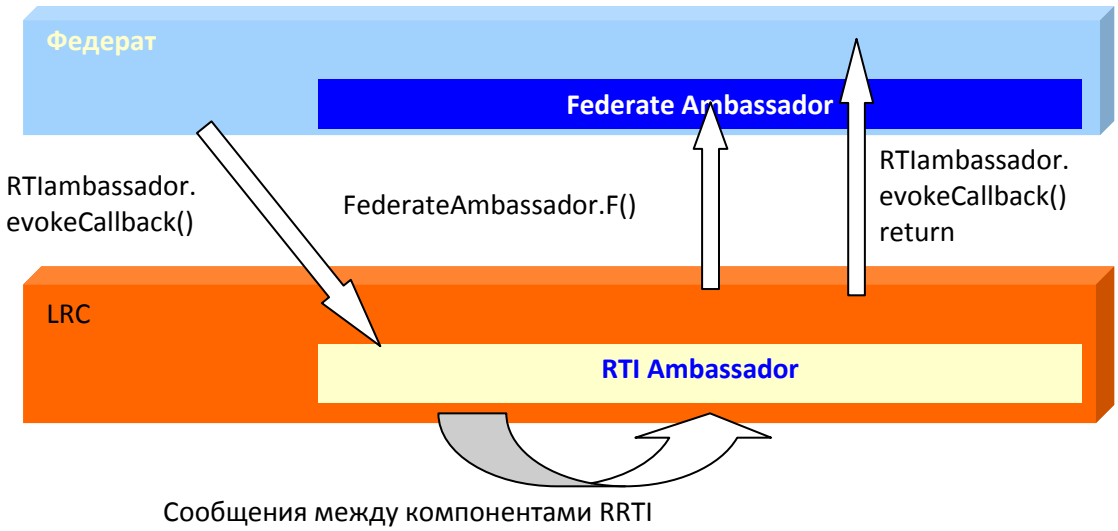

Рисунок 8 – Модель обратных вызовов по требованию

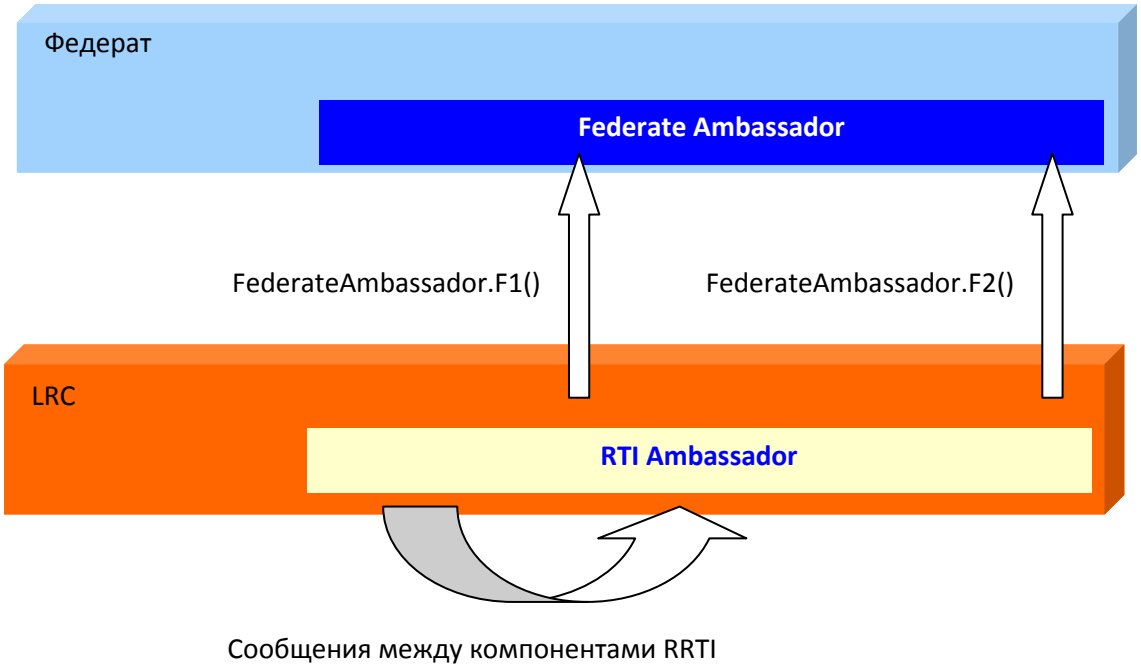

Рисунок 9 – Модель неотложных обратных вызовов

8.2 Потоки библиотеки LRC

Библиотека LRC является многопоточной. Следовательно, независимо от того, является ли многопоточным собственный код прикладной программы, результирующее приложение, скомпонованное с библиотекой LRC, всегда является многопоточным. Как правило, это означает, что при сборке приложения необходимо использовать соответствующие (многопоточные) версии внешних библиотек и опции компилятора.

Для понимания того, как LRC использует потоки, важно иметь в виду следующее.

Прямые вызовы сервисов RRTI осуществляются в том потоке приложения, в котором произошло обращение к сервису, и не порождают других потоков. Допускается вызывать сервисы RRTI из разных потоков приложения, хотя в этом обычно нет необходимости.

Обратные вызовы федерата в модели обратных вызовов по требованию осуществляются в том потоке приложения, в котором было получено требование (вызван метод `evokeCallback` или `evokeMultipleCallbacks`), и не порождают других потоков. Обратные вызовы федерата в модели неотложных обратных вызовов осуществляются в отдельном, но всегда одном и том же потоке LRC.

Таблица 7 содержит описание потоков, создаваемых LRC, а также используемую по умолчанию схему приоритетов. Уровни приоритетов в таблице показаны по отношению к приоритету потока вызывающего федерата.

Таблица 7 – Потоки, создаваемые LRC в контексте процесса федерата

Назначение потока	Уровень приоритета
Обработка входных очередей LRC	выше
Периодическая задача контроля целостности и устойчивости к сбоям	ниже
Задача обратных вызовов федерата в модели неотложных обратных вызовов	выше
Задача обработки асинхронных сообщений транспортного уровня	выше

Стандартная схема приоритетов в RRTI предполагает, что федерату требуется обеспечить возможность быстрее реагировать на входящие события и данные. Поэтому потоки чтения входной очереди LRC и поток обработки обратных вызовов федерата (в модели неотложных обратных вызовов) имеют повышенный приоритет. Задача отправки маяков контроля целостности федерации, наоборот, имеет пониженный приоритет.

Такая схема может быть удачной не для всех федератов. Иногда большой поток входящих событий мешает федерату выполнять собственную работу. Стандартная схема приоритетов может быть отключена с помощью файла конфигурации LRC, параметр `PRIORITY_SCHEME` (см. раздел 4.3). Если выбрано значение параметра `БЕЗ_ПРИОРИТЕТА`, то все потоки LRC выполняются с одинаковым приоритетом, равным приоритету вызывающего федерата.

Замечание для ОС Linux: транспортный уровень RRTI использует сигналы `SIGRTMIN+5` и `SIGRTMIN+6` для собственных нужд. Прикладные программы не должны использовать или маскировать эти сигналы во избежание возможных конфликтов.

8.3 Сервисы и обратные вызовы HLA в RRTI (HLA API)

Реализация программного интерфейса (API) HLA в RRTI полностью соответствует стандарту IEEE Std 1516-2010 (HLA evolved) для языка C++. Описание API HLA содержится в стандарте и в данном документе не повторяется.

Заголовочные файлы для языка C++, содержащие программный интерфейс HLA, являются стандартными и находятся в каталоге `$(RRTI_HOME)/include/HLA2010/RTI`. Для типовых задач федерату достаточно подключить с помощью директивы «`#include`» файл `RTI1516.h`, который включает другие необходимые файлы. Прототипы прямых методов HLA содержатся в файле `RTIambassador.h`, а прототипы обратных вызовов – в файле `FederateAmbassador.h`.

RRTI версии 2 также поддерживает старый API HLA, который применялся в RRTI первой версии. Это обеспечивает выполнение старых федератов в среде RRTI версии 2. Однако, старые федераты не могут работать в одной федерации с новыми. Иначе говоря, федерации должна целиком состоять либо из федератов старого формата, либо из федератов, написанных для интерфейса HLA evolved. Заголовочные файлы, содержащие старый программный интерфейс HLA, находятся в каталоге `$(RRTI_HOME)/include/RRTI1/RTI`.

8.4 Реализация логического времени в RRTI

Логическое время в HLA рассматривается как абстрактный тип данных `LogicalTime`, методы которого должны использоваться федератом для операций над временем. Отдельный тип данных вводится для интервалов логического времени – `LogicalTimeInterval`. Для создания объектов логического времени используется фабрика.

Возможны различные реализации указанных выше абстрактных типов, в том числе, разработанные пользователем, которые должны идентифицироваться именем реализации. Стандарт HLA evolved предусматривает две обязательные реализации логического времени:

- реализация с именем `HLAfloat64Time` основана на представлении значений времени в виде числа двойной точности с плавающей точкой (тип данных **HLAFloat64BE**);

- реализация с именем `HLAinteger64Time` основана на представлении значений времени в виде целого числа двойной точности (тип данных **HLAInteger64BE**).

RRTI включает в себя эти обязательные реализации логического времени. Стандартные заголовки соответствующих типов данных могут быть найдены в каталоге `$(RRTI_HOME)/include/HLA2010/RTI/time`.

RRTI может использовать иные реализации логического времени, если они предоставлены сторонними разработчиками в виде библиотеки `libfedtime1516e`.

Имя реализации времени, которая используется в федерации, задается при создании федерации с помощью сервиса `createFederationExecution`. По умолчанию применяется реализация `HLAfloat64Time`.

Федерат может получить экземпляр фабрики времени, соответствующей используемой в федерации реализации, с помощью метода класса `RTIambassador`:

```
std::auto_ptr<LogicalTimeFactory> getTimeFactory()
```

Полученная таким образом фабрика позволяет федерату создавать новые экземпляры логического времени и интервалов логического времени, как это предусматривается стандартным программным интерфейсом фабрики. Например, метод

```
std::auto_ptr< LogicalTime > makeInitial()
```

создает экземпляр логического времени, содержащий начальное значение времени, а метод

```
std::auto_ptr< LogicalTimeInterval > makeZero()
```

создает экземпляр нулевого интервала логического времени.

8.5 Исключительные ситуации HLA

В соответствии со стандартом HLA, RRTI формирует сообщения об ошибках при выполнении запросов прикладной программы путем генерации исключительных ситуаций. Все используемые RRTI классы исключений основаны на базовом классе `rti1516e::Exception`. Каждый сервис RRTI имеет свой

набор генерируемых исключительных ситуаций, который указан в спецификациях стандарта HLA.

Метод

`std::wstring rti1516e::Exception::what() const_throw()`

позволяет приложению получить текстовое сообщение, уточняющее причину возникновения исключительной ситуации. Для класса `Exception` также определен оператор вывода, позволяющий выводить это текстовое сообщение в поток `std::wostream`.

Федерат при обработке обратных вызовов также может генерировать единственную исключительную ситуацию `FederateInternalError`. RRTI перехватывает это исключение и выводит в журнал LRC сообщение уровня «ПРЕДУПРЕЖДЕНИЕ» следующего вида: «`Исключение в федерате при обработке обратного вызова`».

8.6 Типичная структура простого федерата

Простой федерат можно представить в виде двух частей, одна из которых содержит обращения федерата к сервисам RTI через экземпляр класса `RTIambassador`, а другая предоставляет RTI интерфейс для обратных вызовов федерата через реализацию абстрактного класса `FederateAmbassador`. Допускается также обращение к большинству сервисов RTI из обработчиков обратных вызовов. Рассмотрим сначала вторую часть федерата.

Класс `NullFederateAmbassador` представляет собой набор пустых обработчиков всех обратных вызовов HLA. Типичный федерат содержит класс, производный от класса `NullFederateAmbassador`, с реализацией только тех обратных вызовов, которые нужны федерату. Приведенный ниже пример позволяет обнаруживать объекты, созданные другими федератами, и получать обновленные значения атрибутов, как с логическим временем, так и без него. Конкретные действия федерата по этим событиям определяются реализациями соответствующих методов.

```
class MyFederateAmbassador : public NullFederateAmbassador
{
```

```
public:

// Обнаружение экземпляров объектов
void discoverObjectInstance (
        ObjectInstanceHandle theObject,
        ObjectClassHandle theObjectClass,
        std::wstring const & theObjectInstanceName,
        FederateHandle producingFederate)
        throw (FederateInternalError);

// Получение значений атрибутов объектов без меток времени
void reflectAttributeValues
    (ObjectInstanceHandle theObject,
     AttributeHandleValueMap const & theAttributeValues,
     VariableLengthData const & theUserSuppliedTag,
     OrderType sentOrder,
     TransportationType theType,
     SupplementalReflectInfo theReflectInfo)
      throw (FederateInternalError);

// Получение значений атрибутов объектов с метками времени
void reflectAttributeValues
    (ObjectInstanceHandle theObject,
     AttributeHandleValueMap const & theAttributeValues,
     VariableLengthData const & theUserSuppliedTag,
     OrderType sentOrder,
     TransportationType theType,
     LogicalTime const & theTime,
     OrderType receivedOrder,
     SupplementalReflectInfo theReflectInfo)
      throw (FederateInternalError);

// Получение значений атрибутов объектов, отправленными как TSO
```

```
void reflectAttributeValues
    (ObjectInstanceHandle theObject,
     AttributeHandleValueMap const & theAttributeValues,
     VariableLengthData const & theUserSuppliedTag,
     OrderType sentOrder,
     TransportationType theType,
     LogicalTime const & theTime,
     OrderType receivedOrder,
     MessageRetractionHandle theHandle,
     SupplementalReflectInfo theReflectInfo)
      throw (FederateInternalError);
};
```

Аналогично добавляются другие обратные вызовы, которые интересны федерату. Рассмотрим теперь обращение федерата к сервисам HLA. Такие обращения могут быть успешными или нет. Если сервис не может быть выполнен, RTI сообщает об этом федерату, поднимая программные исключения. Поэтому все вызовы HLA должны производиться внутри блока try-catch. Ниже приводится пример, когда все исключения обрабатываются одинаково, а именно: выводится сообщение оператору. В других случаях приложение может иметь отдельные catch-блоки для разных исключений, что позволяет уточнить причину отказа в выполнении запроса и отреагировать соответственно.

```
// Сервисы HLA находится в этом пространстве имен
using namespace rti1516e;
. . .
try {
. . .
// Здесь размещаем вызовы RTI
. . .
}
catch(Exception& ex) {
std::wcout << L"RTI exception: " << ex << std::endl;
```

```
}
```

Прежде всего, федерату требуется получить от RTI экземпляр класса `RTIambassador`, через который федерат будет обращаться к сервисам HLA.

```
// Фабрика
RTIambassadorFactory ambFactory;
// Указатель на экземпляр RTIambassador
std::auto_ptr <RTIambassador> rtiAmb = ambFactory.createRTIambassador();
```

Следует учитывать следующую особенность реализации RRTI на ОС Linux. При завершении работы потока, в котором был создан экземпляр `RTIambassador`, автоматически чистится разделяемая память и обмен между LRC и LSRC нарушается. Поэтому необходимо создавать экземпляр `RTIambassador` (т.е. осуществлять вызов `createRTIambassador()`) обязательно или в основном потоке, или в потоке, который завершится только при окончании работы федерата.

Теперь необходимо запросить присоединение федерата к полученной копии `RTIambassador`, одновременно информируя RTI, через какой объект федерат будет получать обратные вызовы RTI:

```
// Создание своего экземпляра класса FederateAmbassador
MyFederateAmbassador fedAmb;
// Присоединение к RTI. Будем использовать модель обратных вызовов evoked
rtiAmb-> connect(fedAmb, HLA_EVOKED);
```

Теперь федерат может попытаться создать федерацию и присоединиться к ней.

```
// Дополнительный блок предназначен для перехвата ситуаций,
// когда федерация уже существует
try {
rtiAmb-> createFederationExecution(L“MyFederationName”, L”MyFOM.xml”);
}
```

```
catch(FederationExecutionAlreadyExists& ex) {
// Это не ошибка, просто федерация уже создана другим федератом
}
rtiAmb-> joinFederationExecution(L”MyFederateType”, L“MyFederationName”);
```

Дальнейшие действия федерата определяются его назначением. Например, федерат может подписаться на атрибуты объектных классов и ждать обнаружения объекта класса.

```
// Запрашиваем дескриптор объектного класса по его имени
ObjectClassHandle theClass = rtiAmb-> getObjectClassHandle(L“MyClassName”);
// Запрашиваем дескриптор атрибута объектного класса по его имени
AttributeHandle theAttribute =
      rtiAmb-> getAttributeHandle(L“MyClassName”, L“MyAttributeName”);
AttributeHandleSet hSet;
hSet.insert(theAttribute);
// Подписываемся на атрибуты
rtiAmb->subscribeObjectClassAttributes(theClass, hSet);

// Ожидаем обнаружения объекта (обратного вызова discoverObjectInstance)
while (/* до тех пор, пока объект не обнаружен*/)
{
      // запрашиваем доставку обратных вызовов
      rtiAmb->evokeMultipleCallbacks(0.1, 0.2);
}
```

Завершение федерата выполняется в обратном порядке.

```
// Выходим из состава федерации
rtiAmb->resignFederationExecution(DELETE_OBJECTS);
// Пытаемся уничтожить федерацию
try {
rtiAmb-> destroyFederationExecution(L“MyFederationName”);
```

```
}
catch(Exception & ex) {
// Возможно есть другие подсоединенные федераты
// или федерация уже уничтожена другим федератом
}
// Выходим из соединения с RTI
rtiAmb-> disconnect();
```

Примеры федератов можно найти в тестовых федерациях, которые можно найти в архивном файле `sample_federates.zip`.

8.7 Компиляция и сборка федератов с RRTI

8.7.1 Заголовочные файлы

Заголовочные файлы, содержащие программные интерфейсы RRTI и необходимые для компиляции прикладных программ с библиотеками RRTI, расположены в каталоге `$(RRTI_HOME)/include/HLA2010` (`$(RRTI_HOME)/include/RRTI1` – для старого интерфейса HLA). Необходимо включить этот каталог в пути поиска заголовочных файлов компилятором.

Для компилятора `gcc` это достигается добавлением опции `-I$(RRTI_HOME)/include/HLA2010`.

Для MSVS это можно сделать, добавив `$(RRTI_HOME)/include/HLA2010` в диалоге установок проекта (`Configuration Properties / C/C++ / General / Additional Include Directories`).

8.7.2 Макроопределения препроцессора

Программный интерфейс RRTI версии 1 отличался от стандартного, поскольку являлся промежуточным между интерфейсом HLA версий 2000 и 2010 годов. Для использования старого интерфейса необходимо, чтобы была определена константа препроцессора `RRTI`. Это можно сделать с помощью директивы «`#define`» или ключей компилятора, например, с помощью опции `-DRRTI` компилятора `gcc`.

При использовании интерфейса HLA evolved никаких требований к определению констант препроцессора нет.

8.7.3 Сборка с библиотеками RRTI

8.7.3.1 Платформа MS Windows

Библиотеки динамической компоновки RRTI расположены в каталоге `$(RRTI_HOME)/bin`, в том числе:

- `librti1516e` - библиотека LRC;
- `librti1516ed` - библиотека LRC, неоптимизированная версия с информацией для отладки;
- `libfedtime1516e` - библиотека реализации логического времени;
- `libfedtime1516ed` - библиотека реализации логического времени, неоптимизированная версия с информацией для отладки.

Аналогичные библиотеки для старого интерфейса HLA (RRTI версии 1):

- `librrti` - библиотека LRC;
- `librrtid` - библиотека LRC, неоптимизированная версия с информацией для отладки;
- `librrtitime` - библиотека встроенной реализации логического времени;
- `librrtitimed` - библиотека встроенной реализации логического времени, неоптимизированная версия с информацией для отладки.

Соответствующие библиотеки импорта расположены в каталоге `$(RRTI_HOME)/lib`.

При построении `debug`-версии федерата необходимо использовать отладочные библиотеки, имена которых завершаются символом «`d`», а при построении `release`-версии федерата – соответствующие версии библиотек без информации для отладки.

Все версии библиотек RRTI используют при компоновке многопоточные библиотеки MSVS, т.е. использованы опции для `C/C++ / Code Generation / Runtime Library`:

- `Multi-threaded Debug DLL (/MDd)` – для библиотек с отладочной информацией;

- `Multi-threaded DLL (/MD)` – для библиотек без отладочной информации.

При возникновении проблем необходимо убедиться, что проект федерата имеет совместимые установки для этой опции.

Указанные библиотеки должны быть доступны и входить в пути поиска компилятором на этапе компоновки и в пути поиска ОС на этапе выполнения приложений, использующих RRTI. В частности, в проекте федерата надо добавить путь `$(RRTI_HOME)\lib` в графе `Linker / General / Additional Library Directories`, а также добавить выбранные библиотеки RRTI в графе `Linker / Input / Additional Dependencies`.

8.7.3.2 Платформа Linux

Библиотеки динамической компоновки RRTI расположены в каталоге `$(RRTI_HOME)/bin`, в том числе:

- `librti1516e` - библиотека LRC;
- `libfedtime1516e` - библиотека встроенной реализации логического времени.

Аналогичные библиотеки для старого интерфейса HLA (RRTI версии 1):

- `librrti` - библиотека LRC;
- `librrtitime` - библиотека встроенной реализации логического времени.

Указанные библиотеки должны быть доступны и входить в пути поиска компилятором на этапе компоновки и в пути поиска ОС на этапе выполнения приложений, использующих RRTI. Это может быть достигнуто разными способами, например:

- установкой опций компилятора в `Makefile` федерата (рекомендуемый способ) `-L$(RRTI_HOME)/bin -Wl,-rpath,$(RRTI_HOME)/bin;`

- добавлением путей к библиотекам динамической компоновки к значению переменной среды `LD_LIBRARY_PATH`.

8.8 Объектная модель федерации (FOM)

Стандарт HLA включает в себя описание двух видов объектных моделей:

- объектная модель системы моделирования (федерации) - Federation object model (FOM);

- объектная модель компонента системы моделирования (федерата) – Simulation object model (SOM).

Обе объектные модели используют общий шаблон Object Model Template (OMT), детально описанный в стандарте. Основная цель спецификации шаблона в стандарте – поддержка организации взаимодействия между моделями и упрощение их повторного использования.

RRTI в процессе своего выполнения использует только объектную модель FOM. Поддерживаются модульные объектные модели, введенные в стандарте HLA evolved.

Объектная модель FOM содержит стандартизированное описание всех данных, обмен которыми в принципе возможен в системе моделирования, включая классы объектов и их атрибуты, а также классы событий (взаимодействий) и их параметры. Объектную модель FOM можно рассматривать как информационную модель федерации, которой обязаны следовать все ее потенциальные члены.

Объектная модель FOM содержит как данные, необходимые программной инфраструктуре RTI для организации взаимодействия федератов, так и данные, адресованные разработчикам и пользователям федерации. Подмножество данных FOM, используемое RTI, образует схему данных федерации (Federation Document Data - FDD).

Минимальный объем FDD указан в стандарте и включает в себя:

- структуру дерева объектных классов;
- структуру дерева классов взаимодействий;
- описание объектных классов и их атрибутов в части типа транспортировки, порядка доставки и допустимых фильтров по значению `dimensions`;
- описание классов взаимодействий и их параметров в части типа транспортировки, порядка доставки и допустимых фильтров по значению `dimensions`;
- описание доступных фильтров по значению `dimensions`;
- описание доступных типов транспортировки;
- описание частот обновления в части имени и значения максимальной частоты;

- состояние переключателей.

RRTI работает с документами FOM стандартного вида – файлами в формате XML, созданными в соответствии со спецификациями шаблона OMT DIF (или шаблона предыдущей версии стандарта для старых федератов, использующих интерфейс RRTI версии 1). Для создания и редактирования документа FOM можно использовать любое программное обеспечение, которое поддерживает создание объектных моделей в соответствии со стандартом IEEE 1516.

Данные FDD используются RRTI для организации информационного обмена между членами федерации прозрачным для них образом, а также для управления собственным функционированием (в частности, на основании значений переключателей). Федераты, создающие федерацию или добавляющие модули FOM к существующей федерации, должны иметь доступ к файлам, содержащим соответствующие модули. LRC осуществляет поиск файлов модулей FOM в соответствии с указаниями прикладной программы, а именно: приложение должно передать LRC путь к файлу при обращении к сервису `createFederationExecution` или (опционально) `joinFederationExecution`.

9 Создание веб-федератов для работы с RRTI

9.1 Способы реализации веб-федерата

Веб-федерат, взаимодействует с RRTI через сетевой интерфейс веб-сервиса, который обеспечивается компонентом WSPRC, как это описано в разделах 2.4 и 2.5.

Веб-федерат может быть реализован одним из двух способов:

- путем прямой реализации сетевого интерфейса в соответствии со спецификациями, содержащимися в стандарте IEEE 1516-2010 для языка WSDL;

- через использование C++ API библиотеки WSRTIAmbassador, входящей в состав RRTI.

Сетевой интерфейс должен соответствовать спецификациям, описанным в `hla1516e.wsdl` (составная часть стандарта IEEE 1516-2010), и документации на платформу Apache AXIS2. Библиотека WSRTIAmbassador уже содержит в себе реализацию сетевого интерфейса, а разработчику предоставляет прикладной программный интерфейс, в основном повторяющий API LRC.

Архитектура веб-сервисов предполагает использование взаимодействий типа «запрос-ответ» при связи клиента и сервера. Исходя из этого, модель неотложных обратных вызовов (`immediate`) недоступна для веб-федерата. Таким образом, веб-федераты ***всегда*** использует модель обратных вызовов по требованию (`evoked`).

9.2 Использование библиотеки WSRTIAmbassador в приложении

Библиотека WSRTIAmbassador может быть использована при построении веб-федерата для реализации сетевого обмена с веб-сервисом. Разработчику приложения предоставляется интерфейс прикладного программирования на языке C++.

Способ взаимодействия приложения с библиотекой схож со способом взаимодействия приложения с Локальным компонентом RTI (см. раздел 8.1). Наборы типов данных и исключительных ситуаций, используемые в интерфейсе, полностью совпадают с таковыми в интерфейсе библиотеки LRC.

Различия есть в наборе входных параметров некоторых методов (в частности, `connect` и `evokeMultipleCallbacks`). Кроме того, отсутствует метод `evokeCallback` (его заменяет вызов метода `evokeMultipleCallbacks` с единицей в качестве входного параметра).

Прямые вызовы HLA объединены в класс `WSRTIAmbassadorAPI`, который является аналогом стандартного класса `RTIAmbassador`. Представитель этого класса создается напрямую.

Обработка обратных вызовов производится методами класса `WSFederateAmbassador`, являющегося аналогом стандартного класса `NullFederateAmbassador`. Класс содержит пустые обработчики всех обратных вызовов, предусмотренных стандартом. Для реализации собственных обработчиков обратных вызовов разработчик приложения должен создать свой класс, наследующий `WSFederateAmbassador`, и в нем заменить соответствующие методы. Представитель класса `WSFederateAmbassador` (или его наследника) должен быть создан до создания представителя `WSRTIAmbassadorAPI`, так как конструктору последнего требуется указатель на него.

Перед использованием представитель `WSRTIAmbassadorAPI` должен быть проинициализирован с помощью вызова метода `init`. Параметром метода служит URI веб-сервиса в виде текстовой строки.

Веб-федерату доступна только модель обратных вызовов по требованию (`evoked`). Поэтому указание модели обратных вызовов отсутствует среди параметров метода `connect`. Также из параметров метода убран указатель на класс-обработчик обратных вызовов (он передается в конструкторе).

Метод `evokeMultipleCallbacks` в качестве параметра принимает количество ожидающих в очереди обратных вызовов, которые необходимо обработать. Метод возвращает количество оставшихся в очереди необработанных обратных вызовов. Если значение входного параметра превышает количество вызовов в очереди, происходит обработка всех доступных вызовов, после чего метод завершает работу и возвращает значение 0.

В остальном интерфейс библиотеки WSRTIAmbassador повторяет API локального компонента RTI.

Заголовочные файлы интерфейса библиотеки находятся в каталоге `$(RRTI_HOME)/include/WebService`.

Библиотека динамической компоновки `WSRTIAmbassador` расположена в каталоге `$(RRTI_HOME)/bin`. Для ОС Windows также имеется неоптимизированная версия библиотеки с включенной отладочной информацией – `WSRTIAmbassadord`. Соответствующие библиотеки импорта находятся в каталоге `$(RRTI_HOME)/lib`.

10 Список сокращений

API	- Application Program Interface (Интерфейс прикладных программ)
CRC	- Central RTI Component (Центральный компонент RTI)
DDM	- Data Distribution Management (управление распространением данных)
FOM	- Federation Object Model (объектная модель федерации)
HLA	- High Level Architecture (высокоуровневая архитектура)
LAN	- Local Area Network (локальная сеть)
LRC	- Local RTI Component (локальный компонент RTI)
LSRC	- Local Server RTI Component (локальный серверный компонент RTI)
MSVS	- Microsoft Visual Studio
MOM	- Management Object Model (объектная модель управления)
OMT	- Object Model Template (шаблон объектной модели)
RTI	- Run-Time Infrastructure (инфраструктура времени выполнения)
RRTI	- Russian RTI (Российская RTI)
TSO	- Time Stamp Order (порядок доставки сообщений в соответствии с метками логического времени)
WAN	- Wide Area Network
URI	- Uniform Resource Identifier – унифицированный идентификатор ресурса
WSDL	- Web Service Definition Language – язык описания веб-сервисов
WSPRC	- Web Service Provider RTI Component – компонент RTI - провайдер веб-сервисов
XML	- eXtensible Markup Language – расширяемый язык разметки.