Применение технологии UniTESK для функционального тестирования инфраструктурного ПО Грид

Н.В. Пакулин, С.А. Смолов
{npak, ssedai}@ispras.ru

Аннотация. В статье рассматриваются вопросы тестирования инфраструктурного программного обеспечения (ИПО, middleware) Грид-систем на соответствие стандарту. Грид-системы в настоящий момент являются одним из приоритетных направлений в вычислительной технике. Потому первоочередной задачей становится эффективное использование их преимуществ, что неразрывно связано с проблемой переносимости программного обеспечения Грид-систем. Одним из наиболее простых решений этой проблемы является введение стандартов на программные комплексы. Следовательно, соответствие стандарту представляется одним из основных требований, предъявляемых к системе, а потому его выполнение должно быть подтверждено с высокой достоверностью. В статье рассматривается разработка тестовых наборов для ИПО Грид средствами технологии автоматизированного тестирования UniTESK. Приведены результаты тестирования программного пакет Globus Toolkit 4.2 на соответствие базовому стандарту WSRF 1.2.

1. Введение

Грид (англ. *grid* – решетка, сеть) – это согласованная, открытая и стандартизованная компьютерная среда, которая обеспечивает гибкое, безопасное и скоординированное использование вычислительных ресурсов и данных. Термин “Грид” появился в начале 1990-х гг. в сборнике “The Grid: Blueprint for a new computing infrastructure” [1] под редакцией Яна Фостера как метафора о такой же легкости доступа к вычислительным ресурсам, как и к электрической сети (англ. *power grid*).

Создание Грид-систем было продиктовано необходимостью повышения вычислительных мощностей ресурсов. Так как требования к точности получаемых результатов и скорости работы вычислительных комплексов неуклонно растут, то возникает вопрос: как удовлетворить таким требованиям с наименьшими затратами? Один из возможных способов разрешения данной проблемы заключается в объединении различных ресурсов в одну систему. Причем необходимо, чтобы “объединенный” ресурс работал как единое целое даже при отсутствии централизованного управления. Не должна также

оказывать никакого отрицательного влияния на работоспособность системы и разнородность ресурсов, хотя особенности каждого из ресурсов также должны учитываться (для оптимизации времени выполнения).

Что же касается конечного пользователя, то он может и не иметь никакого понятия о том, как именно устроена система. Зато предполагается возможность коллективного разделяемого режима доступа к ресурсам и связанным с ними услугами в рамках глобально распределенных виртуальных организаций, состоящих из предприятий и отдельных специалистов, совместно использующих общие ресурсы.

Согласно Яну Фостеру, Грид-система [2] (далее ГС) – это система, которая:

а) координирует использование ресурсов при отсутствии централизованного управления этими ресурсами;
б) использует стандартные, открытые и универсальные протоколы и интерфейсы;
в) нетривиальным образом обеспечивает высококачественное обслуживание.

ГС, следуя определению, является универсальной инфраструктурой обработки и хранения распределенных данных, в которой функционируют различные службы, называемые Грид-сервисами. Последние не только позволяют решать конкретные задачи, но и могут предоставлять определенные услуги, например поиск ресурсов, сбор информации об их состоянии, хранение и доставка данных. Ясно также, что из определения проистекает высокая специфичность соответствующего ПО ГС.

Особого внимания заслуживает второй пункт определения. В нём указано, что различные ГС могут и должны поддерживать стандартные протоколы и интерфейсы, несмотря на расхождения в архитектуре или особенности реализации. Иными словами, любая ГС должна соответствовать определенному набору стандартов. Цель стандартизации ГС – обеспечить переносимость вычислительных приложений между различными Гридами, в том числе построенных на различных инфраструктурных программных пакетах.

1.1. Стандартизация Грид

Реализации ГС начали появляться с 1995 года, когда появился инфраструктурный программный пакет Globus Toolkit, ныне являющийся де-факто стандартом ГС. Он был выпущен организацией Globus Alliance [3] – крупнейшим международным консорциумом в области Грид. В 1997 году был начат европейский проект по созданию программного пакета для ГС, приведший к созданию ИПО UNICORE[4]. В 2004 году под эгидой проекта EGEE (Enabling Grids for E-sciencE) был выпущен пакет gLite[5]. С появлением большого числа несовместимых между собой реализаций ИПО Грид необходимость унификации и стандартизации стала актуальной, и началась активная работа над созданием стандартов Грид. В 2004 году

компания OASIS объявила о выходе стандарта WSRF [7] (Web Services Resource Framework), а в 2005 году Global Grid Forum – о стандарте OGSA [6] (Open Grid Services Architecture). В том же 2005 году компания Microsoft выпустила ещё один стандарт, определяющий управление разнородными ресурсами – WS-Management[8].

Таким образом в настоящее время действуют три группы стандартов для ИПО Грид.

1.2. Вопросы тестирования реализаций Грид

Одной из особенностей предметной области является наличие ряда, вообще говоря, несовместимых стандартов и нескольких независимых реализаций. Заметим, что в данной работе не решается задача анализа адекватности стандартов, т.е. не только не утверждается, но и не проверяется гипотеза о том, что реализации, соответствующие одному и тому же стандарту, совместимы. Однако ясно, что весьма актуальной задачей является тестирование реализации на соответствие стандарту.

Также стоит отметить, что в предметной области имеет место специфичность средств взаимодействия пользовательских приложений и самой ГС – в данном случае это Грид- и Web-сервисы, а также удаленные вызовы процедур. Web-сервис[10] – это программная система, идентифицируемая строкой URI (Uniform Resource Identifier – некоторая последовательность, однозначно определяющая физический или абстрактный ресурс), чьи общедоступные интерфейсы определены на языке XML (eXtensible Markup Language – расширяемый язык разметки, предоставляющий возможность хранения структурированных данных для обмена ими между программами). Описание этой программной системы может быть найдено другими программными системами, причем последние могут с ней взаимодействовать, обмениваясь сообщениями в формате XML с помощью протоколов Интернета (например, SOAP[13]) .

Наиболее распространенными подходами к тестированию ГС являются следующие:

1. Модульное тестирование (Unit testing) – тестирование различных модулей исходного кода ПО. При таком подходе тесты пишутся для каждой нетривиальной функции или метода в отдельности. В частности, разработчики ИПО Globus toolkit широко используют JUnit для проверки реализации.

2. Интеграционное тестирование (Integration testing, оно же тестирование взаимодействия) – тестирование выполнения приложений на сборке ИПО Грид. Типичными примерами интеграционных тестовых сценариев являются пересылки больших массивов данных и проведение типовых расчетов[9].

Оба подхода направлены на выявление ошибок реализации. Однако у них есть существенный недостаток в контексте тестирования совместимости: отсутствует связь между тестами и требованиями соответствующих стандартов. То есть по результатам тестирования нельзя сделать вывод о соответствии или несоответствии реализации стандарту.

Таким образом, важным вопросом тестирования ГС является тестирование на соответствие стандартам. Сложность этого вопроса заключается в наличии ряда жестких условий на используемые технологии тестирования и получаемые тесты. Технологии разработки тестов должны предоставлять возможность определения покрытия требований – без выполнения этого условия ценность получаемых результатов весьма сомнительна. Сами же тесты должны представлять собой последовательности вызовов процедур или обращений к сервисам, детали реализаций которых могут быть неизвестны.

1.3. Технология автоматизированного тестирования UniTESK

С 1994 года в ИСП РАН разрабатывается технология автоматизированного тестирования UniTESK, которая с успехом использовалась для тестирования различных классов программных систем – программных интерфейсов, телекоммуникационных протоколов, аппаратного обеспечения. Доступ к ИПО Грид реализуется посредством различных механизмов удаленных вызовов процедур и служебных протоколов, что близко к области применимости UniTESK, поэтому данная технология была выбрана в качестве технологической платформы для разработки тестов.

Мы не будем здесь подробно описывать данную технологию, а остановимся лишь на основных этапах разработки тестов с её применением. Разработка тестов с применением технологии UniTESK ведется в следующие 7 этапов:

1. анализ требований и их формализация, построение формальных спецификаций;
2. формулировка требований к качеству тестирования;
3. разработка тестовых сценариев, реализующих заданное покрытие;
4. привязка тестовых сценариев к конкретной целевой системе посредством разработки медиаторов;
5. получение готового тестового набора (трансляция и компиляция тестов);
6. отладка и исполнение тестов;
7. анализ результатов тестирования.

Важным преимуществом технологии UniTESK является возможность оценки качества проведенного тестирования посредством вычисления покрытия требований. Этим она отличается от подавляющего большинства тестовых

наборов для ГС. Автоматизированы как вычисление покрытия требований, так и построение так называемых оракулов, проверяющих соответствие целевой системы спецификации.

Таким образом, технология UniTESK позволяет проводить широкомасштабное тестирование широкого класса программных систем и, в частности, разрабатывать тестовые наборы для решения задач соответствия.

1.4. Стандарты OGSA и WSRF. Ограничение области определения задачи

Рассмотрим характерные особенности стандартов, использующихся при разработке ИПО Грид: OGSA и WSRF.

Стандарт OGSA описывает инфраструктурное ПО (далее ИПО) OGSI (Open Grid Services Infrastructure), занимающее “промежуточное” положение между приложениями пользователей и непосредственно вычислительными ресурсами. Это означает, что приложения не могут взаимодействовать с ресурсами напрямую, а только с ИПО. Таким образом, от пользователей скрывается внутренняя структура ГС, и им совершенно нет нужды вникать в детали реализации вычислений на самих ресурсах, приложение взаимодействует только с этой промежуточной средой. В основе архитектуры OGSI лежит понятие Грид-сервиса, который представляет собой механизм удаленных вызовов, разработанный специально для Globus Toolkit версии 3. Посредством удаленного доступа к методам Грид-сервиса приложение получает определенный вид обслуживания. Таким образом, унифицируются различные функции: доступа к вычислительным ресурсам, ресурсам хранения, базам данных и к любой программной обработке данных.

Однако, при всех своих достоинствах, концепция грид-сервисов имеет ряд недостатков. Во-первых, само понятие Грид-сервиса недостаточно формализовано, а это означает, что в различных реализациях Грид-сервисами могут быть совершенно разные, зачастую даже несовместимые сущности. Во-вторых, наиболее близким аналогом Грид-систем можно считать Web. Соответственно, представляется разумным обеспечить их совместимость, чего столь специфичное решение, как Грид-сервисы, дать не может.

Одновременно с разработкой стандарта OGSA, в 2004 году консорциум OASIS предложил стандарт WSRF. Он также описывает некоторое ИПО, в основе которого, однако, лежат уже не Грид- а Web-сервисы. А в 2005 году был опубликован стандарт WS-Management, фактически, представляющий собой протокол для управления серверами, устройствами, и приложениями, основанный на использовании всё тех же Web-сервисов.

В последней версии Globus Toolkit разработчики отказались от использования Грид-сервисов в пользу Web-сервисов. За основу была взята спецификация WSRF. Соответственно, областью определения данной работы является именно ИПО, основанное на Web-сервисах, так как на этой концепции базируются наиболее широко используемые реализации ИПО Грид.

В данной работе исследуется возможность применения технологии автоматизированного тестировании UniTESK для функционального тестирования ИПО Грид, включая тестирования соответствия реализации стандарту. В частности, рассматриваются следующие вопросы:

1. представление требований в сервисам ИПО Грид в виде формальных спецификаций UniTESK;
2. разработка медиаторов для оказания воздействий на ИПО Грид;
3. разработка тестовых сценариев для ИПО Грид.

2. Формализация требований к реализациям ИПО Грид

2.1. Регламентирующие документы и требования к реализациям ИПО Грид

Как упоминалось во введении, в настоящее время при разработке ИПО Грид используются два семейства стандартов: OGSA и WSRF. Рассмотрим особенности формализации требований указанных стандартов.

При изучении стандарта OGSA оказалось, что:

1. стандарт носит описательный характер, требования нечётко выражены;
2. стандарт практически не содержит функциональных требований;
3. в стандарте не приводятся описания форматов сообщений и протоколов удаленного обращения к сервисам;
4. стандарт признан устаревшим по сравнению с новыми подходами к организации удаленных вызовов, основанных на Web-сервисах.

По этим причинам стандарт OGSA не подходит в качестве основы для разработки формальных спецификации и основанного на них тестового набора.

Стандарт WSRF больше подходит для формализации. Стандарт содержит 5 спецификаций:

1. WS-BaseFaults – определяет формат сообщений об ошибках и механизм их обработки;
2. WS-Resource – определяет само понятие WS-Resource, форматы сообщений и семантику сервисов управления ресурсом;
3. WS-ResourceLifetime – определяет механизмы прекращения существования WS-Resource;
4. WS-ResourceProperties – определяет, как WS-Resource связан с

интерфейсом, описывающим Web-сервис, а также позволяет извлекать, изменять и уничтожать свойства WS-ресурса;

5. WS-ServiceGroup – определяет интерфейс к набору гетерогенных Web-сервисов.

Рассмотрим понятие ресурса (Resource) и WS-ресурса (WS-Resource), введенных в WS-Resource. Ресурс – это логическая сущность, обладающая следующими характеристиками:

1. идентифицируемость;
2. наличие “времени жизни”;
3. наличие множества (быть может пустого) свойств, представимых в XML InfoSet [14].

Теперь перейдем к понятию WS-ресурса. WS-ресурс представляет собой композицию ресурса и Web-сервиса, посредством вызова методов или полей которого осуществляется доступ к ресурсу. WS-ресурс также должен быть идентифицируемым (роль идентификатора должен играть XML элемент типа EndpointReference, описание которого содержится в стандарте WS-Addressing[15]), а множество свойств ресурса, ассоциированного с сервисом, – представимым в XML InfoSet.

Стандарт WSRF весьма удобен для анализа по причине его структурированности. Так, например, большая их часть функциональных требований снабжена ключевыми словами стандарта RFC 2119 [10] – MUST, SHOULD, MAY. Кроме того, большинство требований снабжено блоками объяснений и примерами, написанными на псевдокоде, имеющем много общего с языком WSDL 2.0 [11] описания Web-сервисов.

Отдельные части стандарта имеют различные уровни обязательности в градации RFC 2119. А именно: WS-ресурс обязан (MUST) реализовывать требования спецификаций WS-Resource и WS-ResourceProperties, ему следует (SHOULD) руководствоваться требованиями WS-BaseFaults и он может (MAY) удовлетворять требованиям WS-ResourceLifetime. Следовательно, при создании тестового набора наиболее пристальное внимание нужно уделить именно первым двум обязательным спецификациям. Действительно, такой подход существенно упрощает как структуру тестового набора, так и его размер, при этом сохраняя корректность тестов как решения задачи соответствия.

Итак, спецификация WS-Resource содержит определения ресурса и WS-ресурса. Также она содержит описания двух сообщений об ошибках, которые WS-ресурс может (MAY) возвращать в ответ на запросы. Что же касается спецификации WS-ResourceProperties, то в ней описан следующий способ хранения свойств WS-ресурса. Каждое свойство WS-ресурса представляет собой XML-элемент, полями которого являются уникальное имя свойства (в

терминах спецификации WS-ResourceProperties – QName), значение свойства и, возможно, служебные параметры. Все свойства ресурса объединены в некоторую композицию, называемую Resource Property Document (далее RPD), имеющую собственный идентификатор. Иными словами, свойства ресурса являются дочерними элементами по отношению к некоторому корневому элементу – фактически, идентификатору RPD. Вместе с тем не указывается, каким именно образом сервис должен реализовывать свой RPD.

В спецификации WS-ResourceProperties описаны следующие обмены сообщениями:

1. GetResourcePropertyDocument – получение всех свойств WS-ресурса;
2. GetResourceProperty – получение определенного свойства WS-ресурса;
3. GetMultipleResourceProperties – получение нескольких свойств WS-ресурса;
4. QueryResourceProperties – выяснение структуры свойств WS-Resource и выполнение запросов к RPD и вычислений над его элементами (отдельно указывается диалект, на котором записано вычисляемое выражение, примером такого диалекта может служить язык XPath[16]);
5. PutResourcePropertyDocument – замена всего RPD WS-ресурса “новым” RPD;
6. SetResourceProperties – изменение нескольких свойств WS-ресурса (фактически, некоторая композиция следующих трех обменов сообщениями);
7. InsertResourceProperties – добавление новых свойств WS-ресурса;
8. UpdateResourceProperties – изменение значений свойств WS-ресурса;
9. DeleteResourceProperties – удаление свойств WS-ресурса.

В ходе анализа стандартов были выделены 325 функциональных требований, из них 29 относятся к спецификации WS-BaseFaults, 12 – к WS-Resource, 51 – к WS-ResourceLifetime, 159 – к WS-ResourceProperties, 73 – к WS-ServiceGroup.

2.2. Разработка формальной спецификации

Каждый обмен сообщениями в WSRF представляет собой пару <запрос – ответ> где в качестве ответа может быть сообщение с возвращаемым значением или сообщение об ошибке. Тем самым обмены сообщениями WSRF

можно представить как вызовы функций с возвращаемым значением, а сообщения об ошибках моделировать как исключения.

Благодаря этому в рассматриваемом методе формализации требований к ИПО Грид Web-сервис моделируется как объект некоторого класса, а обмены сообщениями между клиентом ИПО и реализацией представляются как вызовы методов объекта. Модельное состояние сервиса формализуется как набор полей объекта.

Такой способ моделирования неявно подразумевает синхронный сценарий взаимодействия между клиентом и сервисом: после отправки запроса клиент ожидает ответа. Не допускаются сценарии, в которых клиент может послать серию запросов, не дожидаясь ответов. Однако, анализ сценариев использования Грид-систем показал, что большинство современных клиентских приложений разрабатываются в синхронной парадигме. Более того, сам стандарт WSRF описывает семантику поведения сервиса в синхронном стиле. По этим причинам в рассматриваемом методе к формализации требований мы моделируем обмены сообщениями между клиентом и ИПО Грид как процедурные вызовы.

В модельное состояние ресурса необходимо включить переменные для представления свойств ресурса и окружения:

1. EndpointReference – модель идентификатора WS-ресурса, т.н. ссылка на конечную точку, удовлетворяющая требованиям стандарта WS-Addressing адресации Web-сервисов;
2. ResourcePropertyDocument – модель RPD WS-ресурса как множества элементов, моделирующие отдельные свойства (properties) ресурса;
3. набор параметров модели, учитывающих отсутствие поддержки ряда необязательных обменов сообщениями в реализации;
4. CurrentTime – переменная, определяющая текущее системное время;
5. TerminationTime – переменная, определяющая время прекращения существования WS-ресурса.

Наличие первого поля определяется требованием спецификации WS-Resource об идентифицируемости. Что же касается второго и третьего пунктов, то на них стоит остановиться поподробнее.

Свойство ресурса как элемента RPD в спецификации моделируется классом со следующими полями: идентификатор-имя свойства (QName), значение свойства и минимальное количество его включений в RPD. Наличие последнего поля продиктовано рядом требований стандарта, касающихся тех свойств WS-ресурса, у которых оно является нулевым, т.е. наличие этого свойства, вообще говоря, необязательно.

Что касается третьего пункта, то его наличие связано со спецификой организации стандарта WSRF 1.2. Дело в том, что требования стандарта, находящиеся даже в одной и той же его части, совершенно необязательно имеют один и тот же “ранг” в градации RFC 2119, т.е. ключевые слова в них различные. В таблице 1 изображено распределение требований по частям стандарта в зависимости от их “ранга”:

	WS-Base Faults	WS-Resource	WS-Resource Properties	WS-Resource Lifetime	WS-Service Group
MUST	17	6	115	30	41
SHOULD	1	0	12	8	15
MAY	11	6	32	13	17

Таблица 1. Распределение требований по отдельным частям стандарта WS-RF.

Таким образом, в спецификации было формализовано около половины всех требований стандарта WSRF (порядка 160 требований), и порядка 60% содержащихся в частях WS-Resource, WS-ResourceLifetime и WS-ResourceProperties стандарта WSRF требований. Реализация остальных требований осуществлялась в медиаторе.

Стоит также упомянуть о полях CurrentTime и TerminationTime класса WSResourceSpecification. Они используются для формализации требований “времени жизни” из спецификации WS-Resource, а также с требований спецификации WS-ResourceLifetime.

Сигнатура метода, моделирующего отдельный обмен сообщений стандарта WSRF, определяется структурой сообщений запросов и ответов. Запросы представляются в виде набора параметров спецификационного метода, ответы представляются как возвращаемые значения метода, а сообщения об ошибках моделируются исключениями.

Всё множество требований можно разделить на две основные группы: синтаксические и функциональные. Синтаксические требования налагают ограничения на структуру сообщений и связи между полями одного сообщения. Функциональные требования представляют собой ограничения на функциональность обработки запросов и связь между содержимым запроса и ответом на запрос.

Функциональные требования к реализациям WSRF представляют собой описания простых операций, таких как изменение значения поля ресурса, добавление или удаление свойств (операции с множеством), детерминированных и полностью определенных. Такие требования в рамках предлагаемого метода моделируются в явном виде с использованием блоков **update**, поддерживаемых в реализации UniTESK – инструменте JavaTESK. **update**-блок содержит инструкции, которые явным образом изменяют

значения полей модельного состояния в соответствии с текстом требования стандарта.

	WS-Base Faults	WS-Resource	WS-Resource Properties	WS-Resource Lifetime	WS-Service Group
Синтаксис	27	1	50	12	5
Семантика	2	11	109	39	68

Таблица 2. Соотношение между синтаксическими и семантическими требованиями стандарта WS-RF.

Синтаксические требования мы предлагаем проверять в медиаторах на этапе разбора сообщений, полученных от целевой системы.

3. Разработка медиатора

Основная функция медиатора, как компонента тестового набора, – это установление соответствия между модельным объектом (объектом спецификационного класса) и целевой системой. В случае тестирования ИПО Грид возможность непосредственного доступа к полям и методам системы отсутствует, потому воздействовать на ГС можно было лишь посредством отправки соответствующих сообщений-запросов, а реакции принимать – обрабатывая отклики. Соответственно, главной функцией медиатора является генерация сообщений-запросов и разбор сообщений-откликов с целью выявления полезной информации о выполнении операции.

В рассматриваемом методе медиаторы преобразуют параметры вызова спецификационного метода в XML сообщение, преобразуют в сообщение протокола SOAP/HTTP и отсылают его в целевую систему по установленному TCP соединению. Полученные ответы медиатор разбирает, проверяет соответствие ответа синтаксическим требованиям и формирует возвращаемое значение спецификационной функции.

Одна из особенностей разработки медиаторов для ИПО Грид заключается в том, что существующие реализации (в частности, Globus Toolkit 4.2) не удовлетворяют синтаксическим требованиям стандарта. Для проведения тестирования потребовалась адаптация медиаторов под нарушения стандарта, допускаемых реализацией. К примеру, Globus Toolkit 4.2 реализует один из последних предварительных проектов стандарта (draft) и поддерживает устаревшее пространство имен XML элементов (namespace); в результате реализация отвергала запросы, сформированные в соответствии со стандартом.

Также путем анализа исходного кода ИПО ГС Globus Toolkit 4.2, а также ряда экспериментов было выяснено, что реализацией не поддерживаются

следующие обмены сообщениями – PutResourceProperties, SetResourceProperties.

4. Разработка тестового сценария

Следующим этапом разработки тестового набора являлась разработка тестовых сценариев. Тестовым сценариям в технологии UniTESK отводится немаловажная роль. Как уже указывалось ранее, для их построения используется конечно-автоматная модель целевого ПО. При таком выборе модели актуальны ответы на следующие вопросы:

1. Как вычислять состояния тестируемой системы?
2. Как осуществляются переходы между состояниями?
3. Как задается автомат теста и как производится обход графа переходов автомата?

Ответ на первый вопрос не может быть дан, исходя только из общих соображений. Для каждой целевой системы состояние должно быть смоделировано с учётом её особенностей и требований, предъявляемых к качеству тестового набора вообще.

В случае ИПО Грид в качестве объекта тестирования выступает WS-ресурс. Основной его характеристикой выступает, очевидно, RPD, как наиболее полно описывающая ресурс сущность. Следовательно, RPD может быть предъявлен в качестве описателя состояния WS-ресурса. Однако, такой подход является избыточным, а тестовые сценарии на его основе, вообще говоря, могут выполняться неопределенно долгое время. Действительно, представим себе, что WS-ресурс обладает n свойствами, каждое из которых может принимать только значение “1”, и пусть изначально его RPD пуст. Рассмотрим конечный автомат, построенный на основе такого ресурса, причем в качестве модели состояния будем использовать его RPD, а в качестве переходов – два метода: add и remove, обозначающие, соответственно добавление нового свойства и удаление некоторого старого. Тогда из начального состояния автомат сможет перейти в n состояний, соответственно, образуется 2n дуг на графе состояний. Из состояний “ранга 1” (т.е. таких, при которых RPD WS-ресурса содержит только одно свойство), которых ровно C_n^2, в состояния “ранга 2” уже будет $2*n*C_n^2 = n*(n-1)$ переходов. Из состояний “ранга 2” в состояния “ранга 3”, которых имеется C_n^3 переходов уже будет $2*C_n^2*C_n^3 = (n-1)^2*(n-2)^2*(n-3) \sim n^5$ при больших n. Это означает, что граф состояний будет иметь 2^n вершин, а число ребер – будет расти экспоненциально с ростом n, что приведет к крайне медленной работе обходчика а, следовательно, и замедлит работу тестов. Другим важным недостатком такого подхода является его избыточность. Действительно, с

точки зрения тестирования методов add и remove совершенно необязательно обходить все ребра такого графа (т.к. все свойства такого WS-ресурса с “точки зрения” данных методов эквивалентны). Достаточно обработать лишь некоторые (например, ребра, идущие из состояния, в котором все n свойств WS-ресурса равны 1 и ребра, идущие из начального состояния).

Потому в ряде случаев представляется разумным выбирать другую модель состояния тестируемой системы. При разработке тестового сценария для тестирования соответствия ИПО Грид стандарту WSRF в качестве идентификатора состояния автомата теста мы предлагаем использовать мощность RPD WS-ресурса, т.е. количество свойств WS-ресурса. С одной стороны, такой способ существенно уменьшает размер предполагаемого графа состояний а, следовательно, и время работы тестов. С другой стороны, при таком задании состояния сохраняется детерминированность графа, что принципиально для корректной работы обходчика UniTESK.

Ответ на второй вопрос традиционен для большинства приложений UniTESK – переходы между состояниями автомата осуществляются посредством подачи стимулов в целевую систему. В качестве стимулов в случае построения тестового набора для ИПО Грид выступают, естественно, вызовы медиаторных методов. Последние же представляют собой не что иное, как отправки соответствующих XML-сообщений целевому ресурсу.

Теперь перейдем к обсуждению третьего вопроса – заданию автомата теста. Так как используется синхронная модель целевой системы, то для каждого спецификационного метода можно задать отдельный сценарный метод, который будет перебирать параметры метода и вызывать (неявно) медиатор для оказания тестового воздействия и оракул для проверки правильности возвращаемого значения.

5. Опыт практического тестирования реализации ИПО Грид

Представленный выше метод к разработки тестовых наборов для ИПО Грид использовался при создании тестового набора для проверки соответствия реализаций ИПО Грид стандарту WSRF.

5.1. Обзор целевой системы

Объектом тестирования в данной работе является программный пакет Globus Toolkit 4.2. Ниже перечислены основные компоненты данного инструментария:

1) Компоненты поддержки времени выполнения (Common runtime components):
   1. C Core Utilities – обеспечение переносимости;
   2. C WS Core – поддержка разработки Web-сервисов и выполнения клиентских приложений на C;

   3. Java WS Core – поддержка разработки Web-сервисов и выполнения клиентских приложений на Java;
   4. CoG jglobus – поддержка безопасности и выполнения не Web-сервисной части Java;
   5. Core WS Schema – поддержка схемы стандартов WSRF и WSN;
   6. Python Core – разработка и исполнение WS и не-WS клиентских приложений на языке Python;
   7. XIO – расширяемые библиотеки ввода-вывода на C.

2) Управление данными (Data Management):
   1. GridFTP – протокол передачи файлов;
   2. OGSA-DAI – инфраструктура, основанная только на java-сервисах, для получения доступа к ресурсам и интеграции их в Грид;
   3. Reliable File Transfer – технология надежной передачи файлов, основанная на Web-сервисах;
   4. Replica Location – технология копирования и обнаружения данных;
   5. Data Replication – технология идентификации групп файлов в среде Грид, и их локального копирования.
3) Управление выполнением (Execution Management):
   1. GRAM – обнаружение местоположения, инициализация выполнения, наблюдение за работой и завершение удаленных задач на Грид-ресурсах;
   2. GridWay – коллективное использование вычислительных ресурсов;
   3. MPICH-G2 – реализация стандарта MPI [16].
4) Информационные сервисы (Information Services):
   1. MDS4 – технология для слежения за ресурсами и их поиска, включающая сервисы индексирования и триггеры.
5) Компоненты безопасности (Security):
   1. C Security – технология поддержки безопасности;
   2. CAS\SAML Utilities – технология, относящиеся к авторизации сообществом;
   3. Delegation Service – технология, предоставляющая хосту учетные данные;
   4. GSI-OpenSSH – модифицированная версия OpenSSH[18], которая поддерживает аутентификацию сертификатов и их предоставление, удаленный зарегистрированный доступ и сервис передачи файлов;
   5. MyProxy – технология хранения и извлечения учетных данных из репозитория.

Остановимся на некоторых принципиальных особенностях архитектуры Globus Toolkit 4.2.

Сервисно-ориентированная архитектура. Globus Toolkit 4.2 создан для поддержки приложений, в которых множества сервисов взаимодействуют посредством стандартных протоколов. ПО включает и сами сервисы полностью, и библиотеки, реализующие стандартные протоколы.

Сервисы инфраструктуры. Globus Toolkit 4.2 включает встроенные сервисы для организации, наблюдения, управления и контроля доступа к таким элементам инфраструктуры, как ресурсы данных и вычислительные ресурсы.

Web-сервисы. Globus Toolkit 4.2 использует протоколы стандартных Web-сервисов и механизмы описания сервисов, обнаружения, контроля доступа, аутентификации и авторизации.

Контейнеры. ПО Globus Toolkit 4.2 включает компоненты, которые могут быть использованы для конструирования контейнеров для “помещения” в них Web-сервисов, написанных на Java, C, или Python.

Безопасность. Подсистема безопасности выполняет задачи защиты сообщений, аутентификации, авторизации и передачи полномочий.

Компоненты. Компоненты Globus Toolkit 4.2 не отвечают, вообще говоря, нуждам конечного пользователя напрямую: большинство из них выступает скорее как TCP\IP библиотека или реализация Web-сервера, чем как Web-браузер. Вместо этого, Globus Toolkit 4.2 предоставляет широкий диапазон компонент и инструментов, которые обеспечивают высокоуровневые возможности, необходимые определенным сообществам пользователей.

Согласно утверждению Globus Alliance, компонент Globus Toolkit 4.2 Java WS Core реализует требования стандарта WSRF. В работе задача тестирования соответствия решалась именно для этого компонента.

В рамках Java WS Core Web-сервис – это просто Java-объект, а поддерживаемые им обмены сообщениями соответствуют методам класса. Для вызова методов класса следует посылать сервисам XML-сообщения по протоколу SOAP/HTTP. Стоит также заметить, что в реализации возможны передача сообщений как с применением шифрования, так и без него.

5.2. Тестирование реализации

Разработанный с помощью инструмента JavaTESK тестовый сценарий включает сценарии для семи методов (они же сценарные методы): Get-ResourcePropertyDocument, GetResourceProperty, GetMultipleResourceProperties Insert-, Update- и DeleteResourceProperties, а также ImmediateDestroy. Сценарий для GetResourceProperty предельно прост: в нём устанавливаются нижний и верхний пределы мощности RPD WS-ресурса, и пока мощность находится в заданном диапазоне, выбирается один из элементов RPD, т.е. некоторое свойство WS-ресурса. Имя свойства выбирается не произвольно, а как поле элемента массива, в который конвертируется RPD. Осуществляется это посредством применения метода toArray() библиотеки java.util.Set. Запрос искомого свойства осуществляется с помощью вызова одноименного медиаторного метода. Аналогичны сценарии для методов GetResourcePropertyDocument и GetMultipleResourceProperties.

Сценарий для InsertResourceProperties тоже несложен: в нём устанавливается только верхний предел мощности RPD WS-ресурса (нижний устанавливать

бессмысленно, т.к. новое свойство может быть добавлено даже в RPD мощности 0), а затем с помощью одноименного медиаторного метода отправляется искомое XML-сообщение. В данном сценарии случайным образом генерируется имя свойства и его значение (с помощью функций класса java.util.Random). В некотором смысле аналогичен ему сценарий для метода UpdateResourceProperties – также устанавливается верхний предел мощности RPD WS-ресурса, но вместе с ним устанавливается и нижний предел (т.к. невозможно изменить значение свойства у WS-ресурса, который не обладает ни одним свойством). Далее с помощью одноименного медиаторного метода отправляется XML-сообщение. Новое значение свойства также генерируется случайно.

Наконец, сценарий для обмена сообщениями DeleteResourceProperties таков: устанавливается нижний предел мощности RPD WS-ресурса, а затем вызывается одноименный медиаторный метод. Как и в сценарии для обмена GetResourceProperty, имя удаляемого свойства WS-ресурса не произвольно, а является именем элемента массива, в который конвертируется RPD с помощью метода toArray() библиотеки java.util.Set.

Соответствующий граф состояний тестируемой системы имеет следующий вид (Рис.1):

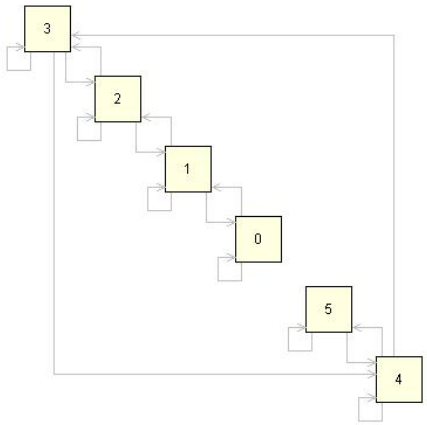

Рис.1. Граф состояний тестируемой системы

Дуги, соответствующие переходам, приводящим к увеличению номера состояния, соответствуют вызовам сценарного метода InsertResourceProperties, обратные – DeleteResourceProperties, а циклические – вызовам методов Get-ResourcePropertyDocument, GetResourceProperty, GetMultipleResourceProperties.

Стоит пояснить, почему именно 7 вышеуказанных обменов сообщениями были выбраны в качестве основных для написания соответствующего генератора XML-сообщений. В процессе разработки тестового набора

предполагалось, что с его помощью будут протестированы сервисы контейнера компонента Java WS Core инструментария Globus Toolkit 4.0. По умолчанию, всего в контейнере содержится 34 Web-сервисов, из которых 23 поддерживают некоторые обмены сообщениями из разделов WS-ResourceProperties и WS-ResourceLifetime стандарта WSRF. В частности, все они поддерживают обмен типа GetResourceProperty, 9 – ImmediateDestroy а также Insert-, Update- и DeleteResourceProperties, 14 – QueryResourceProperties, 13 – GetMultipleResourceProperties, и 7 – SetTerminationTime (обмен сообщениями из раздела WS-ResourceLifetime стандарта WSRF, соответствующий установке времени окончания работы WS-ресурса). Соответственно, данные 7 обменов сообщениями являлись как достаточно простыми в смысле написания спецификационных методов (чего нельзя сказать, к примеру, об обмене QueryResourceProperties), так и позволяли в полной мере провести тестирование соответствия стандарту WSRF 1.2.

Отдельный сценарий был реализован для метода ImmediateDestroy. В этом сценарии просто вызывается соответствующий медиаторный метод. Соответствующий граф состояний таков(Рис.2):

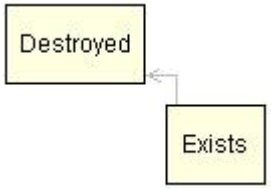

Рис 2. Граф состояний для сценария ImmediateDestroy

Переход здесь всего 1, результатом выполнения сценарного метода является уничтожение WS-ресурса (состояние “Destroyed”).

5.3. Результаты тестирования

В данной работе была протестирована реализация ИПО ГС Globus Toolkit 4.0 средствами технологии UniTESK (JavaTESK). В качестве результатов тестирования выступали сгенерированные инструментом отчеты о покрытии требований. Всего было покрыто 60% функциональности, в чём данный тестовый набор не уступает тестам, применяемым разработчиками Globus Toolkit'a; последние для оценки качества тестирования измеряют покрытие тестами кода с помощью инструментов JUnit и Clover, и данные инструменты позволили разработчикам установить, что их юнит-тесты покрывают 60% функциональности компонента Java WS Core. Однако, как уже говорилось ранее, эти тесты ничего не могут сообщить о соответствии реализации Globus Toolkit 4.0 какому-либо стандарту. Таким образом, существующие и разработанные тесты не только дополняют друг друга, но и позволяют с разных точек зрения рассматривать реализацию.

В разработанный тестовый набор не вошли спецификации и тесты для необязательных требований спецификации WS-ServiceGroup и требования к операции QueryResourceProperties.

При тестировании реализации были выявлены несоответствия семантическим требованиям стандарта WSRF 1.2 в обменах сообщениями InsertResourceProperties и UpdateResourceProperties. В частности, эти методы позволяют добавлять или изменять значения свойств WS-ресурса с разными идентификаторами (QName), что запрещается требованиями стандарта.

6. Существующие методы и подходы тестирования ИПО Грид

В 2006-м году Европейский институт стандартизации телекоммуникаций (ETSI) организовал экспертную группу по разработке тестового набора для тестирования совместимости ИПО Грид [23,24].

Метод, разрабатываемый в ETSI, основывается на разработке большого числа тестов (test cases) на языке TTCN-3. Каждый тест предназначен для проверки отдельной цели тестирования – тестовой ситуации, сформулированной полуформально на языке TPL (Test Purpose Language). В настоящее время ведется выделение целей тестирования.

Прототип тестового набора для ИПО Грид на языке TTCN3 представлен в работе [25]. Тестовый набор не связан с каким-либо стандартом для ИПО Грид. Вместо проверки требований стандарта тестовый набор проверяет *применимость* Грид в типовых сценариях использования – постановки вычислительной задачи в очередь, выполнение задачи на одном из вычислительных узлов, доставка результатов вычисления клиенту.

В рамках проекта построения распределенной системы EGEE обработки экспериментальных данных с Большого адронного коллайдера проводилось тестирование пакета gLite, разработанного в CERN. В качестве основного метода тестирования используется интеграционное тестирование – выполнение задач по пересылке крупных массивов данных[26]. В случае тестирования gLite этот метод тестирования оправдан, так как основное назначение EGEE – быстрая передача больших объемов данных с датчиков коллайдера в вычислительные системы. Задачу обеспечения совместимости с другими ИПО Грид разработчики EGEE не ставят.

Подход к тестированию ИПО Грид, близкий к представленному в данной статье, развивается в работе [27]. Авторы предлагают использовать формализм автоматов с абстрактным состоянием (Abstract State Machine, ASM) и автоматически генерировать тестовые последовательности из обхода автомата модели. Вопросы анализа стандартов ИПО Грид, выделения требований и построения формальной модели требований не рассматриваются.

7. Заключение

В данной работе решалась задача тестирования соответствия реализации ИПО ГС Globus Toolkit 4.2 стандарту систем управления распределенными ресурсами WSRF 1.2. Данный стандарт был проанализирован и на его основе составлен его каталог требований. Также была исследована структура и интерфейсы компонента Java WS Core программного пакета Globus Toolkit 4.2. На основе полученных данных был разработан тестовый набор для указанного программного пакета с применением технологии тестирования UniTESK (JavaTESK) и проведено тестирование. Тестирование показало, что реализация Globus Toolkit 4.2 соответствует стандарту WSRF, хотя выявило отсутствие выполнения ряда необязательных требований, как семантических, так и синтаксических. Также тестирование показало, что технология UniTESK применима для тестирования ИПО ГС, в частности, имеют место следующие особенности её применения:

1. Обмены сообщениями с Web-сервисами естественным образом моделируются спецификационными функциями;
2. Для разработки тестового набора нужна библиотека классов для автоматизации построения различных (в том числе некорректных) сообщений Web-сервисов.

В рамках данной работы были получены следующие результаты:

1. показана применимость технологии UniTESK для разработки тестовых наборов для инфраструктурного программного обеспечения (ИПО, middleware) Грид-систем;
2. разработан метод формализации требований и разработки тестов для стандартов ИПО Грид, основанных на архитектуре Web-сервисов;
3. разработан прототип тестового набора для проверки соответствия реализаций ИПО Грид базовому стандарту WSRF версии 1.2:
    a. проведен анализ базового стандарта WSRF, составлен каталог требований;
    b. на основе каталога требований разработана формальная спецификация;
    c. проведено исследование реализации, на основе которого разработан медиатор;
    d. разработан тестовый набор для реализации ИПО ГС Globus Toolkit 4.0 средствами технологии UniTESK (JavaTESK);

    e. проведено тестирование реализации ИПО ГС Globus Toolkit 4.0;
    f. при тестировании реализации был обнаружен ряд несоответствий базовому стандарту WSRF.

В качестве направлений для дальнейшей работы мы рассматриваем разработку тестового набора, проверяющего специфические требования к сервисам ИПО Грид, такие как сервис передачи больших массивов данных, сервис создания и управления вычислительными задачами и др.

Литература

[1] Ian Foster The Grid: Blueprint for a New Computing Infrastructure. — Morgan Kaufmann Publishers. — ISBN 1-55860-475-8
[2] Ян Фостер. Что такое грид? http://gridclub.ru/library/publication.2004-11-29.5830756248/publ_file/
[3] Веб-сайт консорциума Globus Alliance. http://globus.org/
[4] Пакет ИПО Грид UNICORE http://www.unicore.eu/
[5] Пакет ИПО Грид gLite. http://glite.web.cern.ch/glite/
[6] Open Grid Services Architechture http://www.ogf.org/documents/GFD.80.pdf
[7] Web Service Resource Framework http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
[8] WS-Management standard. http://msdn2.microsoft.com/en-us/library/aa384470.aspx
[9] В.В. Галактионов, Н.А. Кутовский Тестирование GT4 в ОИЯИ. http://lit.jinr.ru/Reports/annual-report05/gt4-48.pdf
[10] RFC 2119 Key words for use in RFCs to Indicate Requirement Levels. http://www.ietf.org/rfc/rfc2119.txt
[11] Web Services Description Language ver. 2.0 http://www.w3.org/TR/wsdl20/
[12] Web Services Activity конрциума W3. http://www.w3.org/2002/ws/
[13] Single Object Access Protocol. http://www.w3.org/TR/soap/
[14] XML Information Set. http://www.w3.org/TR/xml-infoset/
[15] Web Services Addressing. http://www.w3.org/Submission/ws-addressing/
[16] XML Path Language (XPath). http://www.w3.org/TR/xpath
[17] A Message-Passing Interface Standard (MPI). http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html
[18] OpenSSH project. http://www.openssh.org/
[19] А.В. Баранцев, И.Б. Бурдонов, А.В. Демаков, С.В. Зеленов, А.С. Косачев, В.В. Кулямин, В.А. Омельченко, Н.В. Пакулин, А.К. Петренко, А.В. Хорошилов. Подход UniTESK к разработке тестов: достижения и перспективы. Труды Института системного программирования РАН, №5, 2004. URL: http://www.citforum.ru/SE/testing/UniTesK
[20] Bertrand Meyer. Applying 'Design by Contract'. IEEE Computer, vol. 25, No. 10, October 1992, pp. 40-51.
[21] Software Testing Framework JUnit. http://www.junit.org/
[22] Code Coverage Analysis “Clover”. http://www.atlassian.com/software/clover/
[23] S. Schulze. Achieving Grid Interoperability: The ETSI Approach. The 20th Open Grid Forum - OGF20/EGEE 2nd User Forum. Manchester, UK. May 7 - 11, 2007

[24] GRID Activity. European Telecommunication Standardization Institute. http://www.etsi.org/WebSite/Technologies/GRID.aspx
[25] T.Rings, H.Neukirchen, J.Grabowski. Testing Grid ApplicationWorkflows Using TTCN-3. First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway, April 9-11, 2008.
[26] E.Slabospitskaya, V.Petukhov, and G.Grosdidier. Glite I/O Storm Testing in EDG-LCG Framework. Nuclear Electronics and Computing 2005. Varna, Bulgaria, 12-18 September 2005.
[27] Lamch, D.; Wyrzykowski, R. Specification, Analysis and Testing of Grid Environments Using Abstract State Machines. International Symposium on Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. 13-17 Sept. 2006 Pages:116 - 120