УДК 004.75

РЕАЛИЗАЦИЯ ПРОГРАММНОГО ИНТЕРФЕЙСА ГРИД-СЕРВИСА PILOT НА ОСНОВЕ АРХИТЕКТУРНОГО СТИЛЯ REST

А. П. Демичев[1], В. А. Ильин[1], А. П. Крюков[1], Л. В. Шамардин[1]

Рассматриваются особенности реализации грид-сервисов на основе архитектурного стиля REST. Приводятся базовые элементы интерфейса прикладного программирования сервиса запуска многошаговых заданий Pilot проекта ГридННС. Описываются общие принципы построения интерфейса для таких сервисов и методы идемпотентного вызова сложных операций. Рассматриваются разные механизмы аутентификации клиентов. Работа выполнена при поддержке Федерального агентства по науке и инновациям (государственные контракты № 01.647.11.2004 и 02.740.11.0388) и РФФИ (проект № 10–07–00323-а).

Ключевые слова: веб-сервисы, грид-сервисы, грид, REST, API.

1. Введение. Использование архитектурного стиля REST (REpresentation State Transfer) для построения грид-сервисов [1] открывает возможность создания сервисов с простыми в реализации интерфейсами прикладного программирования, не требующими сложных дополнительных библиотек или средств автоматической генерации кода. В настоящей статье мы рассмотрим общие принципы построения интерфейса прикладного программирования грид-сервиса на примере грид-сервиса запуска многошаговых заданий Pilot, разработанного в рамках проекта ГридННС [2].

Сервис выполнения многошаговых заданий Pilot работает с заданиями, которые представляют собой объекты сложной структуры, состоящие из элементарных задач. Каждая элементарная задача является запуском какой-либо программы на выполнение на грид-ресурсе по протоколу Globus WS-GRAM. Интерфейс прикладного программирования этого сервиса должен предоставлять возможность создания ресурсов заданий и задач, изменения их свойств, выполнения операций над заданиями (запуск, отмена), получения информации о состоянии заданий и задач, а также получения учетной информации о выполнении заданий и задач. Таким образом, основными ресурсами сервиса являются задания, задачи и учетная информация.

2. Организация коллекций ресурсов. Сервис выполнения многошаговых заданий Pilot занимается выполнением заданий, состоящих из задач. Описание задания является направленным ациклическим графом, определяющим порядок выполнения задач и зависимости между ними. Задачи запускаются на грид-ресурсы, информация о которых доступна из центрального реестра ресурсов. Для выбора ресурсов реализуется простое динамическое планирование в момент запуска с поиском подходящих ресурсов по критериям из описания задачи. В случае неуспешного завершения задачи возможно продолжение выполнения задач, не зависящих от аварийно завершенной, либо остановка выполнения всего задания в целом.

Все запуски заданий и задач, производимые сервисом, протоколируются. Внешние сервисы мониторинга и учета либо администраторы виртуальных организаций могут обращаться к сервису выполнения многошаговых заданий за учетной информацией о выполнении заданий и задач.

В дальнейшем при описании структуры ресурсов сервиса префикс `pilot/` в записи будет использоваться для обозначения адреса сервиса запуска многошаговых заданий. Например, для сервиса, расположенного по адресу `https://tb01.ngrid.ru/`, полному URI `https://tb01.ngrid.ru/jobs/` будет соответствовать запись `pilot/jobs/`.

Ресурсы сервиса выполнения многошаговых заданий группируются в коллекции согласно рекомендациям из [3]: отдельные элементы пути в URI будут разделять уровни иерархии ресурсов сервиса или задавать путь в направленном графе связанных ресурсов. Заметим, что URI ресурсов будут соответствовать отдельным ресурсам, но не операциям над ресурсами.

[1] Научно-исследовательский институт ядерной физики им. Д. В. Скобельцына, Московский государственный университет им. М. В. Ломоносова (НИИЯФ МГУ), Ленинские горы, д. 1, стр. 2, 119991, Москва; А. П. Демичев, ст. науч. сотр., e-mail: demichev@theory.sinp.msu.ru; В. А. Ильин, зав. лабораторией, e-mail: ilyin@theory.sinp.msu.ru; А. П. Крюков, ведущий науч. сотр., e-mail: kryukov@theory.sinp.msu.ru; Л. В. Шамардин, мл. науч. сотр., e-mail: shamardin@theory.sinp.msu.ru

2.1. Задания и задачи. Задания пользователя объединяются в коллекцию `pilot/jobs/`. Запросы `GET` к этой коллекции возвращают списки заданий пользователя, при этом каждое задание получает URI вида `pilot/jobs/<jobid>`. Для создания нового задания клиент генерирует идентификатор задания `<newid>` и направляет запрос `PUT`, используя предполагаемый идентификатор нового задания, во входную очередь сервиса по адресу `pilot/submission/<newid>`. В случае если идентификатор задания не является уникальным, сервис возвращает клиенту код ошибки `409 Conflict`. Для восстановления после такой ошибки клиент может повторить попытку создать задание, сгенерировав новый идентификатор. При создании задания клиент может использовать запрос с заголовком `Expect` протокола HTTP/1.1 [4], позволяющий получить от сервера подтверждение возможности создания задания с таким идентификатором без пересылки описания задания и другой информации от клиента. Для генерации идентификаторов используется алгоритм, подобный time-based UUID из раздела 4.2 в [5]. Такой порядок создания задания гарантирует идемпотентность операции и позволяет использовать простой механизм восстановления после сбоев [1]. Ответ сервиса содержит URI созданного задания.

Приведем следующий пример запроса на создание задания от клиента и ответ сервиса на этот запрос (ответы сервиса выделены курсивом):

PUT /submission/jfKDIaWQ HTTP/1.1
Host: tb01.ngrid.ru:5053
Expect: 100-continue

HTTP/1.1 100 Continue

Connection: close
Content-MD5: 2iuaMUZlhPEaS33ticwJhg==
Content-Type: application/json
Content-Length: 6117
User-Agent: pilot_cli-0.1dev
{"definition": . . .,
"proxy": . . .}

HTTP/1.1 201 Created
Server: Pilot-xxx Python/2.4.6
Date: Fri, 28 May 2010 08:23:03 GMT
Pragma: no-cache
Cache-Control: no-cache
Location: https://tb01.ngrid.ru:5053/jobs/jfKDIaWQ

Каждое задание представляет собой не только самостоятельный ресурс, который может быть изменен при помощи запросов PUT или удален при помощи запроса DELETE, но и коллекцию ресурсов — отдельных задач задания. Эти ресурсы создаются, удаляются и изменяются автоматически при создании задания. Задачи получают URI вида `pilot/jobs/<jobid>/<taskid>`.

Запросы `GET` по адресам заданий предоставляют информацию о состоянии заданий, а также оглавление коллекции задач. Запросы GET по адресам задач предоставляют информацию о состоянии задач. Для удаления задания используются запросы `DELETE` по URI задания. Удаление отдельных задач не предусмотрено.

2.2. Учетная информация. Учетная информация представляет собой динамическую коллекцию, адреса ресурсов в которой имеют вид `pilot/accounting/<ts1>-<ts2>`, где `<ts1>` и `<ts2>` — дата и время начала и конца временно́го интервала во временно́й зоне UTC, учетная информация для которого предоставлена в данном ресурсе. В качестве `<ts2>` возможно указание специального ключевого слова current, обозначающего текущий момент времени.

По умолчанию для всех ресурсов сервиса запуска многошаговых заданий используется представление в формате JSON. Если клиент указывает другой желаемый формат представления ресурса через заголовок `Accept`, то сервис возвращает представление в запрашиваемом формате. Например, для большинства ресурсов поддерживается формат `text/html`, позволяющий получить представления ресурсов, пригодные для отображения в обычном веб-браузере, что допускает его использование в качестве самого простого клиента грид-сервиса запуска многошаговых заданий.

3. Выполнение сложных операций. Некоторые операции, выполняемые над ресурсами, могут требовать длительного времени для выполнения. Для сервиса запуска многошаговых заданий такими операциями являются, например, запуск или приостановка выполнения задания. Выполнение этих опе-

раций реализуется при помощи протокола, описываемого ниже на примере выполнения операции запуска задания.

Каждое задание имеет атрибут, соответствующий списку операций, выполненных над заданием или находящихся в очереди на выполнение. Элементы этого списка имеют такие атрибуты, как

— выполняемая операция;

— результат выполнения операции (если операция была выполнена, успешно или не успешно);

— идентификатор операции.

Для выполнения операции запуска задания клиент отправляет сервису запрос на модификацию задания, содержащий новую создаваемую операцию, при этом клиент должен сгенерировать новый идентификатор операции, уникальный в рамках одного задания. Новая операция, отмеченная как невыполненная, помещается в список операций задания. В случае если запрос на изменение списка операций задания содержал идентификатор операции, который уже использован, клиент получает ответ с указанием на ошибку и новая операция не создается.

Таким образом, реализуется протокол создания новых операций задания, аналогичный протоколу создания новых заданий и удовлетворяющий требованиям идемпотентности операций из [1] и [4].

4. Аутентификация запросов. Протокол HTTP (HyperText Transfer Protocol) имеет стандартный механизм аутентификации пользователей, использующий заголовки `Authorization` и `WWW-Authenticate` (см. [4]). Начальная аутентификация через этот механизм производится стандартной последовательностью запрос–вопрос–ответ (request–challenge–response). Клиент посылает запрос, требующий аутентификации, сервису и получает ответ `401 Unauthenticated` с заголовком `WWW-Authenticate`, содержащим по крайней мере один вопрос (возможно, содержащий дополнительные параметры), ответ на который клиент должен сообщить серверу в заголовке `WWW-Authenticate`, повторив запрос. При последующих запросах клиент может заблаговременно передавать заголовок `Authorization` в запросах, используя для вычисления ответов информацию исходного вопроса.

Для работы с этими заголовками определяются две стандартные схемы аутентификации, Basic и Digest, подробно описанные в [6]. Эти схемы используют в качестве параметров доступа имя пользователя и пароль. В схеме Basic имя пользователя и пароль передаются непосредственно сервису с каждым запросом, а в схеме Digest с каждым запросом передается результат вычисления некоторой криптографической функции от имени пользователя, пароля, параметров запроса и дополнительных случайных параметров, которые сервер и клиент выбирают для защиты авторизационной сессии. Поскольку в схеме аутентификации Basic данные передаются в форме, позволяющей перехватить имя пользователя и пароль, использовать эту схему рекомендуется только поверх защищенных соединений, например с использованием HTTP через TLS (Transport Layer Security), т.е. с использованием протокола HTTPS (Hypertext Transfer Protocol Secure).

Схема аутентификации с использованием заголовков texttttAuthorization / `WWW-Authenticate` предусматривает возможность использования любых расширенных механизмов аутентификации. Например, помимо стандартных схем, определенных в [6], опубликована также схема аутентификации [7], переносящая параметры сообщений WSSE [8] внутрь авторизационного заголовка.

Однако существует еще один метод авторизации запросов, использующий для аутентификации протокол, расположенный на более низком уровне, а именно использование установления соединения (handshake) с аутентификацией клиента по сертификату на уровне протокола TLS при доступе к сервису через HTTPS. Большинство уже существующих грид-сервисов, как построенных на протоколах WSRF/OGSA, так и использующих GSI из Globus Toolkit, применяют клиентские сертификаты X.509 и прокси-сертификаты [9] для аутентификации пользователей, поэтому аутентификация пользователей RESTful-грид-сервисов при помощи клиентских сертификатов или прокси-сертификатов является оправданной. Более того, такой способ аутентификации открывает возможность использования информации из атрибутных сертификатов [10] для принятия решений об авторизации. В частности, атрибутные сертификаты VOMS [11] получили достаточно широкое распространение в проектах EGEE/WLCG.

Сложность такого подхода заключается в том, что наиболее распространенные реализации HTTPS-серверов, включая Apache mod_ssl, не поддерживают аутентификацию с использованием прокси-сертификатов. Однако для сервера Apache существует дополнительный модуль расширения mod_gridsite [12], включающий в себя не только поддержку прокси-сертификатов для аутентификации клиентов, но и предоставляющий приложениям информацию из атрибутных сертификатов с VOMS-расширениями. К недостаткам mod_gridsite можно отнести зависимости от библиотек Globus Toolkit. Для упрощения реализации RESTful-грид-сервисов на языке программирования Python авторами настоящей публикации был разработан сервер приложений, совместимый со спецификацией Python WSGI [13], предоставляю-

щий приложениям полный доступ к информации о клиентском сертификате, цепочке делегирования и VOMS-расширениях.

5. Заключение. При разработке интерфейса прикладных программ для грид-сервисов на основе архитектурного стиля REST основным принципом является выделение и группировка ресурсов в иерархические коллекции, при этом часть ресурсов могут быть зависимыми и создаваться (актуализироваться) автоматически при создании их "родительских" ресурсов. Простые операции над ресурсами выполняются при помощи стандартных методов HTTP.

Для выполнения сложных или длительных операций возможно построение протокола регистрации операций, удовлетворяющего требованиям идемпотентности спецификации HTTP и [1].

Аутентификация клиентов RESTful-грид-сервисов может производиться как при помощи стандартных для HTTP механизмов (Basic, Digest), так и при помощи средств, стандартных для традиционных грид-сервисов, таких как сертификаты X.509 или прокси-сертификаты. В последних случаях возможно использование специфических атрибутных сертификатов, таких как расширения VOMS.

На основе принципов, изложенных в настоящей статье, разработан грид-сервис выполнения многошаговых заданий Pilot, используемый в проекте ГридННС для запуска сложных заданий на распределенных грид-ресурсах.

СПИСОК ЛИТЕРАТУРЫ

1. *Демичев А., Крюков А., Шамардин Л.* Принципы построения грид с использованием restful-веб-сервисов // Программные продукты и системы. 2009. № 4.
2. Проект ГридННС (http://www.ngrid.ru/).
3. *Richardson L., Ruby S.* RESTful web services. Milwaukee: O'Reilly Media, 2007.
4. *Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T.* Hypertext Transfer Protocol — HTTP/1.1. Tech. rep. IETF Network Working Group. 1999. June. RFC2616 (http://tools.ietf.org/html/rfc2616).
5. *Leach P., Mealling M., Salz R.* A Universally Unique IDentifier (UUID) URN namespace. Tech. rep. IETF Network Working Group. 2005. July. RFC4122 (http://tools.ietf.org/html/rfc4122).
6. *Franks J., Hallam-Baker P., Hostetler J., Lawrence S., Leach P., Luotonen A., Stewart L.* HTTP authentication: basic and digest access authentication. Tech. rep. IETF Network Working Group. 1999. June. RFC2617 (http://tools.ietf.org/html/rfc2617).
7. *Pilgrim M.* Atom authentication. Available online (http://www.xml.com/pub/a/2003/12/17/dive.html).
8. *Nadalin A., Kaler C., Monzillo R., Hallam-Baker P.* Web Services Security UsernameToken Profile 1.1. Tech. rep. OASIS Standard, 2006 (http://www.oasis-open.org/committees/download.php/16782/wss-v1.1- spec-os-UsernameTokenProfile.pdf).
9. *Tuecke S., Welch V., Engert D., Pearlman L., Thompson M.* Internet X.509 Public Key Infrastructure (PKI) proxy certificate profile. Tech. rep. IETF Network Working Group. 2004. June. RFC3820 (http://tools.ietf.org/html/rfc3820).
10. *Farrell S., Housley R., Turner S.* An internet attribute certificate profile for authorization. Tech. rep. IETF Network Working Group. 2010. January. RFC5755 (http://tools.ietf.org/html/rfc5755).
11. *Groep D.* The VOMS attribute certificate format. OGF Draft, artf6312, 2010 (http://forge.gridforum.org/sf/go/artf6312).
12. *McNab A.* The gridsite web/grid security system // Grid Security Workshop. Oxford, 2004.
13. *Eby P.J.* Python Web Server Gateway Interface v1.0. PEP 333, 2004 (http://www.python.org/dev/peps/pep-0333/).

Поступила в редакцию
28.05.2010