АРХИТЕКТУРА И РЕАЛИЗАЦИЯ СЕРВИС-ОРИЕНТИРОВАННОЙ НАУЧНОЙ СРЕДЫ MATHCLOUD*

О.В. Сухорослов
Центр грид-технологий и распределенных вычислений ИСА РАН
E-mail: os@isa.ru

Аннотация

В докладе рассматриваются принципы реализации распределенной научной среды MathCloud. Целями данной среды являются предоставление унифицированного доступа к проблемно-ориентированным вычислительным сервисам и поддержка интеграции данных сервисов при решении прикладных задач. Во главу предлагаемого подхода к реализации среды MathCloud ставятся удобство разработки сервисов, простота доступа к сервисам пользователей и использование открытых технологий. Архитектура среды основана на представлении сервиса в виде функции с заданным набором входных и выходных параметров и применении стиля REST для описания унифицированного интерфейса сервиса. Данный интерфейс поддерживает обмен данными в формате JSON, асинхронную обработку запросов и получение описания сервиса. Разработан контейнер сервисов, реализующий указанный интерфейс и поддерживающий быстрое преобразование в сервисы приложений с интерфейсом командной строки. Каждый сервис, развернутый в контейнере, доступен пользователям среды через веб-браузер. Произведена интеграция контейнера сервисов с грид-инфраструктурой EGEE, позволяющая преобразовывать в сервисы MathCloud существующие грид-приложения. Для поддержки интеграции сервисов среды при решении прикладных задач реализованы редактор композитных приложений и система управления сценариями на основе workflow-подхода.

Введение

При решении научных задач современный исследователь все чаще сталкивается с отсутствием требующихся ему ресурсов на своем компьютере, будь то информационные ресурсы, вычислительные мощности или программное обеспечение. При этом необходимые ресурсы могут быть найдены на серверах, вычислительных системах или компьютерах коллег, распределенных административно и территориально. Современные сетевые технологии позволяют избежать переноса всех требуемых ресурсов на один компьютер путем организации удаленного доступа к данным ресурсам с последующим их объединением в рамках распределенных приложений.

Первой вехой на пути к повсеместной интеграции ресурсов научного сообщества стало появление в начале 90-х годов World Wide Web. Сейчас Web является крупнейшей распределенной системой, предоставляющей доступ к огромному количеству информационных ресурсов. Успех Web обусловлен рядом важных особенностей ее архитектуры. Клиент-серверная модель Web обеспечивает неотчуждаемость ресурсов от их владельцев, позволяет оперативно обновлять содержимое ресурсов и контролировать доступ

* Работа выполнена при поддержке РФФИ (грант 08-07-00430-а) и Президиума РАН (программа П1).

к ним. В то же время, использование гипертекста с односторонними ссылками позволяет легко ссылаться на ресурсы Web, не требуя при этом участия их владельцев. Web основана на открытых стандартах, что позволяет создавать независимые реализации серверов и клиентов.

Одновременно с появлением Web начали делаться первые шаги по интеграции распределенных вычислительных ресурсов. В рамках концепции метакомпьютинга впервые было предложено использовать географически распределенные суперкомпьютеры как единую вычислительную среду для решения сложных научных задач. Широкое распространение получили проекты добровольных вычислений, объединяющие ресурсы простаивающих компьютеров. В 2000-х годах начали формироваться глобальные грид-инфраструктуры, ориентированные на интеграцию высокопроизводительных вычислительных ресурсов для поддержки крупных научных проектов.

Несмотря на внушительный объем агрегированных ресурсов, круг сегодняшних пользователей и приложений грид относительно узок. Обусловлено это главным образом недостатком сервисов, позволяющих пользователю сформулировать интересующую его задачу через привычный проблемно-ориентированный интерфейс и преобразующих данную задачу в вычислительные задания, запускаемые в грид. Стоит также отметить сложность освоения и использования низкоуровневого ПО грид, что затрудняет разработку подобных сервисов.

Современные грид-системы ориентированы на интеграцию высокопроизводительных вычислительных ресурсов для решения задач с предельно высокими требованиями к подобным ресурсам, а также задач, допускающих декомпозицию на множество небольших независимых подзадач. Представляется, что концепция грид-вычислений может быть использована для решения более широкого класса задач, отличительной особенностью которых является возможность их декомпозиции на относительно «крупные» типовые подзадачи. Данный класс фактически охватывает широкий спектр вычислительных задач математики, физики, химии, биологии и т.д. Для решения таких задач требуется набор сервисов решения типовых вычислительных математических задач, связанных друг с другом в соответствии со схемой решения исходной задачи. Появление возможности решения сложных задач путем композиции распределенных проблемно-ориентированных сервисов способно вывести распределенные вычислительные среды на качественно новый уровень.

В настоящее время все большее распространение получает модель научной кооперации и разделения труда, основанная на совместной работе территориально распределенных, часто - международных, коллективов исследователей. В подобных “распределенных” научных проектах остро стоит проблема совместного использования наработок сторон, вовлеченных в проект. Чаще всего речь идет о тех или иных вычислительных пакетах, приложениях и моделях, а также архивах и базах данных. Эта проблема может быть решена путем преобразования данных наработок в удаленно доступные проблемно-ориентированные сервисы. Другими ситуациями, в которых может возникать потребность в создании подобных сервисов, являются публикация и обмен результатами научных исследований, внедрение и коммерциализация полученных результатов и создание образовательных ресурсов.

Предлагаемый подход, охватывающий фактически все этапы научных исследований,

состоит в построении распределенных вычислительных сред нового поколения, предоставляющих доступ к проблемно-ориентированным сервисам и образующих универсальную инфраструктуру для научной кооперации. Данная инфраструктура базируется на сервис-ориентированном подходе: пользователи преобразуют свои приложения в удаленно доступные сервисы, которые могут быть обнаружены и использованы другими пользователями для решения интересующих их задач. Данный подход обобщает идеи совместного использования вычислительных ресурсов в грид, расширяя при этом возможности системы и ликвидируя разрыв между прикладными задачами и вычислительной инфраструктурой.

В основе предлагаемого подхода лежит понятие сервис-ориентированной архитектуры (Service Oriented Architecture, SOA). Как подход к организации распределенной вычислительной инфраструктуры, SOA заключается в построении данной инфраструктуры, отталкиваясь от решаемых задач, а не используемых при этом технологий. Акцент делается на совместном и повторном использовании типовой функциональности, оформленной в виде доступных по сети сервисов. Новые приложения могут создаваться путем обнаружения и композиции существующих сервисов. Функциональность сервиса может одновременно использоваться в контексте сразу нескольких приложений. При этом сервису может быть не известно о том, в контексте каких приложений он используется. Кроме того, при выполнении запросов сервис может использовать функциональность других сервисов. Таким образом, в SOA достигается переход от монолитных распределенных приложений к приложениям, состоящим из набора слабо связанных (loosely coupled) распределенных компонентов, обнаруживаемых динамически в сети.

Предлагаемый подход также согласуется с получившей широкое распространение в последние годы моделью «приложение-как-сервис» (Software-as-a-Service, SaaS), заключающейся в оформлении приложения в виде удаленно доступного сервиса. Данная модель обладает рядом преимуществ по сравнению с традиционной моделью распространения приложений. Поставщик приложения сохраняет полный контроль над приложением и средой его выполнения, что упрощает поддержку и обновление приложения, а также его коммерциализацию. Пользователю приложения, в свою очередь, не требуется производить закупку необходимого оборудования, установку, настройку и постоянную поддержку приложения. Для коммерческих сервисов затраты пользователя могут быть сведены к оплате за использование сервиса, рассчитываемой на основе объема фактически использованных клиентом ресурсов.

Актуальность предлагаемого направления исследований подтверждается сформулированной в 2005 году концепцией «сервис-ориентированной науки» (Service-Oriented Science) [1], автором которой выступил Иэн Фостер, один из основположников грид-вычислений. В соответствии с данной концепцией, сервис-ориентированный подход позволяет организовать повсеместный доступ к разнородным научным ресурсам и автоматизировать процесс научных исследований, тем самым, повышая производительность исследований и открывая новые возможности для науки в целом. Концепция “сервис-ориентированной науки” перекликается с более широкой концепцией «электронной науки» (e-Science), сформулированной в 1999 году Джоном Тэйлором как «глобальное сотрудничество в ключевых областях науки и обеспечивающая его инфраструктура нового поколения».

Следует подчеркнуть, что сервис-ориентированные научные среды не являются отрицанием грид-систем, а, напротив, - должны базироваться на вычислительной инфраструктуре грид, используя ее для проведения сложных вычислений и хранения больших массивов данных. Таким образом, речь идет о естественной эволюции грид и реализации новых системных уровней над уже созданной инфраструктурой. Новизна же предлагаемого подхода состоит в смещении акцента от агрегации вычислительных ресурсов на решаемые с помощью агрегированных ресурсов задачи. Если грид-системы развивались снизу вверх, начиная с «сырых» ресурсов, то среды нового поколения нацелены на отображение прикладных задач на доступные в грид ресурсы путем создания проблемно-ориентированных сервисов.

В настоящее время отсутствуют проработанные подходы к реализации сервис-ориентированных научных сред. Обусловлено это как относительной новизной данного направления, так и рядом технологических проблем, стоящих на пути реализации подобных сред, таких как:

- унификация и обеспечение интероперабельности сервисов на уровне протоколов, интерфейсов, форматов и семантики данных;
- организация безопасного доступа к сервисам, включая защиту передаваемых по сети данных, аутентификацию и авторизацию пользователей, учет использования ресурсов и т.д.;
- снижение технологического барьера для потенциальных разработчиков сервисов за счет упрощения процедур создания и размещения сервисов в сети;
- обеспечение масштабируемости сервисов, в том числе за счет динамического подключения внешних вычислительных ресурсов;
- снижение технологического барьера для потенциальных пользователей сервисов путем реализации проблемно-ориентированных интерфейсов («рабочих пространств»), скрывающих техническую сторону функционирования сервисов и сложность низлежащей вычислительной инфраструктуры;
- создание механизмов публикации, аннотации и оценки качества сервисов, позволяющих пользователю быстро найти интересующий его сервис;
- создание механизмов формирования сообществ пользователей (виртуальных организаций), образующих контексты для разделения сервисов.

Несмотря на имеющийся опыт решения подобных проблем при создании грид-систем, существующие технологии зачастую сложны в использовании и требуют своего пересмотра или поиска новых решений. Сервис-ориентированные научные среды, за счет расширения круга ресурсов и приложений, должны обеспечивать функционирование множества виртуальных сообществ среднего и малого размера. При этом требуется радикально упростить процедуры развертывания соответствующего ПО, формирования сообществ пользователей, разработки сервисов и их размещения в среде. Существующие средства разработки грид-сервисов, такие как Globus Toolkit, изначально ориентированы на разработчиков базовых сервисов грид, сложны в использовании и основаны на обладающих рядом недостатков спецификациях веб-сервисов.

Помимо технологических проблем, стоящих на пути реализации сервис-ориентированных научных сред, существуют проблемы организационно-методологического характера. Так, например, отсутствуют общепринятые практики и механизмы вовлечения исследователей в процесс создания проблемно-ориентированных сервисов, который в перспективе может стать одним из способов публикации и апробации научных результатов.

Для решения описанных проблем предлагается воспользоваться опытом разработки нового поколения Web-приложений, обозначаемого термином Web 2.0. Суть Web 2.0 состоит в уходе от модели пассивного потребителя информации и в преобразовании Web в платформу для создания приложений. Большое внимание уделяется формированию виртуальных сообществ пользователей для совместного создания, редактирования и обработки информации. Важным аспектом Web 2.0 является распространение Web-сервисов, предоставляющих удаленный программный доступ к приложениям посредством протокола HTTP. Многие популярные Web-приложения, например, Google Maps, имеют не только пользовательский Web-интерфейс, но и интерфейс прикладного программирования (Web API), позволяющий интегрировать данный сервис в произвольное приложение. Собственно это и превращает Web в платформу, обладающую теми же преимуществами для пользователей и поставщиков сервисов, что и Web-приложения. Развиваются средства композиции Web-сервисов, такие как Yahoo! Pipes, позволяющие буквально «собрать» новое приложение (т.н. mashup) из существующих сервисов и затем сделать его доступным в виде нового сервиса. Данный аспект является чрезвычайно важным, поскольку именно он запускает механизм саморазвития и роста системы. Ориентация на удобство конечного пользователя и простоту разработки сервисов привели к отказу от тяжеловесных технологий и стандартов (таких, например, как SOAP, WSDL и UDDI) в пользу лежащего в основе Web архитектурного стиля REST.

В статье рассматривается распределенная среда MathCloud [2], реализующая концепцию сервис-ориентированных научных сред. Целями среды являются предоставление унифицированного доступа к проблемно-ориентированным вычислительным сервисам и поддержка интеграции данных сервисов при решении прикладных задач. Во главу предлагаемого подхода к реализации среды MathCloud ставятся удобство разработки сервисов, простота доступа к сервисам пользователей и использование открытых технологий. Для этого используются архитектурный стиль REST, современные Web-технологии и наработки Web 2.0. Сервис-ориентированный подход позволяет пользователю MathCloud абстрагироваться от конкретных ресурсов, требуемых для решения задачи, и формулировать запрос к системе в терминах его предметной области. Тем самым, повышается уровень удобства использования среды для неподготовленного пользователя. Данный подход также идеально подходит для интеграции программных ресурсов, таких как математические и вычислительные пакеты. В случае, если для выполнения запроса сервису требуются вычислительные ресурсы, данный запрос может преобразовываться в вычислительные задания, запускаемые на кластере или в грид. Тем самым, реализованный подход обобщает идеи совместного использования вычислительных ресурсов в грид, расширяя при этом возможности среды и ликвидируя существующий разрыв между прикладными задачами и вычислительной инфраструктурой.

1. Архитектура среды MathCloud

Архитектура среды MathCloud состоит из нескольких уровней, изображенных на Рис. 1.

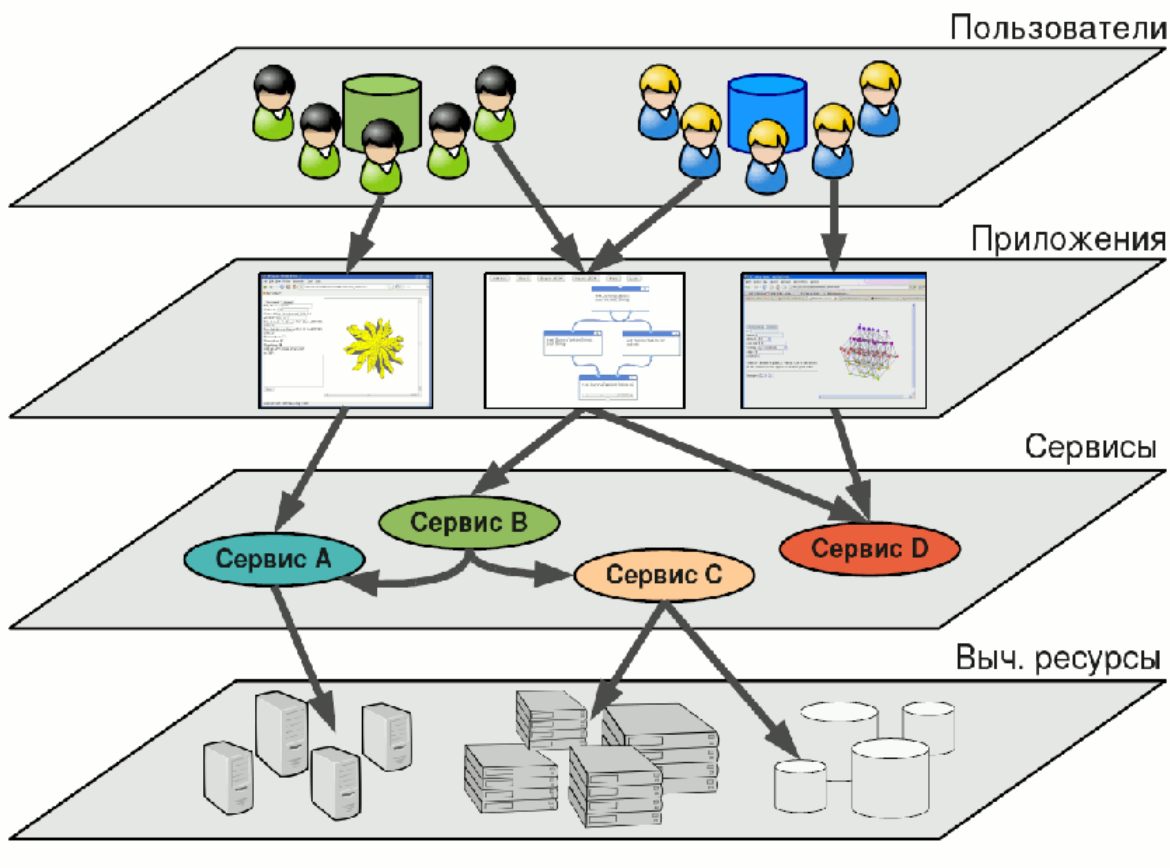

Рис. 1. Уровни архитектуры MathCloud.

1.1 Вычислительная инфраструктура

Нижний уровень архитектуры MathCloud содержит вычислительные ресурсы, используемые для функционирования сервисов среды. Формально данный уровень не относится к самой среде MathCloud, а формируется из доступных разработчикам и пользователям ресурсов. Унифицированные механизмы интеграции и доступа к вычислительным ресурсам уже сформированы в рамках грид-инфраструктур. Однако стоит отметить, что данные инфраструктуры ориентированы на запуск ограниченных по времени вычислительных заданий и не поддерживают размещение постоянно функционирующих сервисов. В то же время, в рамках коммерческих систем, таких как Amazon EC2, активно развивается модель аренды виртуализованных аппаратных ресурсов (cloud computing), позволяющая размещать на этих ресурсах и сами сервисы. В рамках среды MathCloud предполагается использование обоих видов вычислительных инфраструктур.

Доступ к вычислительным ресурсам может осуществляться с помощью соответствующих интерфейсов грид-инфраструктур. Однако, учитывая сложность освоения и использования промежуточного ПО грид-систем, в рамках MathCloud предполагается создание адаптеров, упрощающих взаимодействие сервисов с грид. Данные адаптеры

позволят прозрачным для разработчика сервиса способом транслировать запросы к сервису в вычислительные задания, запускаемые в грид. Это также позволяет значительно упростить процесс преобразования в сервисы существующих грид-приложений.

1.2 Уровень сервисов

На уровне сервисов реализуется удаленный программный доступ к некоторой востребованной пользователями среды функциональности. Проблемно-ориентированные сервисы, являющиеся главными компонентами среды, нацелены на решение определенного класса прикладных задач и служат основой для создания приложений. Помимо проблемно-ориентированных сервисов, в среде могут присутствовать системные сервисы, реализующие универсальную функциональность, такую как публикация и поиск сервисов. В дальнейшем под сервисами MathCloud будут подразумеваться проблемно-ориентированные сервисы. Уровень сервисов, как уже было отмечено, опирается на низлежащую вычислительную инфраструктуру, используемую для проведения сложных вычислений или хранения данных.

Остановимся подробнее на модели проблемно-ориентированного сервиса.

Сервис MathCloud представляет собой доступный по сети программный компонент, поддерживающий решение определенного класса задач с помощью соответствующих вычислительных алгоритмов. В соответствии с моделью клиент-сервер, сервис обслуживает приходящие к нему запросы клиентов на решение конкретных задач. Запрос клиента содержит параметризованное описание задачи, формулируемое в виде конечного набора входных параметров. После успешной обработки запроса сервис возвращает клиенту результат, оформленный в виде конечного набора выходных параметров.

Для взаимодействия с сервисом клиенту необходимо знать список входных и выходных параметров сервиса. Данная информация является частью описания сервиса, публикуемого в общедоступном месте или предоставляемого сервисом по запросу клиента. Описание параметров сервиса определяет форматы сообщений (контракт), которым должны следовать клиент и сервис. Данное описание должно поддерживать машинную интерпретацию, например для валидации сообщений, генерации клиентского кода или реализации динамических вызовов.

Отметим, что описанная модель является довольно общей и может быть, вообще говоря, применена к большому классу сервисов, не обязательно предоставляющих доступ к вычислительным алгоритмам. В частности, это могут быть сервисы доступа к базам данных или сервисы, осуществляющие обработку и преобразование данных. В общем случае, речь может идти о произвольном программном компоненте, оформленном в виде сервиса. Тем не менее, в рамках среды MathCloud предполагается рассмотрение в первую очередь сервисов, нацеленных на решение научных и прикладных вычислительных задач. Рассмотрим характерные особенности вычислительных сервисов.

Во-первых, каждый запрос обрабатывается сервисом независимо от других запросов. Иными словами, результат запроса определяется исключительно значениями передаваемых клиентом входных параметров и не зависит от результатов других запросов. Фактически данная особенность играет роль ограничения, накладываемого на рассматриваемые сервисы. А именно, за рамками описываемой модели сервиса остаются клиент-серверные приложения, в которых требуется поддерживать состояние (сессию) между последовательными запросами клиента к серверу. Исключение из рассмотрения подобных случаев позволяет существенно

упростить интерфейс и реализацию сервисов, а также, что самое главное, повысить масштабируемость и отказоустойчивость среды. Отметим, что данное ограничение часто накладывается на сервисы в рамках сервис-ориентированной архитектуры.

Вторая важная особенность вычислительных сервисов заключается в том, что обработка запроса клиента (решение задачи) может потребовать длительных вычислений, в том числе на многопроцессорных вычислительных комплексах или в грид. В этом случае сервис не может вернуть результат сразу же после получения запроса. Обработка подобных запросов должна вестись в асинхронном режиме, путем преобразования поступающих запросов в задания. В ответ на вызов клиенту передается идентификатор соответствующего задания, используя который клиент может опрашивать статус задания (т.н. режим pull) и получить результат. Клиент также может получать от сервиса уведомления об изменении статуса задания (режим push).

Кроме того, растет количество вычислительных приложений, принимающих на вход или генерирующих большие объемы данных. Для подобных сервисов необходимо организовать эффективную передачу параметров большого размера в виде файлов с использованием соответствующих механизмов передачи данных по сети. В этом случае запрос к сервису или возвращаемый сервисом результат могут содержать не сами значения параметров, а ссылки на соответствующие файлы.

Доступное клиентам описание вычислительного сервиса должно содержать информацию не только о параметрах сервиса, но и о классе решаемых сервисом задач, используемых при этом алгоритмах и численных методах, точности получаемых решений и т.п. Формальное описание данной информации в машинно-интерпретируемом виде является сложной задачей, имеющей специфические требования для конкретных прикладных областей. В общем случае данная информация может предоставляться в произвольной форме, доступной для просмотра пользователем.

Для упрощения повторного использования и композиции сервисов в рамках различных приложений требуется унификация механизма удаленного доступа к сервисам на уровне протоколов и форматов данных. Для этих целей предлагается использовать архитектурный стиль REST (Representational State Transfer), хорошо зарекомендовавший себя в рамках Web. Для среды MathCloud разработан и подробно описан унифицированный REST-интерфейс (см. главу 2), который должны реализовывать сервисы среды.

Среда MathCloud является открытой распределенной системой, для участия в которой разработчику сервиса необходимо и достаточно реализовать унифицированный REST-интерфейс. При этом детали реализации сервиса остаются полностью на усмотрение его разработчика. Тем не менее, необходимо наличие готовых средств, упрощающих разработку сервисов, в первую очередь для разработчиков с минимальной квалификацией. Подобные средства особенно актуальны для быстрого преобразования в сервисы существующих приложений. Исходя из этих соображений была реализована универсальная среда выполнения или контейнер сервисов Everest (см. главу 3).

1.3 Уровень приложений

На уровне приложений реализуется доступ пользователей к сервисам среды через проблемно-ориентированные интерфейсы. В качестве стандартной среды выполнения приложений предлагается использовать веб-браузер, что обладает рядом преимуществ:

приложение сразу готово к работе без установки на компьютер пользователя, разработчик приложения сохраняет полный контроль над ним, включая возможность обновлений и учета использования. Так же как в случае сервисов, архитектура не накладывает существенные ограничения на используемые для разработки приложения средства программирования.

Отметим, что реализация некоторого сервиса может быть уже снабжена интерфейсом для работы с сервисом пользователей. Однако в общем случае понятия сервиса и приложения следует различать. Если сервис предоставляет программный интерфейс для приложений, то приложение предоставляет интерфейс конечному пользователю. Таким образом, один и тот же сервис может использоваться в рамках нескольких приложений. И наоборот - в рамках одного приложения могут быть задействованы сразу несколько сервисов. Остановимся на последнем моменте подробнее.

Ключевой функцией предлагаемой сервис-ориентированной среды является поддержка объединения или композиции сервисов, позволяющая пользователям среды «собирать» из существующих сервисов новые приложения и, что особенно важно, новые сервисы, развивая, тем самым, среду. Именно для этого необходим унифицированный интерфейс доступа к сервисам. В общем случае, композиция сервисов может осуществляться с применением произвольных средств программирования, например скриптовых языков. Однако, также как и в случае с разработкой сервисов, в рамках MathCloud предусмотрены готовые средства, упрощающие композицию сервисов и доступные пользователям с минимальной квалификацией. Для поддержки интеграции сервисов среды при решении прикладных задач реализованы редактор композитных приложений и система управления сценариями на основе workflow-подхода (см. главу 4).

1.4 Уровень пользователей

На уровне пользователей сконцентрирован социальный аспект среды. Здесь решаются задачи, связанные с формированием и обеспечением функционирования сообществ пользователей. Предполагается, что данные сообщества, как правило, будут формироваться вокруг той или иной области исследований, аналогично виртуальным организациям в грид. Идеальной средой для формирования таких сообществ представляются Web-порталы с функциональностью социальных сетей, учитывающие научную специфику проекта. Будучи созданными, подобные сообщества могут образовывать контексты для разделения ресурсов и сервисов. Во-первых, в рамках сообщества могут коллективно вестись публикация, каталогизация и учет сервисов, относящихся к тематике данного сообщества. Во-вторых, аналогично грид, доступ к сервисам может регламентироваться на основе принадлежности пользователя к определенному сообществу. В-третьих, интерфейсы приложений могут быть интегрированы в портал сообщества, предоставляя удобный доступ к сервисам среды. Данный уровень архитектуры MathCloud является в данный момент наименее проработанным, поскольку текущие усилия сосредоточены на нижних, технологических уровнях среды.

2. Интерфейс сервиса MathCloud

Описанная архитектура носит достаточно общий характер и может быть реализована с помощью различных технологий построения распределенных систем. Выбор той или иной

технологии зависит от ряда факторов, таких как распространенность, удобство разработки и открытость исходного кода. В рамках научной сервис-ориентированной среды, допускающей участие различных лиц и организаций, очень важно использование открытых стандартов и наличие нескольких независимых реализаций базовой технологии. Важно также, чтобы используемая технология как можно ближе соответствовала описанной выше модели проблемно-ориентированного вычислительного сервиса.

В настоящее время доминирующей технологией для построения сервис-ориентированных систем являются Web-сервисы на основе протокола SOAP и спецификаций WS-*, которые будем далее называть WS-сервисами. Спецификации WS-сервисов являются открытыми стандартами, изначально ориентированными на реализацию сервис-ориентированной архитектуры. Наконец, что немаловажно, данные технологии активно поддерживаются крупными компаниями-разработчиками, что привело к повсеместному их распространению и появлению множества реализаций, в том числе и с открытым кодом.

Распространенной критикой WS-сервисов является их чрезмерная сложность и некорректное использование протокола HTTP, что ставит под сомнение приставку "Web-" в их названии. При этом преимущества от использования WS-сервисов заметны только в определенном классе приложений, ориентированных на поддержку сложных бизнес-процессов, встречающихся в корпоративных и государственных системах и слабо распространенных в технических и научных проектах. Это привело к появлению альтернативных, более простых в использовании подходов к реализации Web-сервисов, основанных на прямом использовании протокола HTTP.

В 2000 году Рой Филдинг, один из главных разработчиков протокола HTTP и других спецификаций Web, опубликовал диссертацию [3] с описанием эталонной модели архитектуры Web. Данная модель или архитектурный стиль, получивший название Representational State Transfer (REST), содержит ряд ключевых принципов, определенных в виде накладываемых на архитектуру ограничений. Центральными понятиями REST являются понятия ресурса, идентификатора и представления ресурса. В рамках REST вводятся следующие ограничения на архитектуру системы: клиент-серверная архитектура, хранение состояния приложения на стороне клиента, кэширование ответов на запросы, унифицированный интерфейс доступа к ресурсам, многоуровневая архитектура. Данные ограничения позволяют обеспечить такие свойства архитектуры Web, как масштабируемость, расширяемость и открытость. Стиль REST носит общий характер и может быть применен при реализации как систем на основе Web, так и других распределенных систем.

Благодаря унифицированному интерфейсу доступа к ресурсам, использованию открытых стандартов (HTTP, URI) и наличию множества проверенных временем реализаций (Web-серверы, библиотеки для работы с HTTP для всех современных языков программирования), REST обеспечивает максимальную свободу для независимой разработки Web-сервисов и соответствующих клиентских приложений. Немаловажным обстоятельством является простота разработки REST-сервисов в сравнении с другими рассмотренными технологиями. Все это позволяет максимально расширить как круг потенциальных разработчиков, так и пользователей сервисов, обеспечив при этом совместимость различных реализаций и широкое повторное использование сервисов. Это привело к широкому распространению Web-сервисов на основе стиля REST (т.н. RESTful Web-сервисов), особенно в рамках Web 2.0 приложений.

Исходя из вышеописанного, в качестве базовой платформы для организации удаленного доступа к сервисам MathCloud предлагается использовать протоколы и технологии Web, основанные на архитектурном стиле REST. Разработан и подробно описан REST-интерфейс сервиса MathCloud [4], использующий в качестве протокола доступа к сервисам протокол HTTP. Разработанный интерфейс учитывает характерные особенности вычислительных сервисов, такие как длительная обработка запросов и передача больших объемов данных в виде файлов. Также в соответствии с сервис-ориентированным подходом разработанный интерфейс поддерживает интроспекцию, т.е. получение информации о сервисе. Выбранные базовые технологии допускают независимые реализации описанного интерфейса на различных языках программирования. При этом детали реализации сервиса остаются полностью на усмотрение его разработчика.

2.1 Общая схема интерфейса

В соответствии с принципами REST, интерфейс сервиса MathCloud образован совокупностью идентифицируемых при помощи URI ресурсов, поддерживающих стандартные методы протокола HTTP (Табл. 1). Данными ресурсами являются:

- сервис, идентифицируемый с помощью SERVICE_URI;
- задание, идентифицируемое с помощью JOB_URI;
- файл результата задания, идентифицируемый с помощью FILE_URI;
- сервер, идентифицируемый с помощью SERVER_URI (является необязательным ресурсом).

Таблица 1. Ресурсы и методы интерфейса сервиса MathCloud.

Ресурс	**GET**	**POST**	**DELETE**
Сервис SERVICE_URI	Получение описания сервиса	Запрос к сервису	---
Задание JOB_URI	Получение статуса и результатов задания	---	Отмена задания, удаление результатов задания
Файл результата задания FILE_URI	Загрузка файла	---	---
Сервер SERVER_URI	Получение списка сервисов, размещенных на сервере	---	---

Ресурс-сервис поддерживает два метода. Метод GET возвращает клиенту представление данного ресурса, содержащее описание сервиса. Метод POST служит для отправки сервису нового запроса. В теле запроса клиент передает набор значений входных параметров задачи. В ответ сервис создает новый ресурс-задание, являющийся подчиненным

по отношению к ресурсу-сервису, и возвращает идентификатор ресурса-задания и его текущее представление клиенту.

Ресурс-задание поддерживает методы GET и DELETE. Метод GET возвращает представление данного ресурса, содержащее информацию о текущем статусе задания. Данная информация должна обязательно включать состояние задания, которое может принимать следующие значения:

- WAITING — выполнение задания еще не началось;
- RUNNING — задание выполняется;
- DONE — задание выполнено успешно;
- FAILED — выполнение задания завершилось ошибкой.

Если выполнение задания завершено успешно (состояние DONE), то в представление ресурса-задания также включается результат задания в виде набора значений выходных параметров. Часть значений выходных параметров может содержать идентификаторы ресурсов-файлов.

В случае, если в процессе выполнения задания стали известны значения части выходных параметров, данные значения могут включаться в представление ресурса-задания еще до окончания выполнения задания.

Метод DELETE ресурса-задания позволяет клиенту отменить выполнение запроса или, если выполнение уже завершено, удалить результаты задания. После вызова DELETE данный ресурс-задание, а также подчиненные ему ресурсы-файлы, перестают существовать.

Ресурс-файл представляет собой часть результата задания и поддерживает метод GET для получения содержимого файла клиентом.

Дополнительный ресурс-сервер предназначен для случаев, когда в рамках одного HTTP-сервера размещено несколько алгоритмических сервисов. В этих случаях может быть полезным получение списка сервисов, размещенных на сервере. Для этих целей зарезервирован метод GET. Ресурс-сервер в данном случае является родительским по отношению к ресурсам-сервисам.

Отметим, что в рамках описанной схемы интерфейса не предписываются конкретные шаблоны для URI ресурсов, которые могут варьироваться между реализациями. При построении URI рекомендуется соблюдать указанные иерархические отношения между ресурсами.

Описанный интерфейс поддерживает обработку запросов как в синхронном, так и асинхронном режиме. Действительно, если результат запроса может быть сразу возвращен клиенту, то он передается внутри возвращаемого в ответ клиенту представления ресурса-задания с указанием состояния DONE. Если же для обработки запроса требуется время, то это указывается внутри возвращаемого клиенту представления ресурса-задания с помощью указания соответствующего состояния задания (WAITING или RUNNING). В этом случае клиент использует переданный URI ресурса-задания для дальнейшего опроса состояния задания и получения результата.

Подобная схема, с одной стороны, обеспечивает максимальную свободу сервиса в выборе стратегии обработки каждого отдельного запроса. С другой стороны, при реализации клиента требуется учесть оба сценария развития событий. Почему одни запросы могут быть обработаны быстро, а другие нет? Это может быть связано как с зависимостью времени

выполнения алгоритма от значений входных параметров, так и просто с текущей загрузкой сервиса входящими запросами и их возможной приоритетизацией. В общем случае трудно априори предсказать режим обработки запроса и разделить запросы на синхронные и асинхронные. Исходя из этих соображений и была выбрана данная схема.

2.2 Форматы представления данных

За рамками описанной выше схемы интерфейса сервиса остался вопрос о том, какие форматы представления данных будут использовать во время взаимодействия между клиентом и сервисом. Наиболее распространенными форматами представления данных в Web являются HTML, XML и JSON. Первый формат используется, главным образом, для отображения данных пользователю через Web-браузер. Два других формата используются при реализации программных интерфейсов к Web-сервисам. Наибольшее распространение получил XML, как универсальный формат обмена данными между программными системами. В последние несколько лет набирает популярность формат JSON, который является компактной альтернативой XML и хорошо сочетается с языком JavaScript, на котором пишется большинство клиентских Web-приложений.

В качестве основного формата представления данных в интерфейсе сервиса MathCloud предлагается использовать JSON, исходя из следующих соображений:

- более компактное представление структур данных, в то время как XML ориентирован на представление произвольных документов;
- число библиотек для работы с JSON на различных языках программирования приближается к таковому для XML;
- удобство работы с JSON-данными на языке JavaScript.

Известным недостатком JSON является отсутствие стандартного средства описания схем, сопоставимого с языком XML Schema. Однако данный недостаток носит временный характер, так как в настоящее время ведутся активные работы над подобным языком JSON Schema [5].

Протокол HTTP поддерживает указание формата запроса с помощью заголовка «Content-Type», а также динамическое согласование формата ответа с помощью заголовка «Accept». Значениями указанных заголовков являются идентификаторы MIME-типов, регистрируемых организацией IANA. Данные возможности протокола HTTP позволяют поддерживать в рамках описанного интерфейса сразу несколько форматов представления данных, используемых в зависимости от типа клиента или других обстоятельств.

Например, для поддержки работы пользователей с сервисом через Web-браузер, в качестве дополнительного формата представления данных в интерфейсе сервиса может использоваться формат HTM (text/html). При передаче запроса к сервису через форму в Web-браузере может использоваться стандартный формат «multipart/form-data». Конкретные представления запросов и ответов в данном случае выходят за рамки спецификации программного интерфейса сервиса и определяются разработчиком сервиса.

В Табл. 2 указаны возможные форматы представления запроса и ответа для каждого ресурса и метода интерфейса сервиса MathCloud. В дальнейшем описанный интерфейс

может быть легко расширен путем добавления новых форматов.

Таблица 2. Возможные форматы представления данных для ресурсов и методов интерфейса сервиса MathCloud.

Ресурс	**GET**	**POST**	**DELETE**
Сервис SERVICE_URI	Accept: • aplication/json • text/html	Content-Type: • application/json • multipart/form-data Accept: • aplication/json • text/html	---
Запрос REQUEST_URI	Accept: • aplication/json • text/html	---	---
Файл результата запроса FILE_URI	Content-Type: • определяется типом файла	---	---
Сервер SERVER_URI	Accept: • aplication/json • text/html	---	---

В последующих разделах главы приводится детальное описание возможных запросов и ответов для интерфейса сервиса MathCloud. В качестве формата представления данных используется JSON.

2.3 Получение описания сервиса

Формат запроса

```
GET SERVICE_URI
Accept: application/json
```

Формат ответа

Если сервис с данным URI существует, то сервер возвращает описание сервиса в следующем виде:

```
200 OK
Content-Type: application/json

{
  "name":"SERVICE_NAME",
  "description":"SERVICE_DESCRIPTION",
  "inputs": {
    "IN_PARAM1_NAME":{"type":"...","title":"...",...},
```

```
      "IN_PARAM2_NAME":{"type":"...","title":"...",...},
      …
    },
    "outputs": {
      "OUT_PARAM1_NAME":{"type":"...","title":"...",...},
      "OUT_PARAM2_NAME":{"type":"...","title":"...",...},
      …
    }
}
```

Поясним назначение отдельных атрибутов описания сервиса:

- атрибут ***name*** содержит краткое имя сервиса;
- атрибут ***description*** содержит краткое текстовое описание сервиса и/или URI документа с подробным описанием сервиса;
- атрибут ***inputs*** содержит массив описаний входных параметров сервиса;
- атрибут ***outputs*** содержит массив описаний выходных параметров сервиса.

Описание параметра состоит из имени параметра и его схемы, оформленной в соответствии с JSON Schema. Наиболее важными атрибутами схемы являются:

- атрибут ***type***, указывающий тип значения параметра;
- атрибут ***title***, содержащий краткое текстовое описание параметра;
- атрибут ***description***, содержащий полное текстовое описание параметра;
- атрибут ***optional*** (применим только для входных параметров), указывающий на то, является ли данный параметр обязательным (значение false) или нет (значение true);
- атрибут ***default***, содержащий значение параметра по умолчанию.

Приведем пример описания тестового сервиса Echo:

```
{
  "name":"echo",
  "description":"Test service echoes back input params",
  "inputs":{
    "input1":{"type":"integer","title":"Integer input"},
    "input2":{"type":"string","title":"String input","optional":true},
    "input3":{"type":"file","title":"File input","optional":true}
  },
  "outputs":{
    "output1":{"type":"integer","title":"Integer output"},
    "output2":{"type":"string","title":"String output"},
    "output3":{"type":"file","title":"File output"}
  }
}
```

В приведенном выше описании используется специальный тип параметра file, отсутствующий в JSON Schema. Данный тип предназначен для передачи значения параметра в виде отдельного файла.

Коды ошибок

Предусмотрены следующие коды ошибок:

- 404 Not Found - сервис с данным URI не существует;
- 500 Internal Server Error - ошибка на серверной стороне.

2.4 Запрос к сервису

Формат запроса

```
POST SERVICE_URI
Content-Type: application/json

{
  "IN_PARAM1_NAME":IN_PARAM1_VALUE,
  "IN_PARAM2_NAME":IN_PARAM2_VALUE,
  …
}
```

В запросе должны быть указаны значения всех обязательных входных параметров сервиса.

Приведем пример запроса к тестовому сервису Echo:

```
POST http://somehost.com/echo
Content-Type: application/json

{
  "input1":1637673,
  "input2":"some string",
  "input3":{"uri":"http://otherhost.com/files/test.bin"}
}
```

В приведенном выше запросе для файлового параметра input3 указывается URI удаленного файла. В данном случае перед началом выполнения запроса сервис производит загрузку содержимого файла с помощью протокола HTTP. Данный подход позволяет организовать передачу данных большого объема вне тела запроса.

Предусмотрен также вариант передачи содержимого файла непосредственно внутри запроса в кодировке Base64:

```
POST http://somehost.com/echo
Content-Type: application/json

{
  ...
  "input3":{"data":"BASE64_ENCODED_FILE_DATA"}
}
```

В будущем описанная схема передачи файловых параметров может быть расширена путем добавления других протоколов доступа к удаленным файлам, таких как FTP/GridFTP.

Формат ответа

В случае если запрос клиента принят к выполнению, то сервис возвращает следующий ответ:

```
202 Accepted
Location: JOB_URI
Content-Type: application/json
```

```
Текущий статус запроса в формате JSON (см. раздел 2.5)
```

Статус «202 Accepted» означает, что запрос принят к выполнению и его статус можно отслеживать с помощью нового ресурса-задания, URI которого содержится в заголовке Location. В теле ответа передается текущее представление ресурса-задания. Это позволяет клиенту сразу получить результат в случае, если запрос может быть выполнен сервисом в течение короткого времени.

Приведем пример ответа на запрос к тестовому сервису Echo:

```
202 Accepted
Location: http://somehost.com/echo/job2706049717
Content-Type: application/json

{
  "state":"WAITING"
}
```

В данном случае задание еще не выполнено и находится в состоянии WAITING.

Коды ошибок

Предусмотрены следующие коды ошибок:

- 404 Not Found - сервис с данным URI не существует;
- 400 Bad Request – некорректный запрос (в запросе не указаны значения всех обязательных входных параметров, значение параметра не соответствует его схеме и т.п.);
- 500 Internal Server Error - ошибка на серверной стороне.

2.5 Получение статуса и результатов задания

Формат запроса

```
GET JOB_URI
Accept: application/json
```

Формат ответа

В случае если задание с данным URI существует и находится в состояниях WAITING или RUNNING, то сервис возвращает ответ следующего вида:

```
200 OK
Content-Type: application/json

{
  "state":"JOB_STATE",
  "info":"JOB_STATUS_INFO",
}
```

В случае если известны значения части выходных параметров или задание находится в состоянии DONE, то добавляется атрибут result:

```
200 OK
Content-Type: application/json

{
  "state":"DONE",
  "info":"JOB_STATUS_INFO",
  "result": {
    "OUT_PARAM1_NAME":OUT_PARAM1_VALUE,
    "OUT_PARAM2_NAME":OUT_PARAM2_VALUE,
    …
  }
}
```

Аналогично, если задание находится в состоянии FAILED, то добавляется атрибут error:

```
200 OK
Content-Type: application/json

{
  "state":"FAILED",
  "info":"JOB_STATUS_INFO",
  "error":"ERROR_INFO"
}
```

Поясним значение каждого из используемых атрибутов:

- обязательный атрибут state содержит текущее состояние задания, которое может принимать одно из следующих значений: WAITING, RUNNING, DONE, FAILED;
- необязательный атрибут info предназначен для передачи дополнительной информации о статусе выполнения задания, отображаемой пользователю;
- атрибут result содержит, возможно частичный, результат выполнения задания в виде набора значений выходных параметров;
- атрибут error содержит информацию об ошибке, приведшей к сбою при выполнении задания.

Приведем пример ответа для тестового сервиса Echo в случае, когда выполнение задания завершено успешно:

```
200 OK
Content-Type: application/json

{
  "state":"DONE",
  "result": {
    "output1":1637673,
    "output2":"some string",
    "output3":{"uri":"http://somehost.com/echo/job2706049717/out.bin"}
  }
}
```

На приведенном выше примере видно, что файловые выходные параметры,

аналогично входным параметрам, передаются при помощи указания URI файла. В данном случае клиенту передается адрес файла, соответствующего выходному параметру output3. Данный подход позволяет организовать передачу результатов большого объема. Описанная схема передачи файловых параметров может быть расширена путем добавления других протоколов доступа к удаленным файлам.

Предусмотрен также вариант передачи содержимого выходного файла в кодировке Base64 непосредственно внутри тела ответа. Для этого к REQUEST_URI необходимо добавить query-параметр data:

```
GET REQUEST_URI?data
Accept: application/json
```

Ответ сервиса в данном случае выглядит следующим образом:

```
200 OK
Content-Type: application/json

{
  "state":"DONE",
  "result": {
    ...
    "output3":{"data":"BASE64_ENCODED_FILE_DATA"}
  }
}
```

Коды ошибок

Предусмотрены следующие коды ошибок:

- 404 Not Found - запрос с данным URI не существует;
- 500 Internal Server Error - ошибка на серверной стороне.

2.6 Отмена выполнения задания и удаление результатов

Клиент может отменить выполнение задания с помощью метода DELETE ресурса-задания. В случае если задание уже выполнено успешно, происходит удаление всех данных, связанных с результатом задания. Таким образом, клиент дает знать сервису, что больше не нуждается в этих данных.

Формат запроса

```
DELETE JOB_URI
```

Формат ответа

В случае если задание с данным URI существует и успешно удален, то сервис возвращает ответ

```
200 OK
```

Коды ошибок

Предусмотрены следующие коды ошибок:

- 404 Not Found - запрос с данным URI не существует;

- 500 Internal Server Error - ошибка на серверной стороне.

2.7 Получение файла результата задания

После успешного выполнения задания каждый выходной параметр типа file размещается сервисом под некоторым FILE_URI, который передается клиенту в результатах задания. Клиент может загрузить файл с помощью метода GET ресурса-файла.

Формат запроса

```
GET FILE_URI
```

Формат ответа

В случае если файл с данным URI существует, то сервис возвращает содержимое файла в теле ответа:

```
200 OK
Content-Type: FILE_MIME_TYPE

FILE_DATA
```

Значение заголовка Content-Type соответствует типу файла и определяется, например, исходя из расширения файла.

Коды ошибок

Предусмотрены следующие коды ошибок:

- 404 Not Found - файл с данным URI не существует;
- 500 Internal Server Error - ошибка на серверной стороне.

2.8 Получение списка сервисов, размещенных на сервере

Формат запроса

```
GET SERVER_URI
```

Формат ответа

Сервер возвращает список размещенных сервисов с указанием URI каждого сервиса:

```
200 OK
Content-Type: application/json

{
  "SERVICE1_NAME":"SERVICE1_URI",
  "SERVICE2_NAME":"SERVICE2_URI",
  ...
}
```

Коды ошибок

Предусмотрены следующие коды ошибок:

- 404 Not Found - сервер с данным URI не существует;

- 500 Internal Server Error - ошибка на серверной стороне.

3. Среда выполнения сервисов

Наличие готового и простого в использования инструмента для создания сервисов очень важно с точки зрения расширения круга потенциальных разработчиков сервисов. Для упрощения процесса разработки сервисов MathCloud была реализован контейнер сервисов Everest. Данный контейнер представляет собой универсальную среду выполнения сервисов, реализующую описанный выше интерфейс сервиса.

Универсальность контейнера заключается в том, что он позволяет легко, во многих случаях без написания программного кода, преобразовать в сервис MathCloud широкий спектр существующих приложений. Кроме того, реализованный в контейнере механизм плагинов позволяет подключать произвольные реализации обработчиков запросов. В частности, путем преобразования запроса к сервису в вычислительное задание, на уровне плагина может быть реализован доступ к вычислительной инфраструктуре.

Everest реализован на базе библиотеки Jersey, являющейся в свою очередь открытой эталонной реализацией спецификации JAX-RS (Java API for RESTful Web Services). На Рис. 2 изображена архитектура Everest. Взаимодействие с клиентами осуществляется с помощью встроенного в контейнер Web-сервера Jetty. Входящие HTTP-запросы передаются библиотеке Jersey и, затем, реализации Everest. Сопряжение между Jersey и Everest осуществляется с помощью четырех Java-классов (ServerResource, ServiceResource, JobResource и FileResource), которые являются реализациями соответствующих ресурсов из REST-интерфейса сервиса MathCloud.

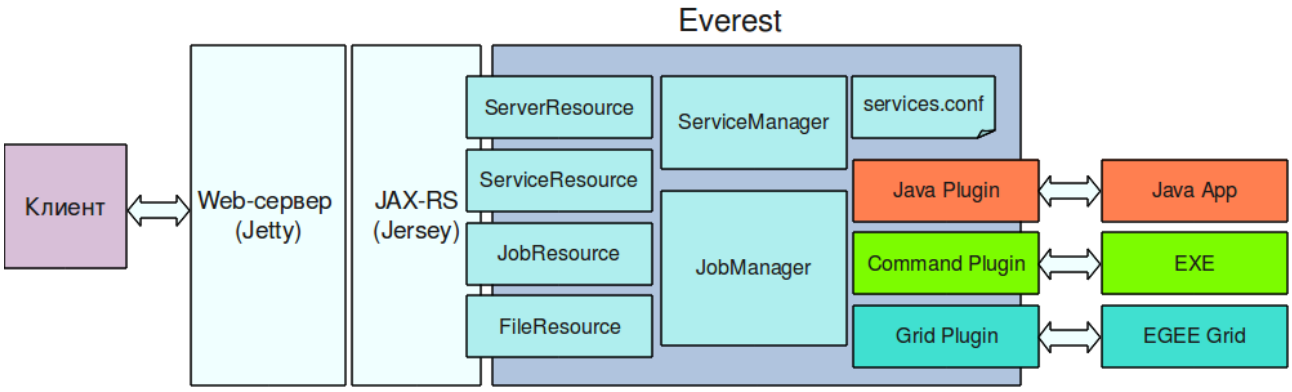

Рис. 2. Архитектура Everest.

Контейнер осуществляет обработку запросов клиентов в соответствии с конфигурационной информацией. Компонент ServiceManager хранит список сервисов, размещенных на сервере, и их конфигурацию. Данная информация считывается при запуске сервера из конфигурационных файлов. Конфигурация каждого сервиса состоит из двух частей:

- внешнее описание сервиса, передаваемое клиентам (текстовая аннотация сервиса, описания входных и выходных параметров);
- внутренняя конфигурация сервиса, содержащая информацию о реализации сервиса

(плагин и его конфигурационные параметры).

Компонент JobManager осуществляет управление обработкой запросов к сервисам. Запросы преобразуются в задания и размещаются в очереди, откуда затем извлекаются конфигурируемым пулом потоков-обработчиков. При выполнении задания поток-обработчик вызывает тот или иной плагин, в соответствии с внутренней конфигурацией сервиса.

Компоненты, реализующие обработку запросов к сервисам (заданий), оформляются в виде подключаемых плагинов. Каждый плагин реализует стандартный программный интерфейс, с помощью которого контейнер передает плагину параметры запроса, контролирует состояние выполнения задания и получает результаты. Реализация плагина, как правило, преобразует полученный запрос в вызов внешнего приложения. После успешного завершения приложения плагин сохраняет полученные результаты и изменяет состояние задания на DONE.

В настоящее время реализовано три универсальных плагина, поддерживающих интеграцию следующих типов приложений:

- приложение с интерфейсом командной строки (плагин Command);
- приложение на языке Java (плагин Java);
- грид-задание, запускаемое в инфраструктуре EGEE (плагин gLite).

Плагин Command осуществляет преобразование запроса к сервису в запуск команды в отдельном процессе операционной системы. Во внутренней конфигурации сервиса необходимо указать запускаемую команду в специальном формате, содержащем информацию о том, каким образом параметры сервиса отображаются в аргументы команды и внешние файлы. Ниже приведен пример конфигурации сервиса, использующего плагин Command.

```
{
    name: povray,
    description: "POV-Ray raytracer.",
    inputs: {
        scene:  {type: file,   title: "Scene file"},
        ini:    {type: file,   title: "Ini file", optional: true},
        params: {type: string, title: "Rendering parameters",
                optional:true, description: "Try +W320 +H240"}
    },
    outputs: {
        image:  {type: file, title: "Rendered image"},
        log:    {type: file, title: "Runtime log",
                description: "This is the raw output of POV-Ray"}
    },

    implementation: {
        type: command,
        command: "povray $ini +I$scene +Oimage.png +FN $params",
        input2file: {
            scene: scene.pov, ini: scene.ini
        },
        file2output : {
            image.png: image, stderr: log
        }
    }
```

```
}
```

Плагин Java осуществляет вызов заданного Java-класса в рамках текущей виртуальной машины, передавая в вызове параметры запроса к сервису. Используемый Java-класс должен реализовывать стандартный программный интерфейс. Во внутренней конфигурации сервиса требуется указать имя соответствующего Java-класса.

Плагин gLite осуществляет преобразование запроса к сервису в запуск вычислительного задания в грид-инфраструктуре EGEE, основанной на ПО gLite. Данный плагин может использоваться как для преобразования в сервис существующего грид-приложения, так и для переноса существующей реализации сервиса в грид, например из соображений производительности и масштабируемости. Во внутренней конфигурации сервиса требуется указать рабочую виртуальную организацию, путь к файлу с описанием грид-задания в формате JD, а также информацию об отображении параметров сервиса в аргументы и файлы грид-задания.

Отметим, что плагины Command и gLite позволяют преобразовывать существующие приложения в сервисы без разработки дополнительных программных компонентов. Данная возможность позволяет быстро преобразовать в сервисы широкий спектр существующих приложений, не обладая при этом навыками программирования.

Каждый размещенный в контейнере сервис становится доступным через стандартный интерфейс сервиса MathCloud. Кроме того, Everest поддерживает дополнительную HTML-версию данного интерфейса, позволяющую пользователям непосредственно обращаться к сервисам через веб-браузер. На Рис. 3 изображен автоматически сгенерированный сервером Web-интерфейс для отправки запроса к тестовому сервису Echo.

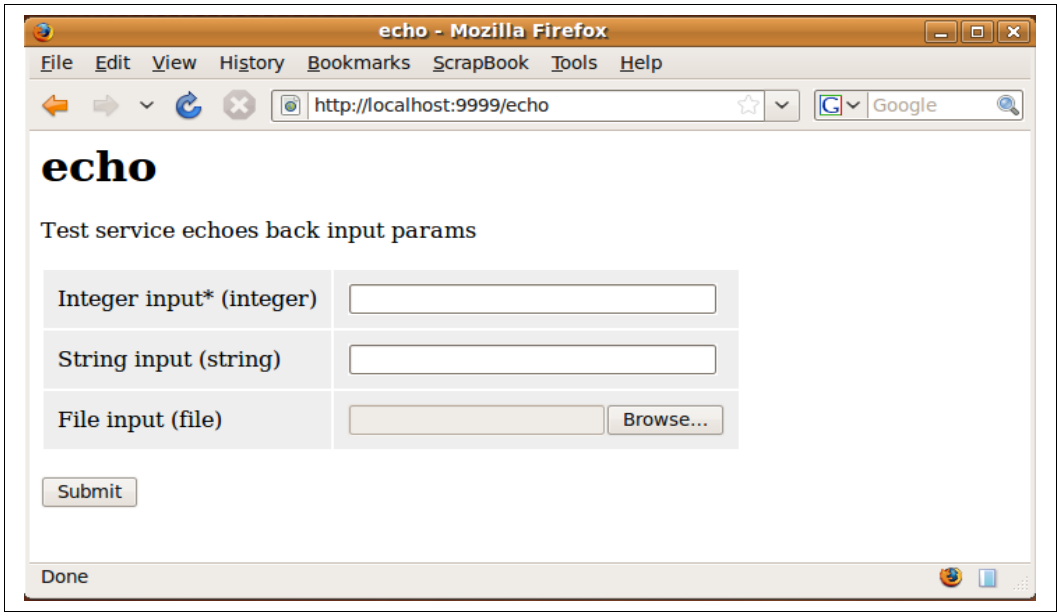

Рис. 3. Web-интерфейс сервиса, сгенерированный контейнером.

4. Композиция сервисов и система управления сценариями

Для поддержки интеграции сервисов среды MathCloud при решении прикладных реализована система управления сценариями на основе workflow-подхода [6]. Данная

система реализует описание, хранение, выполнение и публикацию сценариев совместного использования сервисов среды. Сценарии описываются в виде ориентированных ациклических графов при помощи визуального редактора. Описанный сценарий может быть затем преобразован в новый сервис среды и запущен на выполнение путем вызова данного сервиса. Предлагаемый подход позволяет скрыть от пользователя детали реализации вызовов сервисов и передачи данных между ними, оставив только необходимость правильного соединения сервисов друг с другом. Таким образом, многие задачи, решение которых сводится к компоновке типовых сервисов, становятся доступными для пользователей, не обладающих навыками распределенного программирования. Графическое представление сценария позволяет сделать наглядными связи между сервисами. Кроме того, такое представление позволяет быстро вносить изменения в уже работающие сценарии - такие, как добавление новых блоков сценария или замена отдельных блоков другими, получая на выходе новые сервисы с новыми параметрами.

4.1 Архитектура системы управления сценариями

Рассмотрим архитектуру и основные принципы реализации системы управления сценариями для среды MathCloud. Разработанная система имеет распределенную клиент-серверную архитектуру (Рис. 4). Клиентская часть системы включает редактор сценариев, а серверная — сервис управления сценариями и среду выполнения сценариев.

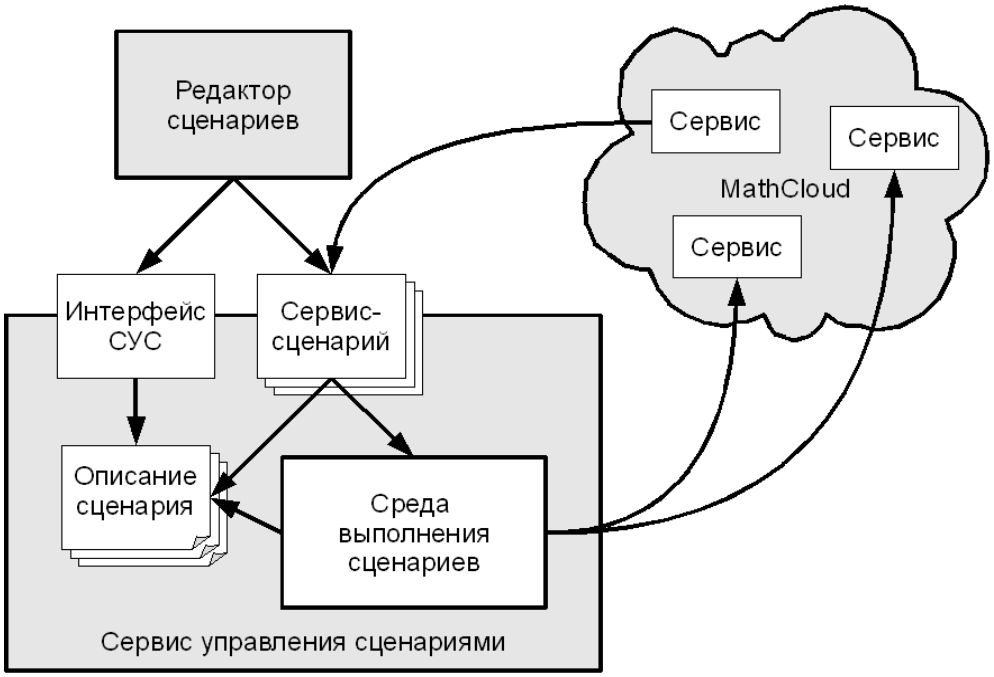

Рис. 4. Архитектура системы управления сценариями MathCloud.

Редактор сценариев реализует графический интерфейс, предназначенный для взаимодействия с системой пользователей. Редактор является клиентом сервиса управления сценариями и может функционировать на любой машине в сети. Основными функциями редактора являются просмотр и редактирование сценариев. Созданный с помощью редактора сценарий может быть сохранен и запущен на выполнение на серверной стороне. При этом в

редакторе отображается состояние выполнения сценария.

Сервис управления сценариями реализует удаленный программный интерфейс для хранения сценариев, созданных с помощью редактора. В соответствии с сервис-ориентированным подходом, сервис управления сценариями также обеспечивает развертывание каждого сохраненного пользователем сценария в виде нового сервиса среды MathCloud. Последующий запуск сценария осуществляется путем вызова соответствующего сервиса через унифицированный интерфейс (см. главу 2).

Среда выполнения сценариев осуществляет обработку запросов к сервисам-сценариям. Каждый подобный запрос приводит к запуску нового экземпляра соответствующего сценария. Среде выполнения сценариев передается описание сценария и значения его входных параметров. Среда осуществляет интерпретацию описания сценария, производит указанные в сценарии действия (в том числе вызовы внешних сервисов), отслеживает состояние выполнения сценария и возвращает результаты его выполнения.

4.2 Редактор сценариев

Редактор сценариев реализован в виде веб-приложения на языке JavaScript. Благодаря этому редактор может использоваться без предварительной установки на любом компьютере, где установлен современный веб-браузер. Редактор создан с широким использованием технологий AJAX. В его основе лежат такие JavaScript-библиотеки, как WireIt, YUI и InputEx. Это позволило сделать интерфейс редактора легковесным, интуитивным и удобным для пользователя.

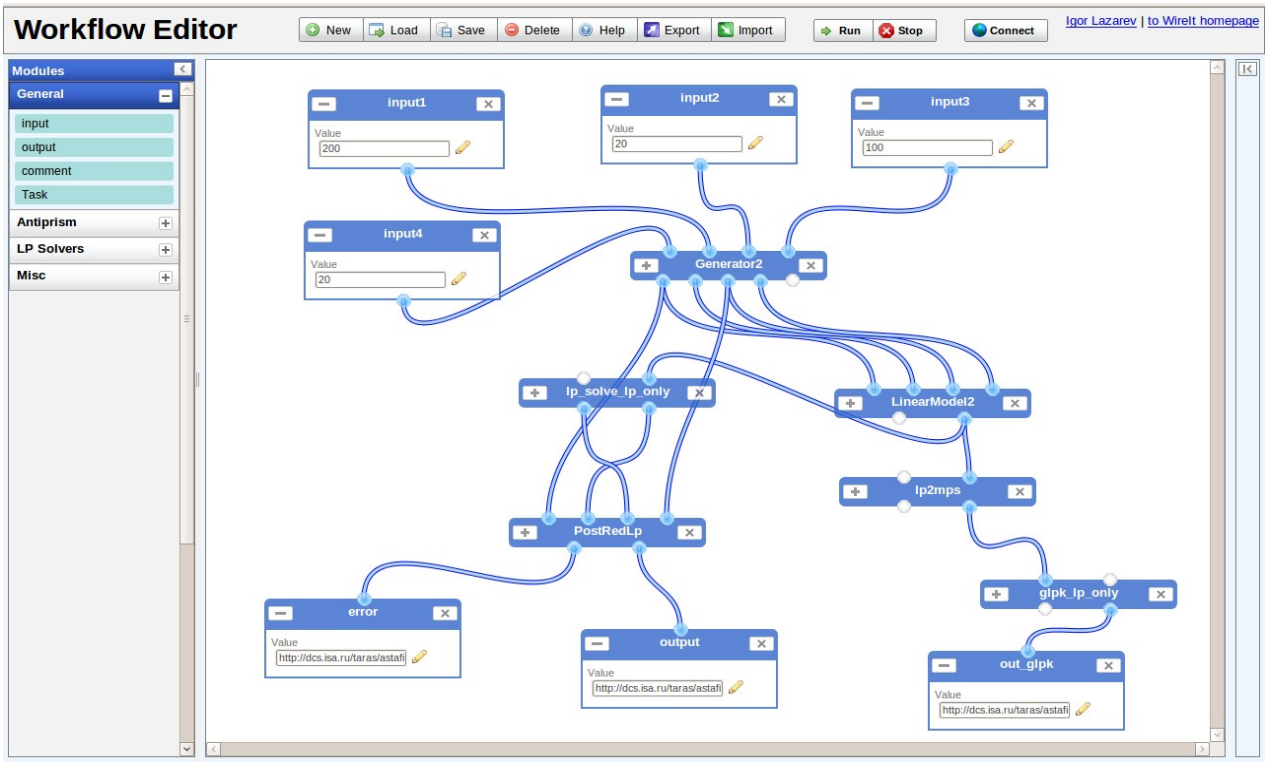

Рис. 5. Общий вид редактора сценариев.

На Рис. 5 изображен общий вид редактора сценариев. Правая часть редактора

содержит список доступных сервисов и прочих элементарных блоков, из которых пользователь может компоновать сценарий. Верхняя часть представляет собой главное меню, предоставляющее доступ к основным операциям со сценариями — открытие, сохранение, запуск и т.д. В центральной части редактора размещается графическое представление сценария.

Сценарий представлен в виде ориентированного ациклического графа, вершинами которого являются элементарные блоки сценария, а ребрами — соединения, задающие потоки данных между блоками. Каждый блок имеет набор входов и выходов, отображаемых в виде круглых портов соответственно сверху и снизу блока. Блок реализует некоторую логику обработки поступающих на вход данных. Передача данных между блоками реализуется путем соединения выхода одного блока с входом другого. С каждым выходом и выходом связан определенный тип данных. При соединении блоков проверяется совместимость типов входа и выхода.

В настоящее время реализованы следующие типы блоков сценария:

- Input — служит для задания значения входного параметра сценария;
- Output — служит для отображения значения выходного параметра сценария;
- Service — осуществляет вызов заданного сервиса MathCloud;
- JavaScript — осуществляет выполнение заданного кода на языка JavaScript;
- Python — осуществляет выполнение заданного кода на языке Python;
- Comment — служит для размещения комментариев к сценарию.

Добавление в сценарий сервиса MathCloud осуществляется путем создания нового блока Service и ввода пользователем URI требуемого сервиса. Редактор запрашивает описание данного сервиса, откуда извлекает информацию о количестве, типах и именах входных и выходных параметров сервиса. После чего динамически формируются соответствующие входы и выходы блока, а в заголовке блока отображается имя сервиса. В случае неудачи при соединении с сервисом блок окрашивается в красный цвет.

Для упрощения создания блоков Input и Output, редактор позволяет автоматически создавать данные блоки путем указания пользователем требуемого входа или выхода блока.

Блоки JavaScript и Python позволяют увеличить гибкость при описании сценария путем внедрения в него фрагментов кода (скриптов). Скрипт оформляется в виде функции, сигнатура которой используется для динамической генерации входов и заголовка блока. Выход блока соответствует возвращаемому значению функции. Выполнение скрипта осуществляется на серверной стороне средой выполнения сценариев.

Редактор поддерживает передачу и сохранение созданного сценария на стороне сервиса управления сценариями. Также с помощью редактора можно просматривать список сохраненных сценариев, открывать сохраненный сценарий, удалять сценарий, осуществлять импорт и экспорт описания сценария в формате JSON.

Важной функцией редактора является запуск сценария на выполнение. Для этого необходимо задать значения всех входных параметров сценария с помощью соответствующих блоков Input. После нажатия пользователем кнопки Run, редактор производит вызов сервиса-сценария с заданными входными параметрами. Далее редактор осуществляет периодический опрос состояния запущенного задания, содержащего

информацию о состоянии отдельных блоков сценария. Данная информация отображается пользователю путем окрашивания блоков сценария в цвет, соответствующий текущему состоянию (Рис. 6). После успешного завершения сценария, значения выходных параметров сценария отображаются в соответствующих блоках Output. Каждый запущенный сценарий имеет уникальный URI, используя который можно в любой момент открыть в редакторе выполняющийся сценарий. Данная возможность особенно важна в тех случаях, когда сценарий выполняется в течение длительного времени.

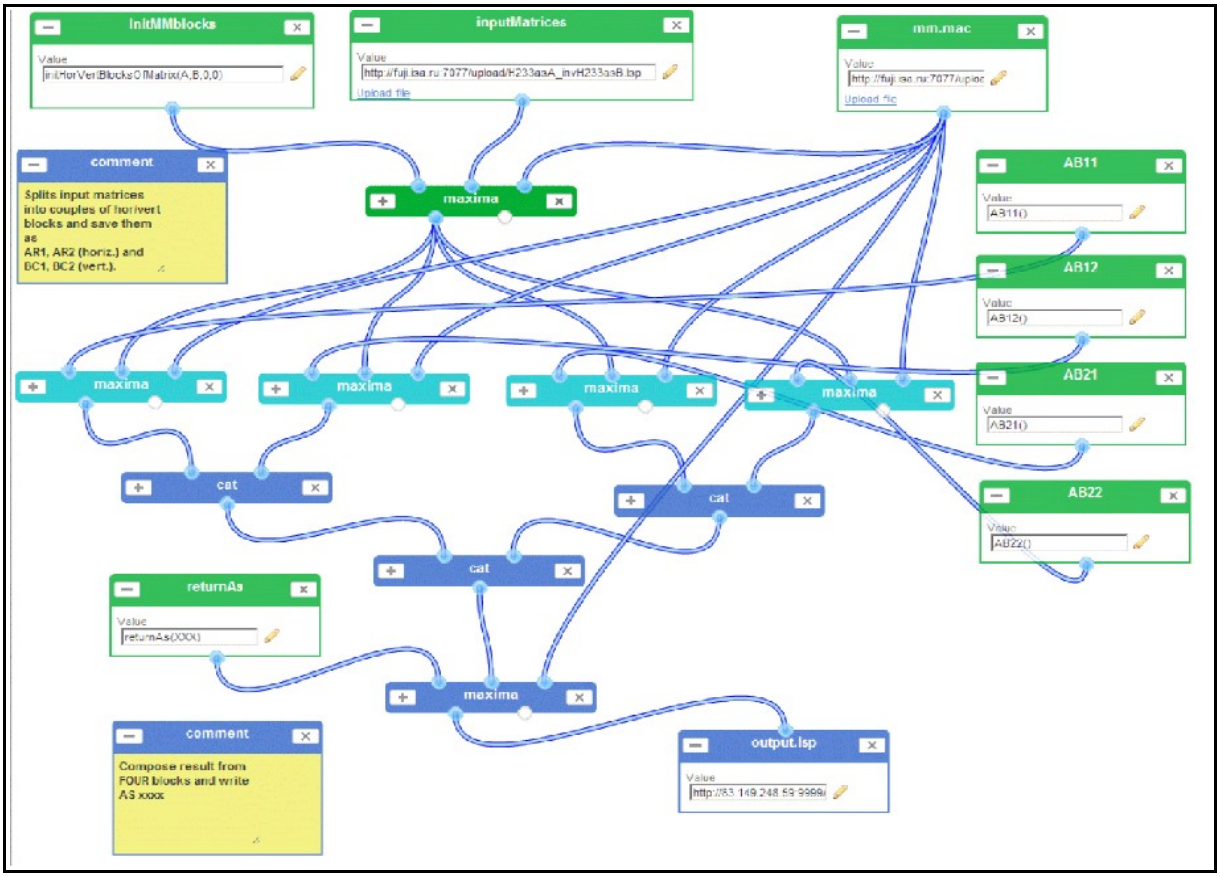

Рис. 6. Отображение состояния выполняющегося сценария
(выполненные блоки сценария окрашены зеленым цветом, выполняющиеся — голубым, ожидающие выполнения — синим).

4.3 Сервис управления сценариями

Сервис управления сценариями реализует удаленный программный интерфейс для хранения сценариев, созданных с помощью редактора, а также осуществляет развертывание сохраненных сценариев в виде новых сервисов MathCloud.

Удаленный интерфейс сервиса реализован на основе протокола HTTP и архитектурного стиля REST, что позволяет удобным образом взаимодействовать с сервисом из редактора сценария. В соответствии с принципами REST интерфейс сервиса образован совокупностью идентифицируемых при помощи URI ресурсов, поддерживающих стандартные методы протокола HTTP (Табл. 3). Данными ресурсами являются:

- сервис управления сценариями, идентифицируемый с помощью

WFMS_SERVICE_URI;

- сценарий, идентифицируемый с помощью WF_URI;
- сервис сценария, идентифицируемый с помощью WF_SERVICE_URI;
- вспомогательный прокси, идентифицируемый с помощью WFMS_PROXY_URI.

Таблица 3. Ресурсы и методы интерфейса сервиса управления сценариями.

Ресурс	**GET**	**POST**	**PUT**	**DELETE**
Сервис управления сценариями WFMS_SERVICE_URI	Получение списка хранимых сценариев	Создание нового сценария	__	__
Сценарий WF_URI	Получение описания сценария	__	Обновление сценария	Удаление сценария
Сервис сценария WF_SERVICE_URI	Получение описания сервиса	Запуск сценария	__	__
Прокси WFMS_PROXY_URI	Получение описания сервиса с указанным URI	__	__	__

Ресурс-сервис управления сценариями поддерживает следующие методы HTTP:

- метод GET возвращает список всех хранимых сервисом сценариев в формате JSON; для каждого сценария указывается его URI, имя и краткое описание;
- метод POST предназначен для сохранения нового сценария. В теле запроса клиент передает описание сценария в формате JSON. Сервис сохраняет полученное описание, создает ресурс-сценарий, развертывает сервис сценария, после чего возвращает клиенту идентификаторы созданных ресурса-сценария и сервиса сценария.

Ресурс-сценарий поддерживает следующие методы HTTP:

- метод GET возвращает представление сценария в формате JSON, содержащее описание сценария и идентификатор сервиса сценария;
- метод PUT предназначен для обновления описания сценария. В теле запроса клиент передает новое описание сценария. Сервис обновляет описание сценария и производит повторное развертывание сервиса сценария;
- метод DELETE предназначен для удаления сценария из системы. В ответ на вызов данного метода сервис удаляет описание сервиса и свертывает сервис сценария. После вызова DELETE данный ресурс-сценарий, а также сервис сценария перестают существовать.

Сервис сценария реализует унифицированный REST-интерфейс сервиса MathCloud (см. главу 2). Таким образом, с точки зрения клиентов сервис сценария ничем не отличается от обычных сервисов и запуск сценария на выполнение осуществляется путем отправки запроса к его сервису. Это позволяет, например, легко использовать существующий сценарий

в качестве одного из элементов при создании нового сценария. Единственное дополнение в унифицированный REST-интерфейс для сервиса-сценария состоит в том, что в представление ресурса-запроса включается дополнительная информация о состоянии отдельных блоков сценария. Данная информация используется редактором при визуализации состояния выполняющегося сценария.

Вспомогательный ресурс-прокси используется редактором сценариев для того, чтобы получить описание добавляемого в сценарий сервиса. Дело в том, что из-за политики безопасности браузера, Web-редактор не может напрямую обратиться по протоколу HTTP к сервису, размещенному не на одном Web-сервере с редактором. Поэтому редактор отправляет запрос GET ресурсу-прокси, выполняющемуся на одном Web-сервере с редактором, указывая в query-параметре URI интересующего сервиса. Ресурс-прокси загружает описание данного сервиса и возвращает его редактору.

4.4 Среда выполнения сценариев

Среда выполнения сценариев осуществляет обработку запросов к сервисам-сценариям путем интерпретации описания сценария с заданными значениями входных параметров.

Основой среды выполнения является разработанная библиотека WorkflowRuntime на языке Java, поддерживающая выполнение произвольных сценариев. Сценарий задается с помощью интерфейса прикладного программирования (API) библиотеки в виде ориентированного ациклического графа, вершинами которого являются задания. Каждое задание имеет набор входных и выходных параметров определенного типа. Ребра в графе связывают между собой выходные и входные параметры заданий, тем самым устанавливая зависимости по данным между заданиями. При соединении между собой заданий типы соответствующих параметров должны совпадать.

Задания реализуются в виде Java-классов, расширяющих библиотечный класс Task. Внутри метода run() класса задания программисту требуется реализовать логику обработки значений входных параметров и вычисления значений выходных параметров. Сценарий (библиотечный класс Workflow) также расширяет класс Task, что позволяет использовать готовый сценарий в виде задания в рамках другого сценария. Класс сценария также содержит метод для добавления в сценарий заданий.

Библиотека WorkflowRuntime осуществляет выполнение сценария путем последовательного запуска отдельных заданий сценария и передачи значений параметров между заданиями в соответствии с установленными зависимостями. Задание может быть запущено как только будут готовы значения всех его входных параметров. Выполнение заданий осуществляется с помощью конфигурируемого пула потоков, что позволяет одновременно осуществлять запуск нескольких заданий. Данная возможность может значительно сократить полное время выполнения сценария, в случае наличия в нем нескольких независимых заданий, требующих длительного времени для своего выполнения. Примерами подобных заданий могут служить запросы к сервисам MathCloud. Во время выполнения сценария WorkflowRuntime позволяет получить текущие состояния отдельных заданий сценария.

Описанная выше модель сценария в WorkflowRuntime является достаточно

универсальной и, в то же время, хорошо согласуется с моделью сценария в среде MathCloud, что позволяет использовать WorkflowRuntime для выполнения сценариев MathCloud. Для этого было реализовано преобразование описания сценария MathCloud в формате JSON в объектную модель сценария WorkflowRuntime. Кроме того, был реализован специальный тип задания, соответствующий вызову сервиса MathCloud.

Заключение

Одним из важных аспектов архитектуры среды MathCloud, оставшимся за рамками данной статьи, является вопрос обеспечения безопасности. Предполагается, что многие из сервисов MathCloud будут открыты для публичного доступа. Тем не менее, очевидно, что в ряде случаев ограничения круга пользователей сервиса будут необходимы, и что жизнеспособность предлагаемой среды будет зависеть от наличия полноценного механизма безопасности. Сюда входят аутентификация пользователей и сервисов, авторизация пользователей и контроль доступа к сервисам, защита передаваемых по сети данных и делегирования прав доступа. В настоящее время ведется разработка механизма безопасности среды MathCloud, основанного на применении протокола HTTPS и цифровых сертификатов. Также предусмотрен упрощенный способ аутентификации пользователей при помощи публичных провайдеров учетных записей и стандарта OpenID.

Другим важным компонентом среды MathCloud, работа над которым еще не завершена, является каталог сервисов, поддерживающий публикацию, поиск и мониторинг сервисов среды.

Данные вопросы будут подробно рассмотрены в рамках дальнейших работ.

ЛИТЕРАТУРА

[1] I. Foster. Service-Oriented Science // Science. – 2005. – V. 308, N. 5723. - P. 814-817.

[2] Астафьев А.С., Афанасьев А.П., Лазарев И.В., Сухорослов О.В., Тарасов А.С. Научная сервис-ориентированная среда на основе технологий Web и распределенных вычислений. // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). – М.: Изд-во МГУ, 2009. – 524 с. (с. 463-467)

[3] Fielding, R.T. Architectural styles and the design of network-based software architectures. PhD Dissertation. Dept. of Information and Computer Science, University of California, Irvine, 2000.

[4] Сухорослов О.В. Унифицированный интерфейс доступа к алгоритмическим сервисам в Web. // Проблемы вычислений в распределенной среде / Под ред. С.В. Емельянова, А.П. Афанасьева. Труды ИСА РАН, Т. 46. - М.: КРАСАНД, 2009. - 304 с. (с. 60-82)

[5] JSON Schema. http://www.json-schema.org/

[6] Лазарев И.В., Сухорослов О.В. Реализация распределенных вычислительных сценариев в среде MathCloud. // Проблемы вычислений в распределенной среде / Под ред. С.В. Емельянова, А.П. Афанасьева. Труды ИСА РАН, Т. 46. - М.: КРАСАНД, 2009. - 304 с. (с. 6-23)