I. РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Реализация распределенных вычислительных сценариев в среде MathCloud *

И. В. Лазарев[1], О. В. Сухорослов[2]

[1] *Московский физико-технический институт*
[2] *Центр Грид-технологий и распределенных вычислений Института системного анализа РАН*

В работе рассматривается реализация системы управления сценариями распределенной среды MathCloud, позволяющая описывать сценарий совместного использования нескольких сервисов MathCloud при помощи визуального редактора. Описывается архитектура и основные компоненты системы: визуальный веб-редактор сценариев, сервис управления сценариями и среда выполнения сценариев.

Введение

Распределенная среда MathCloud [1] ориентирована на поддержку математических исследований и базируется на веб- и грид-технологиях. Целью среды MathCloud является предоставление унифицированного доступа к сетевым сервисам решения различных классов математических задач. Во главу предлагаемого подхода к реализации данной среды ставится удобство разработки сервисов, преобразования существующих ресурсов в сервисы, интеграции сервисов со сторонними приложениями и, наконец, доступа конечных пользователей к сервисам и приложениям. Для этого ак-

* Работа выполнена при поддержке РФФИ (№ 08–07–00430–а, 07–07–00071–а), Президиума РАН (программа фундаментальных исследований П1) и Совета по грантам Президента Российской Федерации (№ НШ–5511.2008.9).

тивно используются современные веб-технологии. 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.

Поддержка объединения или композиции сервисов является ключевой функцией среды MathCloud, позволяющей пользователям среды «собирать» из существующих сервисов новые приложения и, что особенно важно, новые сервисы, развивая, тем самым, среду. Композиция сервисов может осуществляться с применением произвольных средств программирования, например скриптовых языков. Однако в рамках MathCloud предусмотрены готовые средства, упрощающие композицию сервисов и доступные пользователям с минимальной квалификацией.

В данной статье рассматривается система управления сценариями среды MathCloud, позволяющая описывать сценарий совместного использования нескольких сервисов MathCloud при помощи визуального редактора. Описанный сценарий может быть затем преобразован в новый сервис MathCloud и запущен на выполнение путем вызова данного сервиса. Предлагаемый подход позволяет скрыть от пользователя детали реализации вызовов сервисов и передачи данных между ними, оставив пользователю только необходимость правильного соединения сервисов друг с другом. Таким образом, многие задачи, решение которых сводится к компоновке стандартных 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.

1. Архитектура системы

Рассмотрим архитектуру и основные принципы реализации системы управления сценариями (СУС) для среды MathCloud. Разработанная система имеет клиент-серверную архитектуру (рис. 1). Клиентская часть системы включает редактор сценариев, а серверная — сервис управления сценариями и среду выполнения сценариев.

Редактор сценариев предназначен для взаимодействия пользователей с СУС. 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.

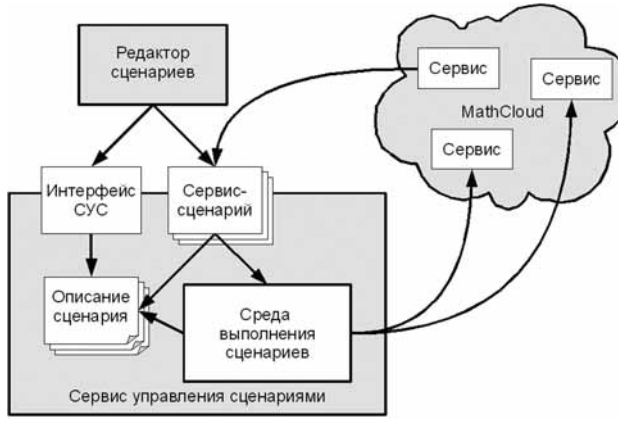

Рис. 1. Архитектура системы управления сценариями

В соответствии с сервис-ориентированным подходом, сервис управления сценариями должен обеспечить развертывание каждого сохраненного сценария в виде нового сервиса среды MathCloud. Последующий запуск сценария осуществляется путем вызова соответствующего сервиса через унифицированный REST-интерфейс [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.

2. Редактор сценариев

Редактор сценариев СУС реализован в виде веб-приложения на языке JavaScript. Такая конфигурация позволяет сделать пользовательский интерфейс нетребовательным к ресурсам (в силу особенностей языка JavaScript) и предельно переносимым — фактически, веб-интерфейс может использоваться без предварительной установки на любом компьютере, где установлен современный веб-браузер. Это означает совместимость с подавляющим большинством современных операционных систем. Кроме того, такая конфигурация позволяет отделить серверную часть СУС от пользовательского интерфейса. Фактически, веб-интерфейс является клиентской программой для сервиса управления сценариями. Таким образом, интерфейс и сервис управления сценариями могут работать на разных машинах.

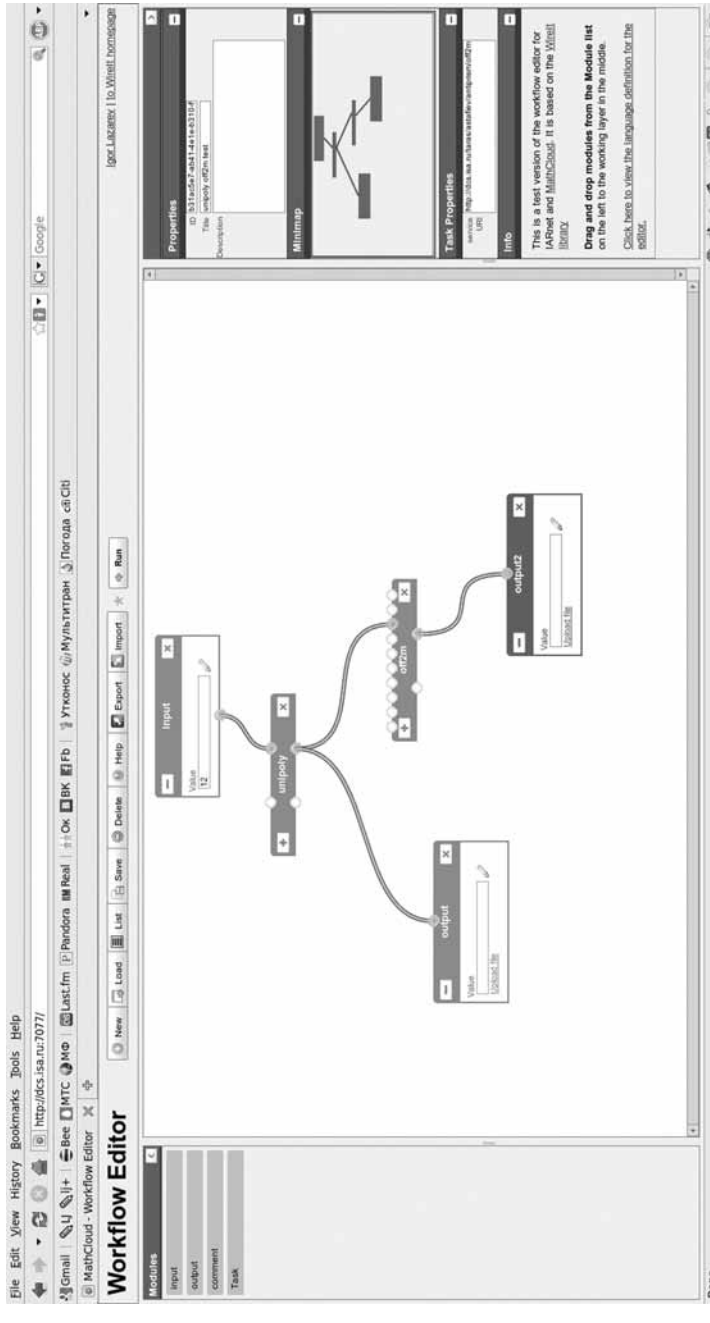

Рис. 2. Редактор сценариев

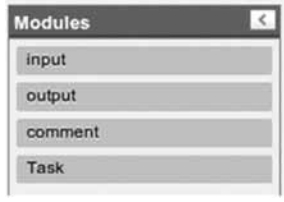

Рис. 3. Панель контейнеров

Рис. 4. Свернутый контейнер

Также это позволяет впоследствии реализовать любой другой пользовательский интерфейс для СУС.

Однако все это не означает, что данный веб-интерфейс, подобно многим из существующих браузерных приложений, проигрывает в удобстве, быстродействии и функциональности традиционным инсталлируемым приложениям. Благодаря JavaScript-библиотекам WireIt [3], YUI [4] и Inputex [5], на которых основан программный код редактора сценариев, удалось сделать интерфейс редактора легковесным, интуитивным и удобным для пользователя.

2.1. Общие сведения

Рассмотрим основные элементы веб-интерфейса. Первое, что видит пользователь, открывая в браузере его адрес — это *редактор сценариев* (рис. 2).

В левой части страницы расположен список доступных *контейнеров* (рис. 3). В центре — тело сценария. Перетаскивая модули из списка контейнеров в тело сценария и соединяя их связями, пользователь формирует описание сценария.

Каждый контейнер имеет входные и выходные *порты*, посредством которых контейнер принимает и отдает данные, причем для каждого порта указан *тип данных*, которые могут передаваться через него. Тип данных порта отображается на экране в формате **имя:тип** при наведении на порт курсора. Связи, которыми соединяются порты контейнеров, называются *проводами*.

В данный момент провода могут соединять только порты разных контейнеров и одинаковых типов. По понятным причинам из каждого выходного порта может исходить неограниченное число проводов, но к каждому входному порту можно подключить только один провод.

В настоящее время веб-интерфейс может оперировать данными следующих типов: integer, number, array, string, text, url, boolean, file.

2.2. Контейнеры

После добавления в тело сценария у нового контейнера появляются входные и выходные порты, количество которых зависит от конкретного контейнера и содержимое, которое меняется в зависимости от вида контейнера.

Каждый контейнер имеет кнопки **collapse** и **close** (рис. 4). Первая из них (обозначена значком «минус») сворачивает контейнер в узкую полоску, содержащую только его название, для экономии места на экране. После сво-

рачивания кнопка **collapse** заменяется кнопкой **expand** (обозначена значком «плюс»), позволяющей снова развернуть контейнер. Кнопка **close** (обозначена крестом) удаляет контейнер из сценария.

На текущий момент доступны всего четыре вида контейнеров — это **input**, **output**, **Task** и **comment**. Несмотря на малое число видов контейнеров, их выразительности уже сейчас достаточно для того, чтобы конструировать довольно сложные и развернутые сценарии.

Рис. 5. Контейнер в состоянии MISSING

Во время редактирования каждому контейнеру присваивается *состояние*, указывающее на соответствие или несоответствие данного кон-тейнера требованиям среды выполнения. В режиме редактирования контейнеры могут принимать три состояния: **READY** (обозначается синим цветом контейнера), означающее, что контейнер готов к запуску сценария, **UNREADY** (обозначается серым), означающее неготовность к запуску из-за недостатка каких-либо данных, и **MISSING** (обозначается красным цветом, рис. 5), означающее отсутствие связи с соответствующим сервисом MathCloud, если вид контейнера предполагает такую связь.

Контейнеры типа **input** или *входные контейнеры* используются для того, чтобы задавать входные параметры сценария. Каждый такой контейнер имеет один выходной порт. Тело контейнера содержит поле, в которое вводится значение входного параметра, а также кнопку «карандаш». При нажатии на «карандаш» у контейнера появляется дополнительная панель (рис. 6), позволяющая задать тип выходного параметра (т. е. и тип соответствующего порта), а также некоторые другие настройки контейнера — такие, как имя выходного порта. Все входные контейнеры должны иметь уникальные имена в пределах сценария.

При выборе типа входного параметра **file** в контейнер добавляется форма загрузки, позволяющая выбрать файл для загрузки из локальной файловой системы (рис. 7). Пользователь может следить за ходом загрузки файла — он отображается в виде традиционного индикатора закачки. По окончании загрузки URL загруженного файла помещается в поле входного параметра и может передаваться по проводам в качестве данных типа **file**.

Входной контейнер, поле которого пусто, окрашивается в серый цвет и имеет статус UNREADY. Это сигнализирует пользователю о том, что у сценария отсутствуют соответствующие данному контейнеру входные данные.

Контейнеры типа **output** или *выходные контейнеры* используются для отображения результатов выполнения сценария. Каждый такой контейнер имеет один входной порт. По остальным свойствам выходной контейнер очень похож на входной, за тем исключением, что он не содержит

Рис. 6. Входной контейнер в состоянии UNREADY с дополнительной панелью

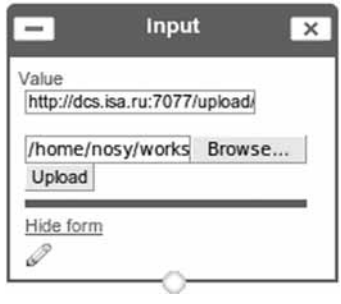

Рис. 7. Входной контейнер с загруженным файлом

Рис. 8. Выходной контейнер со ссылкой для просмотра файла

формы загрузки файлов. Зато, если **output** имеет тип **file** и пришедший на вход файл — известного типа, в контейнер автоматически добавляется ссылка **View** для отображения принятого файла в соответствующем средстве просмотра (рис. 8). Например, для файлов с расширением **.m** (трехмерные объекты), таким средством является Java-апплет LiveGraphics3D. Возможно добавление новых средств просмотра для пользовательских типов файлов.

Выходной контейнер, к порту которого не подсоединен провод, переходит в состояние UNREADY и окрашивается в серый цвет, чтобы оповестить пользователя, что данный контейнер не соответствует никаким выходным данным сценария. Все выходные контейнеры должны иметь уникальные имена в пределах сценария.

Контейнеры типа **Task** или *задания* предназначены для добавления в сценарий вызовов сервисов MathCloud. Для экономии места на экране,

они добавляются в тело сценария в свернутом виде. Развернув такой контейнер, в его тело можно ввести URI соответствующего сервиса (рис. 9). Также URI сервиса можно ввести в поле **service URI**, находящееся в правой панели и дублирующее тело выделенного в данный момент контейнера. Веб-интерфейс совершает Ajax-запрос по этому адресу и, согласно MathCloud REST API [2], получает информацию о количестве, типах и именах входных и выходных параметров данного сервиса. После чего эти параметры добавляются в качестве входных и выходных портов контейнера с соответствующими типами и именами, а заголовок контейнера меняется на имя соответствующего сервиса (рис. 10). В случае неудачи при соединении с сервисом, такой контейнер переходит в состояние MISSING и окрашивается в красный цвет. В качестве сервисов можно подключать и уже существующие сценарии, просто введя URI сохраненного сценария в тело задания. Таким образом, для использования функциональности, уже реализованной в существующем сценарии, достаточно добавить его в новый сценарий в виде задания.

Контейнеры типа **comment** или *комментарии* содержат в себе простую форму для ввода тех или иных комментариев, относящихся к данному сценарию (рис. 11).

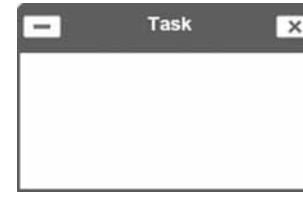

Рис. 9. Только что добавленное задание

Рис. 10. Задание, соединенное с сервисом unipoly

Рис. 11. Комментарий

2.3. Панель свойств

В правой части страницы расположена *панель свойств*, на которой отображаются различные свойства сценария (рис. 12). Благодаря тому, что эта панель создана с использованием YUI Accordion [4], не используемые в данный момент элементы панели можно скрыть от просмотра, чтобы не загромождать экранное пространство.

В верхней части этой панели находится форма **Properties**, состоящая из трех полей. Первое поле **ID** содержит уникальный идентификатор данного сценария в СУС. Содержимое этого поля генерируется автоматически при сохранении сценария. Второе поле **Title** содержит имя сценария, вводимое пользователем. Заполнение этого поля является обязательным.

Рис. 12. Панель свойств

Третье поле **Description** может содержать краткое текстовое описание сценария.

В средней части панели находится элемент **Minimap**, схематически отображающий сценарий в уменьшенном виде. В случае особо крупных сценариев, превышающих размером экран, его можно использовать для навигации по сценарию.

Ниже располагается форма **Task Properties** с единственным полем service URI, которое, как было упомянуто ранее, дублирует тело задания, в текущий момент времени выделенного пользователем. В самой нижней части панели расположен элемент **Info**, который содержит в себе краткую информацию о редакторе сценариев**.**

2.4. Главное меню

В верхней части страницы расположено *главное меню* (рис. 13), при помощи которого пользователь может сохранять созданные им сценарии, загружать сохраненные сценарии из удаленного репозитория, импортировать и экспортировать описания сценариев в формате JSON [6] и запускать сценарий на выполнение.

Рассмотрим подробнее каждую из функций, предоставляемых главным меню редактора сценариев.

Кнопка **New** отвечает за создание нового описания сценария. После ее нажатия очищается содержимое всех форм и тела сценария.

Кнопка **Load** позволяет пользователю загрузить в редактор существующий сценарий, указав его уникальный идентификатор.

Кнопка **List** выводит на экран список всех описаний сценариев, хранящихся в СУС (рис. 14). При выборе одного из описаний, оно загружается в редактор сценариев. Каждый из элементов списка содержит имя сце-

Рис. 13. Главное меню редактора сценариев

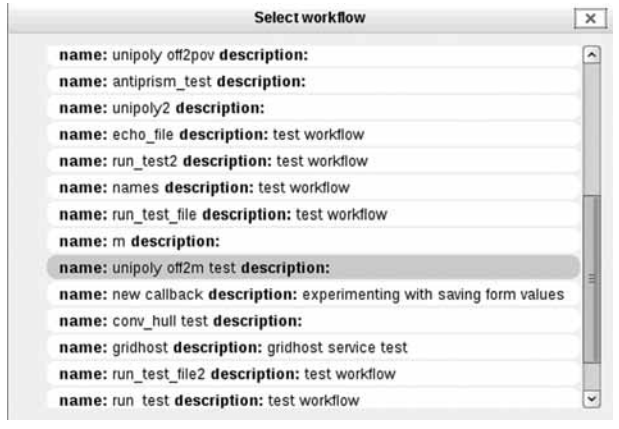

Рис. 14. Панель загрузки сценариев

нария и его краткое словесное описание, введенные пользователем при редактировании.

Кнопка **Save** отвечает за сохранение сценариев в СУС. При сохранении нового сценария СУС присваивает ему уникальный идентификатор. После этого можно запускать сценарий на выполнение. При повторном сохранении уже существующего в СУС сценария редактор предложит пользователю перезаписать имеющееся описание сценария или сохранить данный сценарий под новым идентификатором. Описания сценариев хранятся в СУС в формате JSON.

Кнопка **Delete** удаляет данный сценарий из СУС.

По кнопке **Help** на экран выводится панель с кратким описанием редактора сценариев.

Кнопка **Export** выводит на экран панель с описанием текущего сценария в формате JSON.

Кнопка **Import** позволяет пользователю загрузить в редактор описание сценария в формате JSON.

Между кнопками **Import** и **Run** расположен *индикатор сохранения сценария*, который появляется, если в сценарий были внесены изменения со времени последнего сохранения.

Кнопка **Run** запускает сценарий на выполнение.

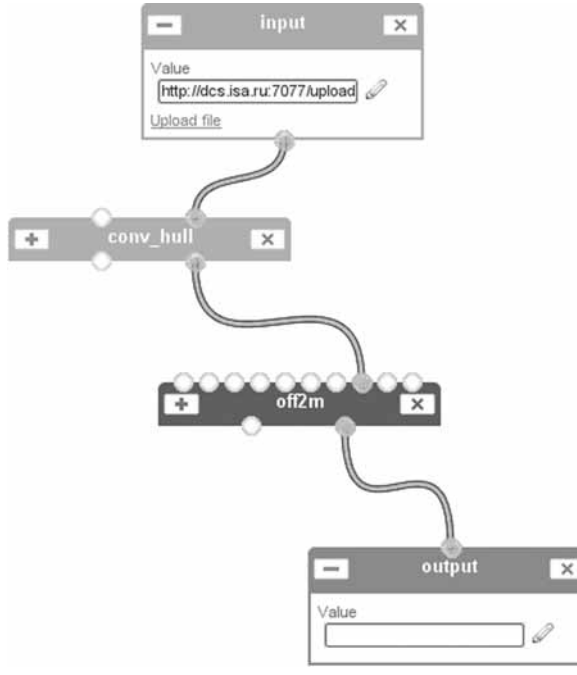

Рис. 15. Выполнение сценария

2.5. Выполнение сценария

Остановимся подробнее на том, что происходит, начиная с момента запуска сценария на выполнение. После нажатия пользователем кнопки **Run** редактор проверяет описание сценария на соответствие следующим требованиям: наличие описания сценария в СУС (проверка на то, сохранялся ли сценарий, в противном случае ему не присвоен идентификатор, необходимый для запуска); наличие у сценария имени; уникальность имен входных и выходных контейнеров; отсутствие в описании сценария контейнеров с состояниями UNREADY и MISSING.

После этого сервису управления сценариями отправляется Ajax-запрос на выполнение сценария. Содержимое входных контейнеров подается в качестве входных параметров запроса. Среда выполнения осуществляет запуск сценария, в то время как веб-интерфейс начинает регулярно (на настоящий момент каждую секунду) опрашивать ее о ходе выполнения сценария и отображать изменения на экране (рис. 15). Данные о состоянии выполнения сценария (состояния отдельных элементов и всего сценария в целом) передаются в формате JSON.

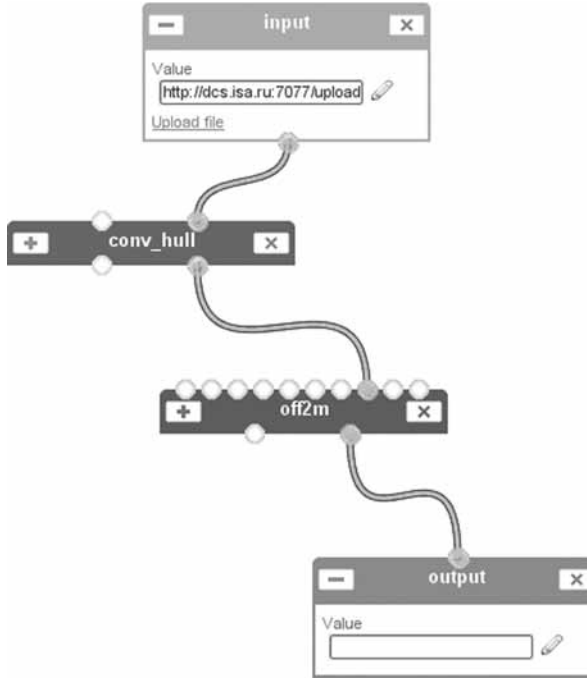

Рис. 16. Сценарий, в ходе выполнения которого произошли ошибки

В режиме выполнения, так же как и в режиме редактирования, контейнеры имеют состояния (рис. 15). Так, задание, ожидающее выполнения, имеет состояние WAITING и окрашено в синий цвет. Выполняемое в текущий момент задание имеет состояние RUNNING и бирюзовый цвет. Выполненное задание имеет состояние DONE и зеленый цвет. Для заданий, выполнение которых завершено неудачно, существует состояние FAILED и красный цвет контейнера (рис. 16).

Для заданий, выполнение которых было отменено в ходе выполнения сценария (например, в результате выбора другой ветви графа после условного перехода), зарезервированы состояние CANCELLED и серая окраска контейнера.

После завершения выполнения сценария в JSON-сообщении о состоянии выполнения сценария появляется атрибут *result*, содержимое которого веб-интерфейс распределяет между выходными контейнерами согласно их именам и типам (рис. 17).

После завершения работы в режиме выполнения, пользователь может вернуться к режиму редактирования сценария, щелкнув мышью

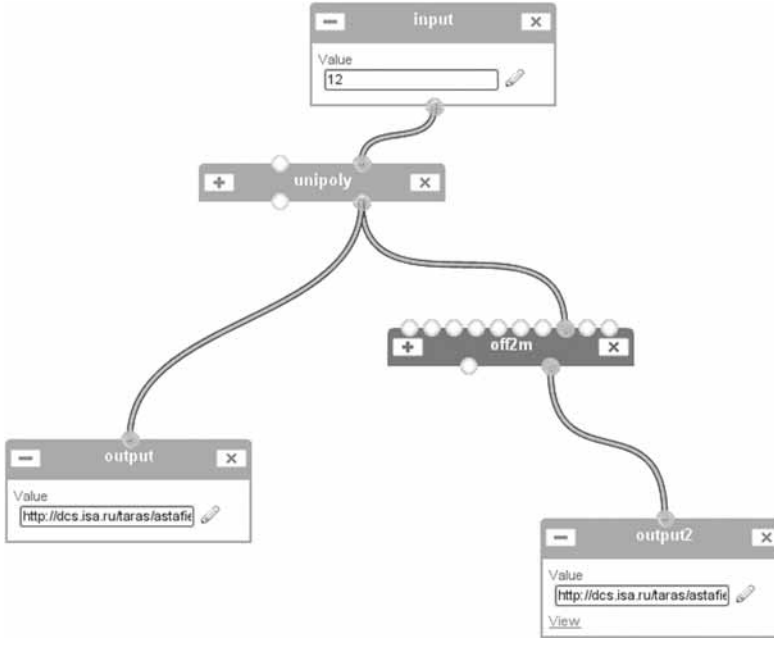

Рис. 17. Выполненный сценарий

на любом свободном месте тела сценария. Все контейнеры из состояния DONE снова перейдут в состояние READY и сменят цвет на исходный.

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 подхода REST (Representational State Transfer) [7], что позволяет удобным образом взаимодействовать с сервисом из Веб-редактора. В соответствии с принципами REST интерфейс сервиса образован совокупностью идентифицируемых при помощи URI ресурсов, поддерживающих стандартные методы протокола HTTP (табл. 1). Данными ресурсами являются:

- сервис управления сценариями, идентифицируемый с помощью WFMS_SERVICE_URI;

Таблица 1

Ресурсы и методы интерфейса сервиса управления сценариям

Ресурс / URI	GET	POST	PUT	DELETE
Сервис СУС 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	—	—	—

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

Ресурс-сервис СУС 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 алгоритмического сервиса [2], используемый для доступа к сервисам MathCloud. 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, Веб-редактор не может напрямую обратиться по протоколу HTTP к сервису, размещенному не на одном Веб-сервере с редактором. Поэтому редактор отправляет запрос GET ресурсу-прокси, выполняющемуся на одном Веб-сервере с редактором, указывая в query-параметре URI интересующего сервиса. Ресурс-прокси загружает описание данного сервиса и возвращает его редактору.

Текущая версия сервиса управления сценариями реализована на языке Java на базе встраиваемого Веб-сервера Jetty и библиотеки Jersey, открытой эталонной реализации спецификации JAX-RS (Java API for RESTful Web Services).

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.

Приведем пример описания и запуска сценария с помощью WorkflowRuntime API:

```
// DESCRIBE
Workflow wf = new Workflow("test");
wf.addInput("in");
wf.addOutput("out");

Task taskA = new TaskA("A");
taskA.setInput("in", wf.getInput("in"));
wf.addTask(taskA);
Task taskB = new TaskBC("B");
taskB.setInput("in", taskA.getOutput("out1"));
wf.addTask(taskB);

Task taskC = new TaskBC("C");
taskC.setInput("in", taskA.getOutput("out2"));
wf.addTask(taskC);
Task taskD = new TaskD("D");
taskD.setInput("in1", taskB.getOutput("out"))
.setInput("in2", taskC.getOutput("out"));
wf.addTask(taskD);

wf.setOutput("out", taskD.getOutput("out"));

// RUN
wf.setInputValue("in", "workflow");
wf.run();
System.out.println("Result: " + wf.getOutputValue("out"));
```

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 одновре-

менно 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 (см. главу 2), 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.

Таким образом, обработка запроса к сервису сценария в СУС MathCloud состоит из следующих шагов:

- преобразование описания сценария в объектную модель сценария WorkflowRuntime;
- установка значений входных параметров для сценария;
- запуск сценария в WorkflowRuntime в отдельном потоке;
- отслеживание и передача клиенту состояния выполняющегося сценария;
- получение и передача клиенту значений выходных параметров выполненного сценария.

Заключение

В работе рассмотрена реализация системы управления сценариями распределенной среды MathCloud, позволяющая описывать сценарий совместного использования нескольких сервисов MathCloud при помощи визуального редактора. Описаны архитектура и основные компоненты системы: визуальный Веб-редактор сценариев, сервис управления сценариями и среда выполнения сценариев. Предложенный в работе подход к реализации композиции сервисов среды MathCloud позволяет скрыть от пользователя детали реализации вызовов сервисов и передачи данных между ними, оставив пользователю только необходимость правильного соединения сервисов друг с другом. Таким образом, многие задачи, решение которых сводится к компоновке стандартных сервисов, становятся доступными для пользователей, не обладающих навыками распределенного программирования.

Литература

1. *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., Новороссийск). М.: Изд-во МГУ, 2009. С. 463–467.
2. *Сухорослов О. В.* Унифицированный интерфейс доступа к алгоритмическим сервисам в Web // В наст. сборнике.
3. WireIt // A JavaScript Wiring Library: http://javascript.neyric.com/wireit/
4. The Yahoo User Interface: http://developer.yahoo.com/yui/
5. Inputex // Field Framework for Web Applications: http://javascript.neyric.com/inputex/
6. JavaScript Object Notation: http://www.json.org/
7. *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.