Использование workflow-методологии для описания процесса распределенных вычислений

И. В. Лазарев, О. В. Сухорослов

В работе рассматривается вопрос использования workflow-методологии для описания и реализации распределенных вычислительных процессов, в рамках которых происходит координированное взаимодействие набора распределенных ресурсов. Предварительно вводятся основные понятия workflow-методологии. Описывается специфика распределенных вычислительных сценариев. Рассматриваются существующие подходы к представлению сценариев и их применимость к распределенным вычислениям.

1. Workflow-методология

На мануфактурах XVI в. любой продукт проходил через руки ремесленников разных специальностей, которые выполняли различные виды работ — например, кто-то ткал сукно, кто-то занимался его отделкой, кто-то окрашивал его, а кто-то шил из ткани готовые вещи. Затем конечный продукт помещался на склады, откуда попадал к торговцам для продажи. Смотрители следили за ходом работы и раздавали новые задания, а владелец предприятия решал, кого из ремесленников нанять, кого уволить, как изменить процесс производства, чтобы сделать его эффективнее.

Немногое изменилось за столетия — управленцы по-прежнему распределяют работу между ресурсами, основываясь на их специализации, «умениях» и «навыках». Поначалу такими ресурсами были только люди, которым позднее начали помогать станки, счетные машины, компьютеры. Некоторые шаги были автоматизированы — например, денежные счета автоматически подытоживались и распечатывались, но только после того, как люди отсортируют их, или введут необходимые данные. Несмотря на то, что ход работы был, по крайней мере, частично автоматизирован, управление процессом, сценарием работы оставалось по сути таким же — управленцы раздавали задания и следили за их исполнением, передавали продукт от инстанции к инстанции. Для слежения за ходом работы составлялись бумажные списки — чтобы отыскать, где именно процесс

пошел по неправильному пути или, например, измерить его производительность. И целая армия людей занималась поиском проблем или ошибок в маршрутизации, чтобы сохранять процесс «на ходу».

С другой стороны, долгое время существовал и существует другой подход к управлению большими коллективами. Например, когда снимается художественный фильм, режиссер имеет на руках сценарий (по большому счету это описание взаимодействия актеров между собой) с набором ролей и коллектив актеров, причем далеко не каждый из них способен сыграть каждую роль, поэтому режиссер производит «сопоставление ресурсов задачам». Похожим образом все происходит и в музыкальных оркестрах, только вместо режиссера там — дирижер, а вместо сценария — партитура. То есть, в этих «организациях» само абстрактное задание процесса отделено от «производственных мощностей» — оркестровое произведение может быть исполнено по тем же нотам другими музыкантами, как и сценарий фильма может быть сыгран другими актерами.

Говоря повседневным языком, *workflow-методология* позволяет привнести элементы второго подхода в традиционный производственный процесс, разыграть его, «как по нотам» или «по сценарию».

За последние два десятилетия были разработаны инструменты, помогающие не только выполнять работу, но и управлять ее процессом. Компьютерные workflow-системы являются шагом вперед по сравнению с обычными процедурными документами. В данных системах производственный процесс задается формально, и ход работы управляется программой, которая раздает задания, передает работу от одного участника процесса к другому и отслеживает, на какой стадии находится ее выполнение [1].

Перечислим основные преимущества данного подхода [1]:

- Работа не направляется «не по адресу» и не задерживается — вмешательство извне нужно только в редких случаях, для исправления сбоев или последствий неправильного управления работой.
- Менеджеры могут уделять больше времени персоналу и таким вопросам, как индивидуальная работоспособность, оптимизация технологического процесса и т. д., вместо того чтобы заниматься рутинным распределением заданий. Больше не требуются специальные кадры для слежения за ходом работы и за тем, на какой стадии она находится.
- Все операции формально документируются и точно исполняются — можно быть уверенным, что работа выполняется именно так, как запланировано руководством, с учетом всех деловых и юридических требований.
- Каждому заданию приписывается лучший для него исполнитель, будь то человек или машина, и наиболее важные задания распределяются в первую очередь. Сотрудникам не приходится биться над выбором, с чего начать работу. Больше не затягивается начало работы над важными, но сложными заданиями.

- Параллельные работы, при которых два и более задания выполняются одновременно, осуществляются гораздо легче, чем в традиционных процессах, управляемых вручную.
- Упрощается работа с данными, которые теперь могут храниться удаленно и использоваться посредством Интернета.

Для дальнейшего описания необходимо ввести основные понятия, относящиеся к данной методологии.

1.1. Основные понятия

Термин “workflow” используется в двух аспектах — как формальное представление некоторого процесса и как некоторый подход к автоматизации процессов, основанный на подобном представлении.

Начнем с первого аспекта. Буквальный перевод термина “workflow” как «поток работ» плохо раскрывает содержание данного понятия, поэтому мы будем использовать термин «сценарий». Под ***сценарием*** мы будем подразумевать формальное представление (модель) некоторого процесса, включающее в себя:

- Описание элементарных *операций*, из которых состоит процесс.
- Описание *исполнителей*, которые выполняют указанные операции.
- Описание зависимостей между операциями, а именно — *потоков управления*, которые определяют последовательность выполнения операций и синхронизацию между ними, и *потоков данных*, которые определяют передачу информации между операциями.
- Описание внешних *событий*, которые могут влиять на ход процесса, и правил их обработки.

Второй аспект термина “workflow” нашел отражение в определении, данном Workflow Management Coalition [2], — «это автоматизация, полностью или частично, бизнес-процесса, при которой документы, информация или задания передаются для выполнения необходимых действий от одного участника к другому в соответствии с набором процедурных правил». Данное определение, несмотря на привязку к бизнес-процессам, хорошо отражает суть ***workflow-методологии*** как некоторого подхода к автоматизации вообще говоря различных процессов.

Системой управления сценариями (*workflow management system*, *СУС*) будем называть систему, позволяющую создавать сценарии, запускать и управлять их выполнением [2]. СУС состоит из набора программных компонентов, предназначенных для хранения и интерпретации описаний процессов (сценариев), создания и управления экземплярами запущенных процессов, а также организации их взаимодействия с участниками процесса и внешними приложениями.

Программное приложение, непосредственно выполняющее интерпретацию и запуск сценария, а также управляющее экземплярами запущенных процессов, будем называть ***средой выполнения сценариев*** (*workflow engine*).

1.2. Место систем управления сценариями в эволюции программных приложений

Хотя сейчас почти повсеместно считается, что компьютерные СУС появились в середине 90-х гг., это не совсем так. В действительности, первые шаги в этой области были сделаны еще в 70-х, в исследовательском центре Xerox PARC [3], которому мы обязаны такими изобретениями, как компьютерная мышь, лазерный принтер, Ethernet, пиктограммы и графический оконный интерфейс.

Сотрудниками Xerox PARC была разработана автоматизированная система документооборота Office Talk, уже имевшая многие черты СУС — она представляла порядок следования заданий в офисе от сотрудника к сотруднику графически; а также система SCOOP, в которой поток заданий представлялся в виде сетей Петри (см. раздел 3.3.2).

Однако эти, новаторские для своего времени, разработки не получили должного развития из-за незрелости существовавших программных технологий. В приложениях того времени пользовательский интерфейс еще не был отделен от основного, рабочего кода, программы еще не были построены по модульному принципу, сетевые технологии только начинали свое существование. Поэтому понадобился перерыв длиной более 20 лет, прежде чем сценарий был выделен в чистом виде из текста программы систем автоматизации, и началась современная история СУС. Развитие программных технологий позволило применять методы офисной автоматизации уже не к людям, а к программным приложениям, а через них и ко всем остальным ресурсам.

На рис. 1 схематически показаны основные этапы эволюции программных приложений в течение последних сорока лет:

- 1965–1975. Разделение данных и программного кода в приложениях (**APPL**ications).
- 1975–1985. Управление базами данных. Для работы с данными используются СУБД (**D**ata**B**ase **M**anagement **S**ystems).
- 1985–1995. Появление пользовательского интерфейса (**U**ser **I**nterface **M**anagement **S**ystems), отделение его от остальной, «рабочей» части программы.

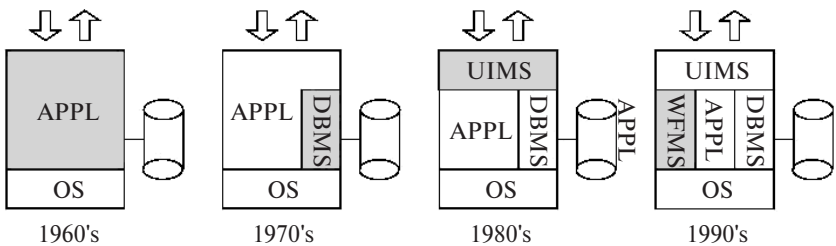

Рис. 1. Этапы эволюции программных приложений [11]

- 1995–2005. Управление сценариями (**W**ork**F**low **M**anagement **S**ystems), выделение сценария в чистом виде из кода приложения.

1.3. Стандарты в области систем управления сценариями

Workflow Management Coalition (WfMC) [2] — некоммерческая международная организация, объединяющая исследователей, разработчиков СУС, пользователей и аналитиков. WfMC призвана способствовать развитию и широкому внедрению worklow-методологии путем создания стандартов, обеспечивающих единую методологическую основу и интероперабельность различных программных решений.

В рамках WfMC была разработана эталонная модель системы управления сценариями (Workflow Reference Model), которая определяет общие характеристики, терминологию и компоненты СУС. Данная модель служит основой для разработки детальных спецификаций, затрагивающих отдельные аспекты СУС.

Эталонной моделью WfMC среде выполнения сценариев приписано пять следующих интерфейсов (см. рис. 2):

1. ***Интерфейс описания процесса,*** предназначенный для описания действий, совершаемых в ходе выполнения сценария, и ресурсов, выполняющих работу.

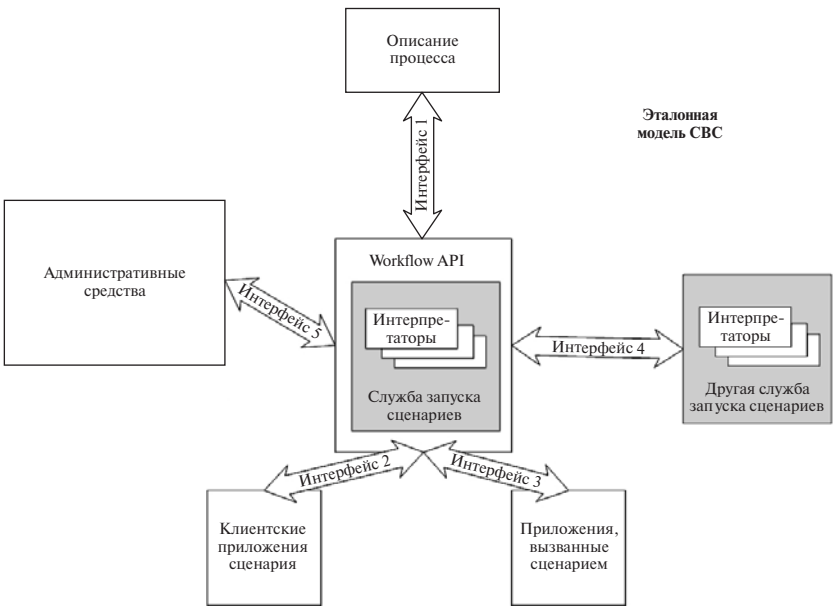

Рис. 2. Эталонная модель сценария WfMC

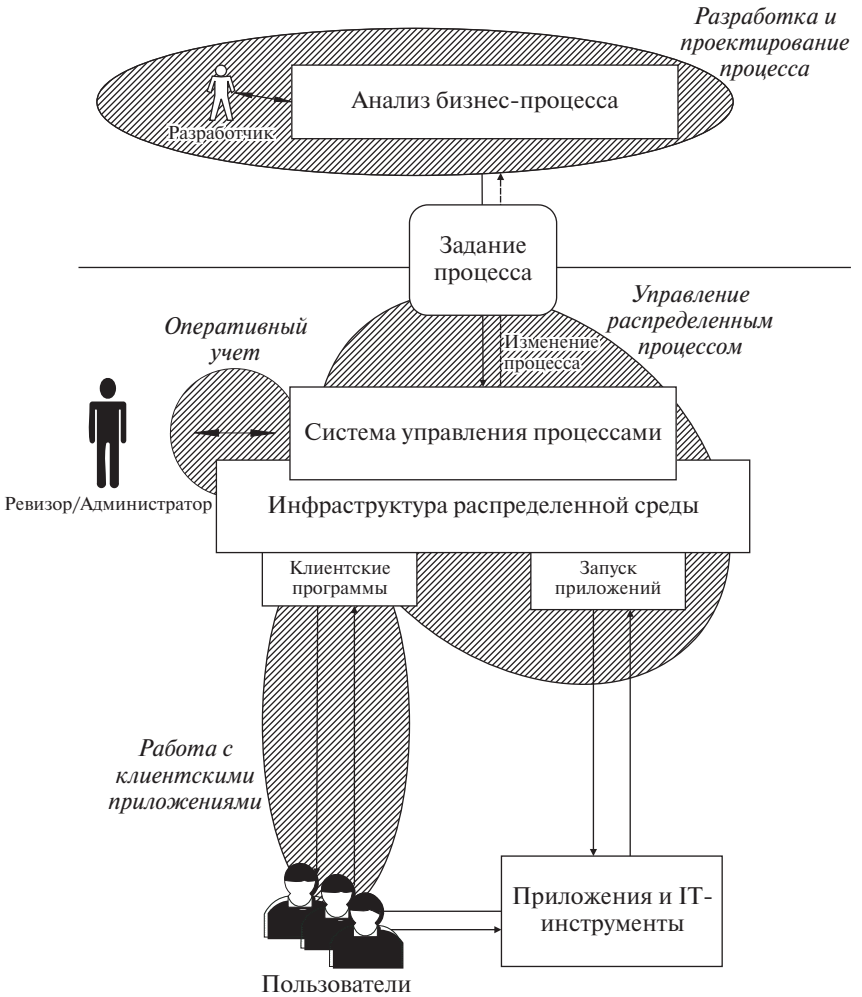

Рис. 3. Область применения эталонной модели WfMC [1]

2. ***Клиентский интерфейс***, определяющий возможные вызовы сценария клиентскими программными приложениями, например запрос на выполнение новой работы или доработку уже сделанной.
3. ***Интерфейс вызова приложений,*** позволяющий в процессе выполнения сценария вызывать внешние программные приложения.
4. ***Интерфейс взаимодействия с внешними сценариями***, определяющий возможные взаимодействия с другими сценариями, взаимодействия между разными СУС внутри одной или нескольких компаний.

5. ***Интерфейс администрирования и мониторинга***, предназначенный для протоколирования хода выполнения сценариев и контроля того, на какой стадии выполнения находится работа.

На рис. 3 показано, как воплощение эталонной модели WfMC выглядит на практике.

2. Научные вычислительные сценарии

На сегодняшний день в Интернете существует достаточное количество мощных вычислительных инструментов, но до недавнего времени не было возможности получить доступ к ним, не прибегая к программированию на низком уровне. Далеко не каждый ученый готов этим заниматься, и для многих из них эти вычислительные мощности оставались фактически недоступными. Также не было возможности координировать их взаимодействие между собой: собирать ресурсы разных типов в единую вычислительную систему, а ведь ресурсы, нужные ученым для работы, нередко располагаются в разных городах, в ведомости разных организаций, и объединить их можно только посредством Интернета.

Заменив в производственном процессе управленца ученым, а его подчиненных — распределенными вычислительными ресурсами, мы получим распределенный вычислительный процесс, также представимый в виде сценария. Это позволит ученому описывать и производить запуск вычислительных экспериментов на качественно новом уровне, с возможностью их повторения и проверки. А значит, вместо того, чтобы разрабатывать «с нуля» новые средства для координирования и доступа к вычислительным ресурсам, можно использовать уже существующую и испытанную методологию сценариев, безусловно, не забывая о чисто научной специфике. Поговорим о ней.

На рис. 4 показано место научных сценариев среди многообразия видов прикладных сценариев. Научные сценарии занимают особое место

Рис. 4. Категории сценариев [4]

среди остальных видов сценариев, охватывая производственные, уникальные (сделанные «под цель») и совместные процессы. Это действительно так — научные процессы часто бывают связаны с производством, с совместной работой различных организаций, и порой являются уникальными, т. е. рассчитанными на узко специфическую область применения.

Приведем основные отличия научных вычислительных процессов от бизнес-процессов, на которые ориентируются традиционные СУС:

- Ориентация на сложные вычислительные ресурсы.
- Количество ресурсов, которые потребуются для выполнения сценария (решения задачи), может быть неизвестно априори, так как для некоторых классов задач трудно оценить предстоящий объем вычислений.
- Необходимость работы в динамичной распределенной среде, в которой ресурсы не известны априори и могут быть подвержены отказам.
- Работа с большими объемами данных.
- Необходимость выполнять большое количество идентичных заданий с переменными параметрами.
- Необходимость следить за выполнением процесса и контролировать его, в том числе внося специальные для конкретных случаев изменения.
- Для многих научных сценариев характерны иерархии подсценариев, создаваемых и уничтожаемых по необходимости.

Указанные особенности научных вычислительных процессов определяют требования, которым должна удовлетворять СУС, подходящая для описания и выполнения данных процессов в виде сценариев. Традиционные СУС, рассчитанные на работу с бизнес-процессами, в подавляющем большинстве не подходят для решения научных задач. Поэтому, требуется разработка новых, научных СУС, с одной стороны опирающихся на сформировавшуюся workflow-методологию, а с другой стороны — специально рассчитанных на требования научных приложений.

Пожалуй, главное, что могут почерпнуть научные СУС из накопленного в данной области опыта, это способы формального представления сценариев. В следующем разделе мы рассмотрим основные подходы к представлению сценариев, причем сделаем это в контексте уже созданных научных СУС для того, чтобы одновременно дать обзор существующих решений в области научных вычислительных сценариев.

3. Различные подходы к представлению сценариев

За время существования методологии сценариев возникло несколько различных способов их формального описания. Можно выделить следующие подходы:

- Использование скриптовых языков.
- Использование графов:
  - — ориентированные ациклические графы;
  - — сети Петри.
- Смешанные подходы.

Кроме того, мы рассмотрим модели, ориентированные на потоки данных, как типичный пример СУС, применяемых сегодня для научных вычислений.

При рассмотрении каждого из подходов основное внимание мы будем уделять тому, как в рамках данного подхода сценарий представлен в формальном виде или отображается пользователю. При этом мы не будем подробно касаться вопросов того, каким образом происходит выполнение сценария и устроен соответствующий интерпретатор, указав только основные принципы того, каким образом следует интерпретировать отдельные конструкции описания сценария.

3.1. Скриптовые языки

Скрипт (*script*) представляет собой набор команд, предназначенный для выполнения специальной программой (называемой *интерпретатором*) без вмешательства пользователя. В отличие от программ, написанных на компилируемых языках программирования, скрипт не компилируется в машинный код, а *интерпретируется* во время его выполнения другой программой.

Соответственно, *скриптовым языком* (*scripting language*) называют язык программирования, предназначенный для написания скриптов. Как правило, подобный язык имеет небольшой словарь и проще для изучения и использования, чем компилируемые языки программирования. Однако, в общем случае, тот факт, что код программы интерпретируется, а не компилируется, не влияет существенно на семантику и синтаксис языка. Некоторые современные скриптовые языки по своей сложности и выразительности сравнимы с компилируемыми языками. Существуют языки, сочетающие в себе интерпретацию и компиляцию, такие как Java. Поэтому подобное разделение языков программирования во многом условно.

Для нас более важно само понятие скрипта, его применение и особенности. Изначально, основным назначением скриптов является объединение различных существующих компонентов для выполнения определенной задачи. Скрипты часто используются для автоматизации часто повторяющихся процессов, как, например, скрипты командной строки MS-DOS batch и Unix Shell или скрипты по сборке кода Apache Ant. Примечательно, что в переводе с английского *script* означает «сценарий», что, учитывая написанное выше, согласуется с нашим определением сценария.

Другая особенность скриптов — это приоритет простоты и быстроты написания программ-сценариев над эффективностью выполнения. Этим

и объясняется относительно небольшой словарь конструкций скриптовых языков, их простота, отсутствие строгой типизации, сложных типов данных и т. д.

Скриптовые языки в качестве средства представления сценариев могут быть удобны для пользователей, знакомых с языками программирования. В то же время, несмотря на довольно высокий уровень и простоту, они недостаточно наглядны и интуитивны для обычных пользователей.

Использование существующих скриптовых языков для представления научных вычислительных сценариев также ограничено их «словарным запасом». Для описания подобных сценариев необходимо введение дополнительных языковых элементов (например, для описания параллельного или последовательного запуска заданий, вызова удаленных ресурсов и т. д.).

Таким образом, для использования существующих скриптовых языков в научных сценариях требуется их расширение. По этому пути пошли авторы описанной в данном разделе системы GridAnt, адаптировавшие язык описания скриптов Apache Ant для описания научных вычислительных процессов в рамках Grid-среды.

Другой подход состоит в создании нового скриптового языка, специально ориентированного на описание научных вычислительных процессов. Данный подход представляет описанная далее система Karajan.

3.1.1. GridAnt

В качестве примера успешного использования скриптового языка для конструирования сценариев стоит упомянуть систему GridAnt [5], модификацию Apache Ant [6] (популярного средства автоматической сборки и интеграции Java-кода), разработанную для автоматизации процессов Grid-вычислений.

GridAnt — это программный инструмент, основанный на языках Java и XML, состоящий из интерпретатора и набора визуальных компонентов, которые обеспечивают динамическое отображение хода выполнения заданий. GridAnt является расширением Apache Ant [6] и обладает полной обратной совместимостью с ним.

GridAnt был разработан для использования в рамках Grid-среды [7] на основе ПО Globus Toolkit 2 и 3 [8] и поэтому наследует особенности работы с ресурсами в рамках подобной среды. В частности, работа с вычислительными ресурсами сводится набору элементарных шагов сценария, соответствующих низкоуровневым операциям с файлами: копирование файлов с одной машины на другую, запуск исполняемых файлов на удаленных машинах, создание файла, содержащего результаты выполнения исполняемого файла, и т. д.

GridAnt обеспечивает функциональность, которая обычно требуется программам для распределенных Grid-вычислений [7], т. е. безопасную передачу файлов, запуск заданий на удаленных вычислительных ресурсах и просмотр результатов их выполнения.

Запуск скрипта, содержащего описание сценария, производится при помощи специальной программной оболочки вокруг стандартного интерпретатора Apache Ant, которая позволяет следить за потоками данных, выполнять асинхронные вызовы (без ожидания), обработку исключительных ситуаций и ошибок. Файловый монитор предоставляет информацию о новых файлах или их новых версиях.

GridAnt имеет следующие расширенные возможности, по сравнению со стандартным Apache Ant:

- Выполнение логико-арифметических операций.
- Поддержка циклов.
- Обмен данными между ресурсами-участниками сценария и интерпретатором.
- Совместный доступ ресурсов-участников сценария к данным, особенно нужный для синхронизации работы нескольких ресурсов.

GridAnt вырос из многочисленных работ в области распределенных и высокопроизводительных вычислений, и этот опыт позволил сформулировать набор требований, которые должны выполняться любой подобной СУС, рассчитанной на работу с ресурсами Globus Toolkit:

- Независимость от платформы.
- Возможность поддержки вызовов, связанных с shell-скриптами.
- Повторяемые и параллельные конструкции управления.
- Асинхронное взаимодействие.
- Взаимодействие с удаленными ресурсами.
- Модульность — для того, чтобы части системы могли использоваться выборочно и по отдельности.
- Расширяемость — для того, чтобы пользователи могли менять систему или что-то добавлять в нее.
- Легкий перевод графического представления алгоритма в скрипт и обратно, или конструирование сценария с помощью графического интерфейса (если сценарий представляется в виде XML).

Большая часть наиболее часто запускаемых операций может выполняться, отслеживаться и проходить последующую обработку с помощью основных средств Apache Ant. Остальным требованиям отвечают расширения GridAnt, о которых было сказано выше.

Приведем основные элементы и управляющие конструкции языка, а также пример простого сценария, заключающегося в переносе файла с данными с одной машины на другую, запуске программы на второй машине над данными из указанного файла и переносе файла с результатами расчетов на третью машину.

В заключение сформулируем достоинства и недостатки рассматриваемого подхода к представлению сценария и системы GridAnt.

Достоинства:

- Удобство описания сценария для пользователей, знакомых с языками программирования и Apache Ant — они получили аналогичный инструмент для Grid-вычислений.
- Эффективная поддержка работы в Grid-среде на базе Globus Toolkit, позволяющая автоматизировать многие рутинные действия пользователя такой среды.

Недостатки:

- Способ описания сценария сложен и недостаточно интуитивен для рядовых пользователей.
- Привязка к командной строке Globus Toolkit и соответствующей низкоуровневой модели работы в Grid-среде.
- Отсутствует поддержка вызовов Grid-(Web-)сервисов.

Основные конструкции языка GridAnt и пример сценария (см. рис. 5)

- Установка параметров Grid-среды.

```
<grid-environment/>
```

- Аутентификация.

```
<grid-authenticate
directory=''/home/gregor/globus''/>
```

- Передача файлов между машинами.

```
<grid-filetransfer
from = ‘‘gridftp://hot.mcs.anl.gov/home/gregor.a.txt’’
to =''gridftp://cold.mcs.anl.gov/home/gregor/a.txt''/>
```

- Безопасная передача файлов

```
<grid-filetransfer server = ‘‘rft.mcs.anl.gov’’
from = ‘‘gridftp://hot.mcs.anl.gov/home/gregor.a.txt’’
to =''gridftp://cold.mcs.anl.gov/home/gregor/a.txt''/>
```

- Запуск исполняемого файла на удаленной машине

```
<grid-run server = ‘‘hot.mcs.anl.gov’’
executable = ‘‘/usr/home/gregor/climate’’/>
```

- Управление потоком заданий.
- Зависимости между заданиями.

```
<target name = ‘‘A’’>...</target>
<target name = ‘‘B’’ depends = ‘‘A’’>...</target>
```

- Параллельное выполнение заданий.

```
<parallel>
<target name = ‘‘C’’>...</target>
<target name = ‘‘D’’>...</target>
</parallel>
```

```
<target name="sampleWorkflow">
  <sequential>
    <grid-setup/>
    <grid-authenticate/>
    <grid-copy
        name="copyInputFile"
        provider="GT2"
        security="xmlSignature"
        delegation="limited"
        from="gsiftp://server1/inputFile"
        to="gsiftp://server2/inputFile"
        parallelStreams="4"
        tcpBuffer="16384"/>
    <grid-execute
        name="myApplication"
        provider="GT2"
        server = "server2:1234"
        security="xmlEncryption"
        delegation="full"
        executable = "myApplication"
        arguments="-file inputFile"
        directory="/home/test"
        localExecutable="false"
        redirect="false"
        outputFile="outputFile"
        errorFile="outputFile"/>
    <grid-copy
        name="copyOutputFile"
        provider="GT2"
        security="xmlSignature"
        delegation="limited"
        from="gsiftp://server2/outputFile"
        to="gsiftp://server3/outputFile"
        parallelStreams="4"
        tcpBuffer="16384"/>
  </sequential>
</target>
```

Рис. 5

- Последовательное выполнение заданий.

```
<sequential>
<target name = ‘‘E’’>...</target>
<target name = ‘‘F’’>...</target>
</sequential>
```

3.1.2. Karajan

В последнее время авторы GridAnt переключились на разработку нового инструмента — системы Karajan, уже не использующей Apache Ant. Цель ее создателей — разработать легкий в обращении инструмент описания сложных заданий для вычислительных сетей, поддерживающий масштабируемость и предоставляющий некоторые дополнительные средства, такие как обработка ошибок, поддержка контрольных точек (для сохранения состояния сценария с возможностью возврата к сохраненному состоянию), и распределенный запуск заданий.

В основе Karajan лежит XML-подобный язык структурного программирования, на котором задается сценарий. Этот язык содержит специальные элементы для моделирования циклов, логических условий, параллельного и последовательного запуска заданий, а также элементы для работы в среде Grid (например, для передачи файлов и запуска заданий на удаленных ресурсах). Часто запускаемые задания могут быть объединены в шаблоны (*templates*) и затем многократно использоваться повторно.

Работа интерпретатора в Karajan основана на событийной модели (*event model*). Участники сценария реагируют на события (*events*), сгенерированные другими участниками и сами генерируют такие события. События служат для оповещения участников сценария об изменениях в состоянии системы, или могут быть использованы для слежения за состоянием участников. Этот механизм позволяет внешней административной службе осуществлять контроль выполнения сценария. Также становится возможным внесение изменений в ход работы участников сценария на ходу, не влияя на работу других участников.

Karajan поддерживает два способа взаимодействия с пользователем через традиционную командную строку и посредством графического интерфейса (см. рис. 6). Последний содержит инструменты для графического отображения сценариев, отслеживания их состояния и сбора информации об их выполнении.

В заключение сформулируем достоинства и недостатки рассматриваемого подхода к представлению сценария и системы Karajan.

Достоинства:

- Отсутствие привязки к уже существующему продукту (а значит, и к его ограничениям).
- Более выразительный и удобный язык описания сценариев.
- Наличие удобного графического интерфейса.

Недостатки:

- Способ описания сценария сложен и недостаточно интуитивен для рядовых пользователей.
- Привязка к командной строке Globus Toolkit и соответствующей низкоуровневой модели работы в Grid-среде.
- Отсутствует поддержка вызовов Grid-(Web-)сервисов.

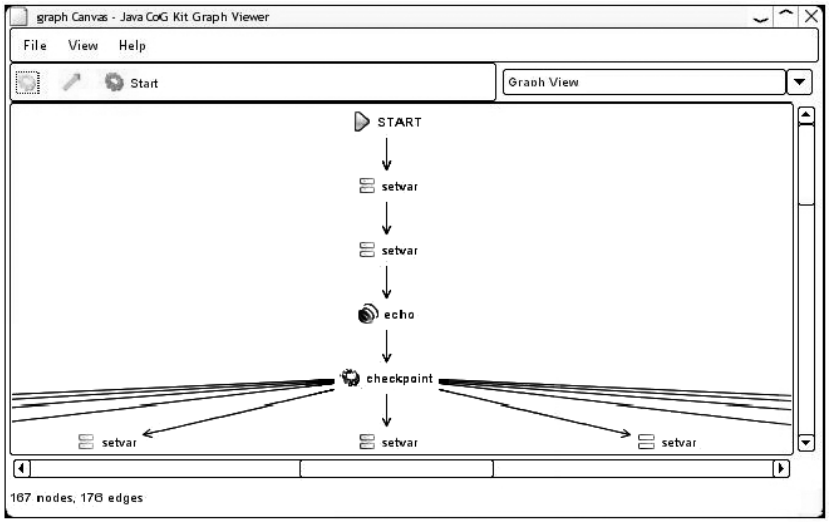

Рис. 6. Графическое отображение сценария в Karajan

3.2. Представление сценариев в виде графов

Следующий способ представления сценариев — это изображение их в виде ***графов***. Изначально, графы — это чисто математическая абстракция, но, тем не менее, они более удобны для неподготовленного пользователя, поскольку представляют сценарий наглядно. Правда, с увеличением сложности сценариев графы «разрастаются», и их становится трудно просматривать. Наглядность в этом случае можно сохранить, используя иерархическое представление графа, позволяющее скрывать детали отдельных его подграфов.

Для представления сценариев широко используются два класса графов: ориентированные ациклические графы и сети Петри.

3.2.1. Ориентированные ациклические графы

Ориентированным ациклическим графом (ОАГ) называется любой ориентированный граф, в котором нет ориентированных циклов [8]. Вершинами графа являются, например, исполняемые программы или выполняемые операции, а ребра устанавливают зависимости между ними. Преимущество таких графов — простота структуры и реализации. Но есть и недостатки: они накладывают ограничения на типы сценариев — например, нельзя явно задать циклы без применения дополнительных конструкций, уже не связанных с графовым представлением. Кроме того, такие графы способны описывать только модель поведения процесса, не фиксируя его состояние во время выполнения.

Примеры приложений, использующих DAG для представления сценариев: Condor [10], Symphony [11], Cactus [12], UNICORE [13].

Condor DAGMan Рассмотрим систему Condor в качестве примера приложения, использующего для представления сценариев ориентированные ациклические графы.

Система Condor [10] предназначена для объединения вычислительных мощностей отдельных машин и кластеров в *виртуальный кластер* с целью проведения ресурсоемких вычислений. Формально Condor представляет собой специализированную *систему пакетной обработки* (*batch processing system*) для заданий, требующих интенсивных вычислений. Как и другие полнофункциональные системы этого класса, Condor включает поддержку механизма организации очереди заданий, политик планирования, мониторинга и управления ресурсами. После того, как пользователь добавил свое задание, Condor помещает его в очередь, выбирает место и время выполнения задания в соответствии с политикой планирования, осуществляет мониторинг его выполнения и, по завершении выполнения, уведомляет об этом пользователя.

Под заданием в Condor подразумевается запуск заданного исполняемого кода (программы) на заданных входных данных. В описании задания указывается путь к программе и входным данным, а также путь к месту, где следует разместить результаты вычислений (выходные данные). Перед выполнением задания исполняемые модули вместе с данными переносятся средствами Condor на выбранные машины. Аналогично происходит перенос результатов с машины, на которой проводились вычисления, в требуемое место.

Для представления набора заданий (сценария), где входные/выходные данные или выполнение каждого из заданий могут зависеть от других заданий, в системе Condor используется ориентированный ациклический граф. Вершинами графа являются исполняемые программы, а ребра устанавливают зависимости между ними.

Condor только находит машины для запуска программ, но не осуществляет планирование их выполнения в соответствии с установленными зависимостями. Для этой цели используется специальный «метапланировщик» *Directed Acyclic Graph Manager* (*DAGMan*). DAGMan передает задания в Condor в порядке, описанном при помощи графа сценария, и обрабатывает полученные результаты. Входными данными для DAGMan является текстовый файл с описанием графа сценария, а также связанные с ним обычные файлы с описаниями всех заданий, входящих в граф. Все задания должны использовать общий системный журнал, который DAGMan использует после передачи заданий в Condor для контроля над выполнением заданий. DAGMan отвечает за планирование, восстановление после сбоев и оповещение о результатах работы заданного набора программ.

Входной файл, используемый DAGMan, включает в себя (см. рис. 7):

- Поименованный список программ с указанием файла-описания здания для каждой из них (узлы графа).
- Указание скриптов, которые могут быть выполнены до передачи задания в Condor (PRE script) или после его выполнения (POST script).

```
# Filename: diamond.dag
#
Job  A  A.condor
Job  B  B.condor
Job  C  C.condor
Job  D  D.condor
Script PRE  A top_pre.csh
Script PRE  B mid_pre.perl  $JOB
Script POST B mid_post.perl $JOB $RETURN
Script PRE  C mid_pre.perl  $JOB
Script POST C mid_post.perl $JOB $RETURN
Script PRE  D bot_pre.csh
PARENT A CHILD B C
PARENT B C CHILD D
Retry  C 3
```

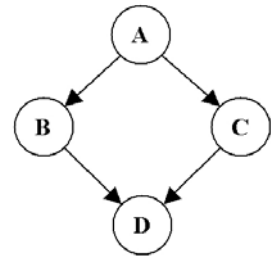

Рис. 7. Пример описания сценария в Condor DAGMan и соответствующий ему граф

- Описание отношений родитель — потомок между заданиями (ребра графа).
- Число попыток повторного запуска при возникновении ошибок.

Таким образом, узел графа включает в себя программу вместе с PRE- и POST-скриптами. Описание скрипта включает в себя его имя и, возможно, аргументы. Все скрипты выполняются на машине, с которой был произведен запуск данного сценария. Основное назначение PRE- и POST-скриптов состоит в подготовке и размещении входных файлов для задания, очистки и удаления временных данных после выполнения задания или подготовке входных данных для последующих заданий.

Для контроля ошибок скрипты и программа должны генерировать выходные значения, которые в случае ошибки отличаются от нуля. Если при выполнении PRE-скрипта произошла ошибка, программа и POST-скрипт не выполняются, и узел помечается как отказавший.

В случае, если PRE-скрипт выполнен успешно, программа передается в Condor. Если выполнение программы закончилось ошибкой, и POST-скрипта нет, то данный узел помечается как отказавший.

Если выполнение программы закончилось ошибкой и есть POST-скрипт, то статус узла определяется по выходному значению POST-скрипта. При этом при успешном завершении POST-скрипта узел помечается как выполненный успешно. Делается это для того, чтобы POST-скрипт мог провести анализ отказа программы и принять решение о том, стоит ли проводить ее выполнение заново. По умолчанию, POST-скрипт выполняется вне зависимости от значения, возвращаемого программой.

Узел, не помеченный как отказавший, после выполнения помечается как успешно выполненный.

Отношения родитель/ребенок в графе задают ограничения на последовательность запуска заданий. Задание может быть запущено только тогда, когда успешно выполнены все его родительские задания.

Число попыток повторного запуска при сбое задания по умолчанию устанавливается равным нулю. В случае, если требуется повторная попытка, все скрипты и программа выполняются в том же порядке заново.

При запуске заданий через DAGMan существуют два ограничения на файлы-описания заданий. Во-первых, каждый файл должен включать добавление только одного задания. Во-вторых, во всех описаниях должен быть указан общий системный журнал, так как DAGMan осуществляет планирование на основе событий, заносимых в него.

В случае отказа одного из узлов выполнение остальной части DAG продолжается до тех пор, пока позволяют установленные зависимости. В момент отказа DAGMan автоматически генерирует так называемый *Rescue DAG* — файл графа, функционально аналогичный исходному, но дополнительно содержащий информацию об успешно выполненных узлах с помощью параметра *DONE* в строке с описанием данного узла. Число попыток для узлов уменьшается на число уже произведенных. В случае, если сценарий запускается заново с использованием Rescue-файла, помеченные как выполненные узлы не будут запускаться заново.

В заключение, сформулируем достоинства и недостатки рассматриваемого подхода к представлению сценария и системы Condor DAGMan.

Достоинства:

- Простота структуры и реализации.
- Система Condor изначально предназначена для вычислительных целей.
- Развитая обработка отказов.

Недостатки:

- Нет средств описания логических конструкций, нельзя явно смоделировать циклы.
- Моделируется только поведение, но не состояние процесса.
- Работа с ресурсами описывается в терминах запуска исполняемых файлов на удаленных машинах, что ограничивает общность подхода и область его применения традиционными параллельными вычислениями.

3.2.2. Сети Петри

Сети Петри [16] — особый класс ориентированных графов. Теория сетей Петри является хорошо известным и популярным формализмом, предназначенным для работы с параллельными и асинхронными системами. Основанная в начале 60-х гг. немецким математиком

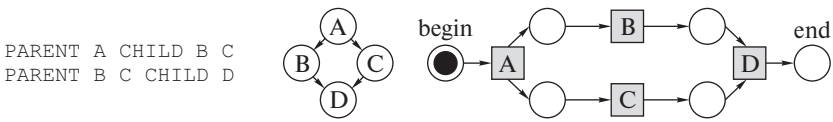

Рис. 8. Ориентированный ациклический граф и эквивалентная ему сеть Петри [15]

Таблица 1

Основные элементы сетей Петри

	**Места**	Файлы, буферы, контрольные места
	**Переходы**	Программные компоненты, контрольные переходы
	**Дуги от мест к переходам**	Место является входным местом перехода
	**Дуги от переходов к местам**	Место является выходным местом перехода
	**Фишки**	Данные, состояние перехода (выполнен/не выполнен)

К. А. Петри, в настоящее время она содержит большое количество моделей, методов и средств анализа, имеющих обширное количество приложений практически во всех отраслях вычислительной техники и даже вне ее [16].

Подробное изложение теории сетей Петри на русском языке можно найти в [17] и [18].

Тип сетей Петри, о котором будет идти речь применительно к сценариям, — это так называемые *цветные* (*раскрашенные*) сети Петри. Цветные сети Петри состоят из *мест* (обозначенных кружками), *переходов* (обозначенных квадратами), *дуг*, идущих от мест к переходам или от переходов к местам, и фишек — объектов, передвигающихся по сети и отличающихся друг от друга разными значениями приписанных им параметров (как разноцветные фишки в настольных играх).

На каждом месте одновременно может находиться несколько фишек. Место **p** называется *входным* (*выходным*) местом перехода **t**, если существует дуга от **p** к **t** (от **t** к **p**). *Начальная разметка* сети описывает положение фишек, которые находятся на местах в исходном состоянии сети. Местам может быть приписана информация о том, какие типы фишек могут на них находиться, а дугам, исходящим из мест — о том, какие типы фишек и при каких условиях могут вызывать *срабатывание* переходов, в которые они ведут.

Срабатыванием переходов описывается выполнение любого действия в системе; такое срабатывание порождает новое размещение фишек в сети. Например, простое условие срабатывания перехода: переход срабатывает, если все его входные места заняты фишками, а выходные — не достигли максимально возможного заполнения.

Благодаря фишкам, сети Петри, в отличие от ориентированных ациклических графов, позволяют описывать не только поведение процесса, но и наглядно представлять его состояние в ходе выполнения сценария.

В терминах сетей Петри сценарии задаются довольно прямолинейно: участники сценария моделируются переходами, условия (при выполнении которых задания передаются тому или иному участнику) — местами, а сами задания — фишками.

Рассмотрим основные преимущества сетей Петри при представлении сценариев.

Графическая природа. Сети Петри — графический язык. Поэтому они интуитивно понятны и легки для изучения. Их графическая природа также удобна для взаимодействия с конечными пользователями.

Формальность описания. Сценарий, описанный в терминах сети Петри, задается строго и точно, потому что семантика классических сетей Петри, как и некоторых дополнений к ним (цвет, время, иерархия) были введены формально.

Выразительность. Сети Петри поддерживают все базисные элементы, необходимые для описания сценария. С их помощью могут быть смоделированы все управляющие конструкции, существующие в современных системах управления сценариями. Более того, точное представление состояний сценария позволяет описывать ситуации неявного выбора и сохранять промежуточные состояния, с возможностью возвращения к ним.

Свойства. В последние десятилетия были подробно изучены основные свойства сетей Петри. Прочные математические основания позволяют делать строгие выводы из этих свойств.

Анализ. Сети Петри отличаются наличием большого числа методов анализа. Это их ценное преимущество с точки зрения использования для описания сценариев — данные методы могут быть использованы для доказательства различных свойств (выполнимости, инвариантности, мертвых точек и т. д.) и для вычисления характеристик выполнения сценариев (время отклика, время ожидания, степень занятости и пр.). Таким образом, становится возможным оценивать альтернативные сценарии, используя традиционные инструменты анализа сетей Петри.

Моделирование основных управляющих конструкций в сетях Петри

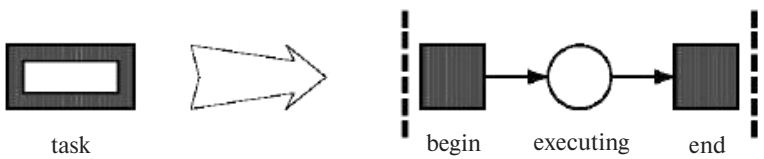

Рис. 9. Задание

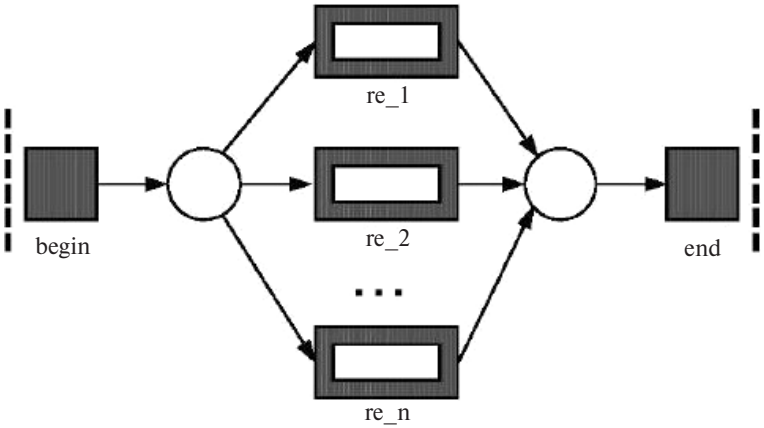

Рис. 10. Выбор

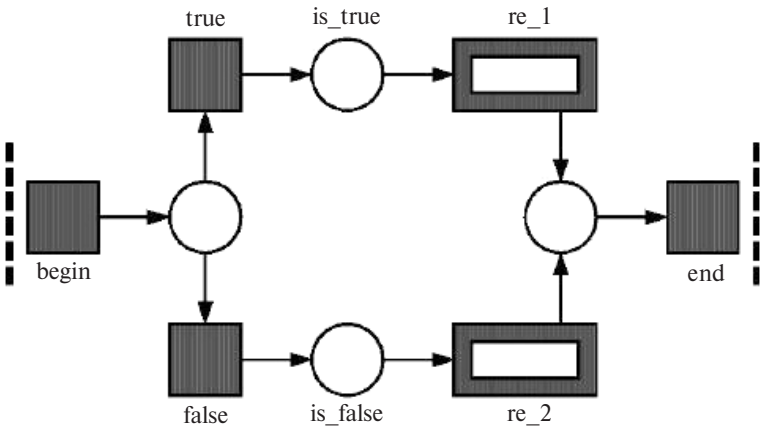

Рис. 11. Условие

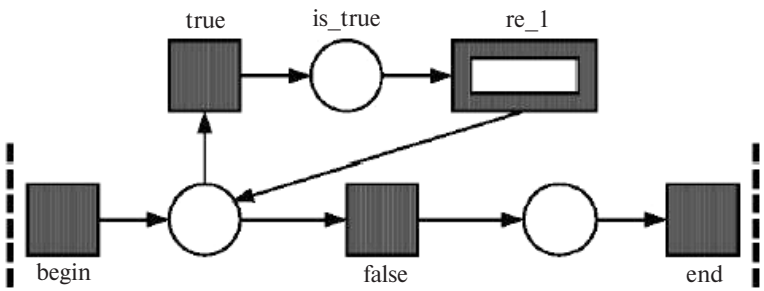

Рис. 12. Цикл while... do

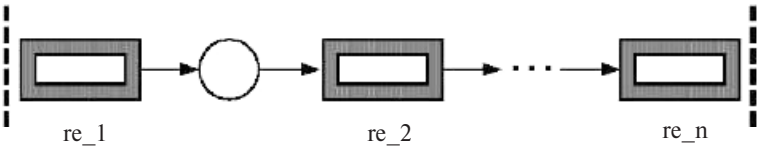

Рис. 13. Последовательность заданий

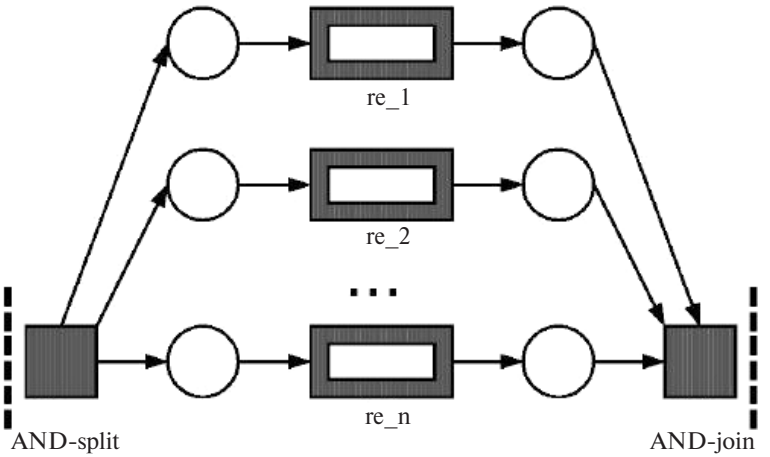

Рис. 14. Параллельный запуск с синхронизацией

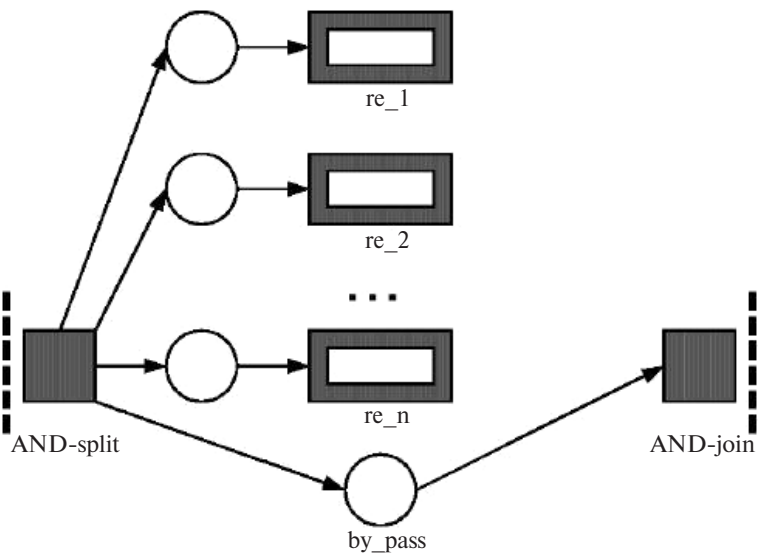

Рис. 15. Параллельный запуск без синхронизации

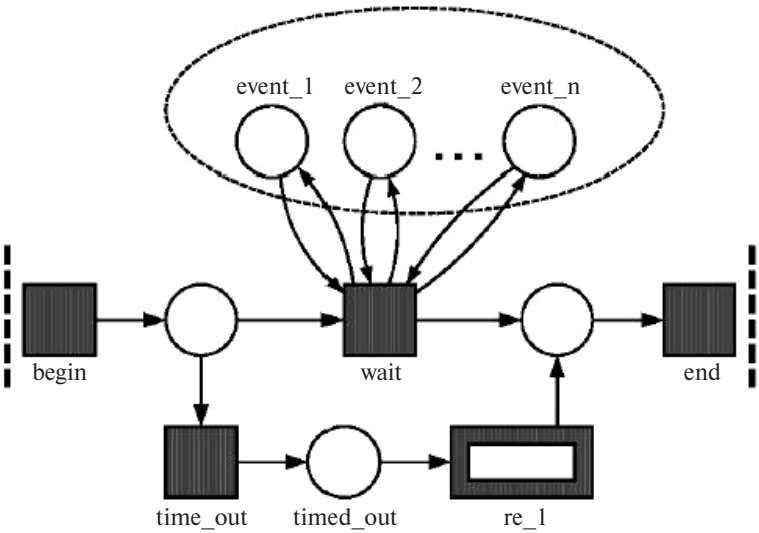

Рис. 16. Ожидание всех событий из списка, с ограничением по времени

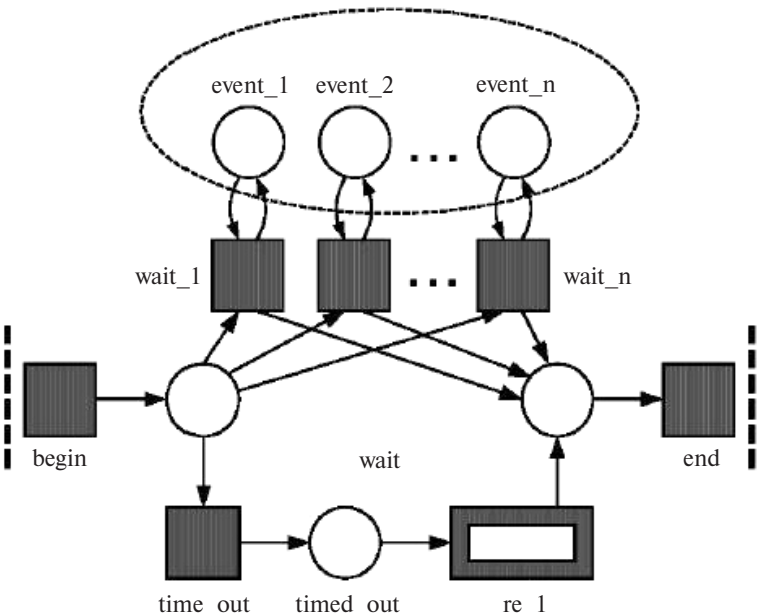

Рис. 17. Ожидание одного события из списка, с ограничением по времени

Grid Job Definition Language. Fraunhofer Resource Grid (FhRG, [19]) — это инициатива в области Grid-вычислений нескольких Фраунгоферовских институтов, которая финансируется Министерством образования и исследований Германии. Главная ее цель — разработать и воплотить в жизнь стабильную и надежную Grid-инфраструктуру в рамках научной организации Fraunhofer-Gesellshaft [20] для интеграции доступных ресурсов и предоставления внутренним и внешним пользователям простого в использовании интерфейса для управления распределенными приложениями и службами в среде Grid. Предполагается, что в будущем программные продукты, разработанные в рамках этой инициативы, будут распространяться в виде открытых исходных кодов под названием eXeGrid [21].

В отличие от других сценарных систем, основанных на ориентированных ациклических графах, для представления своих сценариев FhRG использует формализм сетей Петри. Компонентная среда FhRG состоит из слабосвязанных программных компонентов, причем каждый такой компонент представляет собой исполняемый файл, читающий файлы, поступающие к нему на вход и создающий выходные файлы. Исполнение такого программного компонента называется *атомарным заданием*. В будущем разработчики планируют включить вызов Grid-сервиса в число возможных атомарных заданий, чтобы сделать FhRG совместимым с архитектурой OGSA [22].

В рамках этой инициативы были разработаны два основных пользовательских приложения, ориентированных на научные вычислительные сценарии: *Grid Job Builder*, предоставляющий графический интерфейс для создания и компиляции XML-подобных документов на языке *GjobDL* (*Grid Job Definition Language*), в виде которых формулируются *Grid-задания* (сценарии), и *Grid Job Handler*, служащий для анализа таких документов и приводящий в действие сценарий, содержащийся в Grid-задании.

Разработчики определяют Grid-задание как Grid-приложение, использующее несколько ресурсов согласно определенному сценарию. Grid-задание может состоять из нескольких атомарных заданий как из неделимых компонентов. Документ GJobDL может либо быть сохранен в виде файла, либо непосредственно передан в Grid Job Handler для запуска сценария.

GJobDL-документ включает в себя описание основных ресурсов, необходимых для выполнения Grid-задания, на специальном языке *GresourceDL* (*Grid Resource Definition Language*). GresourceDL поддерживает описание программных ресурсов, аппаратных ресурсов и данных, а также позволяет задавать зависимости между различными видами ресурсов (например, некоторый программный компонент может выполняться только на машинах определенных типов, с определенными операционными системами и при наличии на этих машинах определенных библиотек).

Вторая часть GJobDL-документа — это описание сценария в виде сети Петри. Выполнение сценария в FhRG в большинстве случаев

```
<!-- data: d25 -->
<resource id="d25" type="data">
  <location>
    <resourceRef id="gridNode15" type="hardware"/>
    <directory>/home/fhrgdata</directory>
    <filename>d25.dat</filename>
  </location>
</resource>
...

<!-- workflow description -->
<job type="petriNet" id="concatenateIt">
  <place id="d25">
    <resourceRef id="d25" type="data"/>
    <initialMarking>
      <value type="boolean" op="eq">true</value>
    </initialMarking>
  </place>
  ...
  <place id="d25-27">
    <resourceRef id="d25-27" type="data"/>
  </place>
  <transition id="t_cat1">
    <resourceRef type="software" id="cat"/>
  </transition>
  <transition id="t_cat2">
    <resourceRef type="software" id="cat"/>
  </transition>
  <arc id="arc1" type="P2T">
    <placeRef id="d25"/>
    <transitionRef id="t_cat1">
      <inputRef id="input1" type="file"/>
    </transitionRef>
  </arc>
  ...
</job>
```

Рис. 18. Пример описания сценария на GjobDL

можно представить как движение потока данных, ведь решение о запуске программного компонента часто полностью зависит от наличия или отсутствия входных данных. Фишки в такой сети Петри изображают данные, которые движутся между программными компонентами или Grid-сервисами. Но в некоторых случаях с помощью потоков данных можно описать не все особенности сценариев, и поэтому в добавление к data-местам и data-переходам были добавлены контрольные места и переходы. Соответствующие им фишки содержат логическое состояние элементов сценария (например, «выполнен» или «не выполнен»). Кон-

трольные переходы соответствуют логическим условиям при выполнении сценария.

На данный момент существует несколько подходов к описанию сетей Петри с помощью XML-подобных языков. Широко распространен, в частности, язык Petri Net Markup Language (PNML), разработанный берлинским Университетом Гумбольдта. Разработчики FhRG создали собственный способ описания, сходный с PNML. Описание задания состоит из описания мест, переходов и дуг, составляющих сеть Петри, соответствующую Grid-заданию. Data-места и data-переходы связываются с описаниями ресурсов (программные компоненты, машины, базы данных). Контрольные переходы могут управлять потоком данных через них.

За запуск сценария отвечает Grid Job Handler, он анализирует Grid-задание в форме GjobDL и разбирает зависимости между Grid-ресурсами, описанные при помощи GResourceDL. Ядром Grid Job Handler последовательно выполняются следующие действия:

1. Проверка сети Петри на согласованность, на так называемые подвижность (*liveliness*), мертвые точки (*dead points*) и ловушки (*pits*).
2. Сбор данных обо всех активных переходах сети Петри.
3. Определение условий всех активных переходов.
4. Вызов операций, которые приводятся в действие активными переходами (передача исполняемых файлов, передача данных, разархивация и т. д.).
5. Если переход содержит ссылку на программный компонент, то формируется набор подходящих этому компоненту аппаратных ресурсов.
6. Запрос метапланировщика о том, какие аппаратные ресурсы из указанного набора лучше всего соответствуют политике планирования, чтобы выбрать те, на которых будет запускаться данный программный компонент.
7. Детализация сети Петри, если она необходима (автоматическое добавление дополнительных, более мелких заданий по передаче данных и исполняемых файлов, запуску ПО или обработке ошибок).
8. Запуск программных компонентов на удаленных машинах (или вызов соответствующих методов Grid-сервисов) при помощи средств Globus Toolkit.
9. Переход срабатывает, если соответствующее ему атомарное задание выполнено (done) или завершено неудачно (failed). В любом случае, Grid Job Handler перемещает фишки с входных мест этого перехода на выходные, дописывая в них данные о том, с каким результатом закончилось соответствующее атомарное задание.
10. Повторяет п. 2–9 до тех пор, пока в сети не остается активных переходов.

Действия, выполняемые ядром, универсальны для любого сценария, каким бы сложным ни было соответствующее задание.

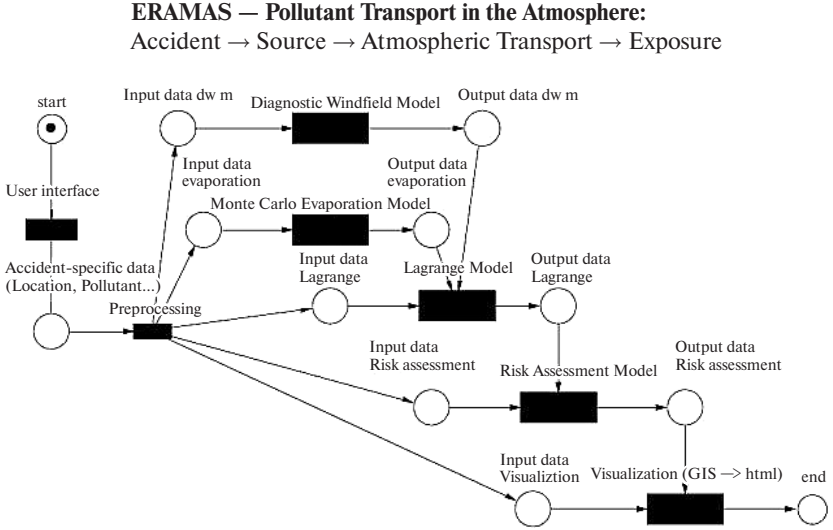

Рис. 19. Изображение в виде сети Петри сценария, моделирующего распространение загрязнений в атмосфере

На рис. 19 приведен пример научного вычислительного сценария, описанного в виде сети Петри и реализованного в рамках FhRG.

В заключение сформулируем достоинства и недостатки рассматриваемого подхода к представлению сценария и системы FhRG (eXeGrid).

Достоинства:

- Формализм цветных сетей Петри позволяет описывать большинство существующих научных вычислительных сценариев.
- Свойства сетей Петри хорошо изучены, есть множество методов их анализа.
- Система изначально предназначена для научных целей.

Недостатки:

- Пока не существует окончательной версии системы.
- Привязка к низкоуровневой модели работы в Grid-среде на базе Globus Toolkit (перенос файлов, запуск исполняемых файлов на удаленных машинах).
- Отсутствие поддержки вызовов Grid-(Web-)сервисов (планируется в будущем).

3.3. Смешанные подходы

Помимо систем, представляющих сценарии при помощи скриптовых языков и в виде графов существуют разработки, в которых сделана попытка синтеза двух этих подходов. В таких разработках в равной степени

присутствуют элементы и того, и другого подхода. Пример смешанного языка — BPEL, созданный на основе использующего графовое представление языка WSFL (IBM) и обладающего элементами языков структурного программирования языка XLANG (Microsoft). Поговорим об этих языках подробнее.

Все три языка опираются на архитектуру Web-сервисов, поэтому мы подробно раскроем здесь это понятие. Web-сервисы являются в настоящее время одним из наиболее активно развивающихся подходов к построению распределенных приложений. World Wide Web Consortium (W3C) [23] определяет Web-сервис как приложение:

- идентифицируемое при помощи URI;
- чьи интерфейсы и связывания с протоколами могут быть определены, описаны и найдены при помощи XML-артефактов;
- которое поддерживает прямое взаимодействие с другими приложениями посредством передачи XML-сообщений поверх стандартных интернет-протоколов.

Главной целью Web-сервисов является обеспечение интероперабельности. Согласно приведенному определению, запрашивающая сторона может получить доступ к сервису при помощи стандартных механизмов. В идеальном случае, любая запрашивающая сторона может взаимодействовать с любым приложением, являющимся Web-сервисом, безотносительно языка программирования и среды выполнения, используемыми каждой из сторон. Это свойство Web-сервисов делает данный подход привлекательным для интеграции программных ресурсов на межорганизационном уровне и построения крупномасштабных распределенных систем.

Одним из ключевых требований, предъявляемых к технологии Web-сервисов, является наличие механизма, позволяющего динамически создавать новые сервисы путем *композиции* уже существующих сервисов. Для того чтобы создать подобный механизм, требуется не только описать порядок, в котором выполняются эти сервисы и их методы, но и представить способ, при помощи которого подобный агломерат может быть представлен в виде сервиса.

Работы в области языков, предназначенных для описания сценариев взаимодействия набора Web-сервисов в виде новых Web-сервисов, активно велись на протяжении последних лет. Описанные далее языки, собственно и стали результатом этой деятельности.

3.3.1. Web Services Flow Language

Web Services Flow Language (WSFL) [22] — это XML-подобный язык описания композиций Web-сервисов, который был предложен фирмой IBM в мае 2001 г. WSFL рассматривает два типа композиций Web-сервисов:

- *Глобальная модель* (*global model*), в рамках которой оговаривается *модель взаимодействия* совокупности Web-сервисов. Данная модель содержит описание всевозможных *партнерских взаимодействий* между

Web-сервисами и служит формальным контрактом, которому должен следовать каждый из партнеров. В контексте Web-сервисов подобный вид композиции сервисов часто называют *хореографией* (*choreography*), поскольку он не подразумевает наличия центрального управляющего механизма.

- *Потоковая модель* (*flow model*), в рамках которой оговаривается *модель использования* совокупности Web-сервисов, для достижения определенной производственной цели. В результате, как правило, получается описание некоторого бизнес-процесса. Данный тип композиции в контексте Web-сервисов часто называют *оркестровкой* (*orchestration*), так как в данном случае подразумевается наличие центрального механизма, управляющего участвующими в процессе Web-сервисами и координирующего их взаимодействия. Этот тип композиции в WSFL соответствует нашему определению сценария. О нем и будем говорить в дальнейшем.

Во втором случае композиция Web-сервисов реализуется путем описания операций сложного сервиса-агрегата, которые должны выполняться группой объединенных Web-сервисов. Каждая операция сложного сервиса формулируется в терминах вызовов операций составных сервисов. Задаются последовательность данных вызовов, потоки данных и логические взаимосвязи между отдельными вызовами и участвующими сервисами.

Фактически новый сложный сервис, описанный при помощи WSFL, реализует некоторый сценарий взаимодействия набора Web-сервисов, направленный на достижение определенной цели. Поэтом в дальнейшем мы часто будем использовать термин «сценарий» вместо терминов «композиция» или «сложный сервис».

Таким образом, в WSFL композиция Web-сервисов может быть представлена в виде нового Web-сервиса, и, следовательно, выступать как компонент новых композиций. Это обеспечивает масштабируемость языка и возможность создавать сценарии как по принципу «от общего к частному», так и наоборот.

WSFL поддерживает различные виды взаимодействий между участниками сценария. Наиболее часто взаимодействие участников происходит при помощи сценария (как некоторого процесса, развернутого на сервере), который в данном случае выступает в роли посредника, направляющего данные от одного сервиса к другому. Кроме того, участники сценария могут взаимодействовать между собой напрямую, минуя процесс сценария. Однако, во втором случае, уже нельзя говорить о том, что ход выполнения сценария полностью контролируется центральным процессом.

Одна из главных целей создателей WSFL — это вписать свою разработку наиболее естественным образом в уже существующую архитектуру Web-сервисов. Поэтому WSFL использует язык описания Web-сервисов WSDL [25] (Web Services Description Language) для описания интерфейсов и связываний с протоколами. Также WSFL опирается на предложенный

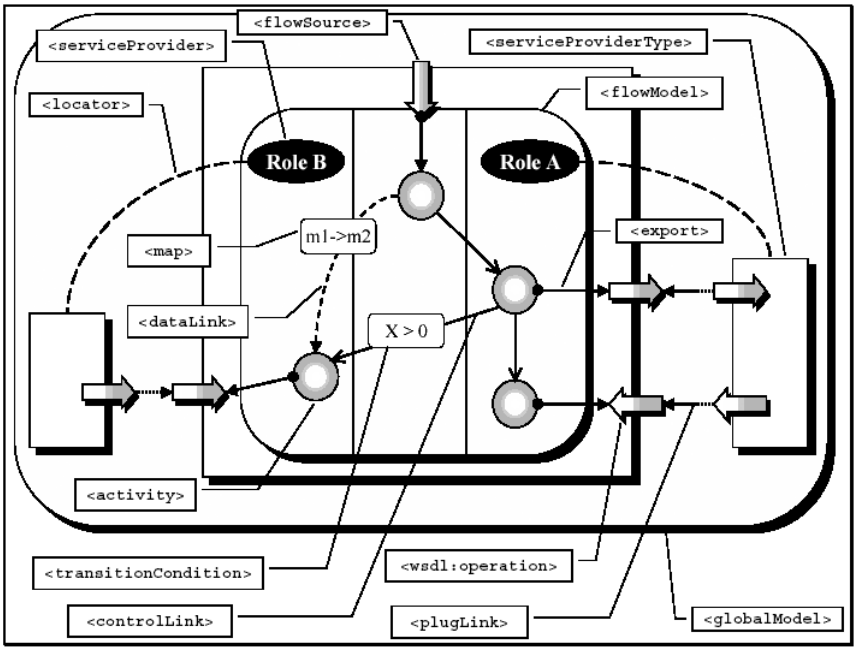

Рис. 20. Представление сценария взаимодействия набора Web-сервисов в WSFL

IBM язык Web Services Endpoint Language (WSEL) для описания дополнительных характеристик точек доступа к сервисам, например — параметров качества сервиса.

Сценарий в WSFL (см. рис. 20) задает структуру выполняемого процесса в виде ориентированного ациклического графа следующим образом: активности (вершины графа, обозначены кружками) описывают шаги процесса, а связи по управлению и связи по данным (*control links* и *data links*, два типа ребер графа) описывают правила следования этих шагов друг за другом и поток данных между активностями. Для каждой активности может быть определен «поставщик услуги» (*service provider*), который отвечает за выполнение данного шага процесса (к примеру, для бизнес-процесса, услуги могут предоставляться при помощи соответствующих Web-сервисов транспортной компанией A или компанией-поставщиком B). Поставщиком активности может быть непосредственно сам сценарий, например — в случае, если требуется обработка данных при передаче их от одного участника другому. Вид сценария в WSFL показан на рис. 20, причем вертикальные «дорожки» на рисунке показывают, каким поставщиком предоставляется соответствующая каждой активности услуга.

Еще одна характерная особенность WSFL — это так называемые *выходные условия* (*exit conditions*), с помощью которых успешно модели-

руются циклы, не возможные при описании сценария исключительно в виде ациклического ориентированного графа. Выходное условие — это булево (логическое) выражение, которое отражает, выполнено ли данной активностью (вершиной графа) задание, с которым она проассоциирована. Если значение этого выражения равно “true”, то активность считается выполненной и процесс выполняется далее, переходя к другим активностям. Если же значение выходного условия равно “false”, то активность выполняется заново. Таким же образом можно смоделировать цикл типа “loop until...” — пока значение выходного условия не станет равным “true”, активность, содержащая цикл будет выполняться снова и снова.

3.3.2. XLANG

Язык XLANG [26] — это расширение языка WSDL [25], предназначенное для описания композиций Web-сервисов. Он был предложен корпорацией Microsoft в мае 2001 г. По словам самих разработчиков, этот язык представляет собой «способ нотации для строгого задания обмена сообщениями между участвующими в сценарии Web-сервисами... Цель XLANG — сделать возможным формальное описание бизнес-процессов как долгосрочных взаимодействий набора Web-сервисов с переменными параметрами».

XLANG описывает поведение участвующих в сценарии Web-сервисов, причем XLANG-описание помещается непосредственно в текст WSDL-описания Web-сервиса.

В отличие от WSFL, язык XLANG использует для описания сценариев подход, основанный главным образом на языках структурного программирования. Перечислим характерные особенности данного языка:

- Последовательные и параллельные исполняемые конструкции.
- Долгосрочные транзакции с возможностью компенсации, т. е. действий в случае сбоя, при необходимости отмены части выполненной работы.
- Асинхронный обмен сообщениями.
- Поддержка исключительных ситуаций.
- Поддержка динамических ссылок на Web-сервисы.
- Поддержка сценариев, в которых Web-сервисам приписано несколько исполняемых ролей.
- Понятие контекста, области видимости (*scope*).
- Базовые действия: `<operation>`, `<delay>`, `<raise>`.
- Управление ходом сценария: `<sequence>`, `<switch>`, `<all>`, `<while>`, `<pick>`.
- При развитии языка были введены переменные для хранения состояния взаимодействия и действия по преобразованию данных.
- Описание контракта между участниками сценария путем установления взаимных соответствий между парами методов сервисов (рис. 21).

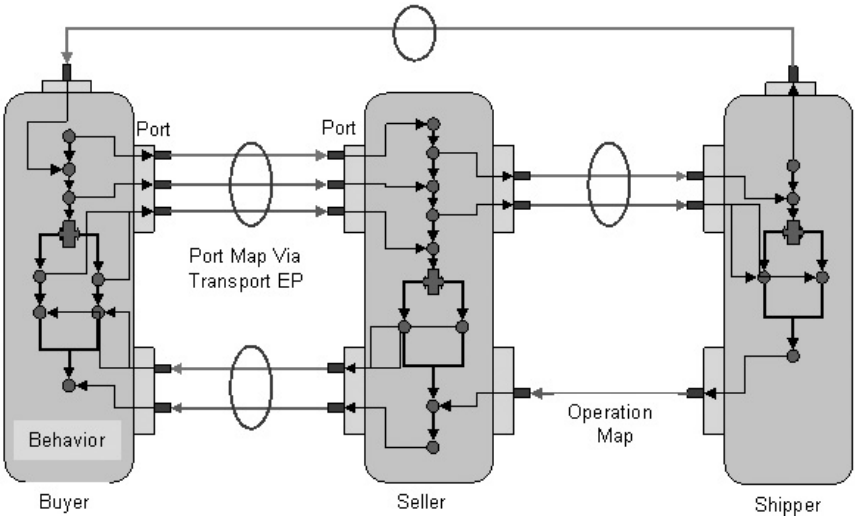

Рис. 21. Взаимодействие между участниками сценария в XLANG

Большинство из этих пунктов были позднее почти без изменения перенесены в BPEL, поэтому подробное их описание будет дано в следующей главе.

Система управления сценариями, поддерживающая язык XLANG, была включена Microsoft в продукт BizTalk Server. Кроме того, на основе Microsoft Visio было создано приложение BizTalk Orchestration Designer, позволявшее создавать сценарии при помощи графических средств Visio, а затем запускать их при помощи встроенного в BizTalk Server интерпретатора XLANG.

3.3.3. BPEL

Язык Business Process Execution Language for Web Services (BPEL4WS, BPEL) [28] возник в результате совместной работы IBM и Microsoft по слиянию их собственных разработок — языков Web Services Flow Language (WSFL) и XLANG. В августе 2002 г. ими была предложена версия 1.0 спецификации BPEL, а в апреле 2003 г. при участии ряда других компании была подготовлена версия 1.1 [27], проходящая процесс стандартизации в рамках консорциума OASIS. Выход стандарта BPEL ожидается в 2005 г.

BPEL представляет собой XML-язык описания (бизнес-)процессов в виде набора согласованных взаимодействий между несколькими Web-сервисами. BPEL позволяет создавать новые Web-сервисы, представляющие собой композиции Web-сервисов и реализующие тот или иной сценарий.

Сценарии, которые могут быть описаны при помощи BPEL, условно делятся на два типа: абстрактные и исполняемые сценарии. *Абстрактные сценарии* служат для описания ожидаемых протоколов взаимодействия и видимых внешнему наблюдателю поведений без перечисления всех деталей сценария (модель хореографии). *Исполняемые сценарии* содержат детальную спецификацию сценария, достаточную для его запуска (модель оркестровки).

BPEL опирается на традиционные способы моделирования сценариев в виде графов (см. язык WSFL), а также использует многие понятия из языков структурного программирования (см. язык XLANG). BPEL опирается на язык Web Service Description Language (WSDL) [25] для описания интерфейсов сервисов, передаваемых сообщений и их структуры, а также использует его в качестве контейнера для дополнительных метаданных. Для описания XML-выражений используется спецификация XPath 1.0 с некоторыми расширениями, а для описания синтаксиса и структуры элементов языка — спецификация XML Schema.

В соответствии с традиционными способами описания сценариев, сценарий в BPEL представляется в виде набора *активностей* (*activity*) или шагов сценария, соответствующих выполнению определенных действий (вызов приложения, действие пользователя и т. д.). Активности могут быть как элементарными, так и сложными, т. е. включающими в себя в качестве параметров другие активности. Примерами сложных активностей могут быть последовательное или параллельное выполнение, условные переходы, циклы и т. д. В качестве представления сценария может быть использована условная запись с перечислением всех активностей или ациклический граф, вершинами которого являются активности, а ориентированные ребра задают порядок выполнения активностей [29].

Потоки данных между активностями могут быть заданы явно при помощи отдельного типа ребер (data links). Данные ребра могут иметь атрибуты, описывающие передаваемые данные и необходимые преобразования.

XML-документ, содержащий описание сценария на BPEL, имеет следующую структуру (рис. 22):

- Описание типов участвующих в сценарии Web-сервисов, т. е. сервисов, которые вызываются из сценария и/или вызывают сам сценарий (элемент `<partners>`, аналог import-конструкций в языках структурного программирования).
- Описание переменных (данных), используемых далее сценарием (элемент `<variables>`, аналог определений переменных в языках структурного программирования).
- Описание наборов данных (correlation sets), используемых для реализации асинхронных взаимодействий (элемент `<correlationSets>`).
- Описание обработчиков ошибок (fault handlers), т. е. действий в случае возникновения той или иной ошибки (элемент `<faultHandlers>`,

```
<process ...>

   <partners> ... </partners>
         <!-- Web services the process interacts with -->
   <variables> ... </variables>
         <!– Data used by the process -->
   <correlationSets> ... </correlationSets>
         <!– Used to support asynchronous interactions -->
   <faultHandlers> ... </faultHandlers>
         <!–Alternate execution path to deal with faulty conditions -->
   <compensationHandlers> ... </compensationHandlers>
         <!–Code to execute when "undoing" an action -->
   <eventHandlers> ... </eventHandlers>
         <!–Code for handling events-->
   (activities)*
         <!– What the process actually does -->

</process>
```

Рис. 22. Структура описания сценария на BPEL

аналог обработчиков исключительных ситуаций в языках структурного программирования).

- Описание обработчиков компенсации (compensation handlers), т. е. действий в случае необходимости отмены результата выполнения набора активностей (элемент `<compensationHandlers>`).
- Описание обработчиков событий (элемент <eventHandlers>, аналог обработчиков событий в языках структурного программирования).
- Описание одной и более активностей, составляющих сценарий (набор элементов для различных видов активностей).

Типы связей между сценарием и участвующими Web-сервисами (партнерами) описываются при помощи специальных элементов `<service-LinkType>`, которые определяют роли взаимодействующих сторон и интерфейсы (portTypes), требующиеся для каждой из ролей (рис. 23). Каждый партнер однозначно определяется при помощи задания его имени, типа связи и его роли в ней.

Переменные в BPEL представлены в виде структурированных WSDL-сообщений. Все входные и выходные данные активностей хранятся в переменных. Обмен данными внутри сценария реализуется при помощи специального типа активности `<assignment>`, который переносит данные из одной переменной в другую.

Среди других основных типов активностей стоит отметить:

- `<invoke ...>` — вызов сценарием операции партнера;
- `<receive ...>` — получение сценарием вызова от партнера;
- `<reply ...>` — отправка сценарием ответа на вызов партнера;
- `<throw ...>` — вызов обработчика ошибок;

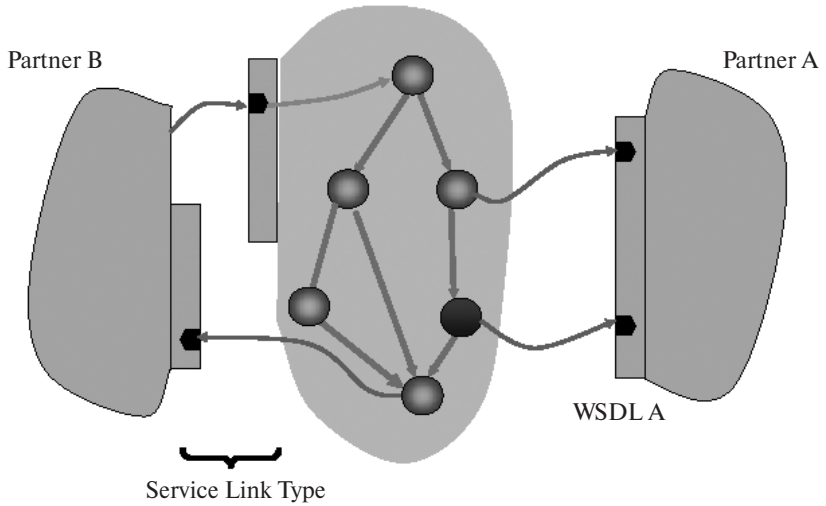

Рис. 23. Связи между сценарием и сервисами-партнерами

- `<terminate/>` — завершение выполнения сценария;
- `<wait ...>` — приостановка выполнения сценария на заданное время;
- `<empty/>` — «пустая» активность.

BPEL содержит следующие структурные (сложные) типы активностей:

- `<sequence>` — последовательное выполнение содержащихся внутри активностей;
- `<flow>` — параллельное выполнение содержащихся внутри активностей;
- `<while>` — цикл с условным переходом;
- `<pick>` — несколько активностей, связанных с определенными событиями, — отбирается первая соответствующая наступившему событию активность.

Специальный элемент `<link ...>` задает зависимость по времени выполнения между двумя активностями.

Из языков структурного программирования BPEL заимствует понятие области видимости. При помощи элемента `<scope>` можно выделить в отдельную область набор активностей. С каждой областью может быть связано три типа обработчиков:

- обработчики ошибок (для разных типов ошибок);
- обработчики компенсации (один на всю область);
- обработчики событий (для разных типов сообщений или сигналов).

Обработчик ошибок определяет альтернативные пути выполнения сценария в случае возникновения ошибки в рамках данной области действия. Для идентификации соответствующего обработчика ошибок используется традиционная стратегия передачи ошибки вверх по иерархии областей видимости.

Обработчик компенсации используется для отката действий, произведенных в рамках *уже выполненной* области. Он может быть вызван только из обработчика ошибок или обработчика компенсации области, непосредственно включающей данную. При запуске компенсирующего обработчика значения всех переменных устанавливаются в значения, бывшие у них сразу после выполнения данной области.

Обработчики событий могут быть настроены на запуск при получении процессом определенных сообщений или временных сигналов на протяжении всего времени, когда данная область действия активна (т. е. выполняется хотя бы одна из содержащихся в ней активностей). Дополнительным условием является отсутствие входящих и исходящих из основного сценария связей `<link>`, пересекающих границу обработчиков событий.

BPEL может использоваться для моделирования долговременных и асинхронных взаимодействий с поддержкой *состояния взаимодействия*. Для этого используются специальные наборы данных *correlation sets* (CS). Каждый CS включает данные, описывающие состояние конкретного взаимодействия в рамках одного «экземпляра» сценария (номера заказа, идентификатор пользователя и т. д.). При помощи анализа CS входящие сообщения могут быть переданы соответствующему экземпляру сценария. Каждый набор CS инициализируется один раз при прохождении области видимости, и его значения не изменяются в ходе взаимодействия.

Несмотря на то, что BPEL пока не является общепризнанным стандартом, уже существует несколько реализаций среды выполнения BPEL на Java. Наиболее известной является разработанный в IBM и свободно распространяемый пакет BPWS4J [30], включающий интерпретатор BPEL Engine, а также средство графического моделирования процессов BPEL Editor. Среди коммерческих разработок лидирует продукт BPEL Process Manager компании Oracle [31].

Поскольку современные Grid-технологии, такие как Globus Toolkit, для организации доступа к ресурсам используют технологии Web-сервисов, то естественным представляется вопрос об использовании BPEL для реализации вычислительных сценариев в рамках Grid. Данный вопрос рассматривается в [32], где делается вывод о том, что в исходном виде BPEL не удовлетворяет всем требованиям Grid-вычислений, однако предусмотренная в стандарте возможность расширения языка позволяет адаптировать его для вычислительных сценариев.

В заключение, рассмотрим достоинства и недостатки рассматриваемого подхода.

Достоинства:

- Удобство и выразительность языка, сочетающего в себе достоинства скриптового и графового описания сценариев.
- Существуют развитые графические среды моделирования процессов на BPEL.

Недостатки:

- Язык несколько непривычен даже для пользователей, знакомых с языками программирования, и требует времени для освоения.
- Привязанность к архитектуре Web-сервисов.
- Несовершенство существующих реализаций интерпретаторов языка, их неприспособленность для вычислительных задач (ориентация на бизнес-процессы).

3.4. Модели, ориентированные на потоки данных

Помимо уже упомянутых методов, существует еще один подход, в котором для представления сценариев могут использоваться как скриптовые языки, так и графы, но который несколько отличается от обсуждавшихся ранее подходов своей спецификой.

Речь идет о системах управления научными вычислительными сценариями, ориентированными на потоки данных. Как уже было сказано, в научных приложениях часто все необходимые действия сводятся к различным операциям над данными, т. е. в подобных процессах потоки управления и потоки данных совпадают. Поэтому нет необходимости вводить специальные элементы языка для описания логических конструкций, а достаточно просто обеспечить средства объединения элементарных *модулей обработки данных* в сеть. Каждый модуль имеет один или несколько входов и выходов, дуги сети соответствуют соединениям выхода одного модуля с входом другого, по которым осуществляется передача данных между модулями. Как только на вход модуля поступили все необходимые данные, происходит запуск программного кода модуля, который производит обработку входных данных, после чего полученные результаты (выходные данные) помещаются в выходы модуля и передаются по соединениям на вход других модулей.

Систем, обеспечивающих описанную функциональность, довольно много: Triana [32], Kepler [34], Discovery Net [35], SCIRun [36], Scitegic Pipeline Pilot [37], Taverna [38]. Но все эти системы довольно схожи как по своим возможностям, так и по принципам, использованным в них. Поэтому здесь мы обсудим подробно только первые две из них.

3.4.1. Triana

Разработчики (Department of Physics and Astronomy, Cardiff University) называют систему Triana «графической средой программирования, которая позволяет пользователям создавать сложные программы из элементарных модулей-алгоритмов (units) путем размещения их на рабочем окне и соединения друг с другом» [32].

Приложение написано на Java и изначально разрабатывалось учеными-астрофизиками для облегчения анализа больших объемов экспериментальных данных. Далее возможности Triana были расширены как с точки зрения поддерживаемых модулей (обработка сигналов, простые математические функции, статистика, обработка текстовых данных и т. д.), так и типов данных (численные данные, изображения, аудиоданные, текст). Triana является расширяемым приложением и содержит средства для создания новых модулей. Запуск программ (сценариев в нашей терминологии), созданных при помощи Triana, возможен прямо из графической среды (Triana GUI) или же из командной строки.

Triana поддерживает возможность распределенного выполнения сценария на нескольких узлах, каждый из которых выполняет определенную его часть (т. е. реализуется распределенный вычислительный сценарий). Кроме того, Triana поддерживает поиск и использование модулей, установленных на удаленных машинах. В случае необходимости, код модуля может быть передан средствами Triana на машину перед запуском сценария. В третьей версии Triana появилась поддержка поиска и использования Web-сервисов в качестве модулей при создании сценариев.

Система доступна бесплатно (на условиях Apache Software License) в виде исходного кода на сайте разработчиков [32].

Рассмотрим кратко архитектуру системы и используемый в ней способ представления сценариев. Архитектура Triana состоит из двух основных типов компонентов: Triana GUI и Triana Service.

Triana GUI. Компонент *Triana GUI* предоставляет простой в использовании графический интерфейс для построения сценариев в виде *сетей* (*networks*), состоящих из отдельных модулей и связей между ними.

Графический интерфейс Triana GUI (рис. 24) предоставляет пользователю средства для конструирования сценариев при помощи простых действий, соединения и группировки модулей, установки и изменения параметров модулей и соединений и т. д. В левой части окна выводится список всех доступных, в том числе и удаленно, модулей.

Каждый модуль имеет четко заданный набор входных и выходных *узлов* (*nodes*) с описанием получаемых и передаваемых параметров. Это позволяет сразу же при соединении модулей (двух узлов этих модулей) автоматически проверять новую связь на соответствие передаваемых и получаемых параметров. При этом некорректные соединения выделяются красным цветом.

Созданный сценарий может быть запущен на выполнение прямо из графической среды или же сохранен в XML-е (TaskGraph) для последующего открытия или запуска через командную строку. Triana GUI также позволяет пользователю указать машины, на которых следует провести выполнение всего сценария или его части.

TaskGraph. Для представления сценария в Triana используется XML-подобный формат, основанный на языке WSFL. Этот язык используется

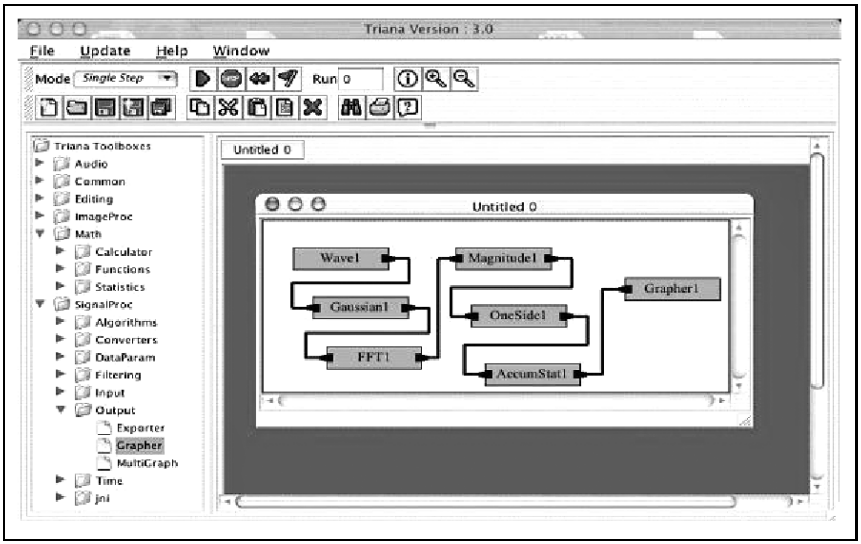

Рис. 24. Графический интерфейс пользователя Triana

для получения основанного на XML текстового описания сценария. В данном виде сценарий передается управляющему Triana-сервису (см. далее), который осуществляет запуск и управление сценарием.

Из-за того, что изначально в Triana не использовались технологии Web-сервисов, была проведена некоторая модификация WSFL с целью интеграции его с архитектурой Triana. Видимо, также сыграл свою роль тот факт, что WSFL не поддерживает особенности Grid-технологий. В планах разработчиков стоит задача интеграции Triana с современными Grid-технологиями в рамках европейского проекта GridLab [34]. Результатом этой работы должна стать возможность использования ресурсов Grid как модулей при создании сценариев.

В результате для представления графа сценария используется WSFL-подобный XML-формат. TaskGraph имеет три типа элементов: Task (подзадача, аналог активности в BPEL), Control Link (связь по управлению — связь между двумя подзадачами, определяющая последовательность их выполнения), Data Link (связь по данным — связь между двумя подзадачами, определяющая обмен данными между ними).

Triana GUI поддерживает экспорт/импорт задания в формате TaskGraph. В планы разработчиков входит добавление поддержки других форматов описания заданий, таких как BPEL.

Triana Service. После того, как в Triana GUI построена сеть, происходит генерация XML-представления соответствующего сценария в формате TaskGraph. Полученный TaskGraph передается *управляющему Triana-*

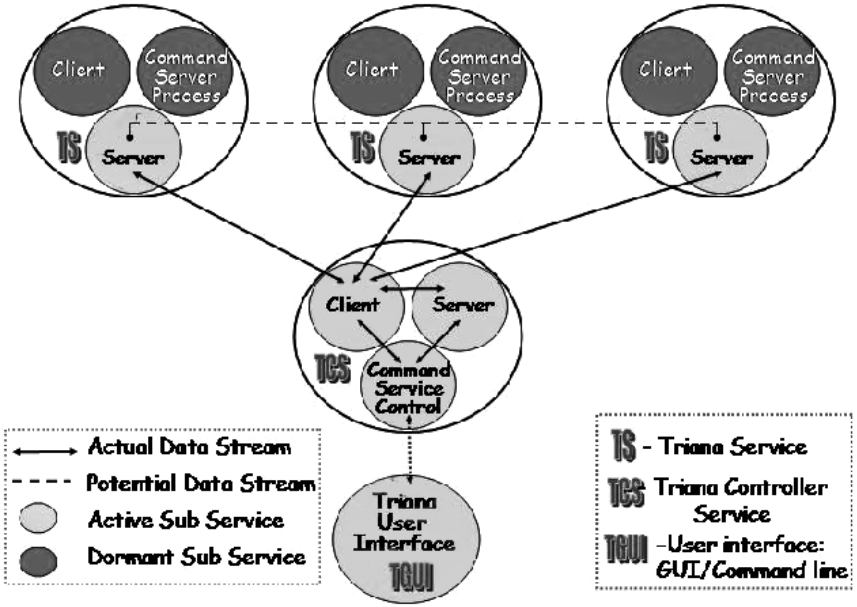

Рис. 25. Схема выполнения сценария в Triana

сервису (*Triana Controller Service*, *TCS*), который осуществляет выполнение сценария путем взаимодействия с другими Triana-сервисами (см. рис. 25).

Каждый *Triana-сервис* состоит из трех частей:

- Клиентская (Client).
- Серверная (Server).
- Управляющий сервер (Command process server).

В типичных условиях *управляющий сервер* активирован только на одном из Triana-сервисов, называемом управляющем Triana-сервисом. *Клиентский компонент* данного Triana-сервиса осуществляет передачу данных и программ *серверным компонентам* других сервисов, которые проводят выполнение полученных заданий и возвращают результаты управляющему сервису.

Triana-сервис отвечает за проверку на корректность входящих заданий (в виде TaskGraph-ов) и их выполнение. Дополнительно Triana-сервис может распределять части графа между сервисами, запущенными на других машинах. Полный список обязанностей Triana-сервиса:

- Интерпретация описания задания.
- Управление экземплярами подзадач.
- Осуществление навигации между подзадачами (последовательная, параллельная, условная и т. д.).

- Поддержка данных, связанных со сценарием.
- Осуществление аудита и журналирования.

Взаимодействие Triana-сервисов организовано в соответствии с пиринговой (peer-to-peer) моделью [40]. На рис. 25 видно, что каждый Triana-сервис содержит полный набор программных компонентов, позволяющих ему функционировать как в качестве клиента, запрашивающего ресурсы у других сервисов, так и в качестве сервера, предоставляющего свои ресурсы другим. При реализации распределенной архитектуры Triana была использована технология JXTA [41] компании Sun Microsystems, представляющая собой единую платформу и набор средств для разработки пиринговых приложений. Для регистрации и поиска Triana-сервисов используется механизм *объявлений* (*advertisements*), для организации взаимодействия и передачи данных между сервисами — механизм *каналов* (*pipes*).

В заключение рассмотрим достоинства и недостатки рассматриваемого подхода и системы Triana.

Достоинства:

- Система изначально предназначена для научных целей.
- Удобная графическая среда моделирования сценариев.
- Возможность запуска сценария на выполнение прямо из среды моделирования.
- Поддержка распределенного выполнения сценариев.
- Пиринговая модель взаимодействия сервисов.

Недостатки:

- Ориентация на описание потоков данных между участниками сценария затрудняет описание сценариев со сложными потоками управления.

3.4.2. Kepler

Система управления научными сценариями Kepler [34] была разработана для описания и выполнения различных сценариев в областях биологии, экологии, геологии, астрофизики и химии. В основе этой системы лежит созданная ранее СУС Ptolemy II [42], к которой разработчики Kepler добавили множество специфических функций, предназначенных для работы с научными сценариями.

Kepler представляет сценарий в виде набора актеров (*actors*, разнообразные средства, используемые при обработке данных), между которыми, от одного актера к другому, движутся потоки обрабатываемых данных. Актеры соединяются друг с другом, образуя ориентированный граф или сеть обработки данных (*data flow process network*). Существует большая библиотека стандартных актеров, равно как и возможность добавления собственных актеров, реализованных в виде Java-классов. Актеры могут вызывать удаленные ресурсы, в том числе и Web-сервисы. Каждый актер

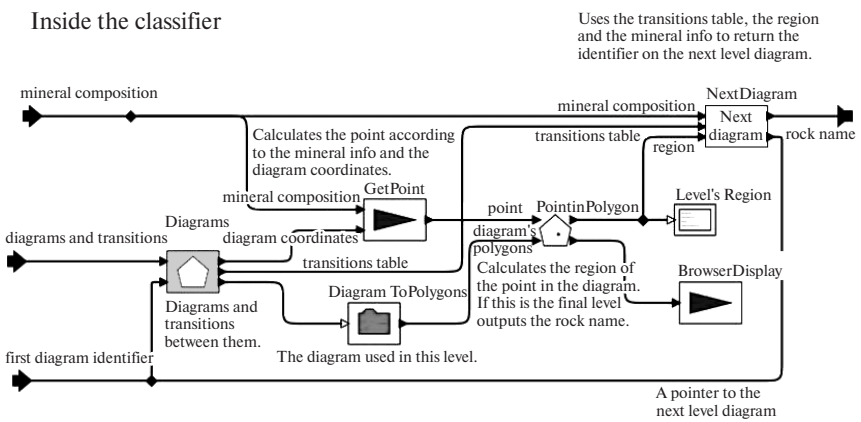

Рис. 26. Внешний вид сценария в Kepler

может иметь несколько входных и выходных портов, через которые движутся потоки данных. Также актеры могут обладать параметрами, которые определяют их индивидуальное поведение.

Аналогично системе Triana, в состав Kepler входит графическая среда (позаимствованная из Ptolemy), позволяющая располагать актеров-участников сценария друг за другом в нужной последовательности прямо в окне редактора (см. рис. 26), а затем и запускать полученный сценарий — среда выполнения сценариев в Kepler встроена в редактор. Созданные сценарии хранятся в виде документов, написанных на XML-подобном языке MoML (Modeling Markup Language), также унаследованном от Ptolemy.

Существующие стандартные актеры Kepler, предназначенные для работы с Web и Grid-сервисами, позволяют ученым использовать доступные удаленные ресурсы для создания собственных распределенных сценариев. Например, обеспечивается бесшовное взаимодействие с Web-сервисами. Существуют специализированные актеры для работы в среде Grid: для аутентификации посредством сертификатов (*ProxyInit*), для запуска задания на вычислительном ресурсе (*GlobusJob*) и доступа к удаленным данным (*DataAccessWizard*, *GridFTP*). Также в состав стандартных актеров входят актеры для работы с базами данных, для связи (возможно несовместимых) Web-сервисов между собой, для работы с Matlab и Python, множество актеров для обработки данных и т. д. Kepler предоставляет пользователям возможность объединения актеров в свои собственные библиотеки, для большего удобства их использования.

В отличие от Triana и других подобных систем, работающим только с одним потоком данных, Kepler поддерживает несколько таких потоков. Таким образом, в рамках одного сценария, через разные входные и выходные порты отдельных актеров одновременно могут проходить разные потоки данных.

В заключение рассмотрим достоинства и недостатки рассматриваемого подхода и системы Kepler.

Достоинства:

- Система изначально предназначена для научных целей.
- Удобная графическая среда моделирования сценариев.
- Поддержка нескольких одновременных потоков данных в рамках одного сценария, в отличие от аналогичных приложений.
- Бесшовное взаимодействие с Web-сервисами.
- Большая библиотека стандартных модулей обработки данных.

Недостатки:

- Ориентация на описание потоков данных между участниками сценария затрудняет описание сценариев со сложными потоками управления.

Заключение

Сделанный в данной статье обзор существующих подходов к представлению научных вычислительных сценариев позволяет сделать вывод о том, что workflow-методология может быть успешно применена не только в сфере бизнес-процессов, но и при организации сложных вычислений, охватывающих набор распределенных ресурсов. Безусловно, данный подход только начинает развиваться в рамках Grid-вычислений, поэтому существующие средства не всегда удобны в применении и обладают рядом недостатков. Тем не менее, можно говорить о том, что приход workflow-методологии в распределенные вычисления способен вывести их на качественно новый уровень так же, как это произошло при эволюции программных приложений.

В рамках работ по созданию системы IARnet [43, 44], реализующей универсальную программную инфраструктуру распределенной вычислительной среды, планируется использовать workflow-методологию для описания прикладных вычислительных сценариев и разработки службы, позволяющей осуществлять выполнение подобных сценариев.

Литература

1. WfMC Workflow Handbook. Future Strategies Inc, 2002.
2. Workflow management coalition (WfMC) (www.wfmc.org).
3. Xerox PARC (www.parc.xerox.com).
4. Production Workflows. Leyman, Roller. Prentice Hall PTR, 1999.
5. GridAnt (www-unix.globus.org/cog/projects/gridant).
6. Apache Ant (ant.apache.org).

7. *Афанасьев А. П.*, *Волошинов В. В.*, *Рогов С. В.*, *Сухорослов О. В.* Развитие концепции распределенных вычислительных сред // Проблемы вычислений в распределенной среде: организация вычислений в глобальных сетях. Сборник трудов ИСА РАН. М.: УРСС, 2004.
8. The Globus Project (www.globus.org).
9. Элементы теории графов, схем и автоматов / Алексеев В. Б., Ложкин С. А. М.: Изд. отдел ф-та ВМК МГУ, 2000. 58 с.
10. Condor Project (www.cs.wisc.edu/condor).
11. Symphony (zuni.cs.vt.edu/symphony).
12. Cactus (www.cactuscode.org).
13. UNICORE (unicore.sourceforge.net).
14. The Application of Petri Nets to Workflow Management. W.M.P. van der Aalst. The Journal of Circuits, Systems and Computers. 1998.
15. User Tools and Languages for Graph-Based Workflows. Andreas Hoheisel, Fraunhofer Institute for Computer Architecture and Software Technology (FIRST), 2004.
16. Экспериментальная версия транслятора сетей Петри в исполняемый код. Лаборатория «Суперкомпьютерных и распределенных вычислительных технологий». ИАПУ ДВО РАН. Ноябрь 2000 г.
17. *Котов В. Е.* Сети Петри. М.: Наука, 1984.
18. *Питерсон Дж.* Теория сетей Петри и моделирование систем. М.: Мир, 1984.
19. Fraunhofer Resource Grid homepage. 2004 (www.fhrg.fhg.de).
20. Fraunhofer-Gesellshaft (www.fraunhofer.de).
21. EXeGrid homepage. 2004 (www.exegrid.net).
22. *Foster I.*, *Kesselman C.*, *Nick J. and Tuecke S.* The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG. Global Grid Forum. June 22, 2002.
23. World Wide Web Consortium (W3C) (www.w3c.org).
24. WSFL specification. IBM, 2001.
25. WSDL specification (www.w3.org/TR/wsdl).
26. XLANG specification (www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm).
27. Business Process Execution Language for Web Services. Version 1.1 (www-106.ibm.com/developerworks/webservices/library/ws-bpel).
28. Business Process with BPEL4WS: Summary (www-106.ibm.com/developerworks/library/ws-bpelcol.html).
29. BPEL4WS (Business Process Execution Language for Web Services) by Francisco Curbera, Frank Leymann, Rania Khalaf. IBM, 2003.
30. BPWS4J (www.alphaworks.ibm.com/tech/bpws4j).
31. Oracle BPEL (www.oracle.com/appserver/bpel_home.html).
32. *Slomiski A.* On Using BPEL Extensibility to Implement OGSI and WSRF Grid Workflows. GGF10 Grid Work Flow Workshop, 2004.
33. Triana (www.triana.co.uk).
34. Kepler: An Extensible System for Scientific Workflows (kepler.ecoinformatics.org).
35. Discovery Net project (www.discovery-on-the.net).

36. SCIRun, Scientific Computing and Imaging Institute, University of Utah (software.sci.utah.edu/scirun.html).
37. Scitegic Pipeline Pilot (www.scitegic.com/products_services/pipeline_pilot.htm).
38. The Taverna Project (taverna.sourceforge.net).
39. GridLab: Grid Application Toolkit and Testbed (www.gridlab.org).
40. *Сухорослов О. В.* Пиринговые системы: концепция, архитектура и направления исследований // Проблемы вычислений в распределенной среде: прикладные задачи. Сборник трудов ИСА РАН. М.: УРСС, 2004.
41. JXTA Project (www.jxta.org).
42. Ptolemy II (ptolemy.eecs.berkeley.edu/ptolemyII).
43. *Афанасьев А. П.*, *Волошинов В. В.*, *Кривцов В. Е.*, *Рогов С. В.*, *Сухорослов О. В.* Использование информационно-алгоритмических ресурсов для организации распределенных вычислений // Проблемы вычислений в распределенной среде: организация вычислений в глобальных сетях. Сборник трудов ИСА РАН. М.: УРСС, 2004.
44. *Волошинов В. В.*, *Естехин О. С.*, *Сухорослов О. В.* Архитектура и принципы реализации системы IARnet (см. в данном сборнике).