Информационная инфраструктура Grid

О. В. Сухорослов

1. Проблема управления информацией в Grid

Концепция Grid, изначально возникшая в контексте довольно узкой проблемы объединения высокопроизводительных вычислительных ресурсов, в настоящее время трансформировалась во всеобъемлющий подход к организации совместного использования ресурсов и координированного решения задач в рамках так называемых виртуальных организаций [13].

Под *виртуальной организацией* (ВО) подразумевается динамическая совокупность географически распределенных организаций, индивидуальных пользователей и принадлежащих им ресурсов, сформированная на основе правил участия и соглашений по совместному использованию ресурсов. Важной особенностью является то, что каждый включенный в состав ВО ресурс по-прежнему управляется только его владельцем. Участники ВО могут использовать ее ресурсы в соответствии с установленными соглашениями и заданными владельцами политиками доступа. Новые организации, пользователи и ресурсы включаются в состав ВО в соответствии с правилами участия.

Конкретные примеры виртуальных организаций могут существенно отличаться по своим целям, сфере деятельности, масштабу, структуре и времени существования. Современные виртуальные организации формируются, как правило, на базе уже существующих консорциумов или сообществ, участники которых имеют общие цели и задачи, из которых собственно и возникает потребность в совместном использовании имеющихся в распоряжении ресурсов. Типичный пример — сообщество исследователей в области физики высоких энергий, активно использующее Grid для решения проблемы сбора, хранения и анализа огромных массивов экспериментальных данных. В то же время, подобные виртуальные организации зачастую носят открытый характер и приветствуют вступление в них новых участников со своими ресурсами.

Практика показывает, что виртуальные организации динамичны по всем своим параметрам: организациям, пользователям, ресурсам, отношениям разделения и политикам доступа. Динамичность состава участников обусловлена открытостью многих ВО и относительно простыми правилами участия. При этом к ВО могут присоединяться новые участники, а старые — выходить из ее состава. Набор ресурсов, доступных в рамках ВО, может изменяться по целому ряду причин, связанных с динамичностью состава участников, отсутствием централизованного управления ресурсами и отказами в работе глобальной сети. Отношения разделения между участниками ВО могут варьироваться во времени как по числу задействованных

ресурсов, так и по правилам доступа к ресурсам, устанавливаемым их владельцами. В общем случае сами виртуальные организации могут иметь спонтанный характер, т. е. создаваться и исчезать динамически внутри некоторой открытой среды.

Современные Grid-технологии направлены на реализацию распределенной программной инфраструктуры Grid, обеспечивающей функционирование описанных выше виртуальных организаций. Говорить о том, что подобная инфраструктура готова полностью, пока рано. В настоящее время ведется активная исследовательская работа и поэтапная разработка инфраструктуры Grid, начатая с низкоуровневых базовых сервисов и продолжаемая выше до уровня приложений и конечных пользователей, использующих ресурсы Grid для решения задач. На этом пути имеется ряд проблем, зачастую требующих разработки новых подходов к своему решению. В настоящей статье рассмотрена одна из таких проблем, связанная с управлением информацией в динамичной гетерогенной среде Grid.

Для эффективной работы в Grid его компоненты, приложения и пользователи должны иметь возможность динамически обнаруживать, определять характеристики и осуществлять мониторинг ресурсов, сервисов, вычислений и других сущностей, присутствующих в Grid. Динамический характер виртуальных организаций, большое количество автономных компонентов и источников информации, ограничения распределенной сетевой среды, использование различных, несогласованных представлений информации существенно затрудняет организацию унифицированного и оперативного доступа к совокупной информации о Grid. Традиционные централизованные решения плохо масштабируются в подобных условиях или просто не применимы.

О какой информации идет речь:

- Параметры Grid-инфраструктуры — текущая конфигурация и состояние базовых сервисов Grid, характеристики сетевых соединений.
- Состав виртуальной организации — находящиеся в ней ресурсы, участвующие организации, профили и группы пользователей, правила доступа к ресурсам.
- Характеристики отдельных ресурсов и сервисов, как относительно статические (описание функциональности, контактная информация, правила доступа), так и сильно динамические (текущий статус, уровень загруженности).
- Состояние выполняющихся в Grid вычислительных задач, задействованные ресурсы.
- Описание хранящихся в Grid наборов данных (структура, содержание, семантика, политики доступа), информация о местонахождении реплик.
- Историческая информация — журналы использования ресурсов, архивы данных мониторинга, записи о взаимодействиях между компонентами (журналы приложений).

Данная информация распределена между множеством гетерогенных источников информации, находящихся в Grid. Это отдельные ресурсы, сервисы, базы данных, файлы, пользователи и т. д.

Приведем примеры сценариев использования данной информации:

- Обнаружение доступных ресурсов на основе информации о составе ВО.
- Планирование выполнения задания, включая подбор подходящих ресурсов на основе их описаний и характеристик, а также лучших реплик данных на основе информации об их местонахождении и состоянии сетевых каналов.
- Адаптация к изменяющимся условиям, включая обнаружение отказов ресурсов и перераспределение вычислений по новым ресурсам.
- Мониторинг, диагностика и обнаружение аномального поведения компонентов Grid на основе информации о динамических характеристиках.
- Отладка и анализ производительности Grid-приложений на основе агрегированной исторической информации.

Несмотря на разнообразие информационных источников, их характеристик и способов их использования, наиболее перспективным представляется построение общей модели данных процессов и создание некоторого целостного подхода к их реализации на практике. Это позволит в дальнейшем произвольным образом сочетать различные сценарии, возникающие в реальных приложениях.

В максимально общем виде задача может быть сформулирована следующим образом [25]. В системе имеются *производители информации* (*producers*), имеющие доступ к источникам информации, и *потребители информации* (*consumers*), желающие получать информацию от производителей. В соответствии со спецификой Grid, производители и потребители информации географически распределены и функционируют автономно, а их состав подвержен постоянным изменениям. Требуются стандартные механизмы, позволяющие потребителям и производителям обнаруживать друг друга, а также организовывать передачу информации от производителей к потребителям на основе общепринятых подходов, таких как запросы (pull) или подписка (push). Данную задачу решают информационные сервисы Grid, рассматриваемые в п. 2.

Тот факт, что источники информации в Grid являются гетерогенными и используют различные модели представления информации, сильно затрудняет интеграцию данных, полученных из нескольких источников. Поэтому важную роль при построении информационной инфраструктуры Grid играет наличие стандартных общепризнанных моделей представления информации о тех или иных сущностях, таких как ресурсы, сервисы, задания, пользователи. Поддержка подобных информационных моделей производителями информации позволяет ее потребителям эффективным образом строить запросы и интегрировать полученную информацию. Информационные модели Grid рассмотрены в п. 3.

В перспективе такие процессы, как поиск и композиция ресурсов для решения задачи в Grid, должны быть максимально прозрачны для пользователя. Он должен сформулировать задачу и требования, предъявляемые к ресурсам, такие как надежность, стоимость, качество обслуживания, и т. п. После чего система должна автоматически, при минимальном участии со стороны пользователя осуществить поиск и отбор ресурсов, способных решить поставленную задачу и удовлетворяющих всем выдвинутым требованиям. В современном Grid пользователю приходится активно участвовать в этом процессе, самостоятельно отбирая подходящие ресурсы на основе предъявляемой ему системой информации. Ситуация осложняется использованием при описании ресурсов различных синтаксически несовместимых схем. Для автоматизации данных процессов необходимо, чтобы информация о семантике элементов описания ресурса и их значений содержалась непосредственно в описаниях ресурсов в форме, допускающей машинную обработку на основе логических выводов. Это позволит вывести на качественно новый уровень возможности по поиску и композиции ресурсов для решения задач в Grid. Данные вопросы рассмотрены в п. 4.

2. Информационные сервисы

Информационные сервисы Grid предназначены для регистрации, агрегации и предоставления унифицированного доступа к информации, связанной с Grid. За последние годы было создано довольно много систем, относящихся в той или иной мере к информационным сервисам Grid. Всех их можно условно разделить на три класса [51]:

- Информационные системы, ориентированные на работу с редко изменяющейся информацией, агрегирующие и кэширующие ее в едином месте для эффективного поиска.
- Системы мониторинга, предназначенные для работы с сильно динамичной и быстро теряющей свою актуальность информацией, где предъявляются особые требования к скорости доставки информации от источника к потребителю.
- Универсальные системы, сочетающие в себе возможности систем первых двух классов.

Для обоих классов систем характерны следующие требования:

- Независимость от конкретных платформ, так как узлы Grid гетерогенны.
- Масштабируемость относительно числа узлов и количества запросов.
- Устойчивость по отношению к отказам узлов и сетевых соединений.
- Отсутствие ограничений на типы используемой информации, возможность определения пользователями своих собственных типов данных.
- Защищенный доступ к информации, включая аутентификацию и авторизацию.

- Минимальное использование системных ресурсов, не мешающее проведению основных вычислений.
- Простой и легкий в использовании прикладной интерфейс.
- Поддержка сложных запросов, таких как «Найти машину с наименее загруженным процессором среди Unix-машин, имеющих как минимум 1 Gb оперативной памяти».
- Поддержка доступа к информации как по запросу (pull), так и по подписке (push).

Дополнительные требования, предъявляемые к информационным системам:

- Возможность кэширования информации вблизи клиентов, что позволяет сократить время обработки запросов, а также уменьшить загрузку сети и источников информации.
- Поддержка иерархической организации информации, например, на основе древовидной структуры связей между узлами системы, где каждый уровень агрегирует поступающую с низлежащих уровней информацию.

Дополнительные требования, предъявляемые к системам мониторинга:

- Низкое время отклика на запросы, сохраняющее актуальность переданной потребителю информации.
- Стабильность времени отклика на запросы, повышающая эффективность работы потребителей информации.
- Любая информация, передаваемая потребителю, должна быть снабжена временной меткой, указывающей на момент ее получения, и временем жизни, определяющим срок ее актуальности.
- Высокая частота измерений, проводимых для получения новой информации, не должна заметно влиять на производительность измеряемого узла.

В работе [44] проведены анализ и классификация существующих реализаций информационных сервисов Grid, поддерживающих мониторинг. Результаты анализа показывают, что, обладая во многом схожей функциональностью, современные системы не поддерживают интероперабельность между собой, в то время как масштабируемость предлагаемых решений не всегда является приемлемой для Grid. В настоящей статье мы ограничимся рассмотрением двух широко используемых в Grid-проектах систем — MDS и R-GMA, входящих в состав пакетов Globus Toolkit и gLite соответственно.

2.1. MDS

Система Monitoring and Discovery Service (MDS) [25] разрабатывается в рамках проекта Globus и входит в состав пакета Globus Toolkit [15]. Как следует из названия, MDS относится к классу универсальных систем.

Архитектура

Архитектура информационных сервисов Grid, описанная в [25] (рис. 1) и легшая в основу MDS, состоит из компонентов двух типов: *поставщиков информации* (*information providers*) и *агрегирующих сервисов* (*aggregate directory services*).

Поставщики информации предоставляют доступ к информации об отдельных сущностях Grid, полученной путем вызова локальных операций или опроса сторонних сервисов, таких как Network Weather Service [36]. Например, поставщик информации о вычислительном кластере может предоставлять информацию о количестве узлов, объеме оперативной памяти, версии операционной системы и загрузке системы. Архитектура не накладывает никаких ограничений на реализацию поставщиков информации. Единственным требованием является поддержка двух протоколов, описанных в архитектуре MDS.

Первый протокол, *Grid Information Protocol* (*GRIP*), предназначен для доступа к информации, предоставляемой поставщиком. При помощи GRIP клиенты могут отправлять поставщику запросы (модель pull) или подписываться (модель push) на получение определенной информации. Реализация протокола GRIP подразумевает выбор некоторого языка запросов, а также стандартного представления возвращаемой поставщиком информации.

Второй протокол, *Grid Registration Protocol* (*GRRP*), используется для уведомления агрегирующих сервисов о наличии поставщика информации. Данный протокол подразумевает временную регистрацию поставщика информации на агрегирующем сервисе, требующую периодического ее продления по инициативе поставщика. Это делается в первую очередь из соображений отказоустойчивости.

Агрегирующие сервисы используют информацию, полученную от поставщиков, для предоставления клиентам различных услуг по поиску информации и мониторингу. Архитектура не накладывает ограничений на детали реализации агрегирующих сервисов, такие как использование индексирования информации или кэширование результатов запросов. Поддержка протоколов GRIP и GRRP агрегирующим сервисом также не является обязательной, поэтому он может использовать собственные модели представления информации и языки запросов.

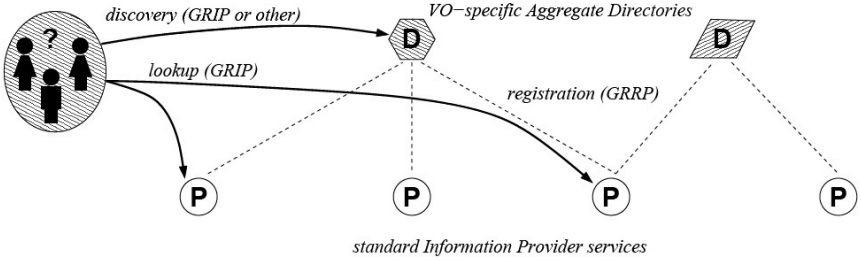

Рис. 1. Архитектура MDS

Предполагается, что агрегирующие сервисы будут функционировать в рамках определенной виртуальной организации. Например, агрегирующий сервис, используемый при планировании запуска вычислительных заданий, может поддерживать список доступных в виртуальной организации вычислительных узлов. В виртуальной организации может быть несколько однотипных или различных агрегирующих сервисов.

«Область покрытия» агрегирующего сервиса не обязательно должна охватывать всю виртуальную организацию. Например, он может предоставлять доступ только к информации о ресурсах некоторой организации или подразделения. Это позволяет организовывать агрегирующие сервисы в иерархическую структуру. Здесь архитектура также не предписывает какие-то конкретные способы реализации подобной иерархии, за исключением того, что для взаимодействия друг с другом агрегирующие сервисы могут использовать протоколы GRIP и GRRP.

Важно отметить, что в рамках описанной архитектуры потребители информации могут обращаться как к агрегирующим сервисам, так и напрямую к поставщикам информации.

MDS2

Реализация MDS2, входящая в состав Globus Toolkit 2, основана на распределенных каталогах, спецификации Lightweight Directory Access Protocol (LDAP) [53] и ее открытой реализации OpenLDAP. Для реализации протоколов GRIP и GRRP были адаптированы протоколы LDAP и формат данных LDIF.

В соответствии с моделью данных LDAP, информация в MDS2 представлена в виде набора объектов, организованных в древовидную структуру. Каждый объект помечен как относящийся к одному или нескольким именованным типам объектов и содержит значения именованных атрибутов, связанным с данными типами (рис. 2).

В рамках MDS2 были реализованы поставщики следующей информации об узле Grid: тип платформы, операционная система, загрузка, оперативная память, файловые системы, количество процессоров и т. д. Информация от поставщиков поступает в формате LDIF в *Grid Resource Information Service* (*GRIS*), конфигурируемый компонент MDS2, предназначенный для упрощения интеграции в систему произвольных локальных поставщиков информации и являющийся единой точкой входа для

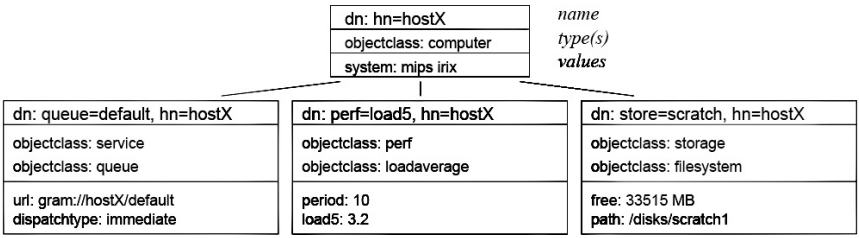

Рис. 2. Модель данных LDAP в MDS2

запросов клиентов. GRIS-сервера поддерживают основанные на LDAP реализации протоколов GRIP и GRRP. MDS2 также включает реализацию агрегирующего сервиса *Grid Index Information Service* (*GIIS*). GISS-сервера поддерживают протоколы GRIP и GRRP, что позволяет организовывать иерархии агрегирующих сервисов.

Будучи одной из первых систем подобного рода в Grid, MDS2 широко применялась и применяется до сих пор в ряде крупных Grid-проектов. За это время был выявлен ряд недостатков данной реализации, главные из которых связаны с отсутствием поддержки передачи данных по подписке (модель push) и использованием стратегии отложенной инициализации при кэшировании данных на агрегирующих серверах. В случае если искомые данные отсутствуют в кэше агрегирующего сервиса, происходят вызовы низлежащих сервисов и поставщиков информации (модель pull), после чего данные помещаются в кэш. В работе [56] было показано, что производительность MDS2 существенно зависит от параметров кэширования данных на агрегирующих сервисах. Отключенное кэширование или слишком малое время жизни кэша, приводит к тому, что запросы к агрегирующему сервису порождают множество запросов, идущих вниз по дереву серверов к поставщикам информации. Что в свою очередь приводит к резкому увеличению времени обработки запросов, повышенному сетевому трафику, и, в конечном счете, — плохой масштабируемости системы. Увеличение времени кэширования данных позволяет исправить ситуацию и снизить накладные расходы. Также рекомендуется размещать информационные сервисы на выделенных узлах, связанных между собой надежной высокоскоростной сетью. При выполнении этих рекомендаций MDS2 может эффективно использоваться для агрегации статических данных.

Описанной проблемы можно было бы избежать, если бы MDS2 поддерживала модель подписки. В этом случае агрегирующие сервисы могли бы подписываться на уведомления об изменении данных на низлежащих сервисах и поставщиках информации. По тем же причинам, несмотря на присутствие слова “monitoring” в названии, MDS2 плохо подходит для мониторинга состояния ресурсов Grid. Использование модели запросов может приводить к образованию узких мест между клиентами и информационными серверами. Для отслеживания динамических характеристик ресурсов более эффективной также является модель подписки.

Другой ряд недостатков MDS2 связан с использованием спецификации LDAP, а именно:

- Иерархическая модель данных не описывает все возможные связи между объектами Grid, и поэтому не является идеальной моделью для описания Grid.
- Структура информационного дерева должна быть предварительно согласована со всеми участниками виртуальной организации, поскольку ее модификация при наличии уже существующих данных достаточно сложна. Традиционные LDAP-каталоги управляются централизованно и не позволяют пользователям создавать и публиковать свои типы данных. Это снижает гибкость системы.

- Иерархические структуры наиболее приспособлены к случаям, когда заранее известны все возможные типы запросов, что позволяет оптимизировать структуру базы данных для быстрой обработки запросов. Запросы к информационным сервисам в Grid не известны заранее и могут существенно варьироваться по своей структуре и сложности.
- Спецификация LDAP не поддерживает способ получения информации по подписке (модель push), и поэтому такая возможность отсутствует в MDS2.
- Также в LDAP не предусмотрены запросы, комбинирующие информацию из объектов различных типов (аналог конструкции JOIN в SQL).

MDS3

Реализация MDS3 появилась в составе пакета Globus Toolkit 3 (GT3), являющегося первой реализацией архитектуры Open Grid Services Architecture (OGSA) [19]. MDS претерпела существенную переработку, в ходе которой произошел отказ от LDAP и переход на технологии Web-сервисов, XML-представление данных и язык запросов XPath.

В рамках OGSA все компоненты Grid представлены в виде Grid-сервисов, являющихся специальным расширением Web-сервисов с поддержкой внутреннего состояния и управляемым временем жизни. Каждый Grid-сервис имеет связанный с ним набор данных, так называемых *Service Data Elements* (*SDE*), представленных в стандартном XML-формате. В OGSA определяются стандартные механизмы, позволяющие клиентам запрашивать у сервиса значения SDE или подписываться на уведомления об изменении их значений. В соответствии с этим, к MDS3 разработчики относят любые части GT3, которые генерируют или каким-либо образом оперируют SDE (индексируют, агрегируют, отслеживают изменения, запрашивают или отображают).

Главным компонентом MDS3 является *Index-сервис* (*Index Service*) с функциональностью аналогичной GIIS из MDS2, но «обернутой» в интерфейс Grid-сервиса. Index-сервис предоставляет следующие возможности:

- Стандартный механизм для динамической генерации SDE при помощи внешних программ-поставщиков (т. н. Service Data Providers) и «вставки» их в экземпляры Grid-сервисов.
- Агрегация и индексация SDE, сгенерированных локальными поставщиками или полученных от других Grid-сервисов. Агрегированные данные представлены в виде SDE самого Index-сервиса, что позволяет запрашивать их значения (pull) или подписываться на их изменения (push) при помощи стандартных механизмов OGSA. Поддерживаются как простые запросы по имени SDE, так и более сложные в виде XPath-выражений.
- Создание *реестра* (*Registry*) для некоторой группы Grid-сервисов (Service Group), в котором регистрируются, с периодическим обновлением своей регистрации, доступные на данный момент сервисы.
- Три предыдущие возможности позволяют создать иерархию или федерацию Index-сервисов наподобие GIIS в MDS2, где каждый сервис

может генерировать информацию с помощью поставщиков, агрегировать информацию, полученную от других сервисов, и отслеживать группу сервисов, составляющих федерацию.

Роль GRIS в MSD3 отводится любым Grid-сервисам, публикующим при помощи SDE информацию о связанных с ними ресурсах. Поставщики информации из MDS2 были адаптированы для динамической генерации XML-данных, становящихся затем значениями SDE-элементов связанного с поставщиком Grid-сервиса. В MDS3 предусмотрено четыре типа поставщиков (Service Data Providers):

- *SimpleSystemInformationProvider* — реализованный на Java поставщик информации о физическом узле, такой как число процессоров, статистика использования памяти, тип операционной системы, дисковые ресурсы.
- *HostScriptProvider* — набор shell-скриптов для Unix-систем, выдающих детальную информацию о физическом узле. Фактически, это скрипты из MDS2, переработанные для выдачи информации в формате XML вместо LDIF.
- *AsyncDocumentProvider* — поставщик, который периодически считывает информацию из XML-документа на диске. Этот поставщик может использоваться в тех случаях, когда компонент, непосредственно генерирующий информацию, не имеет программного интерфейса.
- *ScriptExecutionProvider* — поставщик, получающий информацию при помощи запуска другой программы (обычно это shell-скрипт), которая возвращает данные в XML-формате в стандартный поток вывода.

Пользователи могут создавать своих собственных поставщиков информации на основе перечисленных базовых типов.

MDS4

Последняя на данный момент, четвертая версия MDS входит в состав Globus Toolkit 4 (GT4). Основная функциональность осталась той же, что и в MDS3. Большинство изменений носят концептуальный характер, связанный с переходом от спецификации OGSI в. GT3 к спецификации WS-Resource Framework (WSRF) [50] в GT4. Если в OGSI сервисы предоставляют доступ к SDE, то в WSRF — к *ресурсным свойствам* (*resource properties*, *RP*). По смыслу это абсолютно одинаковые понятия — оба содержат произвольную информацию в XML-формате, связанную с сервисом и доступную по запросу или подписке.

Index-сервис MSD4 функционально аналогичен Index-сервису MDS3 за исключением того, что запросы информации, подписка и регистрация групп сервисов реализованы в соответствии со спецификациями WSRF. Сервисы, поставляющие информацию для Index-сервиса, регистрируются путем вызова специальной команды, добавляющей новую запись в группу агрегируемых сервисов. Каждая подобная запись содержит значение таймаута для полученной информации и параметры механизма, используемого для получения информации. Сервис также может быть сконфигурирован

для автоматической регистрации на локальном Index-сервисе, запущенном в том же контейнере. Сбор информации может осуществляться путем запросов или подписки на значения RP, а также при помощи вызова внешних программ-поставщиков, аналогичных описанным в MDS2 и MDS3. Агрегированная информация публикуется как RP Index-сервиса.

В MDS4 появился дополнительный сервис *Trigger Service*, который помимо агрегации информации производит проверку ее на выполнение определенных условий, указанных в конфигурационном файле. При выполнении условий происходит срабатывание некоторого настраиваемого действия, например, отправка электронного письма администратору с уведомлением о том, что загрузка сервера превысила критическое значение и т. п.

Также в MDS4 была выделена в отдельный компонент общая программная инфраструктура для создания агрегирующих сервисов — *Aggregator Framework*, на базе которой функционируют Index-сервис и Trigger-сервис. Aggregator Framework включает в себя:

- Определение стандартного интерфейса источника информации *Aggregator Source*, используемого агрегирующими сервисами для сбора XML-данных, и несколько его реализаций.
- Конфигурационный механизм для управления информацией о зарегистрированных источниках и их параметрах.
- WSDL-описание записи в группе агрегирующего сервиса (aggregating service group entry), включающей конфигурационную информацию и данные.
- Стандартный механизм регистрации новых записей на агрегирующем сервисе.
- Поддержку времени жизни регистрации, по истечении которого происходит удаление связанных с ней данных.

Aggregator Framework содержит три типа источников информации:

- *Query Aggregator Source*, который получает информацию путем опроса значений RP некоторого сервиса.
- *Subscription Aggregator Source*, который получает информацию путем подписки на уведомления от некоторого сервиса.
- *Execution Aggregator Source*, который получает информацию путем запуска некоторой программы.

В качестве поставщиков информации MDS4 использует следующие компоненты:

- *Hawkeye Information Provider*. Получает информацию о виртуальном пуле ресурсов Condor от системы мониторинга Hawkeye и передает ее в XML-формате схемы GLUE (см. п. 3) сервису WS GRAM (Grid Resource Allocation Manager), который публикует ее в виде своих RP.
- *Ganglia Information Provider*. Получает информацию о вычислительном кластере от системы мониторинга Ganglia и передает ее в XML-формате схемы GLUE сервису WS GRAM, который публикует ее в виде своих RP.

- *WS GRAM*. Сервис GT4, отвечающий за запуск заданий на вычислительных ресурсах. Публикует информацию, полученную от локальной системы управления заданиями, такую как состояние очереди заданий, число свободных процессоров, статистика использования памяти и т. п.
- *Reliable File Transfer Service* (*RFT*). Сервис GT4, отвечающий за передачу файлов. Публикует информацию о числе активных передач, состоянии процессов передачи и т. п.
- *Community Authorization Service* (*CAS*). Сервис GT4, отвечающий за хранение информации о правах доступа пользователей к ресурсам виртуальной организации. Публикует в MDS информацию о виртуальной организации.
- *Любой другой сервис*, который публикует информацию в виде RP. По умолчанию каждый сервис GT4 публикует элемент *ServiceMetaDataInfo*, содержащий время запуска, версию и тип сервиса.

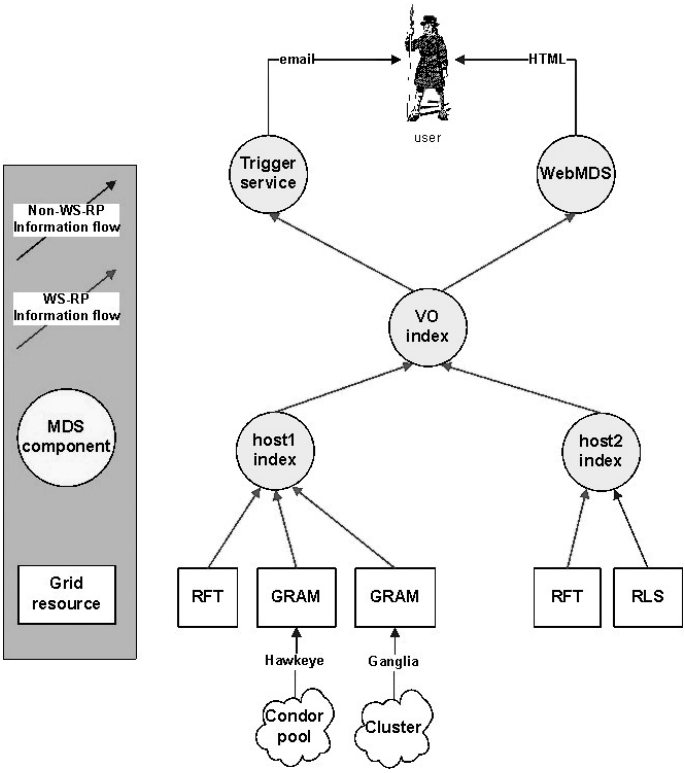

Рис. 3. Схема развертывания компонентов MDS4 в виртуальной организации

В состав MDS4 также входит Web-интерфейс к Index-сервису, *Web-MDS*, который помощи стандартных запросов получает от Index-сервиса опубликованные RP и отображает содержащуюся информацию в браузере.

На рис. 3 приведен пример схемы развертывания компонентов MDS4 на уровне виртуальной организации.

В [22] приведены результаты экспериментов по измерению производительности Index-сервиса MSD4. Для сравнения аналогичные эксперименты были проведены также с MDS2. Эксперименты проводились только над данными, находящимися в кэше Index-сервиса, без учета эффектов, связанных с обновлением данных. Измерения времени обработки запросов и пропускной способности агрегирующего сервиса показали, что MDS4 уступает по своей производительности MDS2. Это объясняется тем, что MDS2 реализована на языке C и использует «легкие» сообщения OpenLDAP, в то время как MDS4 реализована на Java и использует «тяжелый» протокол SOAP/HTTP. Тем не менее, результаты были признаны приемлемыми, учитывая множество других достоинств MDS4 в сравнении с MDS2. Для детального анализа производительности MDS4 требуется проведение дальнейших экспериментов, охватывающих не только Index-сервисы, но и поставщиков информации.

2.2. R-GMA

Система *R-GMA* [3, 41] является другим примером универсальной системы, поддерживающей как агрегацию статичной информацией, так и мониторинг динамических характеристик ресурсов Grid. R-GMA была создана в рамках проекта European DataGrid (EDG). В настоящее время работа над R-GMA продолжается в рамках другого европейского проекта Enabling Grids and e-Science in Europe (EGEE) [12]. R-GMA является информационным компонентом пакета gLite [14], разрабатываемого для EGEE и являющегося стандартным ПО для крупных европейских Grid-проектов. Фактически R-GMA заменил в подобных проектах использовавшуюся до этого реализацию MDS2 на основе LDAP.

R-GMA (relational-GMA) является реализацией архитектуры *Grid Monitoring Architecture* (*GMA*) [4] на основе реляционной модели данных. Архитектура GMA была предложена консорциумом Global Grid Forum (GGF) как некая общая модель информационной инфраструктуры Grid, ориентированная, как следует из названия, в первую очередь на мониторинг. GMA включает три типа компонентов (рис. 4): *потребители* (*consumers*) и *производители* (*producers*) информации, а также *каталог* (*directory service*). Производители регистрируют себя в каталоге, передавая свой адрес и описание генерируемой ими информации. Потребители могут опрашивать каталог для того, чтобы найти производителей интересующей их информации. После того как потребитель с помощью каталога определил местоположение производителя, он может напрямую обратиться к нему для получения требуемых данных. Протокол взаимодействия потребителей и производителей может поддерживать как получение данных по запросу (pull), так и подписку на получение новых данных (push) в потоковом

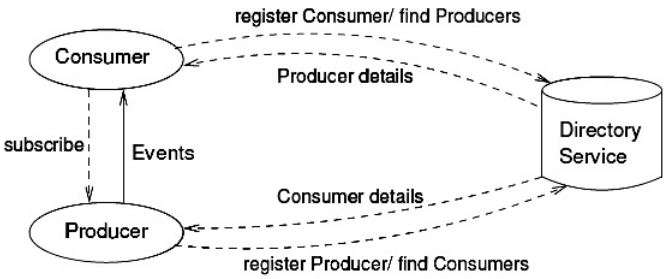

Рис. 4. Архитектура Grid Monitoring Architecture

режиме. В GMA также предусмотрена регистрация в каталоге потребителей информации, позволяющая уведомлять их об изменениях в составе производителей интересующей их информации. Важной особенностью архитектуры GMA является то, что каталог не занимается агрегацией и хранением информации, и передача основных данных происходит напрямую между производителем и потребителем. Кроме того, GMA не накладывает ограничений на используемые при ее реализации протоколы и модели данных.

Несмотря на то, что GMA создавалась для мониторинга, гибкость архитектуры позволила разработчикам R-GMA использовать ее для создания универсальной информационной инфраструктуры. Все публикуемые в R-GMA данные снабжаются временной меткой, указывающей на момент получения данной информации, что позволяет при необходимости использовать их для мониторинга. В то же время эти данные могут использоваться и в рамках информационной системы.

В качестве модели данных при реализации R-GMA была взята реляционная модель, обладающая достаточной мощью и гибкостью при описании структур данных и запросов. По утверждению авторов, R-GMA создает у пользователей ощущение работы с одной большой реляционной СУБД, обслуживающей виртуальную организацию. При этом подчеркивается, что речь идет только о применении реляционной модели в Grid, а не о создании настоящей распределенной СУБД, поскольку производители информации функционируют независимо друг от друга. «Реляционность» системы заключается в применении языка SQL при регистрации производителей (запрос вида SQL INSERT), описании структуры производимых ими данных (запрос вида SQL CREATE TABLE) и запросах информации потребителями (запрос вида SQL SELECT).

Виртуальная база данных R-GMA состоит из следующих компонентов:

- *Репозиторий схем* (*Schema*), хранящий определения таблиц.
- *Реестр* (*Registry*), хранящий список производителей (аналог каталога в GMA).
- *Посредник* (*Mediator*), реализующий подбор производителей для обработки запросов потребителей.

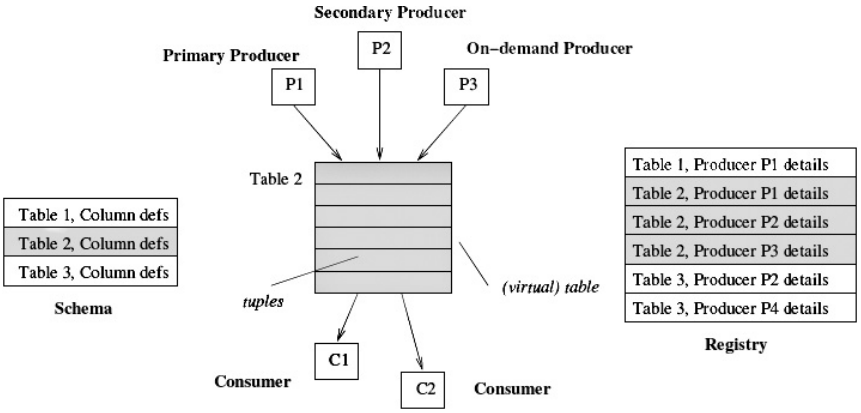

Рис. 5. Организация виртуальной базы данных R-GMA

Производители в R-GMA публикуют данные в виртуальную базу данных путем регистрации в реестре. Для этого используется запрос вида SQL INSERT, в котором указывается имя таблицы, кортежи из которой публикует производитель, и описание производителя. Производитель может ограничить значения публикуемых им кортежей путем указания предиката вида "WHERE ...". В случае если таблица отсутствует в базе данных, необходимо предварительно определить ее структуру путем отправки запроса вида SQL CREATE TABLE репозиторию схем, после чего производители могут публиковать кортежи из данной таблицы. Таким образом, содержимое таблиц виртуальной базы данных формируется из кортежей, публикуемых различными производителями (рис. 5). При этом сами данные не пересылаются от производителей в реестр и не агрегируются в централизованном хранилище.

Потребители в R-GMA позволяют клиентским приложениям выполнять запросы вида SQL SELECT на виртуальной базе данных (рис. 6). Потребитель получает от посредника, который является частью реестра, список производителей, требуемых для выполнения запроса. После чего потребитель рассылает запрос всем производителям из списка, собирает полученные кортежи и сохраняет их во внутреннем хранилище для последующего их извлечения клиентом. Все детали взаимодействия потребителя с производителями и агрегации распределенных данных скрыты от клиента, который работает с системой как с единой базой данных (рис. 7).

Каждый производитель (кроме упомянутых далее производителей по требованию) имеет внутреннее хранилище данных, содержащее публикуемые им таблицы. Таблицы создаются динамически по запросу клиентских приложений, на основе определений таблиц, получаемых от репозитория схем. Данные в хранилище производителя имеют определенный срок хранения, по истечении которого они автоматически удаляются. В зависимости от того, откуда поступают публикуемые производителем данные, в R-GMA различают три типа производителей:

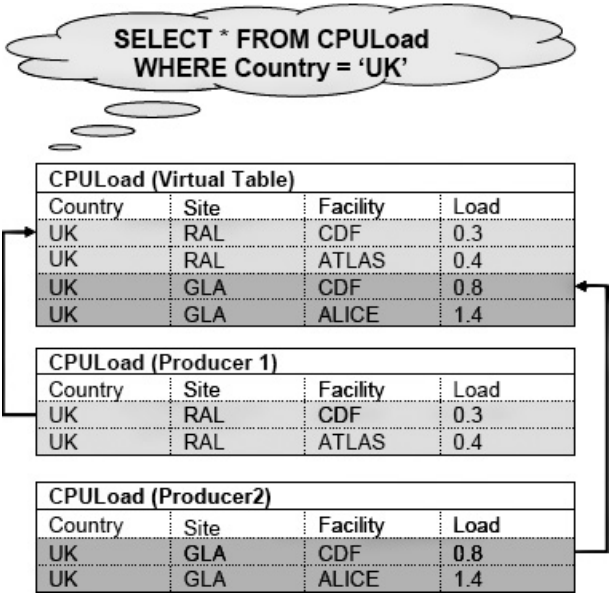

Рис. 6. Агрегация распределенных данных в таблицы виртуальной базы данных R-GMA

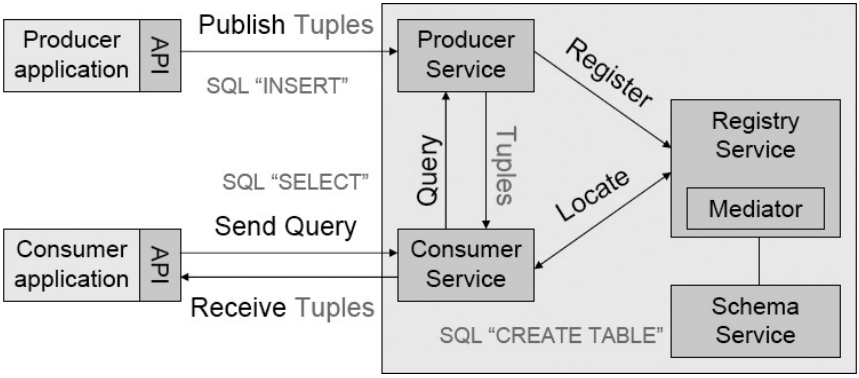

Рис. 7. Архитектура R-GMA

- *Первичный производитель* (*Primary Producer*): код пользователя напрямую добавляет кортежи в хранилище производителя.
- *Вторичный производитель* (*Secondary Producer*): производитель, который агрегирует и повторно публикует кортежи, опубликованные другими производителями. Подобные производители могут использоваться для репликации первичных производителей, архивации данных

от нескольких производителей или для обработки запросов, требующих объединения таблиц нескольких производителей (SQL JOIN).

- *Производитель по требованию* (*OnDemand Producer*): производитель, не имеющий внутреннего хранилища кортежей и пересылающий приходящие ему запросы внешнему приложению, которое возвращает требуемые кортежи.

В R-GMA реализовано четыре типа запросов от потребителей к производителям:

- *Непрерывный мониторинг* (*Continuous*): Все кортежи, удовлетворяющие запросу, будут автоматически передаваться в потоковом режиме потребителю в тот момент, когда они добавляются в таблицу. Данный тип запросов поддерживается только первичными и вторичными производителями.
- *Последние данные* (*Latest*): Потребителю возвращается последняя версия кортежа (с наиболее поздней временной меткой), удовлетворяющего запросу.
- *Предыстория* (*History*): Потребителю возвращаются все имеющиеся кортежи, удовлетворяющие запросу.
- *Статический* (*Static*): Тип запросов, специфичный для производителей по требованию. Представляет собой обычный запрос к базе данных, не содержащий временных меток или интервалов, используемых в R-GMA.

Все запросы, кроме статических, могут содержать ограничение по интервалу времени.

Репозиторий схем R-GMA служит для хранения описаний таблиц виртуальной базы данных. Он поддерживает такие операции, как добавление и удаление таблиц из виртуальной базы данных, или получение информации об атрибутах определенной таблицы.

Отказоустойчивость системы обеспечивается следующим образом:

- Регистрация производителей и потребителей имеет ограниченное время жизни и требует периодического продления со стороны компонента.
- Реестр и репозиторий схем существуют в нескольких экземплярах с автоматической репликацией данных между ними.
- Все компоненты корректно обрабатывают отказы удаленных сервисов путем автоматической отправки повторных запросов и поиска альтернативных сервисов.

Первая реализация R-GMA была построена на технологии сервлетов. Позже был осуществлен переход на технологии Web-сервисов. Система имеет API на языках Java, C, C++ и Python.

R-GMA хорошо масштабируется с ростом числа узлов благодаря поддержке репликации и автоматической синхронизации реестров на нескольких узлах. Поскольку передача основных данных происходит напрямую между потребителями и производителями, масштабируемость системы

по отношению к числу запросов ограничивается возможностями отдельных производителей. При достаточно большом числе запросов, загрузка производителя может достигнуть критического значения. Для решения этой проблемы можно создать вторичного производителя на выделенном узле с высокоскоростным соединением, реплицирующего информацию данного производителя, или же использовать иерархическую организацию производителей наподобие MDS.

Результаты тестирования R-GMA [2] показали хорошую производительность и масштабируемость системы. В то же время, в [57] приводятся результаты измерений производительности R-GMA, выявившие ряд узких мест и недостатков реализации тестируемой версии системы на основе сервлетов. Результаты тестирования реализации R-GMA на технологии Web-сервисов пока отсутствуют.

3. Информационные модели

Рассмотренные выше системы MDS и R-GMA используют определенные модели данных — LDAP, XML или реляционную модель. В то же время, эти системы не накладывают ограничений на модели представления передаваемой через них информации. Для обеспечения информационной совместимости между различными источниками информации в Grid, реализациями информационных сервисов и программными платформами Grid необходимы общие стандарты представления относящейся к Grid информации. Особенно остро потребность в подобных стандартах ощущается в тех случаях, когда требуется обеспечить взаимодействие между уже функционирующими крупномасштабными Grid-инфраструктурами, построенными на основе различных программных платформ. Отсутствие общих информационных моделей и стандартов существенно усложняют реализацию автоматического подбора ресурсов в Grid (см. п. 4). Информационные модели Grid являются в настоящее время предметом активных исследований. В данном пункте мы рассмотрим некоторые из предлагаемых моделей.

3.1. Описание ресурсов

GLUE

Схема *Grid Laboratory Uniform Environment* (*GLUE*) [16] возникла в результате совместной работы проектов DataTAG, iVDGL, Globus и DataGrid над созданием общей модели описания ресурсов Grid. Информационная модель GLUE охватывает основные элементы современной Grid-инфраструктуры и используется такими крупными Grid-проектами, как DataGrid, CrossGrid, Large Hadron Collider Computing Grid (LCG) и EGEE. Модель GLUE описана на языке UML в абстрактном виде, независимом от моделей данных. На сегодняшний момент определены отображения GLUE в три модели данных: LDAP, реляционную модель и XML. Эти модели поддерживаются соответственно системами MDS2, R-GMA и MDS3/4, рассмотренными в предыдущем пункте.

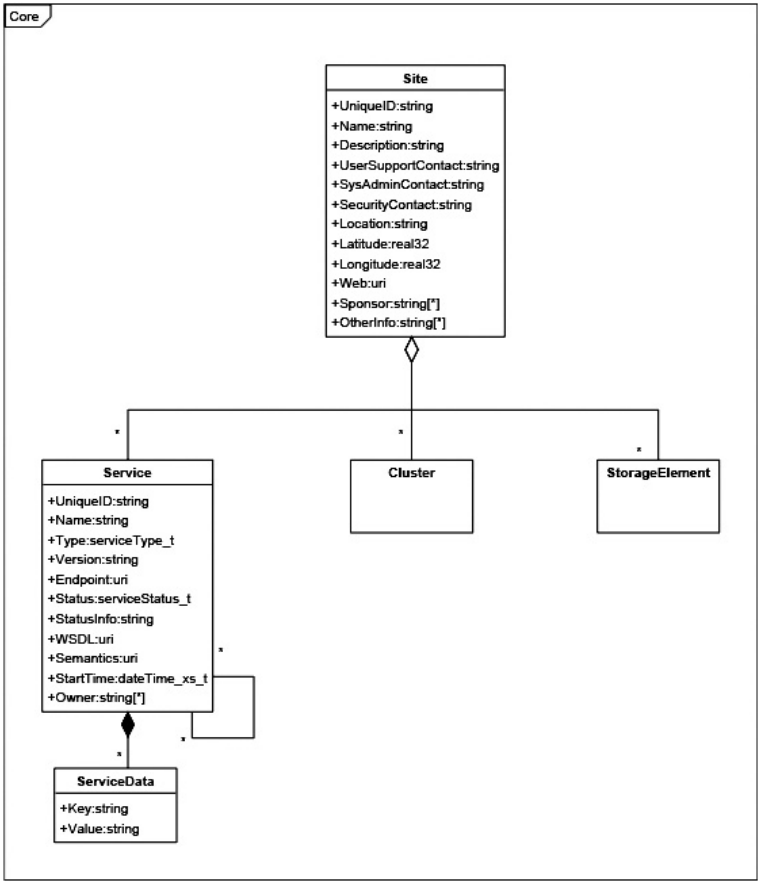

Рис. 8. Базовые сущности модели GLUE

На рис. 8 изображены базовые сущности модели GLUE (версия 1.2, 3 декабря 2005 г.), такие как *узел Grid* (*Site*) и размещенные на нем *сервисы* (*Service*), *вычислительные ресурсы* (*Cluster*) и *ресурсы хранения данных* (*StorageElement*). В будущих версиях GLUE последние два класса (точнее классы *ComputingElement* и *StorageElement*) предполагается сделать подклассами общего высокоуровневого понятия, такого как сервис, в соответствии с современной сервисно-ориентированной архитектурой Grid.

Класс *Service* содержит набор общих атрибутов для описания сервисов, присутствующих в Grid, и служит основой для создания специализированных схем сервисов. Формально он определен как «абстрактное описание программных компонентов в терминах сообщений, которыми обмениваются поставщик и клиент сервиса». Как видно из наличия у данного класса атрибута WSDL, речь идет исключительно о Web-сервисах. Это сни-

жает общность модели, поскольку, несмотря на широкое распространение в Grid технологий Web-сервисов, в общем случае сервисы Grid могут быть реализованы на основе других технологий.

Основное внимание в GLUE уделено моделированию физических ресурсов, таких как вычислительные узлы и серверы (класс *Host*), кластеры (классы *Cluster* и *SubCluster*) и хранилища данных (часть класса *Host*). Классы *ComputingElement* и *StorageElement* описывают сервисы, предоставляющие доступ к подобным физическим ресурсам. Класс *ComputingElement* фактически представляет собой точку входа в систему управления очередями заданий на некотором кластере. Класс *StorageElement* описывает сервис или группу сервисов, управляющих доступом к хранилищу данных, разбитому на несколько логических областей (класс *StorageArea*) со своими политиками доступа, при помощи определенных протоколов (классы *AccessProtocol* и *ControlProtocol*). В GLUE предусмотрена возможность описания отношений между экземплярами классов *ComputingElement* и *StorageElement*, связанных с наличием разделяемой файловой системы или предпочтениями по монтированию хранилищ к вычислительному ресурсу.

В настоящее время работа над схемой GLUE продолжается в рамках проекта EGEE [12].

Common Information Model

Другой подход, развиваемый в рамках организации Global Grid Forum, заключается в расширении промышленного стандарта *Common Information Model* (*CIM*) [9]. CIM представляет собой концептуальную информационную модель, разрабатываемую в рамках организации Data Management Task Force (DMTF) для описания управляемых компонентов IT-инфраструктуры, таких как аппаратные системы, сети, приложения и сервисы. Будучи нейтральной по отношению к производителям, модель CIM служит основой для обеспечения интероперабельности между реализациями различных производителей и унификации управления ресурсами в рамках гетерогенных IT-систем. Под управлением подразумевается активный процесс мониторинга, модификации и принятия решений относительно ресурсов. Для моделирования элементов IT-инфраструктуры в CIM используется объектно-ориентированный подход на базе диаграмм классов UML. Основной упор в CIM, как и в GLUE, делается на описании физических элементов системы, а не абстрактных сервисов. Отличительной особенностью CIM, как и большинства промышленных стандартов, является обширность и сложность спецификаций.

В рамках Global Grid Forum ведутся работы над расширениями CIM, предназначенными для описания ресурсов и сервисов Grid. На сегодняшний момент наибольший прогресс достигнут в описании реляционных баз данных, а также выполняемых в Grid заданий (см. далее Job Submission Information Model).

3.2. Описание заданий

При отправке вычислительного задания на выполнение в Grid, необходимо снабдить его информацией о требованиях к вычислительным

ресурсам, такой как число процессоров, объем доступной памяти и т. п., местоположении файлов с входными данными и т. д. В настоящее время существует несколько языков описания вычислительных заданий в Grid, например — *Resource Specification Language* (*RSL*) в Globus Toolkit, *ClassAds* в Condor (см. раздел 3.3), *Job Description Language* (*JDL*) в gLite.

Для обеспечения интероперабельности различных систем, в рамках организации Global Grid Forum разрабатывается стандартный язык описания заданий *Job Submission Description Language* (*JSDL*) [23]. Спецификация JSDL представляет собой XML-схему для описания пакетных вычислительных заданий и требований к среде их выполнения. JSDL-документы содержат всю информацию, требуемую для из запуска в Grid, такую как: требования к ресурсам, местоположение файлов с входными и выходными данными, способы передачи этих файлов, зависимости между заданиями и т. д. Предполагается, что задание, описанное на JSDL, может быть передано любой системе управления заданиями в Grid, поддерживающей трансляцию из этого абстрактного языка в специфическое для конкретной реализации представление задания.

Другое направление работ — стандартизация информации о состоянии отправленных на выполнение заданий, публикуемой планировщиками Grid. В рамках того же GGF была предложена модель *Job Submission Information Model* (*JSIM*) [24], описывающая структуру и атрибуты пакетных заданий, выполняющихся в Grid. JSIM основана на стандарте Common Information Schema (CIM) (см. раздел 3.1).

3.3. Комбинированные модели

ClassAd

Система Condor [10], предназначенная для организации виртуальных вычислительных кластеров из гетерогенных машин, использует для описания ресурсов и заданий общий язык *ClassAd* (от Classified Advertisement, классифицированные рекламные объявления) [35]. Описания ресурсов и задания пользователей оформляются в виде объявлений (classads), отправляемых центральному планировщику Condor, который производит подбор ресурсов для выполнения заданий путем сопоставления информации из хранящихся объявлений.

Объявление ClassAd содержит набор атрибутов и связанных с ними выражений. Выражения могут быть простыми константами или являться функциями других атрибутов. Определен протокол для оценки выражения из одного объявления по отношению к другому объявлению. Например, оценка выражения «*other.size* > 3» в одном объявлении имеет результат *true*, если в другом объявлении содержится атрибут *size* и связанное с ним выражение имеет значение типа *integer* большее трех. Два объявления считаются совпадающими, если каждое объявление имеет атрибут *Constraint*, оценка выражения которого равна *true* по отношению к другому объявлению.

Поиск совпадающих объявлений применяется в Condor при подборе ресурсов для выполнения заданий пользователей. На каждом ресурсе запущен агент, который периодически публикует объявление, содержащее

информацию о данном ресурсе. С помощью атрибута *Constraint* владелец ресурса может описать в объявлении политику доступа к ресурсу и требования, предъявляемые к выполняемым на ресурсе заданиям. В свою очередь, пользователь оформляет запрос на выполнение задания в виде объявления, в атрибуте *Constraint* которого указывает свои требования к ресурсу. Дополнительные предпочтения владельцев ресурсов и пользователей могут быть описаны с помощью атрибута *Rank* с выражением, используемым для ранжирования совпадающих объявлений.

Спецификация языка ClassAd не накладывает ограничений на используемые в объявлениях конкретные наборы атрибутов (информационные модели). Очевидно, что для эффективного подбора ресурсов на основе объявлений владельцы ресурсов и пользователи должны использовать общую схему, содержащую стандартные имена и значения атрибутов. Используемая в Condor схема ориентирована на описание характеристик вычислительных машин, а также различных ограничений и предпочтений, предъявляемых владельцами и пользователями подобных ресурсов.

Пример объявления ресурса:

```
[
    Type = "Machine";
    Activity = "Idle";
    KeybrdIdle = '00:23:12'; // h:m:s
    Disk = 323.4m; // mbytes
    Memory = 64m; // mbytes
    State = "Unclaimed";
    LoadAvg = 0.042969;
    Mips = 104;
    Arch = "INTEL";
    OpSys = "SOLARIS251";
    KFlops = 21893;
    Name = "foo.cs.wisc.edu";
    ResearchGp = { "raman", "miron", "solomon" };
    Friends = { "calvin", "hobbes" };
    Untrusted = { "rival", "riffraff" };
    Rank = member(other.Owner, ResearchGp) ? 10 :
           member(other.Owner, Friends) ? 1 : 0;
    Constraint = !member(other.Owner, Untrusted) && Rank>=10
    ? true : Rank>0 ? LoadAvg < 0.3 && KeybrdIdle>'00:15' :
             DayTime()<'8:00' || DayTime()>'18:00'
]
```

Пример объявления задания:

```
[
    Type = "Job";
    CompletionDate = undefined;
    RemoteSyscalls = true;
    Checkpoint = true;
    QDate = 'Mon Jan 11 10:53:31 1999 (CST) -06:00';
```

```
    Owner = "raman";
    Cmd = "run_sim";
    Iwd = "/usr/raman/sim2";
    Args = "-Q 17 3200 10";
    ImageSize = 31M;
    Rank = DayTime()>'20:00' && DayTime<'8:00' &&
           other.IsInstructional ? 10 :
           other.KeybrdIdle>'3:00' ? 5 : 0;
    Constraint = other.Type=="Machine" && Arch=="INTEL" &&
           OpSys=="SOLARIS251" && Disk >= 10M &&
           other.Memory >= self.ImageSize
]
```

Реализация спецификации ClassAd доступна в виде отдельных библиотек, которые могут применяться независимо от системы Condor в других приложениях, использующих собственные схемы атрибутов.

4. Применение семантических технологий в Grid

Как наглядно иллюстрирует опыт Web, поиск релевантных с точки зрения пользователя ресурсов существенно усложняется в случае наличия огромного числа динамически изменяемых ресурсов и отсутствия общей модели описания ресурсов. В лучшем случае, этот процесс можно частично автоматизировать так, как это делают современные поисковые машины. При этом пользователь должен принимать активное участие в процессе поиска, определяя истинную релевантность представленных ему ресурсов и уточняя свой запрос. Отсутствие стандартной модели аннотации ресурсов, поддерживающей «понятное» машине описание семантики ресурса, не позволяет радикально повысить качество результатов поиска и автоматизировать рутинные операции, связанные с поиском и интеграцией информации в Web. Указанные обстоятельства послужили причиной появления концепции Semantic Web [49], определяющей дальнейшее развитие Web в направлении семантической аннотации ресурсов и применения интеллектуальных агентов для автоматизации поиска, интеграции и обработки информации в Web.

4.1. Semantic Web

Проект Semantic Web [47] направлен на реализацию нового поколения Web, где информационные ресурсы снабжаются семантическими описаниями, допускающими машинную интерпретацию и проведение логических рассуждений. В конечном итоге, подобная инфраструктура должна стать основой для интеллектуальных сервисов поиска, интеграции и обработки информации в Web, автоматизирующих рутинные операции, проводимые сейчас пользователями вручную.

На рис. 9 изображена многоуровневая архитектура и стек технологий Semantic Web. Рассмотрим кратко основные уровни данной архитектуры.

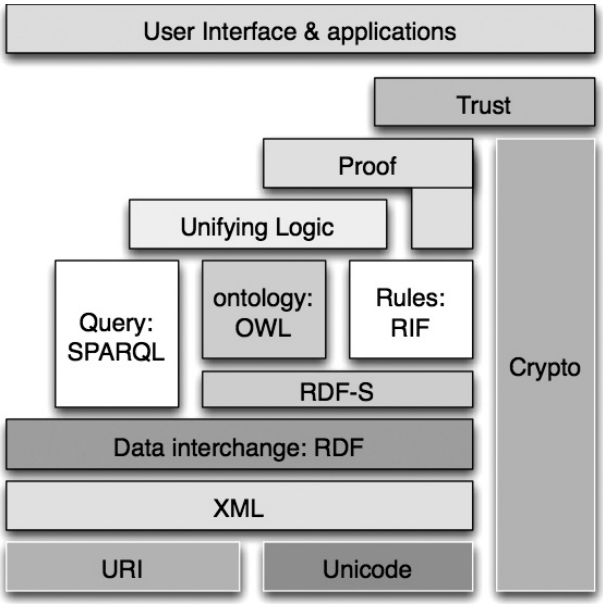

Рис. 9. Многоуровневая архитектура Semantic Web (2006)

Нижние уровни

Технологии Semantic Web базируются на общепризнанных Web-стандартах, таких как Unicode для кодирования текстовой информации, URI для идентификации ресурсов и XML для представления структурированной информации.

Модель данных и синтаксис

Набор спецификаций Resource Description Framework (RDF) [39] определяет стандартную модель данных и синтаксис для описания ресурсов в рамках Semantic Web. В рамках RDF ресурсы описываются при помощи элементарных высказываний, состоящих из трех частей:

- *субъекта* — ресурса, который описывается данных высказыванием;
- *предиката* — определенного атрибута описываемого субъекта;
- *объекта* — значения данного атрибута.

Для идентификации субъектов, предикатов и объектов RDF использует универсальный идентификатор URI [40]. С точки зрения RDF любая сущность, идентифицируемая при помощи URI, является ресурсом и может быть описана при помощи высказываний. Эта же сущность может играть роль объекта или предиката в контексте других высказываний.

Модель данных RDF позволяет представить набор высказываний в виде ориентированного графа, вершинами которого являются субъекты и объекты высказываний, а ребрами — соответствующие предикаты.

Для представления высказываний в виде, пригодном для их передачи и интерпретации программными приложениями, используется основанный на XML синтаксис RDF/XML [38]. RDF/XML представляет универсальный формат для обмена RDF-данными. Существуют другие варианты синтаксиса RDF, такие как Notation 3, N-Triples и Turtle.

Как и любая модель данных, RDF имеет стандартный язык запросов SPARQL [48], предназначенный для унифицированного доступа к RDF-данным. Существует множество альтернативных языков запросов RDF.

Описание онтологий

Онтологии предназначены для спецификации в явном виде понятий и отношений, используемых для представления знаний в некоторой предметной области. В контексте Semantic Web онтологии используются для описания семантики элементов RDF-высказываний, тем самым, позволяя программам отличать синтаксически эквивалентные понятия из разных предметных областей, «понимать» отношения между встречающимися понятиями и проводить логические рассуждения, т. е. выводить новые высказывания на основе уже имеющихся. Для этих целей в Semantic Web используются языки RDFS (RDF Schema) и OWL (Web Ontology Language), созданные на основе модели и синтаксиса RDF.

Язык RDFS [37] позволяет описать классы ресурсов, свойства и отношения между ними, определяющие использование тех или иных свойств при описании экземпляров некоторого класса. В некотором роде, RDFS предоставляет систему типов для RDF. По ряду признаков эта система похожа на системы типов объектно-ориентированных языков программирования. Например, RDFS позволяет определить ресурс как экземпляр одного или нескольких классов, или использовать при описании классов наследование. С другой стороны, описания классов и свойств не формируют некоторую жесткую структуру данных, в которую должна быть занесена информация, а скорее предоставляют дополнительную информацию об описываемых ресурсах. Эта информация может быть использована различным образом в программах. Например, основываясь на описанной в RDFS иерархии классов “RedHat есть подкласс Linux”, программа из высказывания «На сервере X установлен RedHat» может автоматически вывести новое высказывание «На сервере X установлен Linux».

Язык OWL [31] значительно расширяет возможности RDFS по описанию классов и свойств ресурсов. В частности, OWL поддерживает:

- описание кардинальности свойств;
- описание транзитивности свойств;
- описание эквивалентности классов и свойств (т.е. того, что два разных класса или свойства фактически представляют одно и то же понятие);
- описание эквивалентности экземпляров классов (т.е. того, что два разных ресурса фактически представляют один и тот же объект);
- описание ограничений на диапазон или кардинальность значений свойства, которые зависят от класса или ресурса, к которому данное свойство применено;

- описание новых классов путем комбинации (например, объединения или пересечения) других классов, или описание того, что два класса являются дизъюнктными (т.е. ни один ресурс не может быть экземпляром сразу обоих классов).

OWL является формальным языком, основанном на принципах математической логики (точнее т. н. *логики описаний*, *description logic*). Эти принципы позволяют приложениям формально-логическим путем осуществлять автоматический вывод новых фактов из имеющихся высказываний. Существуют три подмножества OWL, отличающиеся своей выразительностью и гарантиями вычислительной полноты и разрешимости:

- *OWL Lite* — предназначен для пользователей, которым необходима лишь классификационная иерархия сущностей и некоторые простые ограничения.
- *OWL DL* (*Description Logic*) — рассчитан на пользователей, которым необходима максимальная степень выразительных возможностей языка без потери вычислительной полноты и разрешимости. Уровень OWL DL ориентирован на существующие сегодня системы описания знаний и логического вывода.
- *OWL Full* — рассчитан на пользователей, которым необходимы все выразительные возможности языка OWL и синтаксиса RDF, но без каких-либо гарантий вычислительной полноты и разрешимости.

Отметим, что модель данных RDF позволяет легко расширять существующие онтологии, описанные на RDFS или OWL, путем добавления новых классов или свойств без каких-либо конфликтов с уже имеющимися описаниями. В масштабах Web приложения могут независимо использовать различные модификации некоторой онтологии или несколько различных онтологий. Язык OWL поддерживает контроль версий и позволяет установить соответствие между понятиями из разных онтологий путем задания эквивалентности классов и свойств.

Правила

Несмотря на то, что существующие языки описания онтологий, такие как OWL, позволяют описывать некоторые правила (аксиомы), возможности эти довольно ограничены. Например, с помощью OWL нельзя записать хорновские формулы наподобие «для любых A, B и C: если “A — родитель B” и “C — брат A”, то “C — дядя B”». Подобные правила могут быть описаны при помощи языков описания правил и затем использованы приложениями в качестве дополнительных знаний при интерпретации RDF-высказываний.

Существует множество языков и систем, основанных на правилах. В контексте Web отметим Metalog, RuleML (Rule Markup Language), WRL (Web Rule Language), SWRL (Semantic Web Rule Language), cwm. Язык SWRL, например, представляет комбинацию языков OWL (DL и Lite) и RuleML (хорновская логика), лишенную уже гарантий вычислительной разрешимости.

В конце 2005 г. в рамках проекта Semantic Web начаты работы по созданию общего формата описания правил (Rules Interchange Format) и его расширений, позволяющих транслировать правила из одного языка в другой и, тем самым, осуществлять перенос правил из одной системы в другую. В этом направлении есть ряд нерешенных вопросов, носящих главным образом теоретический характер. Например, каким образом в рамках целостного подхода охватить все многообразие существующих систем логики.

Верхние уровни

Верхние уровни архитектуры Semantic Web на сегодняшний момент менее всего проработаны, и пока здесь отсутствуют какие-либо наработки или рекомендации W3C. На уровне *Unifying Logic* планируется реализовать единый логический формализм на базе логики предикатов первого порядка. Поверх этого уровня естественным образом может быть реализована поддержка доказательства утверждений (уровень *Proof*). Наконец, доверие (уровень *Trust*) в открытой среде Semantic Web может быть достигнуто путем снабжения результатов запросов соответствующими доказательствами и использования криптографии (цифровых подписей и шифрования).

4.2. Проблемы интеграции информации и поиска ресурсов в Grid

Аналогичная Web картина наблюдается сейчас в Grid. На сегодняшний момент не существует общепризнанных, широко используемых на практике и совместимых друг с другом схем описания информации в Grid. Предлагаемые модели часто пересекаются друг с другом, большинство работ по их стандартизации находится в ранней стадии, что приводит к появлению все большего числа несовместимых друг с другом на информационном уровне программных платформ Grid. Это ставит под сомнение появление, по крайней мере, в ближайшем будущем, некоторой общей схемы описания информации, относящейся к Grid. Кроме того, используемые в современных Grid-системах информационные модели ресурсов являются довольно простыми и не позволяют отразить всю семантику, ограничения на использование или связи с другими ресурсами. Это ограничивает возможности по описанию ресурсов и, как следствие, по реализации эффективного поиска ресурсов в Grid.

Эффективный механизм поиска ресурсов является одним из главных требований, предъявляемых к Grid. Пользователь Grid должен иметь возможность сформулировать запрос на решение его задачи, не обладая детальными знаниями о присутствующих в Grid ресурсах. Необходимо автоматически сопоставлять запрос пользователя с информацией о присутствующих в Grid ресурсах, осуществлять проверку их доступности для данного пользователя и резервировать их для последующего запуска задач. Данный процесс *подбора ресурсов* (resource matching) должен быть реализован в специальном компоненте Grid, называемом обычно *брокером* или планировщиком, избавляя тем самым пользователей и приложения Grid от необходимости выполнять эти действия самостоятельно.

Требования, предъявляемые пользователями к ресурсам, могут варьироваться по своей сложности и охватывать не только функциональность

ресурсов, но и такие характеристики, как надежность, стоимость использования, качество обслуживания и т. п. Пользователь должен иметь возможность указать дополнительные предпочтения, используемые для ранжирования найденных ресурсов. Для того чтобы брокер смог подобрать в точности те ресурсы, которые требуются пользователю, запрос должен быть представлен в форме, сохраняющей всю его семантику и допускающей его передачу и автоматическую обработку (интерпретацию).

С другой стороны, брокер должен иметь доступ ко всей информации о ресурсах, необходимой для выполнения запросов пользователей. Доступ к данной информации может быть организован при помощи описанных в п. 2 информационных сервисов Grid. Информация, используемая брокером для обработки запроса пользователя, должна быть по возможности полной и актуальной, гарантируя тем самым возможность работы пользователя с найденными ресурсами. Например, при наличии информации о политике доступа к ресурсу, брокер может заранее удостовериться, что пользователь действительно сможет получить доступ к ресурсу. Во многих системах такая проверка не производится, что приводит к тому, что, получив от брокера список подходящих ресурсов, пользователь может далее получить отказ при попытке обратиться к ним. Также как и запросы пользователей, информация о ресурсах должна быть представлена в форме, сохраняющей всю семантику и допускающей корректную интерпретацию брокером.

Подбор ресурсов в большинстве современных систем основан на простом синтаксическом сравнении значений атрибутов, присутствующих в описании ресурса, со значениями, указанными в запросе пользователя. Типичным примером подобного подхода является поиск ресурсов в системе Condor с использованием объявлений ClassAd (см. раздел 3.3). Изначально сопоставление объявлений в ClassAd ограничивалось только двусторонним совпадением (bi-lateral match). Это не позволяло, например, оформить запрос сразу на несколько ресурсов, с отдельными требованиями к каждому из них. Впоследствии авторами ClassAd было предложено расширение языка, поддерживающее многостороннее совпадение (gang-matching) [34]. В [6] было предложено расширение ClassAd для поддержки множественного совпадения (set-matching), где число ресурсов неизвестно априори. В этом случае требования формулируются на уровне совокупных характеристик множества ресурсов — например, требуется набор машин с суммарной памятью больше 10 Gb. В [5] предлагается другой подход, основанный на преобразовании исходной задачи подбора ресурсов в задачу поиска допустимого решения, для решения которой могут применяться существующие методы, такие как целочисленное программирование. Созданный авторами язык Redline является более выразительным, чем ClassAd, и изначально поддерживает многостороннее и множественное совпадение.

Обязательным требованием для всех упомянутых выше подходов является использование симметричного набора атрибутов для описания ресурсов и запросов. Требуются жесткие договоренности между поставщиками ресурсов и пользователями, как по именам атрибутов, так и по их возможным значениям. Сопоставление описаний только на синтаксическом

уровне, без учета семантики атрибутов и их значений, а также необходимость координации набора атрибутов между всеми участниками, делает подобные системы негибкими и трудно расширяемыми. Особенно это критично в среде Grid, гетерогенной и в технологическом и организационном плане, где трудно добиться подобной координации и заставить всех участников придерживаться единого синтаксиса.

Проиллюстрируем возникающие проблемы на конкретном примере. В описание вычислительного узла Grid обязательно должна входить информация об используемой операционной системе. Разные участники Grid, в силу технологических или организационных причин, могут использовать для описания данной информации различные варианты имен атрибутов, например, "OperatingSystem" или "OpSys", и значений, например, "Fedora" или "Fedora Core". Первая проблема заключается в том, что всем им придется перейти на использование неких общепринятых имен и значений. Само по себе это разумно, но, учитывая наличие множества технологий с несовместимым синтаксисом описаний и текущее состояние стандартизации в этой области в Grid, сделать это сейчас очень трудно.

Вторая проблема заключается в том, каким образом пользователь должен описать запрос на выполнение задания на любой машине с операционной системой типа Linux. Во-первых, он может явно перечислить в своем запросе все существующие разновидности Linux: OpSys="Fedora" || OpSys="RedHat" || OpSys="ScientificLinux" и т. д. Во-вторых, он может потребовать от всех владельцев ресурсов включить в их описание выражение: OpSysType="Linux", что позволит ему значительно упростить свой запрос. Но что тогда делать пользователю, которому достаточно машины с Unix-совместимой операционной системой, к классу которых относятся и системы типа Linux и, скажем, Solaris или Mac OS X? Таким образом, требование симметричности набора атрибутов при описании ресурсов и запросов может приводить к проблемам, связанных с тем, что для описания требований задания (приложения) бывает более удобно использовать высокоуровневые понятия и характеристики, не встречающиеся явно в описаниях ресурсов.

Технологии Web-сервисов, на которых базируется современная сервисно-ориентированная архитектура Grid, также предоставляют недостаточные возможности для описания сервисов. Используемые стандарты, такие как WSDL, позволяют описать только сигнатуры операций, реализуемых сервисом. Этой информации недостаточно для того, чтобы программные агенты могли автоматически определить точное значение операций и функциональность сервиса. Так два сервиса с полностью совпадающими сигнатурами операций могут в действительности выполнять совершенно разные функции.

Указанные проблемы делают актуальным исследование возможности применения подхода и технологий Semantic Web для решения вопроса синтаксически несовместимых информационных моделей и организации гибкого механизма поиска ресурсов на семантическом уровне. На стыке технологий Grid и Semantic Web начало формироваться новое направление с условным названием "Semantic Grid" [11]. В следующем разделе,

не претендуя на полноту охвата, рассмотрены некоторые проекты в данной области, иллюстрирующие различные аспекты применения семантических технологий в Grid.

4.3. Проекты в области Semantic Grid

Одним из первых подобных проектов был *GRid Interoperability Project* (*GRIP*) [17], нацеленный на обеспечение интероперабельности между платформами Globus Toolkit (GT2 и GT3) и UNICORE [52]. В частности, были построены онтологии, соответствующие информационным моделям ресурсов в Globus Toolkit (схема GLUE) и Unicore, и реализовано средство, осуществляющее автоматическую трансляцию из одной онтологии в другую [20]. Это позволило создать универсальный брокер к системам на базе Globus Toolkit и Unicore.

В [18] описана реализация более сложного брокера ресурсов на основе онтологий, состоящего из следующих блоков:

- Онтологий, служащих для описания модели предметной области, ресурсов и заданий.
- Дополнительных знаний о предметной области, оформленных в виде правил, которые не могут быть описаны явно в рамках онтологий.
- Правил соответствия, определяющих ограничения, при которых запросы и ресурсы «подходят» друг другу.

С помощью языка RDFS описаны три онтологии: ресурсов, заданий и политик доступа. Дополнительные знания и правила соответствия описаны на языке TRIPLE [27]. Операция подбора ресурсов на основе описанных онтологий и правил реализована при помощи дедуктивной базы данных TRIPLE/XSB.

Преимущества данного подхода заключаются в возможности использования различных моделей для описания ресурсов и заданий, гибкости и расширяемости онтологий, поддержке дополнительных знаний в форме правил. Так задания могут быть описаны в высокоуровневых терминах прикладной области, а затем с помощью описанных правил автоматически отображены в требования к ресурсам. Онтологии и правила могут быть также использованы для автоматической проверки корректности описаний ресурсов и заданий. Наконец, правила могут использоваться для описаний соответствий между онтологиями, используемыми различными платформами Grid. Недостатки предложенной системы состоят в необходимости явного описания правил соответствия при каждой модификации онтологий и многоэтапной компиляции правил в инструкции дедуктивной базы данных, а также использовании менее выразительного в сравнении с OWL языка RDFS.

В [28] предлагается другой подход к семантическому описанию, поиску и подбору ресурсов Grid, основанный на использовании языка OWL для описания ресурсов и языка *OWL Query Language* (*OWL-QL*) [42] для описания поисковых запросов. Язык OWL-QL представляет собой формальный язык и протокол для обмена запросами между интеллектуальными агентами, использующими базы знаний в формате OWL. Отличительными осо-

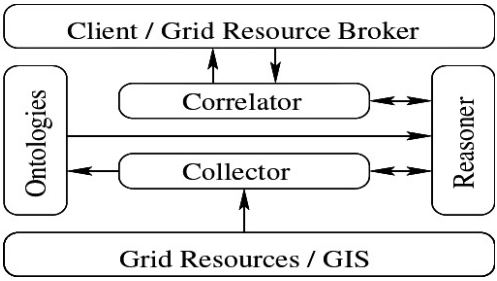

Рис. 10. Архитектура семантического поиска ресурсов в Grid

бенностями OWL-QL является поддержка как точных, так и семантически близких результатов, а также возможность указания в запросе нескольких баз знаний и шаблонов ответов на естественном языке. Предполагается, что обрабатывающий запросы агент использует для ответа на них машину вывода на основе логики описаний (DL-reasoner), в которую загружены базы знаний на языке OWL DL. Например, OWL-QL-сервер RacerManager построен на базе популярной машины вывода RacerPro.

На рис. 10 изображена предложенная в [28] архитектура семантического поиска ресурсов в Grid. Компонент *Collector* осуществляет сбор информации о ресурсах Grid (напрямую или через информационные сервисы), проверку ее на соответствие онтологиям и включение ее в базу знаний. Компонент *Reasoner* представляет машину вывода, которая используется для валидации онтологий и описаний ресурсов, классификации таксономии ресурсов и ответа на OWL-QL-запросы. Компонент *Correlator* принимает запросы от клиентов, проверяет их на корректность, преобразует в случае надобности в формат онтологии ресурсов и формирует запрос к машине вывода. На основе полученных от машины вывода результатов *Correlator* осуществляет окончательный подбор и ранжирование ресурсов, удовлетворяющих запросу.

На пути реализации данной схемы стоит проблема, связанная с тем, что существующие машины вывода для OWL DL ориентированы главным образом на работу с информацией уровня классов и плохо справляются с большими объемами данных уровня экземпляров классов, что требуется при поиске среди описаний ресурсов Grid. Эта проблема будет рассмотрена в конце пункта.

В [55] рассматривается проблема построения базовой онтологии Grid (*core Grid ontology*, *CGO*), определяющей фундаментальные понятия, словари и отношения на основе общей модели Grid. Данная онтология должна с одной стороны быть достаточно абстрактной и не зависящей от конкретных технологий, а с другой — допускать свое дальнейшее расширение специфическим образом. Наличие подобного общего базиса заметно упростило бы задачу интеграции различных информационных моделей и описаний Grid за счет описания их связей с базовой онтологией. В CGO определено 7 базовых классов: Виртуальная организация (*VO*), Ресурс

(*GridResource*), Промежуточное программное обеспечение Grid (*GridMiddleware*), Компонент (*GridComponent*), Пользователь (*GridUser*), Приложение (*GridApplication*) и Сервис (*GridService*). На основе базовых классов определены уточняющие их общие понятия, такие как, например, Вычислительный ресурс (*ComputingResource*) или Информационный сервис (*InfoService*). Наконец, на третьем уровне определены классы, специфичные для конкретных платформ Grid. Например, класс *MDS* определен как подкласс *InfoService*. Онтология CGO описана на языке OWL.

В контексте онтологий также стоит отметить основанную на OWL онтологию Web-сервисов *OWL-S* [32]. Поскольку современная сервисно-ориентированная архитектура Grid базируется на технологиях Web-сервисов, то OWL-S может быть взята в качестве основы для описания сервисов, скрывающих за собой компоненты и ресурсы Grid. Модель сервиса в OWL-S состоит из трех частей:

- *Профиль сервиса* (*ServiceProfile*), содержащий необходимую для поиска и выбора сервиса информацию: предоставляемые сервисом возможности, ограничения по использованию сервиса и качеству обслуживания, предъявляемые к клиентам требования и т. п.
- *Модель сервиса* (*ServiceModel*), описывающая возможные взаимодействия с сервисом в терминах процессов (потоков управления и данных), каждый из которых определяется набором участников, входных и выходных параметров, предусловий и результатов.
- *Реализация сервиса* (*ServiceGrounding*), содержащая информацию о деталях реализации доступа к сервису: транспортных протоколах, форматах сообщений, физических адресах и т. п.

Другой подход к семантическому описанию Web-сервисов представлен в онтологии *Web Service Modeling Ontology* (*WSMO*) [54], использующей для описания сервисов свой собственный язык *Web Service Modeling Language* (*WSML*) и имеющей эталонную реализацию WSMX. Ряд проектов в области Semantic Grid используют OWL-S и WSMO в качестве основы для описания Grid-сервисов.

Помимо поиска и подбора ресурсов, семантические технологии могут быть использованы для автоматизации *вызова* и *композиции* Grid-сервисов. Рассмотрим эти возможности по порядку.

Обладая декларативным описанием семантики операций, аргументов и сообщений, связанных с сервисом, программный агент может путем интерпретации этого описания динамически сформировать вызов данного сервиса. Подобные агенты с адаптивным поведением лучше приспособлены для работы в гетерогенной среде, чем агенты, жестко запрограммированные для вызова определенных сервисов.

Для решения сложных задач в Grid может потребоваться участие сразу нескольких ресурсов (сервисов), взаимодействующих между собой по некоторому *сценарию* (т. н. *workflow*) [1]. Традиционные подходы предполагают, что пользователь вручную осуществляет поиск, подбор и композицию сервисов для реализации сценария решения поставленной задачи. Ручная композиция сервисов сильно усложняется в динамичной и гетерогенной

среде Grid, содержащей большое число ресурсов. Для автоматизации композиции сервисов может быть использована информация, содержащаяся в описаниях сервисов. Данная возможность предусмотрена, например, в OWL-S при помощи декларативных спецификаций входных и выходных параметров, пред- и постусловий вызова операций сервиса, а также процессов взаимодействия с сервисом. На основе данной информации и описания задания программный агент может автоматически провести подбор и композицию сервисов. При автоматической композиции сервисов необходимо установить многостороннее соответствие между описаниями сервисов и обеспечить интероперабельность взаимодействующих сервисов. Последняя может быть достигнута путем адаптации выходных данных одного сервиса к требованиям вызова другого, семантически совпадающего, но синтаксически несовместимого, сервиса [21].

В рамках проекта *Grid Enabled Optimisation and Design Search in Engineering* (*GEODISE*) была реализована полуавтоматическая композиция сервисов [7]. Для решения сложной инженерной задачи пользователь при помощи среды построения сценариев (Workflow Construction Environment) осуществляет композицию сервисов, представляющих собой скрипты с MATLAB-функциями. В процессе построения сценария, среда использует полученные из базы знаний семантические аннотации функций на языке OWL для выдачи рекомендаций пользователю. Так, при добавлении новой функции в область построения сценария, среда автоматически выдает список функций с семантически совместимыми интерфейсами, т. е. функций, чьи входные/выходные параметры семантически совпадают с выходными/входными параметрами данной функции. Кроме того, подробные семантические описания функций позволяют среде предоставлять пользователю информацию о типах параметров, значениях по умолчанию и связях между параметрами функции, альтернативных функциях и т. п. Эти подсказки помогают пользователю выбрать корректный метод решения задачи и правильно настроить его параметры. Построенный сценарий может быть снабжен семантическим описанием, созданным на основе описаний включенных в него функций и дополнительной информации, полученной от пользователя. Описания сценариев сохраняются в базе знаний для дальнейшего использования.

Похожий подход реализован в проекте myGrid [29], нацеленном на реализацию в Grid прикладных сценариев в области биоинформатики. В myGrid реализованы механизмы поиска существующих сценариев по их функциональности (решаемым задачам), типам и форматам входных и выходных данных. Для этого сценарии снабжаются семантическими аннотациями, включающими следующую информацию: функциональность сценария в терминах прикладной области (биологии), тип и формат входных и выходных данных, описание элементарных шагов сценария, общая информация о сценарии (имя, организация, местоположение и т. п.), историческая информация (дата создания, автор, история запусков, их результаты и т. п.). Семантические описания сценариев хранятся в центральном репозитории, который используется при подборе сценария для решения задачи пользователя. Перед запуском сценария пользователю предлагается

автоматически сформированный список доступных сервисов, которые реализуют шаги сценария. Далее сценарий выполняется на выбранной пользователем конфигурации сервисов.

В [43] предлагается архитектура полностью автоматической композиции Grid-сервисов, требующая от пользователя только описания высокоуровневой цели. На первом этапе происходит автоматическое построение *абстрактного сценария*, направленного на достижение поставленной цели. Для этого используются *репозиторий абстрактных сценариев*, *база правил* и *сервис вывода*. В первую очередь производится поиск в репозитории абстрактных сценариев по указанной цели. Если ничего не найдено, используется база правил для семантического поиска в репозитории абстрактных сценариев. Наконец, в крайнем случае, используется сервис вывода для поиска цепочки сервисов, приводящей к достижению указанной цели, на основе входных и выходных данных, пред- и постусловий. На втором этапе абстрактный сценарий преобразуется в *исполняемый сценарий* путем сопоставления абстрактных сервисов с доступными реализациями. Для этого используются *сервис подбора* (*брокер*) и сервис вывода. В случае если реализацию сервиса подобрать не удалось, при помощи сервиса вывода строится цепочка сервисов, достигающая желаемого эффекта. При практической реализации данного подхода для описания сервисов была использована онтология OWL-S, для описания абстрактных сценариев — модель DAML-S Process Model, а исполняемых сценариев — язык BPEL4WS.

Активное использование онтологий в Semantic Grid приводит к необходимости описания стандартного протокола для доступа к хранилищам онтологий. В настоящее время в Semantic Web не существует стандартного механизма доступа к онтологиям, не зависящего от конкретных языков и систем хранения, что приводит к проблемам с интероперабельностью различных систем. В рамках проекта OntoGrid был предложен подобный механизм *WS-DAIOnt* [26], основанный на спецификации доступа к данным в Grid — WS-DAI (Web Services Data Access and Integration). Суть предлагаемого подхода заключается в описании стандартных Grid-сервисов, используемых для доступа к хранилищам онтологий. Для демонстрации подхода была создана реализация WS-DAIOnt-RDF(S) для доступа к онтологиям, описанным на языке RDFS.

В [45] описан информационный сервис S-MDS для Globus Toolkit 4, поддерживающий сбор и хранение семантических данных о Grid-сервисах. Получаемые от первичных источников данные автоматически преобразуются в семантические метаданные, на основе предварительно определенного соответствия между этими данными и онтологией ресурсов. Агрегация семантических метаданных реализована на основе тех же механизмов, что и Index-сервис в MSD4 (см. раздел 2.1). Агрегированные данные помещаются в RDF-репозиторий Jena, поддерживающий язык запросов RDQL. Возможности прототипа S-MDS демонстрируются на примере мониторинга ресурсов вычислительного кластера.

В [8] предлагается общая схема управления семантическими метаданными в Grid (рис. 11), основанная на современных технологиях Semantic Web. Для описания модели метаданных используются онтологии, постро-

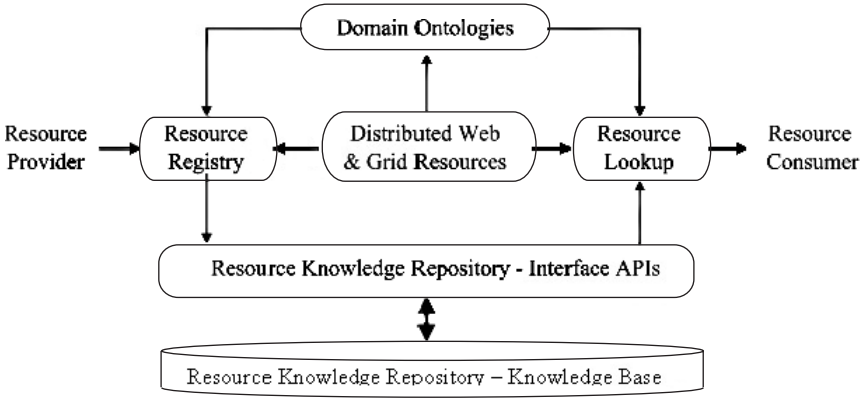

Рис. 11. Общая схема управления семантическими метаданными в Grid

енные на основе знаний о ресурсах Grid. Путем расширения базовой онтологии могут быть построены специализированные онтологии в различных прикладных областях. Для описания онтологий и самих ресурсов предлагается использовать язык OWL. *Реестр ресурсов* (*Resource Registry*) отвечает за добавление семантических метаданных к описаниям ресурсов и публикацию эти метаданных в центральном *репозитории знаний* (*Resource Knowledge Repository*). Репозиторий знаний аккумулирует семантические описания ресурсов, а также описания используемых онтологий, в базе знаний, реализованной на основе реляционной базы данных или специализированного хранилища RDF-данных. Доступ к репозиторию знаний осуществляется через стандартные *прикладные интерфейсы* (*Interface APIs*). *Сервис поиска ресурсов* (*Resource Lookup*) предоставляет пользователем интерфейс для поиска ресурсов по заданным критериям. Поисковый запрос передается машине вывода на логике описаний, которая путем классификации осуществляет подбор ресурсов, соответствующих указанным критериям.

Описанная схема требует одного важного уточнения. Поскольку в Grid присутствует большое количество источников информации, а информация эта может быстро терять свою актуальность, то центральный репозиторий знаний в общем случае должен быть заменен распределенной базой знаний. Данная база знаний формируется из динамического набора поставщиков информации, хранящих отдельные фрагменты совокупных знаний. В п. 2 уже были описаны распределенные информационные сервисы с иерархической организацией. В рамках Semantic Grid получил распространение другой подход, основанный на организации поставщиков RDF-данных в децентрализованные *пиринговые сети* (*peer-to-peer networks*), обслуживающие запросы на одном из языков запросов RDF. Подобный выбор, видимо, объясняется сетевой структурой самих RDF-данных. Кроме того, пиринговые сети хорошо зарекомендовали себя как масштабируемая и отказоустойчивая архитектура для организации распределенного механизма поиска. Для реализации децентрализованного поиска ресурсов

в Grid было предложено множество пиринговых систем, подробный обзор которых можно найти в [33].

В [30] делается попытка описать всеобъемлющую эталонную архитектуру Semantic Grid, названную S-OGSA по аналогии с современной архитектурой Grid — OGSA (Open Grid Services Architecture). Суть подхода заключается в расширении архитектуры OGSA путем встраивания в нее дополнительных элементов, поддерживающих использование онтологий, семантических метаданных и сервисов для реализации концепции Semantic Grid.

В заключение остановимся кратко на критике семантических технологий и Semantic Grid. Помимо очевидных преимуществ, часто отмечаются следующие недостатки.

Во-первых, подразумевается, что все ресурсы Grid будут снабжены подробными семантическими описаниями. У подавляющего большинства ресурсов такие описания сейчас отсутствуют. Это означает, что подобные описания должны быть составлены владельцами ресурсов. В связи с этим возникают вопросы о том, насколько легко это будет сделать владельцам ресурсов, и в какой степени они будут заинтересованы в проведении данной работы. Для решения первой части проблемы разрабатываются удобные интерфейсы для аннотации ресурсов и, там где это возможно, производится автоматическая генерация семантических метаданных. Вторая часть проблемы может быть решена, например, при помощи правил участия в виртуальной организации, требующих наличия соответствующего описания ресурса.

Во-вторых, как уже упоминалось, применяемые при обработке поисковых запросов машины вывода на логике описаний ориентированы на работу с классами и плохо масштабируются для поддержки сотен тысяч и миллионов экземпляров классов, что требуется для организации поиска среди большого числа ресурсов в Grid. В [46] описана система хранения экземпляров, позволяющая при некоторых ограничениях на онтологии и описания экземпляров решить эту проблему. Суть подхода заключается в использовании реляционной базы данных для хранения описаний экземпляров вместе с информацией об их положении в онтологической таксономии, полученной при помощи машины вывода. При обработке запросов в первую очередь используется информация из базы данных, и только в случае необходимости вызывается машина вывода.

5. Заключение

В заключение попытаемся на основе проведенного обзора сформулировать определение информационной инфраструктуры Grid. На наш взгляд, это понятие охватывает совокупность технологий, сервисов и формальных моделей, обеспечивающих эффективное управление информацией в динамичной гетерогенной среде Grid. В состав информационной инфраструктуры Grid входят:

- Базовые информационные сервисы, которые обеспечивают унифицированный доступ к информации, распределенной по множеству источников в Grid.
- Модели данных, информационные модели и онтологии, обеспечивающие формальную основу для согласованного представления информации о ресурсах и других сущностях Grid.
- Базы знаний, хранящие онтологии и другие знания о Grid и прикладных областях.
- Программные агенты, использующие семантические метаданные и базы знаний для предоставления высокоуровневых информационных сервисов, таких как автоматический подбор и композиция ресурсов для решения задач пользователей.

На текущий момент информационная инфраструктура Grid только начала формироваться. Ведется активная исследовательская работа и поэтапная разработка данной инфраструктуры, начатая с базовых информационных сервисов и продолжаемая на уровне семантических технологий. В конечном итоге эта инфраструктура должна повысить удобство и эффективность управления компонентами Grid, разработки Grid-приложений и решения в Grid прикладных задач.

Литература

1. *Лазарев И. В.*, *Сухорослов О. В.* Использование workflow-методологии для описания процесса распределенных вычислений // Проблемы вычислений в распределенной среде: Модели обработки и представления данных. Динамические системы / Труды ИСА РАН. М.: КомКнига/URSS, 2005. Т. 14. С. 26–70.
2. *Cooke A.*, *Gray A.*, *Nut W.*, *Cooke A.*, *Gray A.*, *Nutt W.*, *Cordenonsi R.*, *Byrom R.*, *Cornwall L.*, *Djaoui A.*, *Laurence Field*, *Fisher S.*, *Hicks S.*, *Leakey J.*, *Middleton R.*, *Wilson A.*, *Zhu X.*, *Podhorszki N.*, *Coghlan B.*, *Kenny S.*, *O'Callaghan D.*, *Ryan J.* The Relational Grid Monitoring Architecture: Mediating Information about the Grid // Journal of Grid Computing. Vol. 2. 2004.
3. *Cooke A.*, *Gray A.*, *Ma L.*, *Nutt W.*, *Magowan J.*, *Taylor P.*, *Byrom R.*, *Field L.*, *Hicks S.*, *Leake J.* R-GMA: An Information Integration System for Grid Monitoring // Proceedings of the 11th International Conference on Cooperative Information Systems, 2003.
4. *Tierney B.*, *Aydt R.*, *Gunter D.*, *Smith W.*, *Swany M.*, *Taylor V.*, *Wolski R.* A Grid Monitoring Architecture. GGF Performance Working Group, 2002 (www-didc.lbl.gov/GGFPERF/GMA-WG/papers/GWD-GP-16–2.pdf).
5. *Liu C.*, *Foster I.* A constraint language approach to grid resource selection // Proceedings of the Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC–12). June 2003.
6. *Liu C.*, *Yang L.*, *Foster I.*, *Angulo D.* Design and evaluation of a resource selection framework. In Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC–11). Edinburgh, Scottland. 2002.
7. *Chen L.*, *Shadbolt N.*, *Goble C.*, *Tao F.*, *Puleston C.*, *Cox S. J.* Semantics-Assisted Problem Solving on the Semantic Grid // Computational Intelligence. Vol. 21. № 2. 2005. P. 157–176.

8. *Chen L.*, *Shadbolt N. R.*, *Goble C.*, *Tao F.*, *Cox S. J.*, *Puleston C.* Managing Semantic Metadata for the Semantic Grid // Proceedings of Knowledge Grid and Grid Intelligence (KGGI) workshop. Beijing, China, 2004.
9. Common Information Model (CIM) Standards (www.dmtf.org/standards/cim).
10. Condor Project (www.cs.wisc.edu/condor).
11. *De Roure D.*, *Jennings N. R.*, *Shadbolt N. R.* The Semantic Grid: Past, Present, and Future // Proceedings of the IEEE. Vol. 93. Issue 3. March 2005. P. 669–681.
12. EGEE (Enabling Grids for E-sciencE) project (www.eu-egee.org).
13. *Foster I.*, *Kesselman C.*, *Tuecke S.* The Anatomy of the Grid: Enabling Scalable Virtual Organizations // International Journal of High Performance Computing Applications, 15 (3). 200–222. 2001.
14. gLite (http://glite.web.cern.ch/glite).
15. Globus Toolkit Homepage (http://globus.org/toolkit).
16. GLUE Schema (http://glueschema.forge.cnaf.infn.it).
17. Grid Interoperability project (www.grid-interoperability.org) (accessed Sept. 2004).
18. *Tangmunarunkit H.*, *Decker S.*, *Kesselman C.* Ontology-Based Resource Matching in the Grid — The Grid Meets the Semantic Web / D. Fensel, K. P. Sycara, J. Mylopoulos, editors, Proc. International Semantic Web Conference ISWC'2003. P. 706–737. Springer-Verlag, LNCS 2870, 2003.
19. *Foster I.*, *Kesselman C.*, *Nick J. M.*, *Tuecke S.* The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Open Grid Service Infrastructure WG. Global Grid Forum, June 2002.
20. *Brooke J.*, *Fellows D.*, *Garwood K.*, *Coble C.* Semantic matching of Grid Resource Descriptions / M. D. Dikaiakos, editor, Grid Computing. Second European Across-Grids Conference, AxGrids 2004, Nicosia, Cyprus, January 2004, Revised Papers. Vol. 3165 of Lecture Notes in Computer Science. P. 240–249. Springer 2004.
21. *Hau J.*, *Lee W.*, *Newhouse S.* Autonomic Service Adaptation in ICENI using Ontological Annotation // Proceedings of the 4th International Workshop on Grid Computing. P. 10–17. IEEE Computer Society, April 2003.
22. *Schopf J. M.*, *Raicu I.*, *Pearlman L.*, *Miller N.*, *Kesselman C.*, *Foster I.*, *D'Arcy M.* Monitoring and Discovery in a Web Services Framework: Functionality and Performance of Globus Toolkit MDS4. Preprint ANL/MCS-P1315–0106, Argonne National Laboratory, Argonne, IL, January 2006.
23. Job Submission Description Language (JSDL) Specification v1.0. Global Grid Forum, 2005 (www.gridforum.org/documents/GFD.56.pdf).
24. Job Submission Information Model (JSIM) v1.0. Global Grid Forum, 2004 (www.ggf.org/documents/GFD.28.pdf).
25. *Czajkowski K.*, *Fitzgerald S.*, *Foster I.*, *Kesselman C.* Grid Information Services for Distributed Resource Sharing // Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC–10). IEEE Press, August 2001 (www.globus.org/research/papers/MDS-HPDC.pdf).
26. *Gutirrez M. E.*, *Gmez-Prez A.*, *Garca O. M.*, *Terrazas B. V.* Ontology Access in Grids with WS-DAIOnt and the RDF(S) Realization // Proc. of the 3rd GGF Semantic Grid Workshop in conjunction with GGF 16. 2006. Athens; Greece. February 15.
27. *Sintek M.*, *Decker S.* Triple — an rdf query, inference, and transformation language. In Ian Horrocks and James Hendler, editors, Proc. of the 2002 International Semantic Web Conference (ISWC 2002). Number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

28. *Siddiqui M.*, *Fahringer T.*, *Hofer J.*, *Toma I.* Grid Resource Ontologies and Asymmetric Resource-Correlation. 2nd International Conference on Grid Service Engineering and Management (GSEM'05). 2005. Erfurt, Germany. September 19–22
29. myGrid UK e-Science Project (www.mygrid.org.uk).
30. *Corcho O.*, *Alper P.*, *Kotsiopoulos I.*, *Missier P.*, *Bechhofer S.*, *Goble C.* An overview of S-OGSA: A Reference Semantic Grid Architecture. Journal of Web Semantics. Vol. 4. Issue 2. June 2006. P. 102–115.
31. *van Harmelen F.*, *Hendler J.*, *Horrocks I.*, *McGuinness D. L.*, *Patel-Schneider P. F.*, *Stein L. A.* OWL Web Ontology Language Reference / Dean M., Schreiber G. (Editors). W3C Recommendation, 2004.
32. OWL-based Web Service Ontology (OWL-S) (www.daml.org/services/owl-s).
33. *Trunfio P.*, *Talia D.*, *Fragopoulou P.*, *Papadakis C.*, *Mordacchini M.*, *Pennanen M.*, *Popov K.*, *Vlassov V.*, *Haridi S.* Peer-to-Peer models for Resource Discovery on Grids. CoreGRID Technical Report. 2006. TR-0028. Mar 17.
34. *Raman R.*, *Livny M.*, *Solomon M.* Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching // Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing. Seattle, WA, June 2003.
35. *Raman R.* Matchmaking Frameworks for Distributed Resource Management. PhD thesis. University of Wisconsin-Madison, 2000.
36. *Wolski R.*, *Spring N.*, *Hayes J.* The Network Weather Service: A Distributed Resource Performance Forecasting Service in Metacomputing // Journal of Future Generation Computer Systems. 15 (5–6): 757–768. 1999.
37. RDF Vocabulary Description Language 1.0: RDF Schema (RDFS), Brickley D., Guha R. V. (Editors), W3C Recommendation, 2004 (www.w3.org/TR/rdf-schema).
38. RDF/XML Syntax Specification (Revised), W3C Recommendation, 2004 (www.w3.org/TR/rdf-syntax-grammar).
39. Resource Description Framework (RDF). W3C Semantic Web Activity (www.w3.org/RDF).
40. RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax (www.ietf.org/rfc/rfc2396.txt).
41. R-GMA: Relational Grid Monitoring Architecture (www.r-gma.org).
42. *Fikes R.*, *Hayes P.*, *Horrocks I.* OWL-QL- A Language for Deductive Query Answering on the SemanticWeb. KSL Technical Report 03–14, Stanford University, Stanford, CA, 2003.
43. *Majithia S.*, *Walker D. W.*, *Gray W. A.* Automated Composition of Semantic Grid Services. UK e-Science All Hands Meeting. Nottingham, UK, August 2004.
44. *Zanikolas S.*, *Sakellariou R.* A Taxonomy of Grid Monitoring Services. Future Generation Computer Systems, 21 (1):163–188, January 2005.
45. *Pahlevi S. M.*, *Kojima I.* S-MDS: A Semantic Information Service for Advanced Resource Discovery and Monitoring in WS-Resource Framework // Proc. of the 3rd GGF Semantic Grid Workshop in conjunction with GGF 16. Athens, Greece, February 15, 2006.
46. *Bechhofer S.*, *Horrocks I.*, *Turi D.* The OWL Instance Store: System Description. Proceedings of CADE-20. 2005. Tallinn, Estonia. July 22–27.
47. Semantic Web Activity, W3C (www.w3.org/2001/sw).
48. SPARQL Query Language for RDF, W3C Candidate Recommendation 6. April 2006 (www.w3.org/TR/rdf-sparql-query).
49. *Berners-Lee T.*, *Hendler J.*, *Lassila O.* The Semantic Web, Scientific Am. May 2001. P. 34–43.

50. The WS-Resource Framework. Czajkowski K., Ferguson D. F., Foster I., Frey J., Graham S., Sedukhin I., Snelling D., Tuecke S., Vambenepe W. 2004. March 5 (www.globus.org/wsrf/specs/ws-wsrf.pdf).
51. *Groothuyse T.* Information and Monitoring Systems for Grids: Divide or Unify? B. Sc thesis. Vrije Universiteit Amsterdam, 2004.
52. UNICORE (www.unicore.org).
53. *Yeong W.*, *Howes T.*, *Kille S.* Lightweight Directory Access Protocol. IETF, RFC 1777, 1995 (www.ietf.org/rfc/rfc1777.txt).
54. Web Service Modeling Ontology (www.wsmo.org).
55. *Xing W.*, *Dikaiakos M. D.*, *Sakellariou R.* A Core Grid Ontology for the Semantic Grid. In Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06). 2006. P. 178–184.
56. *Zhang X.*, *Schopf J. M.* Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2 // Proceedings of IEEE IPCCC International Workshop on Middleware Performance (IWMP 2004), 2004.
57. *Zhang X.*, *Freschl J. L.*, *Schopf J. M.* Scalability Analysis of Three Monitoring and Information Systems: MDS2, R-GMA and Hawkeye. Transactions on Parallel and Distributed Computing Journal. June 2006.