РАСПРЕДЕЛЕННЫЕ ИНФОРМ.-ВЫЧИСЛ. РЕСУРСЫ И МАТ. МОДЕЛИРОВАНИЕ
МКВМ-2004 РАБОЧИЕ СОВЕЩАНИЯ С. 95–102

Аспектно-ориентированный подход к созданию GRID приложений

Д.Ю. Новоселов*, С.П. Ковалев†

1. Введение

К настоящему времени недостаточно глубоко исследованы вопросы проектирования архитектуры приложений, предназначенных для выполнения в динамической гетерогенной среде GRID. В частности, нет подходов, позволяющих, основываясь на исходных требованиях к функциональности системы, преобразовывать их в модель приложения, эффективно использующего возможности распределенных вычислительных сетей, таких как GRID [1, 2].

По мнению ведущих IT аналитиков, один из способов создания новых методов в сегодняшних условиях является интеграция уже существующих концепций и применение технологий в необычных контекстах, где объединяются лучшие качества различных подходов. Наблюдается тенденция перехода от компонентов и сценариев к агентам и потокам работ [3]. Попытка применить аспектно-ориентированный подход (АОП) [4] для разработки GRID приложений – это одно из подтверждений существования такой тенденции.

Эффективное использование GRID систем возможно только тогда, когда приложения в достаточной степени интегрированы с инфраструктурой GRID и нижележащими уровнями. Важную роль при этом играет информация о структуре программы, об ее поведении, возможностях вычислительных и сетевых ресурсов, правах доступа к этим ресурсам пользователя, группы пользователей или организации. Возможности подхода АОП в контексте GRID систем открывают новые пути интегрирования приложений и распределенной GRID инфраструктуры.

В докладе обсуждаются вопросы, связанные с применением АОП для проектирования GRID приложений. В первой части приводится короткое введение в архитектуру GRID систем и идеологию АОП. В следующей части доклада рассматриваются вопросы применения АОП при создании функциональности GRID приложений. В заключении приведены дальнейшие планы развития применения АОП для создания GRID приложений.

2. Ключевые свойства GRID сервисов

В основе GRID систем лежит архитектура OGSA [1], которая предлагает подход для интеграции разнородных компьютерных ресурсов и организации доступа к ним со стороны виртуальных организаций [2]. Спецификация OGSA

*Новосибирский государственный университет.
†Институт вычислительных технологий СО РАН.

вводит понятие GRID сервисов и определяет, что это такое, их возможности и на какого рода технологиях они должны базироваться. Архитектура OGSA может быть реализована на основе протоколов, детальная спецификация которых приведена в рамках OGSI [1]. Эти протоколы позволяют решать задачи обеспечения безопасности, учета и выделения распределенных ресурсов. В свою очередь, программное обеспечение Globus Toolkit 3.0 [5, 6] предоставляет необходимые программные компоненты для реализации протоколов и спецификаций, описанных в OGSI и, таким образом, представляет собой один из возможных вариантов реализации OGSA или GRID системы.

В OGSA любая функциональность реализована посредством GRID сервиса [5, 2], который является расширением понятия Web сервис [7]. и отличается от него тем, что GRID сервис – это расширяемый Web сервис со статусом [5].

В следующей версии WSDL аналогичные расширения станут стандартом, и в дальнейшем они не потребуются для описания GRID сервисов.

Для выполнения запросов и передачи результатов используется SOAP, а в качестве транспорта протокол HTTP. Для индексирования GRID сервисов в OGSI применен свой собственный протокол, отличный от UDDI, который свойственен системам, где применяются Web сервисы [7].

Как и Web сервисы, GRID сервисы являются самоописательными. С помощью WSDL клиент может отправить запрос к сервису для получения списка поддерживаемых методов и способов обращения к ним.

Вдобавок, GWSDL сервисы обладают состоянием. Это возможно благодаря введению в описание GRID сервиса дополнительных атрибутов, которые отражают его состояние.

Для последующего изложения важным является то, что несколько клиентов могут одновременно использовать один и тот же GRID сервис. Это достигается наличием состояния у сервиса и введением фабрик. Работа происходит следующим образом: когда клиент хочет получить доступ к определенному GRID сервису, он посылает запрос к фабрике, которая создает экземпляр требуемого сервиса, и уже этот экземпляр используется в дальнейшем через механизм внутренней адресации GRID посредством GSR и GSH [5]. Система GRID осуществляет управление временем жизни экземпляра GRID сервиса и при достижении определенного времени он может освобождать ресурсы, занимаемые неиспользуемыми более экземплярами GRID сервисов.

Создание GRID приложений включает в себя создание GRID сервисов, которые заключают в себе некоторую функциональность, полезную в рамках решаемой задачи. Создание GRID сервиса предполагает следующие операции:

- Описание интерфейса сервиса GRID в терминах GWSDL.
- Программирование функциональности скрытой внутри GRID сервиса.
- Создание необходимых описаний для инсталляции GRID сервиса в GRID системе.

Все задачи программирования GRID сервиса, связанные с описанием WSDL документов и организацией обмена SOAP сообщениями, как правило, создаются автоматически. Таким образом, разработчику необходимо только написать соответствующие методы, которые реализуют специфичную для этого сервиса функциональность. Написание этих методов ничем не отличается от програм-

мирования обычных приложений, которые не предназначены для выполнения в среде GRID. Создаваемый GRID сервис может также использовать вызовы к другим GRID сервисам, обращение к которым ничем не отличается от вызова методов локальных объектов [6].

3. Аспектно-ориентированный подход

В традиционных подходах к задаче декомпозиции (разделения) системы на более мелкие модули, например, в ООП или процедурном подходе, разработчик оптимизирует вызовы процедур, используя имеющиеся абстракции. Довольно часто при применении ООП возникают ситуации, когда схожая функциональность присутствует в нескольких компонентах и не может быть локализована в одном. Разработчик вынужден отслеживать эту функциональность без применения какого-либо системного подхода. Это приводит к неоправданному увеличению объема и уровня сложности исходного кода приложения и, как следствие, к большему количеству ошибок.

Аспектно-ориентированный подход является развитием метода ООП, поскольку наряду с объектами, вводятся так называемые аспекты или потоки функциональности. Эти абстракции возникают в результате явления cross-cut, когда одинаковая функциональность проявляется в методах различных объектов и не поддается локализации средствами объектно-ориентированного подхода [4]. Такую функциональность еще называют сквозной.

При применении аспектно-ориентированного подхода исходный код программы генерируется на основе одной аспектной программы и одной или более объектной программы с помощью достаточно простого аспектного компилятора, называемого сшивателем (weaver).

Важным понятием на этапе генерации конечного кода программы являются точки сшивания, где происходит координация аспектной и объектной части программы. Точки сшивания могут быть определены неявно, например точкой сшивания могут быть места программы непосредственно перед вызовом методов всех классов или сразу после успешного завершения этих вызовов.

Сшивание может выполняться на стадии компиляции программы (Compile time Weaving) или при ее исполнении (Run Time Weaving). В задачу сшивателя не входит оптимизация кода. Аспектный подход возлагает это на разработчика, что позволяет достигать большего эффекта. Сквозные функциональности выделяются и описываются в разных ООП программах, затем они получают свое обозначение в рамках аспектной программы, где также описываются правила склеивания аспектов при различных вариантах композиции объектов.

Правила сшивки позволяют определять, как можно достичь более эффективного выполнения. Примером может служить программа, которая реализует фильтры для операций над графическими изображениями. Каждый из фильтров содержит в себе цикл по всем пикселям изображения. При необходимости последовательного вызова нескольких фильтров, появляется возможность выполнить цикл по всему изображению один раз, но для каждого пикселя будет выполнять композиция операций соответствующих набору необходимых фильтров. Если этот цикл выделить в сквозную функциональность или аспект, то сшиватель будет способен проводить оптимизацию подобного рода.

Аспектный подход позволяет сохранить читабельность исходного кода программы, и в тоже время обеспечить эффективность ее выполнения, что не всегда удается в рамках объектно-ориентированного или функционального подхода.

Фактически, аспектный подход дает возможность сформулировать большее количество метаинформации о поведении алгоритмов и свойствах объектов, чем при применении объектно-ориентированного или функционального подходов. В дальнейшем эта метаинформация применяется для осуществления оптимизации на этапе сшивания. Подобные цели могут быть достигнуты и при применении объектно-ориентированного подхода, но ценой усложнения исходного кода программы и большего количества ошибок.

Например, можно выделить обработку ошибок или ведение журнала во время выполнения программы в отдельный поток функциональности. Коммуникационные аспекты могут заниматься контролем объема данных, передаваемых по сети при запуске удаленных методов. Для этого аспектная программа должна принимать участие в реализации запуска методов, определяя локальные или удаленные методы, и осуществлять необходимый объем копирования в каждом случае, тем самым, обеспечивая оптимальное использование пропускной способности сети.

4. Разработка GRID приложений в парадигме АОП

Как говорилось выше, сквозная функциональность в АОП приложениях может быть реализована в терминах объектно-ориентированного подхода. Функциональность аспекта вплетается в приложение согласно тому, как это описано разработчиком в аспектной части программы. В контексте GRID появляется еще один способ реализации сквозной функциональности – это использование GRID сервисов, которые включают в себя интересующую функциональность. Это делает доступными для разработчика GRID приложений все преимущества применения АОП при проектировании и дальнейшей поддержке приложения и его исходного кода. В точках сшивания происходит вызов GRID сервиса, который реализует поток функциональности, соответствующий данному аспекту. При каждом выполнении аспектной программы можно определять набор аспектов, которые будут использоваться на этот раз, такую возможность дает Run-Time weaving. При выполнении одним из входящих параметров может становиться политика оптимизации, которую необходимо применить программе–сшивателю во время генерации кода для исполнения. Таким способом осуществляется управление изменяющимся набором аспектов, участвующих в работе приложения. Некоторые из аспектов могут не нести важной для приложения функциональности, а осуществлять действия, направленные на улучшение качества работы приложения или предоставлять дополнительную информацию о том, как выполняется основная функциональность. Например, аспект может заниматься измерением продолжительности тех или иных фаз работы сервиса или заниматься подготовкой данных, с которыми вскоре сервис будет работать, для того чтобы уменьшить задержки при доступе к данным. С другой стороны, обратная задача также может быть выделена в аспект, если окажется что выгоднее вычисления доставить поближе к данным.

АОП позволяет объединить слабо связанные (loosely coupled) объектные программы. А в GRID системах интегрируются слабо связанные узлы, взаимодействующие через GRID сервисы стандартного вида. Сервисы GRID обладают рядом свойств, которые полезны при реализации сквозных функциональностей.

Самоописательность GRID сервиса определяется стандартом GWSDL. Это позволяет "узнавать" у GRID сервиса о том, каким образом осуществлять запросы к нему и получать результаты. Это играет большую роль для обеспечения возможности переиспользования однажды созданных GRID сервисов при разработке последующих приложений. Взаимодействие GRID сервисов по стандартным протоколам позволяет достичь высокого уровня платформонезависимости реализации GRID сервисов. Например, программа, написанная на Java и выполняющаяся на платформе Windows, может использовать GRID сервис, созданный на C++ для операционной системы Linux.

Наличие состояния GRID сервиса важно для реализации многих сквозных функциональностей, которые связанны с управлением какими-либо перманентными структурами данных, например, при управлении различных КЭШей или измерении параметров во время исполнения программы.

Существуют также механизмы асинхронного взаимодействия GRID сервисов при помощи сообщений. Приложение или GRID сервис могут "подписаться" на получение сообщений от экземпляра какого-либо сервиса. Подобные возможности полезны при исследовании работы алгоритма во время его выполнения.

Использование аспектов для создания GRID приложений позволяет явным образом сохранять метаинформацию о структуре алгоритмов еще на этапе проектирования и дает возможность использовать ее для оптимального выполнения программы, освобождая компиляторы и сшиватели кода от этих функций. Кроме того, при применении традиционных подходов к проектированию приложений оптимизация на уровне компилятора либо представляет значительные трудности, либо попросту невозможна из-за распределенной природы приложения. Аспекты предоставляют последовательный и строгий путь сохранения этой метаинформации, по сравнению с подходом, предложенным в [8], где исследуется возможность расширения объектно-ориентированного подхода с тем, чтобы эффективно решать задачи оптимизации выполнения приложения. Такое расширение основано на организации хранилища метаинформации о структуре и поведении алгоритмов. Эта метаинформация сохраняется в виде нескольких различных реализаций одной и той же функциональности. В зависимости от параметра оптимизации, при выполнении программы используется та или иная версия компонентов приложения. Авторы преподносят это как эволюцию объектно-ориентированного подхода в некий иной метод, где базовой абстракцией является сервис [5], который можно переиспользовать.

В качестве примера применения АОП для конструирования GRID приложений рассмотрим GRID сервис, который занимается регистрацией времени, затрачиваемого приложением на выполнение работы между обращениями к этому сервису. После окончания выполнения приложения этот сервис предоставляет временную развертку этапов работы приложения. Для ситуации, когда приложение выполняется в несколько потоков, подобный GRID сервис мо-

жет предоставлять статистику, как по отдельным потокам исполнения, так и для всего приложения.

Представление задачи измерения времени как аспект, позволяет легко отключать эту функциональность, когда требуется. Например, применять ее только при отладке приложения и определения узких мест программы, требующих доработки. Отказ от использования этого аспекта не приведет к необходимости исправления исходного кода программы и, таким образом, безопасен для ее основной функциональности и не потребует дальнейшей отладки и тестирования. Кроме того, этот аспект может быть переиспользован в дальнейшем при отладке и других приложений. Применение WSDL для описания интерфейса сервиса, соответствующего аспекту, делает разработчика приложения независимым от того, как происходит обращение к аспекту (GRID сервису), и тем более, от того, как он реализован внутри. Примененная в GRID системах концепция фабрик позволяет использовать один и тот же тип сервиса несколькими приложениями одновременно совершенно изолированным образом, поскольку каждое из приложений использует свой собственный экземпляр GRID сервиса, предоставляемого фабрикой этого сервиса. Взаимодействие параллельных потоков вычисления происходит через механизм сообщений. Сообщения можно рассматривать как точки сшивания аспектной программы [5, 4].

Функциональная декомпозиция GRID приложения приводит к появлению сквозных функциональностей, которые могут быть разделены на несущие в себе специфику предметной области приложения и на те, которые свойственны большинству GRID приложений. Возможность переиспользования аспектов первого типа весьма ограничена и вероятна только при построении приложения в близких предметных областях. Аспекты второго типа применимы всегда, когда речь идет о GRID приложениях.

Дальнейшая классификация аспектов проводится на основе типичных задач, которые решаются с помощью EU DataGrid и описаны в [9].

Специфичные для GRID аспекты предназначены главным образом для анализа процесса выполнения и оптимизирования работы приложения в GRID системе. Они служат для передачи GRID подуровню информации о характере поведения приложения, которая может быть им использована для принятия решений о выделении тех или иных ресурсов GRID для приложения. Иными словами, осуществляют более тесную и эффективную интеграцию приложения с GRID системой.

- Для анализа процесса выполнения приложения в GRID системе можно ввести некоторый аспект, который позволяет отслеживать последовательность вызова определенных процедур. Таким образом, есть возможность, во-первых, получить информацию о задержках при вызове сервисов, вести журналирование событий, происходящих во время выполнения. Во-вторых, на основании этих данных можно проводить анализ эффективности архитектуры приложения, политик оптимизации; проводить отладку и более тонкий учет потребляемых ресурсов, чем тот который предоставляет GRID уровень. В-третьих, речь может идти о прогнозировании времени выполнения приложения и адаптации поведения менеджера GRID ресурсов к реальной загрузке системы.

- Повышение эффективности использования ресурсов GRID системы, путем проведения различного рода кэширования. Так применение GRID сервиса, который позволяет запоминать событие, когда сервис X обработал данные Y и был получен результат Z, может избавить от необходимости многократной обработки одних и тех же данных одним и тем же алгоритмом. Что приведет к заметной экономии ресурсов и уменьшению времени отклика системы. Безусловно, подобный подход должен применяться для сервисов, результат работы которых является только функцией входных данных.
- Интеллектуальный доступ к полям объекта данных. Этот аспект может следить за тем, чтобы производилось копирование только тех атрибутов объекта, которые необходимы для работы приложения. Например, при регистрации новой публикации в электронной библиотеке неизбежно копирование всего объекта в базу данных, но при запросе на чтение подмножества атрибутов объекта нет необходимости копировать весь объект целиком, а только набор требуемых атрибутов.
- Можно выделить аспекты, которые добавляют метаинформацию к объектам данных, связанную, к примеру, с вычислительными затратами на его получение или обработку.
- Аспекты могут предоставлять более изощренную политику безопасности, чем та, которая используется в GRID. Речь может идти о запрете доступа к некоторым полям, как может быть необходимо в медицинской аналитической системе, где имена пациентов должны быть недоступны для всех медицинских аналитиков, кроме ограниченного сообщества лечащих врачей.
- С помощью аспектов удобно решать задачу преобразования формата данных прозрачно для процедур обработки этих данных.

Перечисленные выше классы аспектов, не зависят от специфики предметной области приложения и могут быть использованы для обеспечения более тесной интеграции приложения и GRID.

5. Заключение

Верификация применения аспектно-ориентированного подхода для разработки GRID приложений и сервисов, оценка его эффективности по сравнению с ООП и функциональным подходом производятся в рамках тестовой GRID системы на базе Globus Toolkit v3.0.

В качестве языка программирования применяется Java и его аспектная модификация AspectJ.

Тестовое приложение рассчитано на интенсивную работу с данными, которые распределены в GRID системе. С помощью служебных аспектов производятся измерения задержек и анализ последовательности, в которой происходят вызовы методов, принадлежащих тем или иным объектам. На основании этих данных делается вывод о целесообразности применения тех или иных политик оптимизации на различных шаблонах доступа к данным.

Список литературы

[1] Foster I., Kesselman C., Nick J., Tuecke S. The physiology of the Grid. The Globus Alliance. – http://www.globus.org/research/papers/ogsa.pdf.

[2] Foster I., Kesselman C., Tueckle S. The anatomy of the GRID. The Globus Alliance // Int. J. Supercomputer Applications. – 2001. – http://www.globus.org/research/papers/anatomy.pdf.

[3] Software [R]evolution / A. Rountable, L. Wall, D. Taylor, C. Horn, P. Bassett, P. Ousterhout, M. Griss, R. Soley, J. Waldo, C. Simonyi // IEEE Internet Computer. – May 1999. – P. 48–57.

[4] Aspect-oriented programming / G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, J. Irwin. – Springer-Verlag, 1997.

[5] Tuecke S., Czajkowski K., Foster I., et al. Open Grid Services Infrastructure (OGSI). Version 1.0. – Global Grid Forum, 2003. – http://www.ggf.org/ogsi-wg/.

[6] Sotomayor B. The globus toolkit 3 programmer's tutorial. – March, 2004. – http://www.casa-sotomayor.net/gt3-tutorial/.

[7] Gardner T. An Introduction to Web Services. – http://www.ariadne.ac.uk/issue29/gardner/.

[8] Furmento N., Mayer A., McGough S., et al. Optimization of component-based applications within a Grid environment. – Denver, 2001.

[9] The DataGrid architecture // CERN. Information Society Technology. – 2004. – P. 8–12.