УДК 681.007.05

Р.В. Гребинник, О.В. Липанов

Харьковский национальный университет радиоэлектроники, Харьков

МОДЕЛИРОВАНИЕ SOA СИСТЕМ С ЦЕЛЬЮ ОПТИМИЗАЦИИ ИХ АРХИТЕКТУРЫ

В статье проводится анализ подходов к проектированию оптимальных сервис-ориентированных архитектур и проводится разработка математического аппарата необходимого для моделирования и последующей оптимизации архитектуры сервис-ориентированной системы. Разрабатываемый математический аппарат позволит разработать алгоритмы и системы оптимизации сервис-ориентированных систем.

Ключевые слова: *объект, класс, SOA, метод, архитектура, модель, статистика, система массового обслуживания.*

Введение

Сервис-ориентированная архитектура (SOA, service-oriented architecture) – модульный подход к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами.

В основе SOA лежат принципы многократного использования функциональных элементов информационных технологий, ликвидации дублирования функциональности в ПО, унификации типовых операционных процессов, обеспечения перевода операционной модели компании на централизованные процессы и функциональную организацию на основе промышленной платформы интеграции.

Компоненты программы могут быть распределены по разным узлам сети, и предлагаются как независимые, слабо связанные, заменяемые сервисы-приложения. Программные комплексы, разработанные в соответствии с SOA, часто реализуются как набор веб-сервисов, интегрированных при помощи известных стандартных протоколов (SOAP и т.п.).

Интерфейс компонентов SOA-программы предоставляет инкапсуляцию деталей реализации конкретного компонента (операционной системы, платформы, языка программирования, и т.п.) от остальных компонентов.

Таким образом, SOA предоставляет гибкий и элегантный способ комбинирования и многократного использования компонентов для построения сложных распределённых программных комплексов. SOA хорошо зарекомендовала себя для построения крупных корпоративных программных приложений.

Целый ряд разработчиков и интеграторов предлагают инструменты и решения на основе SOA (например, платформы Intel SOAExpressway, JBoss SOA Platform, IBMWebSphere, SoftwareAG webMethods, Oracle/BEAAqualogic, Microsoft Windows Communication Foundation, SAPNetWeaver, TIBCO).

1.1. Сервис-ориентированное программирование

Архитектура, как таковая, не привязана к какой-то определенной технологии. Организация системы независима от используемой вычислительной платформы (платформ) и независима от применяемых языков программирования. Проектируются сервисы, что не зависят от конкретных приложений, с одинаковыми интерфейсами доступа к ним. Организация сервисов стоится, как система слабосвязанных или не связных компонентов. Главное, что отличает SOA – это использование независимых сервисов с четко определенными интерфейсами, которые для выполнения своих задач могут быть вызваны некоторым стандартным способом, при условии, что сервисы заранее ничего не знают о приложении, которое их вызывает, а приложение не знает, каким образом сервисы выполняют свою задачу.

SOA также может рассматриваться как стиль архитектуры информационных систем, который позволяет создавать приложения, построенные путем комбинации слабосвязанных и взаимодействующих сервисов. Эти сервисы взаимодействуют на основе какого-либо строго определенного независимого интерфейса (например WSDL). Определение интерфейса скрывает языково-зависимую реализацию сервиса. Появление сервис-ориентированного подхода сделало очередную реформу в теории разработки программного обеспечения. Ряд архитектурных особенностей SOA позволяет уменьшить степень связанности различных элементов системы. Для взаимодействия компонентов используется сравнительно небольшой набор простых интерфейсов, которые имеют только общую семантику и доступны всем провайдерам и потребителям. Через эти интерфейсы передаются сообщения, ограничены некоторым словарем. А поскольку данные - только общая структура корпоративной системы, то вся семантика и бизнес-логика, специфичная для приложений, описывается непосредственно в этих сообщениях.

1.2. Практические аспекты применения SOA

Практические аспекты сервис-ориентированной технологии позволяют решить проблемы масштабируемости, интегрировать сети передачи данных и голоса, упростить процедуры проектирования и управления сетями, а также создать другие распределенные приложения, прозрачно взаимодействующие с ресурсами систем с помощью прикладных программных интерфейсов и открытых стандартов.

Программные комплексы компаний Microsoft, SAP, ORACLE и других производителей являются наборами модулей для отдельных задач. У них также существуют проблемы интеграции, как внутри этих условно "монолитных" систем, так и с внешними системами других производителей.

Все представленные решения не "живут" в динамике изменения требований к объединенным информационным системам, не поддерживающих жизненный цикл функциональных систем, требуют колоссальных затрат ресурсов, не учитывают фактор единого времени в поддержании целостности и сопоставимости этих различных интегрированных систем, не эффективны при большом количестве систем.

Появление концепции SOA ознаменовало собой новый виток интеграционных технологий и связанных с ними надежд. Вобрав в себя технологические достижения компонентного программирования и Web-сервисов, SOA предлагает возможность гибкой работы с элементами бизнес-процессов и ИТ- инфраструктурой, лежащей в их основе, как с компонентами, которые можно использовать многократно и гибко комбинировать при изменении требований бизнеса. Благодаря этому SOA способна фундаментальным образом изменить подходы, применяемые в разработке и внедрении программных систем для бизнеса, обеспечивая преимущество в конкурентной борьбе.

Для организации взаимодействия сервисов нужна среда, которая обеспечит динамическую маршрутизацию запросов от прикладного компонента – потребителя сервиса и получения результатов от приложения – провайдера сервиса. Для этого может потребоваться поддержка синхронных и асинхронных коммуникаций низкого уровня между приложениями, трансформация и высокоскоростной распределение данных, трансляция протоколов, кэширование функций Web-сервисов, виртуализация ввода / вывода и т. д.

Конечная цель SOA – обеспечить представление бизнес-процессов, как взаимодействующих сервисов. Средства управления бизнес-процессами обеспечивают интеграцию в нужной последовательности сервисов, которые могут быть как локально - реализованными в ИТ- инфраструктуре компании, так и удаленными, если процесс на определенных этапах обращается к ресурсам партнерских компаний.

Проекты, созданные на базе SOA, – сложные методично, масштабные, затрагивающие основные бизнес-процессы. Кроме этого, задача планирования оптимальной SOA архитектуры не решена в общем виде. Уровень взаимодействия сервисов и компонентов системы сугубо индивидуален и зависит от требований и конкретной реализации системы.

Задача оптимизации взаимодействия компонентов сложной сервис-ориентированной системы не может быть решена в общем виде, потому стоит задача планирования и распределения ресурсов масштабируемой системы. Фактически, вся сложность реализации архитектуры ложится на плечи разработчиков и невозможно дать полноценный анализ или подход к оптимизации. Предлагается создание набора правил и методов для анализа и распределения ресурсов вычислительной среды SOA.

2.1. Анализ оптимальной SOA архитектуры

В сервис-ориентированной архитектуре все достаточно просто организовано: есть поставщики сервисов (приложения), создан реестр, где публикуется информация о поставщиках и куда заходит потребитель, для того, чтобы выбрать необходимый сервис. Потребитель получает нужную ссылку и связывается с поставщиком, чтобы передать запрос на обработку данных и получить результаты. Веб архитектура предполагает такое взаимодействие через Интернет или локальную сеть: например, сервисы через реестр сервисов публикуются в сети (курсы валют, погода, котировки акций). По мере реализации проекта построения интеграционного решения на базе SOA происходит публикация сервисов в реестре, и когда приходит время "связывать" приложения в рамках какого-то бизнес-процесса, это реализуется достаточно просто, потому что все осуществляется централизованно, через реестр сервисов. Таким образом, SOA дает возможность убрать лишние продукты и интеграционные решения, наглядно представить взаимодействия систем, оперативно конфигурировать решение при изменении бизнес-процессов и др.

Уровень взаимодействия сервисов и компонентов системы сугубо индивидуален и зависит от требований и конкретной реализации системы. Единственным существенным требованием есть минимизация связности между сервисами, т е требование разделить логику работы каждого сервиса и сделать их монолитными. Формально, критерий уменьшения связности имеет первоочередный характер в процессе разработки облачной системы, но с другой стороны он не представляет собой аксиоматическую составляющую сервис-ориентированной системы.

Как известно, уровень взаимодействия сервисов носит нерегулярный характер, т.к. зачастую получение данных жестко зависит от предварительного их добавления в серверную систему. Эти зависимости приводят к формированию не только контрактов между клиентом и сервером, но и созданию набора правил взаимодействия компонентов системы. Таким образом, можно сформировать первую аксиому в разработке сервис-ориентированных систем:

Аксиома 1: (о связности сервисов): Сложность представления данных на стороне сервера увеличивает не только количество контрактов между клиентом и сервером, но и повышает уровень связности сервисов.

Определение 1: Контракт – это связь клиентского приложения с сервисом.

Между клиентом и сервером может быть бесконечное количество контрактов. Чем сложнее приложение, тем больше контрактов заключается между клиентом и сервером, что в свою очередь повышает уровень нагрузки на систему т.е.

$$\sum_{i=0}^{N} C_i \to \infty , \qquad (1)$$

где N – это количество контрактов между клиентом и сервером, C – контракт между клиентом и сервером.

Если количество контрактов между клиентом и сервером растет бесконечно, то количество контрактов, которые обслуживаются сервером, растет в геометрической прогрессии. Т.к. количество контрактов напрямую связано с количеством клиентов, которые взаимодействуют с сервером – глобальное количество контрактов может быть выражено в виде:

$$\sum_{j=1}^{M}\sum_{i=1}^{N} C_i , \qquad (2)$$

где M – это количество клиентов, которые связаны с сервером, N – количество контрактов между клиентом и сервером.

Определение 2: Фактический уровень связности сервиса – это количество других сервисов, с которыми он должен взаимодействовать для выполнения своей задачи.

Архитектура сервис-ориентированной системы должна быть организована таким образом, чтобы фактический уровень связности сервиса был равен 1. Т.е. сервисы должны быть независимы друг от друга. Фактический уровень связности сервисов SOA системы можно записать в виде:

$$\sum_{i=1}^{N} R_i \to N , \qquad (3)$$

где N – это количество сервисов системы.

Пример связности двух сервисов: Предположим, что есть сервис A, что выдает информацию о заработной плате сотрудников и сервис B, что возвращает количество часов, отработанных сотрудником в этом месяце. Если сервис A для расчета зарплаты вызывает сервис B – фактический уровень связности сервисов повышается на 1.

Определение 3: Относительный уровень связности сервисов – это уровень логического взаимодействия сервисов, в котором для выполнения запроса к одному сервису, клиент должен предварительно получить данные с другого сервиса.

Относительный уровень связности сервисов абсолютно неконтролируемый и связан непосредственно с техническими требованиями проекта. К примеру, связь между сервисом A, что возвращает Id пользователя по введенному логину и паролю и сервисом B, что считает размер ЗП пользователя по его персональному Id–будет относительной.

Правило связности: Проектируя SOA систему нужно максимально снизить уровень связности сервисов и привести реальную связность к относительной связности.

Исходя из аксиомы 1 или аксиомы о связности, мы можем сделать окончательный вывод, что уровень зависимости сервисов зависит от полученных требований к разработке системы и выполнение критерия связности выполнимо только в самом общем виде.

Критерий связности:

$$Q_N = \sum_{i=1}^{N} R_i , \qquad (4)$$

$$Q_N \to N , \qquad (5)$$

где N – это количество сервисов в системе, R_i – уровень связности i-го сервиса [1, N].

Критерий связности представляет собой основной постулат разработки сервис-ориентированной системы на данном этапе развития SOA. Фактически, критерий связности и есть основным критерием оптимальности; их взаимная дополняемость и масштабируемость являются лишь уровнем профессионализма команды разработчиков. Основополагающим фактором взаимодействия различных систем есть web запрос, что выполняется асинхронно для относительно удаленного приложения и независимо относительно сервера/облака. Конечно, иногда требуются уникальные решения, когда сервер может запрашивать дополнительную информацию у клиента. Такие системы называются двунаправленными, они характеризуются большей связностью и меньшим порогом устойчивости.

На основании всего сказанного, можно выделить следующие факторы, которые формируют качество и оптимальность SOA системы:

1. Оптимальность ООП и аспектно-ориентированной архитектуры системы.

2. Уровень связности сервисов или их взаимной независимости.

3. Оптимизация web-запросов (ws* группа, tcp, http rest ful и т д).

4. Количество ресурсов SOA-сервера или облака.

Как можно увидеть из имеющихся факторов, оптимизация SOA не имеет общих критериев, а только обобщающие факторы. Все это приводит нас к математическому обоснованию теории сервис-ориентированных систем, где мы можем выделить следующие задачи:

1. Математическое представление SOA системы.
2. Уравнение сбалансированности системы.
3. Вероятностная оценка состояния системы или конкретного узла.
4. Общее уравнение сбалансированности системы.
5. Математический аппарат для прогнозирования и оценки облачного сервера.

2.2. Уравнение сбалансированности системы

Представим SOA систему как сетку M×N:

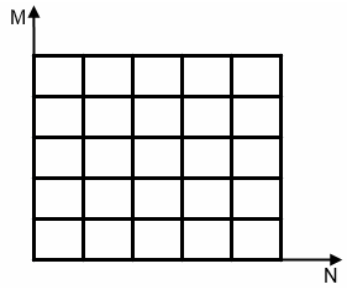

Рис. 1. Сетка SOA системы

Каждая ячейка представляет собой независимый сервис. Распределение нагрузки, показанное на рис. 1, считается идеальным состоянием, что представляет собой эталонный этап планирования сервис-ориентированной системы. В данной формулировке вводится понятие давления.

Определение 4: Давление или уровень нагрузки конкретного сервиса на систему – это количество ресурсов, потребляемое сервисом от общего количества ресурсов сервера или кластера.

Представим уровень нагрузки сервиса на сервер в координатах M×N.

$$df_{N,M} = C_{i,j} ds \left|\overline{Q_{i,j}}\right|, \qquad (6)$$

где $Q_{i,j}$ – вектор нагрузки на систему; $C_{i,j}$ – координаты сервиса в пространстве M×N; ds – общий ресурс SOA системы.

Исходя из уравнения (6) попытаемся выразить общую нагрузку на систему и представить состояние сбалансированности. Понятно, что если $\vec{Q}_{i,j}$ – это вектор нагрузки на систему, то сумма этих векторов даст нам общую нагрузку на сервер. Следовательно, общую нагрузку на систему M×N можно выразить в виде:

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\left|\vec{Q}_{i,j}\right|, \qquad (7)$$

где $Q_{i,j}$ – вектор нагрузки на систему; i, j – координаты сервиса в пространстве M×N.

Можно предположить, что идеальным будет состояние, в котором все ресурсы сервера распределены между всеми сервисами, и мы получим состояние, в котором все сервисы получают необходимое количество ресурсов, но и сервер использует свой потенциал по максимуму. Таким образом, можно получить уравнение сбалансированной нагрузки на систему:

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\left|\vec{Q}_{i,j}\right| = 1, \qquad (8)$$

где $Q_{i,j}$ – вектор нагрузки на систему; i, j – координаты сервиса в пространстве M×N. На рис. 1 представлена равно-распределённая система, в которой каждый отдельный сервис оказывает одинаковый уровень нагрузки на сервер. Не трудно понять, что подобное распределение нагрузки дает возможность безошибочного прогнозирования необходимых ресурсов, а также обеспечивает максимальный уровень отказоустойчивости SOA решения.

Выразим уровень нагрузки (давления) сервиса i,j в пространстве M×N. Предположим, что сервис i,j принадлежит SOA системе ds, тогда нагрузка, порождаемая сервисом i,j, выражается через $C_{i,j}$, что в свою очередь является подмножеством сервис-элементов, составляющих пространство, указанное на рис. 1. Графическое отображение SOA системы дает возможность выразить модуль давления сервиса относительно его пространственных координат в рассматриваемой среде, что позволяет указать зависимость между уровнем потребляемых ресурсов и конкретным сервисом (i, j).

$$C_{i,j} = \left|ds_{i+1} - ds_{i-1}\right| \times \left|ds_{i+1} - ds_{i-1}\right|. \qquad (9)$$

Основная задача состоит в сбалансированности системы. Если сервис требует увеличения ресурсов, то изменение производится за счет изменения размера ячейки (i, j), то есть увеличения модуля (9), как видно на рис. 2.

Как видно из рис. 2, изменение потребляемых ресурсов одним сервисом изменяет размер ресурсов доступных другим сервисам. Основная задача, что стоит перед нами – это минимизировать ситуации, в которых количество запрашиваемых ресурсов будет превышать общие характеристики системы. Конечно, любое управление планировщиками полностью ложится на плечи операционной системы. В данной ситуации мы можем только контролировать уровень нагрузки на систему, ввести очереди и приоритеты запросов, а также информировать поставщиков сервисов о том, что запрашиваемый объём ресурсов в скорости не будет соответствовать уровню запрашиваемых, что, в свою очередь, может снизить производительность SOA решения на X%.

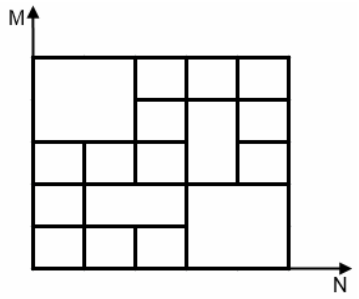

Рис. 2. Изменения размера ячейки

Данные условия приводят нас к формированию основного уравнения сбалансированности системы, которое выглядит следующим образом:

$$\frac{\sum_{i=1}^{M}\sum_{j=1}^{N} C_{i,j}ds\left|\vec{Q}\right|_{i,j}}{\sum_{i=1}^{M}\sum_{j=1}^{N} C_{i,j}ds} = 1, M > 0, N > 0, C_{i,j}ds > 0, \quad (10)$$

где i, j – координаты сервиса в пространстве M×N; $C_{i,j}$ – уровень нагрузки сервиса i, j в пространстве M×N, ds – общий ресурс SOA системы. Уравнение сбалансированности учитывает, что пространство M×N не может быть пустым, т е в системе существует хотя бы один сервис, а так же условие того, что даже на неиспользуемый сервис сервер выделяет отличное от нуля количество ресурсов для поддержки его существования. Фактически это уравнение идеального равновесия системы SOA. Сейчас переопределим уравнение (10) для определения граничного условия или стремления к уровню сбалансированности справа:

$$\left|\frac{\sum_{i=1}^{M}\sum_{j=1}^{N} C_{i,j}ds\left|\vec{Q}\right|_{i,j}}{\sum_{i=1}^{M}\sum_{j=1}^{N} C_{i,j}ds} - 1\right| \to \min, \quad (11)$$

$$M > 0, \quad N > 0, \quad C_{i,j}ds > 0.$$

Как видно из (11) – это условие позволяет нам определять уровень стабильности системы в общем, а главное делать прогнозы на основании введения статистики для этого распределения. Следовательно, чем больший объём статистических данных, накопленных во время работы сервисов, тем выше вероятность точного прогноза относительно уровня запрашиваемых ресурсов со стороны клиентов.

Выводы

Проведенный анализ сервис-ориентированной архитектуры и подходы к ее разработке дали возможность в формальном виде записать основные законы существования SOA системы и ввести некоторые основополагающие определения, которые используются для определения систем уравнений, описывающих облачную систему. Полученные математические выкладки дают возможность анализировать состояние сервера и делать предположения о его стабильности или нестабильности и качестве архитектурных решений, что использовались при проектировании системы. Фактически, был получен базовый набор определений и законов, которые в дальнейшем помогут разработать полноценную систему анализа сервис-орентированных систем, сделают их более надежными и стабильными.

Список литературы

1. Stephen Bennett. SOA Governance: Governing Shared Services On - Premise & in the Cloud / Stephen Bennett, Thomas Erl, Clive Gee. – Hardcover, 2012. – 675 p.

2. David Chou. SOA with .NET & Windows Azure: Realizing Service – Orientation with the Microsoft Platform / David Chou, John deVadoss, Thomas Erl. – Hardcover, 2012. – 893 p.

3. Thomas Erl. SOA Design Patterns / Thomas Erl. – NewYork: Hardcover, 2011. – 865 p.

4. Thomas Erl. Web Service Contract Design & Versioning for SOA / Thomas Erl, Anish Karmarkar, Priscilla Walmsley. – Hardcover, 2011. – 826 p.

5. Thomas Erl. SOA Principles of Service Design / Thomas Erl. – NewYork: Hardcover, 2010. – 573 p.

6. Service Oriented Architecture (SOA): метод SURF [Электронный ресурс] / Сетевой журнал. – Режим доступа к журналу: www/URL: http://msdn.microsoft.com/en-us/library/bb833022.aspx. – Загл. с экрана

7.What Is Service-Oriented Architecture: метод SURF [Электронный ресурс] / Сетевой журнал. – Режим доступа к журналу: www/URL: http://www.xml.com/pub/a/ws/2003/09/30/soa.html. – 09.30.2003. – Загл. с экрана.

Поступила в редколлегию 8.04.2014

Рецензент: д-р техн. наук, проф. И.В. Рубан, Харьковский университет Воздушных Сил им. И. Кожедуба, Харьков.

МОДЕЛЮВАННЯ SOA СИСТЕМ З МЕТОЮ ОПТИМІЗАЦІЇ ЇХ АРХІТЕКТУРИ

Р.В. Гребінник, О.В. Ліпанов

У статті проводиться аналіз підходів до проектування оптимальної сервіс-орієнтованої архітектури і проводиться розробка математичного апарату, необхідного для моделювання і подальшої оптимізації архітектури сервіс-орієнтованої системи. Математичний апарат, що розробляється, дозволить розробити алгоритми і системи оптимізації сервіс-орієнтованих систем.

Ключові слова: *об'єкт, клас, SOA, метод, архітектура, модель, статистика, система масового обслуговування.*

MODELING SOA SYSTEMS TO OPTIMIZE THEIR ARCHITECTURE

R.V. Grebinnyk, O.V. Lipanov

This paper contains development of mathematical model for service oriented architectures and developing approaches for its optimization. Research and create their own mathematical tools for optimizing the workflow of complex service-oriented system. Optimizing workflow of independent services performed by the mathematical apparatus of statistics and queuing theory.

Keywords: *domain objects, object, class, SOA, architecture, methods, models, statistics, queuing system.*