Кондратьев Павел Сергеевич

УлГТУ ФИСТ, ИВТАСмд-11

СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА

Сервис-ориентированная архитектура (SOA, англ. service-oriented architecture) – модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам.

Программные комплексы, разработанные в соответствии с сервис-ориентированной архитектурой, обычно реализуются как набор веб-служб, взаимодействующих по протоколу SOAP, но существуют и другие реализации (например, на основе REST).

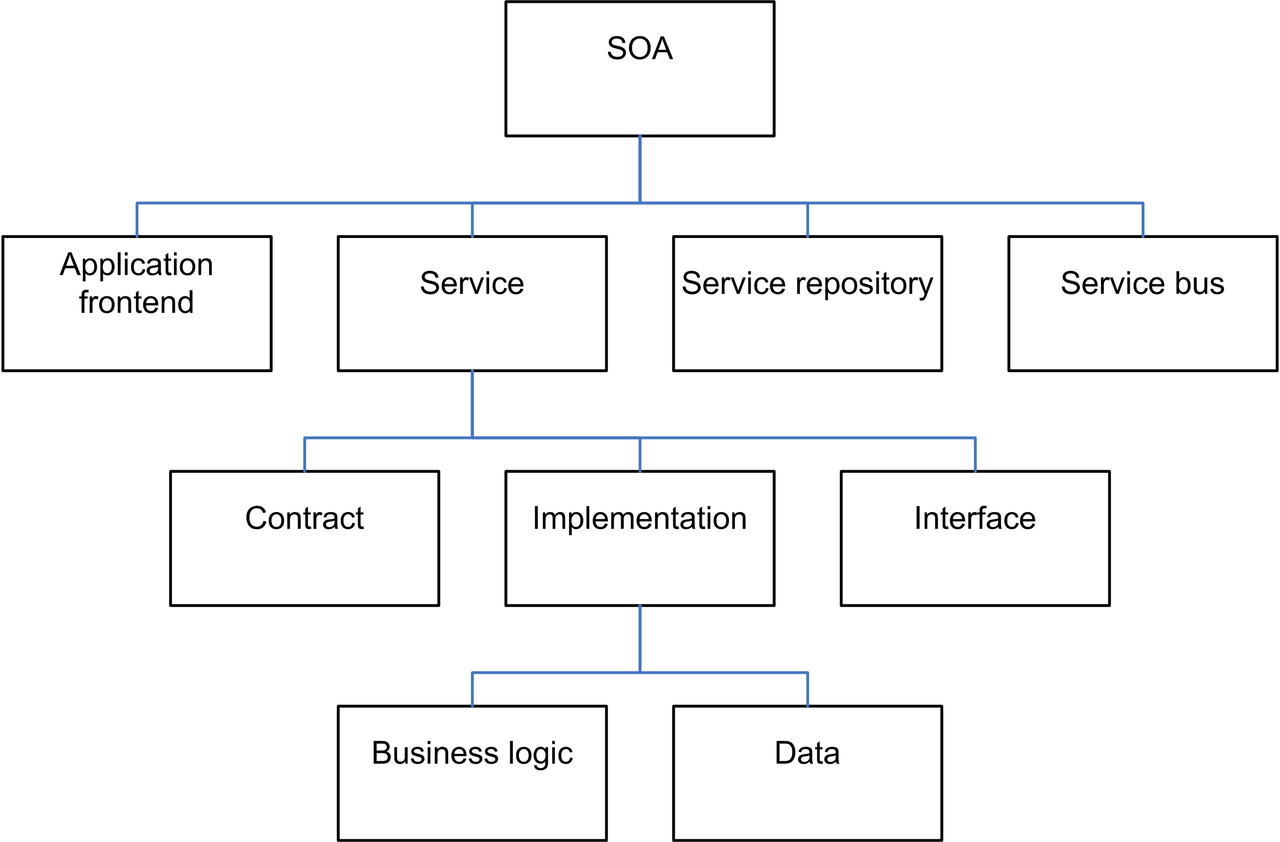


Рисунок 1. Элементы сервис-ориентированной архитектуры, по: Dirk Krafzig, Karl Banke, and Dirk Slama. Enterprise SOA. Prentice Hall, 2005

Интерфейсы компонентов в сервис-ориентированной архитектуре инкапсулируют детали реализации (операционную систему, платформу, язык программирования) от остальных компонентов, таким образом обеспечивая комбинирование и многократное использование компонентов для построения сложных распределённых программных комплексов, обеспечивая независимость от используемых платформ и инструментов разработки, способствуя масштабируемости и управляемости создаваемых систем.

Архитектура не привязана к какой-либо определённой технологии. Она может быть реализована с использованием широкого спектра технологий, включая такие технологии как REST, RPC, DCOM, CORBA или веб-сервисы. SOA может быть реализована, используя один из этих протоколов и, например, может использовать дополнительно механизм файловой системы для обмена данными.

Главное, что отличает SOA – это использование независимых сервисов с чётко определёнными интерфейсами, которые для выполнения своих задач могут быть вызваны неким стандартным способом, при условии, что сервисы заранее ничего не знают о приложении, которое их вызовет, а приложение не знает, каким образом сервисы выполняют свою задачу.

SOA также может рассматриваться как стиль архитектуры информационных систем, который позволяет создавать приложения, построенные путём комбинации слабосвязанных и взаимодействующих сервисов. Эти сервисы взаимодействуют на основе какого-либо строго определённого платформенно-независимого и языково-независимого интерфейса (например, WSDL). Определение интерфейса скрывает языково-зависимую реализацию сервиса.

Таким образом, системы, основанные на SOA, могут быть независимы от технологий разработки и платформ (таких как Java, .NET и т. д.). К примеру, сервисы, написанные на C#, работающие на платформах .Net и сервисы на Java, работающие на платформах Java EE, могут быть с одинаковым успехом вызваны общим составным приложением. Приложения, работающие на одних платформах, могут вызывать сервисы, работающие на других платформах, что облегчает повторное использование компонентов.

SOA может поддерживать интеграцию и консолидацию операций в составе сложных систем, однако SOA не определяет и не предоставляет методологий или фреймворков для документирования сервисов.

**Веб-сервисы**

Для решения уменьшения количества удалённых обращений был созданы веб-сервисы, чтобы повысить производительность системы. Также требовался надёжный канал связи и более простая спецификация обмена сообщениями.

1. Нужен был надёжный канал связи, поэтому:

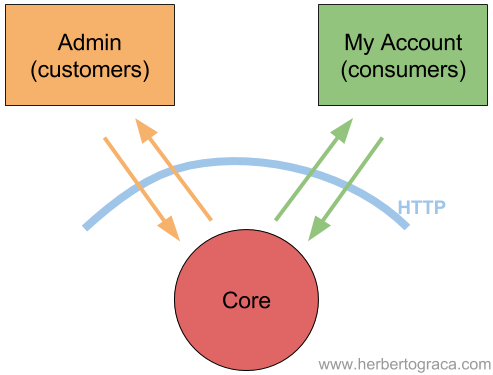
* HTTP стал по умолчанию работать через порт 80.
* Для обмена сообщениями начали использовать платформо-независимый язык (вроде XML или JSON).

1. Нужно было уменьшить количество удалённых обращений, поэтому:

* Удалённые соединения стали явными, так что теперь мы всегда знаем, когда выполняется удалённый вызов.
* Вместо многочисленных удалённых вызовов объектов мы обращаемся к удалённым сервисам, но гораздо реже.

1. Нужно было упростить спецификацию обмена сообщениями, поэтому:

* Первый черновик SOAP появился в 1998-м, стал рекомендацией W3C в 2003-м, после чего превратился в стандарт. SOAP вобрал в себя некоторые идеи CORBA, вроде слоя для обработки обмена сообщениями и «документа», определяющего интерфейс с помощью языка описания веб-сервисов (Web Services Description Language, WSDL).
* Рой Филдинг в 2000-м описал REST в своей диссертации «Architectural Styles and the Design of Network-based Software Architectures». Его спецификация оказалась гораздо проще SOAP, поэтому вскоре REST обогнал SOAP по популярности.
* Facebook разработал GraphQL в 2012-м, а публичный релиз выпустил в 2015-м. Это язык запросов для API, позволяющий клиенту строго определять, какие данные сервер должен ему отправить, не больше и не меньше.



Но нужно понимать, что в рамках SOA веб-сервисы – не просто API общего назначения, всего лишь предоставляющие CRUD-доступ к базе данных через HTTP. В каких-то случаях эта реализация может быть полезной, но ради целостности ваших данных необходимо, чтобы пользователи понимали лежащую в основе реализации модель и соблюдали бизнес-правила. SOA подразумевает, что веб-сервисы являются ограниченными контекстами бизнес-субдоменов (business sub-domain) и отделяет реализацию от решаемых веб-сервисами задач.

**Достоинства**

* Независимость набора технологий, развёртывания и масштабируемости сервисов.
* Стандартный, простой и надёжный канал связи (передача текста по HTTP через порт 80).
* Оптимизированный обмен сообщениями.
* Стабильная спецификация обмена сообщениями.
* Изолированность контекстов доменов (Domain contexts).

**Недостатки**

* Разные веб-сервисы тяжело интегрировать из-за различий в языках передачи сообщений. Например, два веб-сервиса, использующих разные JSON-представления одной и той же концепции.
* Синхронный обмен сообщениями может перегрузить системы.

**Очередь сообщений**

К примеру, у нас есть несколько приложений, которые асинхронно общаются друг с другом с помощью платформо-независимых сообщений. Очередь сообщений улучшает масштабируемость и усиливает изолированность приложений. Им не нужно знать, где находятся другие приложения, сколько их и даже что они собой представляют. Однако все эти приложения должны использовать один язык обмена сообщениями, т. е. заранее определённый текстовый формат представления данных.

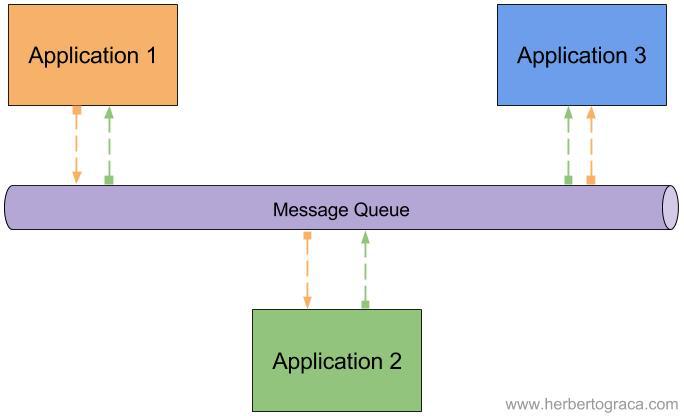
Очередь сообщений использует в качестве компонента инфраструктуры программный брокер сообщений (RabbitMQ, Beanstalkd, Kafka и т. д.). Для реализации связи между приложениями можно по-разному настроить очередь:

**Запрос/Ответ**

* Клиент шлёт в очередь сообщение, включая ссылку на «разговор» («conversation» reference). Сообщение приходит на специальный узел, который отвечает отправителю другим сообщением, где содержится ссылка на тот же разговор, так что получатель знает, на какой разговор ссылается сообщение, и может продолжать действовать. Это очень полезно для бизнес-процессов средней и большой продолжительности (цепочек событий, sagas).

**Публикация/Подписка**

* **По спискам**. Очередь поддерживает списки опубликованных тем подписок (topics) и их подписчиков. Когда очередь получает сообщение для какой-то темы, то помещает его в соответствующий список. Сообщение сопоставляется с темой по типу сообщения или по заранее определённому набору критериев, включая и содержимое сообщения.
* **На основе вещания.** Когда очередь получает сообщение, она транслирует его всем узлам, прослушивающим очередь. Узлы должны сами фильтровать данные и обрабатывать только интересующие сообщения.



Все эти паттерны можно отнести к либо к pull- (polling), либо к push-подходу:

* В pull-сценарии клиент опрашивает очередь с определённой частотой. Клиент управляет своей нагрузкой, но при этом может возникнуть задержка: сообщение уже лежит в очереди, а клиент его ещё не обрабатывает, потому что не пришло время следующего опроса очереди.
* В push-сценарии очередь сразу же отдаёт клиентам сообщения по мере поступления. Задержки нет, но клиенты не управляют своей нагрузкой.

**Достоинства**

* Независимость набора технологий, развёртывания и масштабируемости сервисов.
* Стандартный, простой и надёжный канал связи (передача текста по HTTP через порт 80).
* Оптимизированный обмен сообщениями.
* Стабильная спецификация обмена сообщениями.
* Изолированность контекстов домена (Domain contexts).
* Простота подключения и отключения сервисов.
* Асинхронность обмена сообщениями помогает управлять нагрузкой на систему.

**Недостатки**

* Разные веб-сервисы тяжело интегрировать из-за различий в языках передачи сообщений. Например, два веб-сервиса, использующих разные JSON-представления одной и той же концепции.

**SOAP как средство интегрирования технологий RPC с Интернет - технологиями**

Изначально протокол SOAP разрабатывался с тем расчетом, что он будет функционировать «над» HTTP (дабы упростить интеграцию SOAP в Web-приложения), однако теперь могут быть задействованы и иные транспортные протоколы, например, SMTP, используемый для асинхронной отправки сообщений. Таким образом,

SOAP – это протокол на базе языка XML, который определяет правила передачи сообщений по Internet между различными прикладными системами.

Разработанный в рамках XML-технологий простой «связующий» протокол SOAP, и представляющий собой надстройку над протоколом HTTP, обеспечивает удаленный вызов операций приложений и возврат результатов их выполнения. SOAP предлагает минимальный набор условий, позволяющий приложению передавать сообщения: клиент посылает сообщение для того, чтобы вызывать программный объект, а сервер возвращает результаты этого вызова.

SOAP довольно прост: сообщения представляют собой документы XML, содержащие команды SOAP. Хотя теоретически SOAP может быть привязан к любому транспортному протоколу для приложений, как правило, он используется вместе с HTTP.

Данные инкапсулируются в HTTP или в какой-то другой транспортный протокол и передаются адресату, в качестве которого обычно выступает сервер. Этот сервер выделяет данные SOAP из пакета, выполняет требуемую обработку и возвращает результаты в виде ответа SOAP.

Если проанализировать протоколы, созданные в рамках Web технологий, то можно выделить три уровня протоколов: TCP/IP, HTTP/HTML и SOAP/XML. Эти уровни надстроены последовательно друг над другом и являются совместимыми.

Первый уровень, протокол TCP/IP, связан, прежде всего, с передачей данных в пакетах по кабелю. Будучи протоколом, который гарантирует пересылку данных по общественным сетям, TCP/IP обеспечивает физическую связь и надежную транспортировку данных. Первоначально TCP/IP представлял собой «замазку», скрепляющую отдельные сети, тогда как сейчас это базовый протокол Web, на котором основаны высокоуровневые стандартные протоколы типа HTTP.

Второй уровень, HTTP/HTML, является уровнем представления и с помощью интерфейса браузера занимается поиском, выборкой и совместным использованием информации. Здесь основное внимание уделяется GUI-навигации и манипуляции форматов представления. Во многих отношениях HTML носит больше демонстративный, чем функциональный характер, и испытывает недостаток как в расширяемости, так и в истинной программной мощности. Тем не менее, совместное использование гипертекстовых документов в среде с интерфейсом браузера полностью изменил способ передачи текстовой информации между людьми.

Третий уровень основан на использовании при обмене данными технологии XML, поддерживающей формат обмена со строгим контролем типов. Этот вид обмена данными приобрел статус протокола SOAP и предоставляет новое измерение уровню HTTP/ HTML, в котором осуществление межкомпьютерного взаимодействия обеспечивается посредством стандартных интерфейсов.

При использовании SOAP уже не требуется, чтобы две взаимодействующие системы запускались на одной платформе или писались на одном языке программирования. А вместо вызова методов по бинарному протоколу, принятого в CORBA и DCOM, пакет SOAP использует открытый, основанный на тексте стандартный синтаксис вызова методов на основе XML.

Таким образом, вся информация между запрашивающим приложением и принимающим объектом пересылается в виде данных, заключенных между тегами, в XML-потоке по HTTP.

С точки зрения Web-сервисов SOAP реализуется в виде клиента и сервера. Клиент SOAP – это программа, которая создает XML-документ, содержащий информацию, необходимую для удаленного вызова метода в распределенной системе.

Сервер SOAP – это специальный код, который слушает SOAP сообщения и действует как распределитель и интерпретатор SOAP документов. Внешние Web-сервисы могут взаимодействовать с приложениями-серверами, которые обрабатывают запросы SOAP от множества клиентов.

Серверы SOAP гарантируют, что документы, полученные по HTTP соединению, будут преобразованы в формат языка, который понятен объекту и всем окружающим. Поскольку все коммуникации осуществляются в форме XML, то объекты, написанные на одном языке (скажем, Java), могут связываться через SOAP с объектами, написанными на другом языке (например, C++). Работа SOAP-сервера заключается в том, чтобы удостовериться, что конечные пункты понимают обслуживающий их SOAP.

**SOAP и REST**

SOAP более применим в сложных архитектурах, где взаимодействие с объектами выходит за рамки теории CRUD, а вот в тех приложениях, которые не покидают рамки данной теории, вполне применимым может оказаться именно REST ввиду своей простоты и прозрачности. Действительно, если любым объектам вашего сервиса не нужны более сложные взаимоотношения, кроме: «Создать», «Прочитать», «Изменить», «Удалить» (как правило — в 99% случаев этого достаточно), возможно, именно REST станет правильным выбором. Кроме того, REST по сравнению с SOAP, может оказаться и более производительным, так как не требует затрат на разбор сложных XML команд на сервере (выполняются обычные HTTP запросы — PUT, GET, POST, DELETE). Хотя SOAP, в свою очередь, более надежен и безопасен.