Сервис-ориентированные архитектуры

SOAP XML веб-сервисы

API сообщений

Очереди сообщений

Глеб Игоревич Радченко

06.02.2021

SOAP XML Веб-сервисы

SOAP XML Веб-сервисы

- XML Веб-сервисы – это основанная на языке XML платформо-независимая технология, поддерживающая разработку распределенных приложений, которая была предложена в 2000 году консорциумом во главе с Microsoft.
- XML Веб-сервисы обеспечивает создание независимых, масштабируемых, слабосвязанных приложений.
- Они основаны на протоколах HTTP, XML, XSD, SOAP, WSDL.
- Реализации SOAP/WSDL:
  - The Java API for XML Web Services (JAX-WS)
  - Apache CXF
  - Microsoft’s Windows Communication Foundation (WCF)
  - …

Серверная часть Web служб

Borja Sotomayor . The Globus Toolkit 4 Programmer's Tutorial. University of Chicago Department of Computer Science

WSDL: Web Service Definition Language

- WSDL – это стандартный XML-документ, описывающий фундаментальные свойств Web службы, как то:
  - Что это – описание методов, предоставляемые Web службой;
  - Как осуществляется доступ – формат данных и протоколы;
  - Где он расположен – сетевой адрес службы (URI).

Адресация XML Web служб

- Адрес XML Web службы – это стандартный URI (Uniform Resource Identifer).

- http://live.capescience.com/ccx/GlobalWeather

    - BTW: URL (Uniform Resource Locator) является одним из видов URI, и является прародителем URI.

Пример Web службы

Простой пример Web службы (Java)

```
@WebService
public class MyMath
    {
        @WebMethod
        public int squared(int x)
        {
            return x * x;
        }
    }
```

Элементы WSDL-документа

- Блок types – типы данных, используемые Web-службой
- Блок message – сообщения, используемые Web-службой
- Блок portType – методы, предоставляемые Web-службой
- Блок binding – протоколы связи, используемые Web-службой

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://DefaultNamespace"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://DefaultNamespace"
xmlns:intf="http://DefaultNamespace"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:message name="squaredRequest">
      <wsdl:part name="in0" type="xsd:int"/>
   </wsdl:message>
   <wsdl:message name="squaredResponse">
      <wsdl:part name="squaredReturn" type="xsd:int"/>
   </wsdl:message>
   <wsdl:portType name="MyMath">
      <wsdl:operation name="squared" parameterOrder="in0">
         <wsdl:input message="impl:squaredRequest" name="squaredRequest"/>
         <wsdl:output message="impl:squaredResponse" name="squaredResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="MyMathSoapBinding" type="impl:MyMath">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/ soap/http"/>
      <wsdl:operation name="squared">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="squaredRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/"
               namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="squaredResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/"
               namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="MyMathService">
      <wsdl:port binding="impl:MyMathSoapBinding" name="MyMath">
         <wsdlsoap:address location="http://localhost:8080/axis/testaccount/MyMath"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

WSDL-документ

Namespaces

messages - сообщения

portTypes - методы

binding - связи

Определение службы

Порты WSDL <portType>

- Элемент <portType> является наиболее важным элементом WSDL.
- Он определяет саму Web-службу, предоставляемые ей операции и используемые сообщения.
- Можно сравнить с библиотекой функций, в которой указаны входные параметры и результаты работы функции.

Пример блока <portType>

```
<wsdl:portType name="MyMath">
        <wsdl:operation name="squared" parameterOrder="in0">
                <wsdl:input message="impl:squaredRequest“ name=">
                <wsdl:output message="impl:squaredResponse“ name=">
                </wsdl:output>
                </wsdl:input>
        </wsdl:operation>
</wsdl:portType>
```

Сообщения WSDL <message>

- Элемент <message> определяет элементы данных операции.

- Каждое сообщение может содержать одну или несколько частей. Эти части можно сравнить с параметрами вызываемых функций в традиционных языках программирования.

Пример блока <message>

```
<wsdl:message name="squaredRequest">
       <wsdl:part name="in0" type="xsd:int" />
</wsdl:message>
<wsdl:message name="squaredResponse">
       <wsdl:part name="squaredReturn" type="xsd:int" />
</wsdl:message>
```

Связи WSDL <binding>

- Элемент <binding> определяет формат сообщения и детали протокола для каждого порта.

- Отвечает за то, каким образом элементы абстрактного интерфейса в блоке <portType> преобразуются в массивы информации в формате протоколов взаимодействия, например SOAP.

Пример блока <binding>

```
<wsdl:binding name="MyMathSoapBinding" type="impl:MyMath">
        <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/
soap/http" />
        <wsdl:operation name="squared">
                <wsdlsoap:operation soapAction="" />
                <wsdl:input name="squaredRequest">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
                </wsdl:input>
                <wsdl:output name="squaredResponse">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
                </wsdl:output>
        </wsdl:operation>
</wsdl:binding>
```

Блоки <port> и <service>

- Данные блоки определяют, где находится служба.

- port – описывает расположение и способ доступа к конечной точке
- service – именованная коллекция портов

Пример блока <service>

```
<wsdl:service name="MyMathService">
      <wsdl:port binding="impl:MyMathSoapBinding“ name="">
      <wsdlsoap:address
location="http://localhost:8080/axis/myaccount/MyMath" />
      </wsdl:port>
</wsdl:service>
```

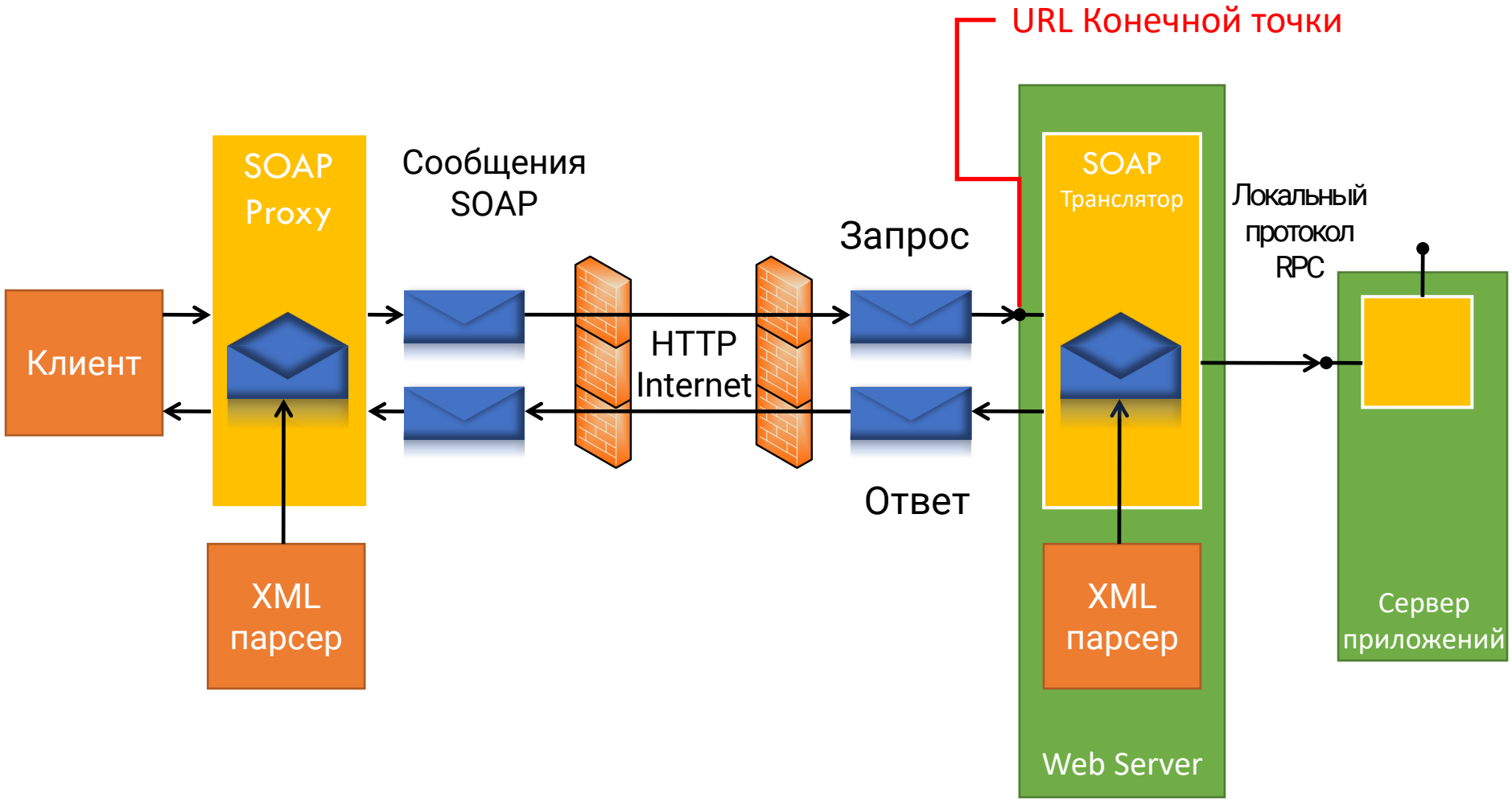
SOAP (уже не только Simple Object Access Protocol)

- SOAP – это протокол, основанный на обмене XML-документами.
- SOAP определяется следующим образом: «SOAP это основанный на XML протокол обмена информацией в децентрализованной распределенной среде.

Сообщение SOAP

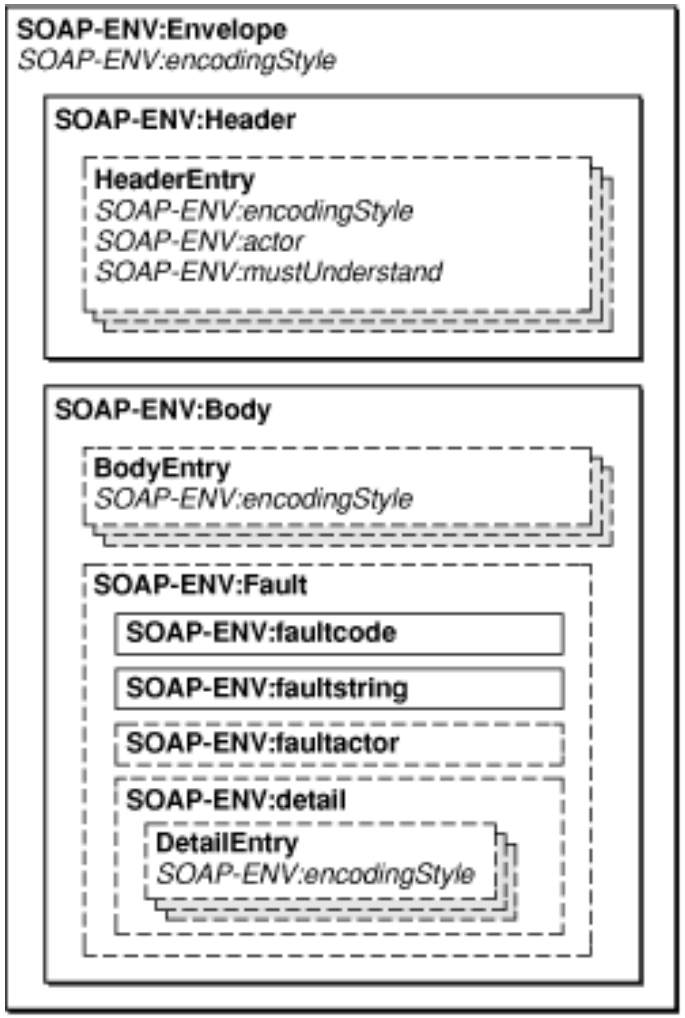
- Сообщением SOAP называют одностороннюю передачу информации между узлами SOAP: от источника к приемнику.
- Сообщения SOAP являются фундаментальным строительным блоком, обеспечивающим возможности более сложных шаблонов взаимодействия:
  - Запрос/ответ
  - «диалоговый» режим
  - и т.п.

Архитектура SOAP



Элементы сообщения SOAP

- Envelope (конверт) – корневой элемент сообщения.
- Header (заголовок) – не обязательный элемент сообщения. Может содержать дополнительную информацию для приложения, обрабатывающего запрос.
- Body (Тело) – обязательный элемент сообщения. Содержит вызовы необходимых методов и передаваемые параметры.



Шаблон сообщения SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap=http://www.w3.org/2001/12/soap-envelope
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Пример заголовка SOAP

- В заголовке мы можем ввести новый элемент, не предусмотренный стандартом SOAP. Например, номер транзакции.
- Атрибуты:
  - mustUnderstand – получатель обязан обрабатывать этот элемент;
  - actor – указывает конкретное приложение-получатель при обработке сообщения в цепочке.

```
<soap:Header>
   <trans:Transaction xmlns:trans="http://
   www.host.com/namespaces/space/"
   soap:mustUnderstand="1">
   12
   </trans:Transaction>
</soap:Header>
```

Тело сообщения SOAP

Запрос

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productID>12345</productID>
     </getProductDetails>
   </soap:Body>
 </soap:Envelope>
```

Ответ

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productID>12345</productID>
         <productName>Стакан граненый</productName>
         <description>Стакан граненый. 200 мл.</description>
         <price>9.95</price>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
 </soap:Envelope>
```

Связывание SOAP и HTTP

- Передача SOAP сообщений происходит поверх HTTP – протокола посредством запроса POST (начиная со стандарта SOAP 1.2 возможно применение GET)
- Стандартный протокол связи:
  - Клиент посылает запрос
  - Сервер отвечает OK

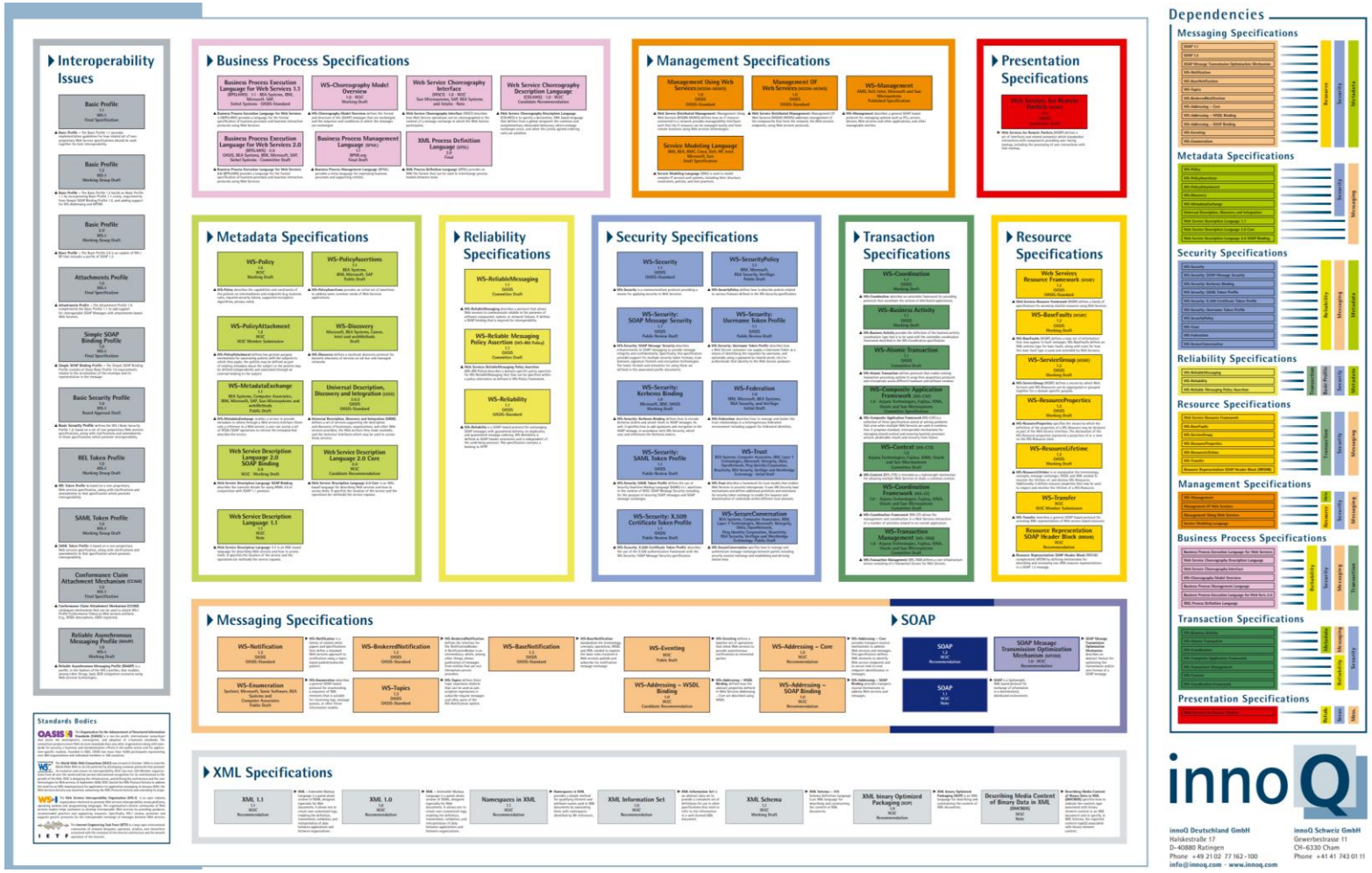
```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml;
charset=utf-8
Content-Length: nnn
…

HTTP/1.1 200 OK
Content-Type: application/soap;
charset=utf-8
Content-Length: nnn
…
```

Семейства WS-* стандартов

- Начиная с 2003 года, консорциумы интернет-компаний занялись развитием стандартов XML веб-сервисов в надежде сделать их пригодными для корпоративного использования. Новые стандарты были ориентированы на следующие аспекты работы веб-сервисов:
  - Безопасность и шифрование
  - Работа с состояниями и удаленными ресурсами
  - Расширенные возможности передачи сообщений
  - Надежность в передаче сообщений
  - Работа с транзакциями
  - И др.

Карта стандартов WS-*



Причины провала WS-* стандартов

- Борьба за рынки привела к появлению дублирующих стандартов от конкурирующих корпораций
- Большинство стандартов были либо плохо проработаны, либо чрезвычайно специфичны
- Большинство стандартов не были реализованы в популярных библиотеках и фреймворках для разработки XML веб-сервисов, а те что были реализованы, имели большие проблемы с совместимостью
- Язык XML, лежащий в основе XML веб-сервисов, не позволял организовать эффективный обмен данными. Так, реализация шифрования и подписи сообщений на базе стека протоколов WS-Security увеличивала **размер передаваемых сообщений в 2-2.5 раза, а латентность – в 4(!!!) раза.**

API Сообщений

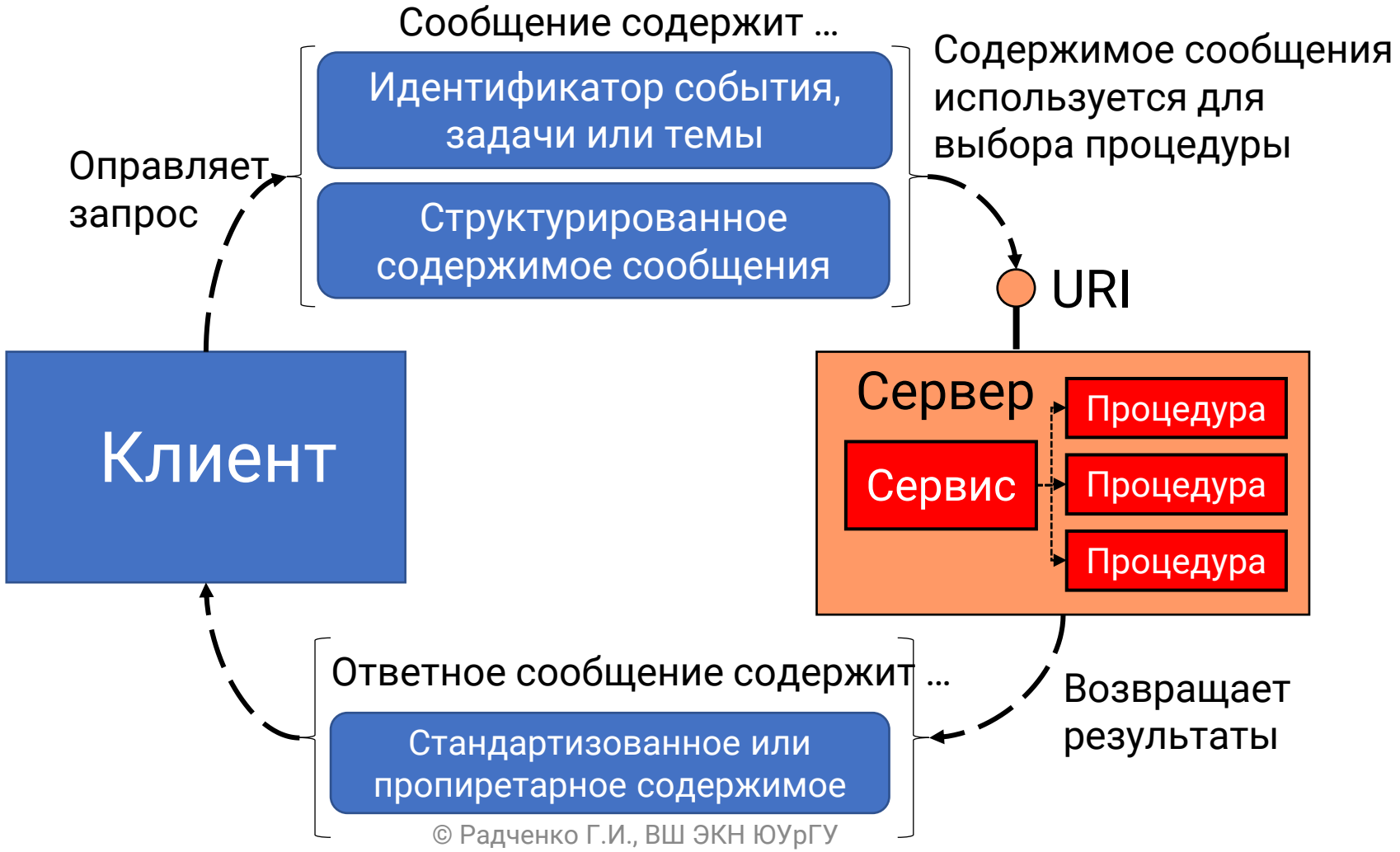
Стили API Веб-сервисов

Стиль Веб-сервиса	Решаемая проблема
1. RPC API	Как клиенты могут выполнить удаленные процедуры посредством HTTP?
2. API сообщений	Как клиенты могут отправлять сообщения (команды, нотификации или другую структурированную информацию) удаленной системе по HTTP, не привязывая себя к удаленным процедурам?
3. API ресурсов	Как клиент может манипулировать данными, предоставляемыми удаленной системой по HTTP, без связывания себя с удаленными процедурами, а также без необходимости создания специального проблемно-ориентированного API?
4. Графовый API	Как сформировать запрос к графу объектов, доступных на удаленном сервере, запросив конкретные поля, необходимые клиенту для обработки?

Стили API Веб-сервисов: API сообщений

- Если API сервиса основывается на реализуемых процедурах, сложно обрабатывать изменение такого API, т.к. необходимо менять код на всех клиентах. Если клиенты и сервисы управляются разными организациями, такой подход может стать не приемлемым.
- **Решение** - необходимо определить сообщения для обмена, которые бы не раскрывали интерфейс удаленных процедур.
- Данные сообщения должны содержать информацию об определенных событиях, задачах, которые необходимо выполнить.
- Клиент должен отправить сообщение с этой информацией по указанному URI, а сервер, на основе анализа сообщения должен выполнить определенные процедуры.

Стили API Веб-сервисов: API для сообщений



Стили API Веб-сервисов: API сообщений

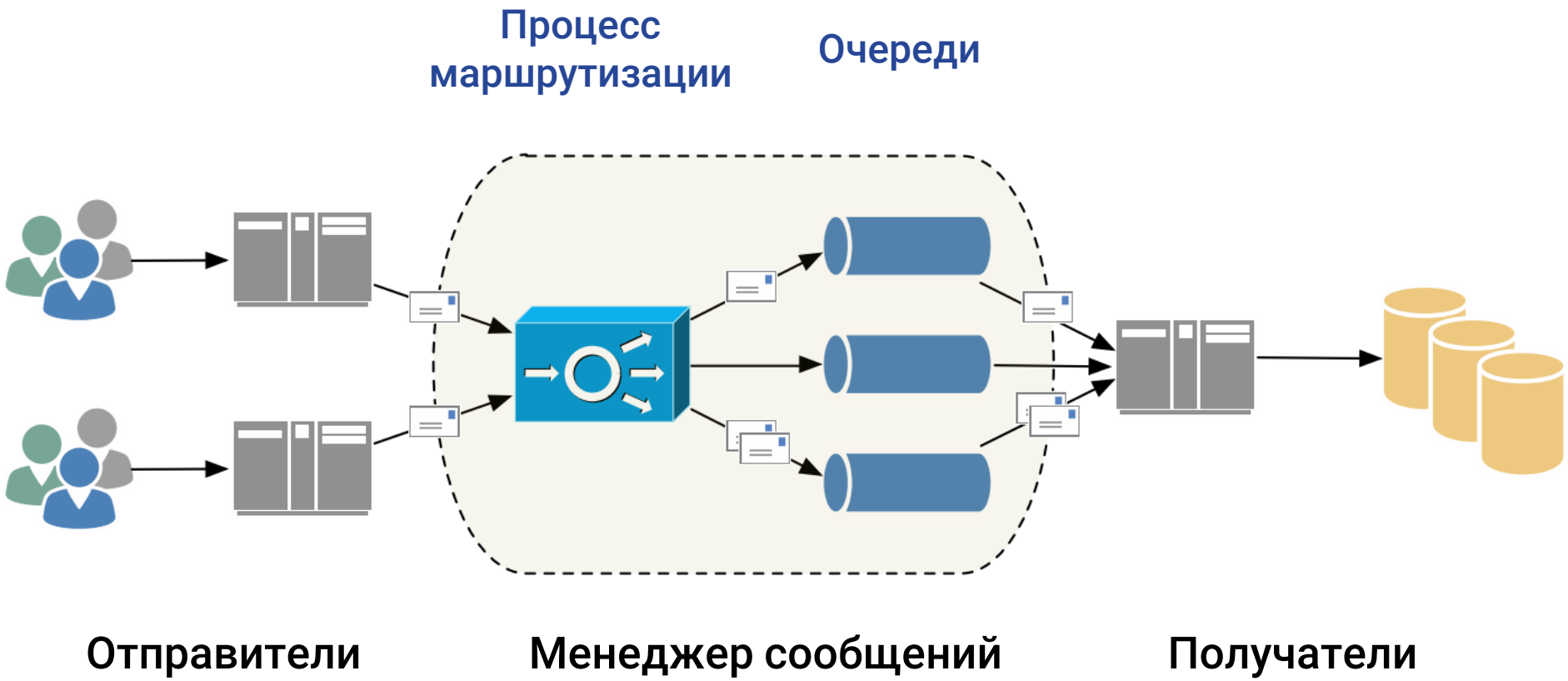
- Фактически, такой подход строится на использовании тех же технологий, что и RPC API. Отличается метод проектирования системы:
  - Проектирование начинается с определения формата сообщений, которыми обмениваются удаленные компоненты.
  - Они могут быть определены на некотором промежуточном языке, например посредством OpenRPC, XML Schema Language или Google Protocol Buffers IDL.
- Сервисы создаются после того, как сообщения были определены.
- Отличием сервисов для API сообщений от сервисов для RPC является количество аргументов в интерфейсе сервиса. Чаще всего, при использовании API сообщений, каждый метод имеет только 1 параметр.

Очереди сообщений

Менеджеры (очереди) сообщений

- Очередь сообщений – это ПО промежуточного слоя, обеспечивающие управление взаимодействием между компонентами распределенной системы, посредством сбора, хранения и маршрутизации сообщений между процессами.
- Очередь сообщений позволяет работающим в различное время приложениям взаимодействовать в гетерогенных сетях и системах, которые могут временно отключаться от сети.
- Приложения отправляют, получают и считывают (то есть читают без удаления) сообщения из очередей.

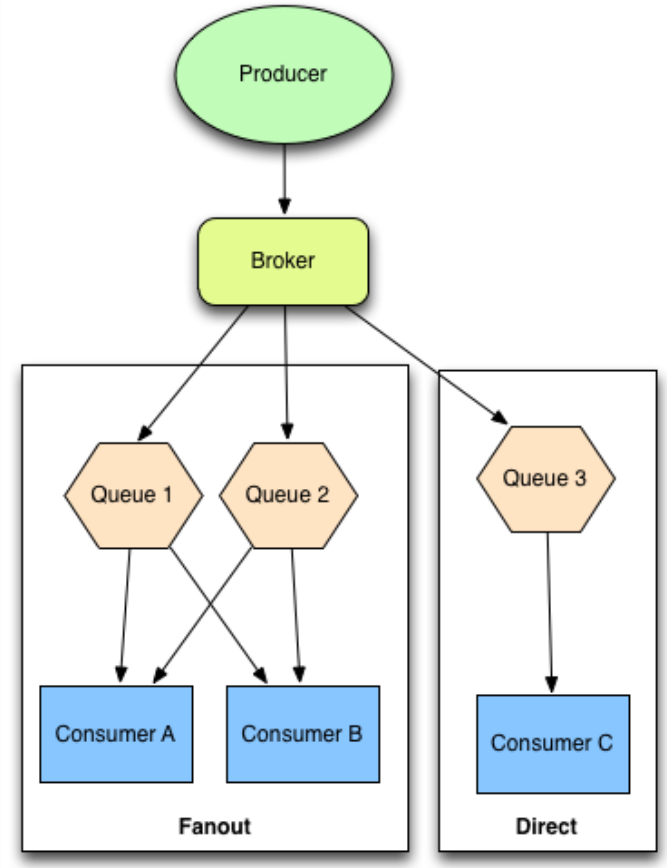
Использование менеджера сообщений



(c) NICOLAS NANNONI. Message-oriented Middleware for Scalable Data Analytics Architectures http://kth.diva-portal.org/smash/get/diva2:813137/FULLTEXT01.pdf

Менеджеры (очереди) сообщений

- Очередь сообщений предоставляет интерфейс для поставщика (producer) и потребителя (consumer) сообщений.
- Менеджер сообщений может обеспечить распределение сообщений в различные очереди, обеспечивая распределение нагрузки между задачами и возможность горизонтального масштабирования.

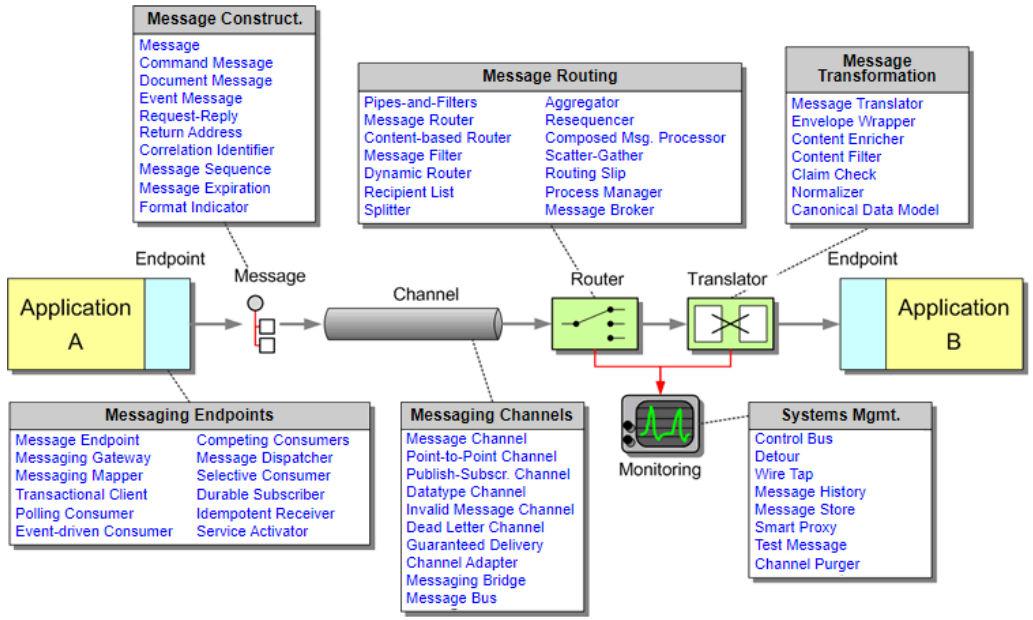


© **Nathan Esquenazi.** Asynchronous Processing in Web Applications

Паттерны очередей сообщений

Ключевые аспекты организации очередей сообщений

- Выделяют не менее 65 паттернов, связанных с очередями сообщений в следующих областях:
  - Организация самих сообщений
  - Организация каналов обмена сообщениями в очередях
  - Применение очередей сообщений для реализации различных шаблонов взаимодействия
  - Роутинг сообщений
  - Преобразование сообщений
  - Мониторинг и управление очередями сообщений

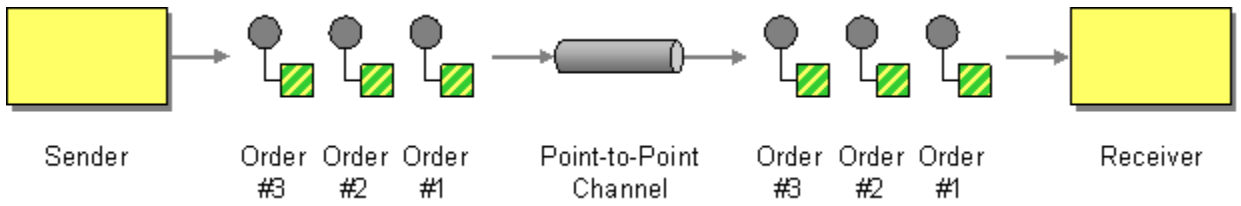


Ключевые типы сообщений

- Выделяют следующие типы сообщений:
  - Командные сообщения *(Command Message)*: используются для инициализации определенных действий на сервере (например, обработать запись).
  - Сообщения о событиях *(Event Message)*: оповещают получателя о произошедших у клиента событиях (например, закончилось место на складе).
  - Документы-сообщения *(Document Message)*: содержат информацию о тех или иных бизнес-сущностях (например, бланк заказа).
- Ответы сервера также определяются сообщениями: либо полноценным сообщением с ответом, либо простейшим подтверждением о получении сообщения.
- Для того, чтобы мы могли идентифицировать, на какое сообщение был получен ответ, и к сообщению и к ответу на него необходимо привязать идентификатор корреляции (Correlation Identifier)

Канал «точка-точка»
Point-to-Point Channel

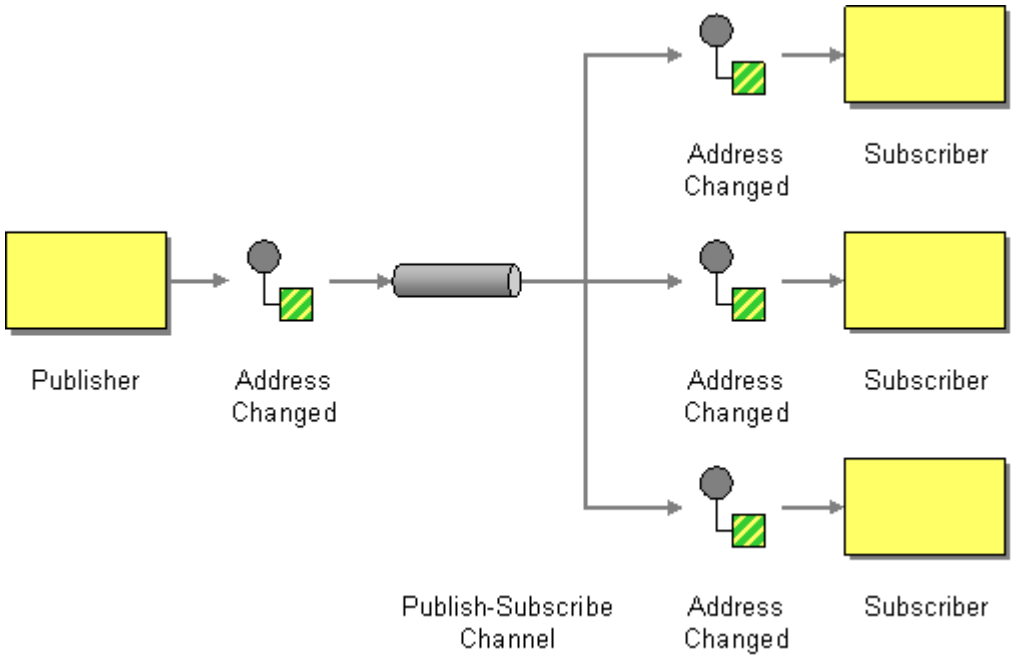
- Отправка сообщения по каналу "Точка-точка" гарантирует, что только один потребитель получит конкретное сообщение.
- Если канал имеет несколько потребителей, только один из них может успешно принять определенное сообщение.
- Если несколько потребителей пытаются принять одно сообщение, канал гарантирует, что только один из них успешно примет это сообщение, поэтому потребители не должны согласовывать это друг с другом.



Enterprise Integration Patterns Gregor Hohpe and Bobby Woolf ISBN 0321200683

Канал «публикация-подписка» Publish-Subscribe Channel

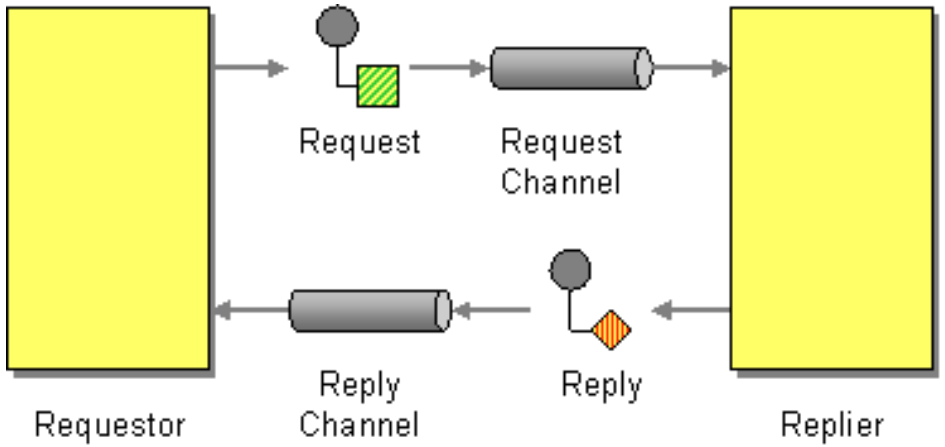
- Оправка сообщения по каналу, организованному в формате pub/sub, доставляет копию определенного события каждому подписанному потребителю.
- Такой канал имеет один входной и множество (по одному на каждого подписчика) выходных каналов.
- Когда сообщение публикуется во входном канале, то его копия доставляется в каждый из выходных каналов.
- В каждом выходном канале есть только один подписчик, которому разрешается принимать сообщение только один раз. Таким образом, каждый подписчик получает сообщение только один раз, а потребляемые копии исчезают из их каналов.



Enterprise Integration Patterns Gregor Hohpe and Bobby Woolf ISBN 0321200683

Запрос-ответ
Request-reply

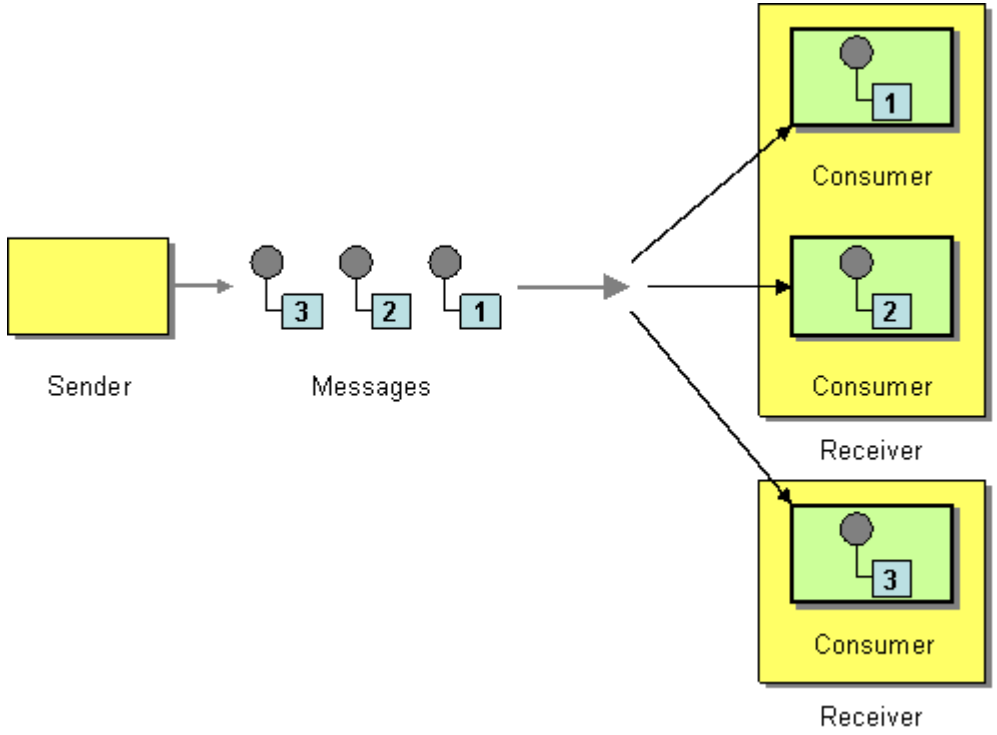
- Очередь задач может быть использована для организации взаимодействия в формате «Запрос-ответ», в том числе и для моделирования RPC.
- Чаще всего для организации этого применяют два независимых канала: канал для отправки запросов и канал для получения ответов.



Enterprise Integration Patterns Gregor Hohpe and Bobby Woolf ISBN 0321200683

Очередь задач
Work queue

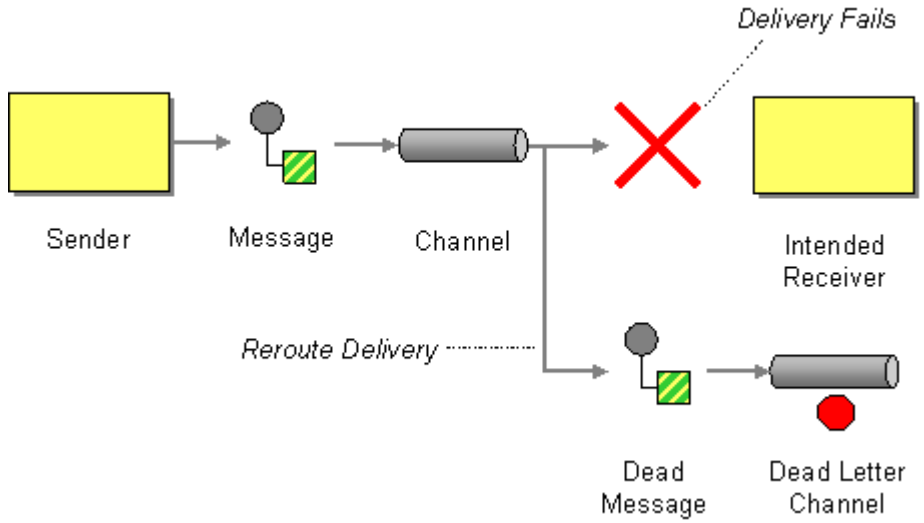
- Обеспечивает возможность конкурентной обработки сообщений, создавая множество конкурирующих обработчиков на одном канале, которые могут обработать несколько поступающих сообщений одновременно
- Мы запускаем одновременно несколько потребителей которые получают сообщения с одного канала "точка-точка".
- Когда канал доставляет сообщение, любой из потребителей может его получить. Реализация системы сообщений определяет, какой потребитель на самом деле получает сообщение, но на самом деле потребители конкурируют друг с другом за то, чтобы получить сообщение. Как только потребитель получает сообщение, он может делегировать его обработку далее.



Enterprise Integration Patterns Gregor Hohpe and Bobby Woolf ISBN 0321200683

Канал «невостребованных писем» Dead Letter Channel

- Что делать, если обработка сообщения не удалась?
- Обычно, когда потребитель получает сообщение из очереди, сообщения остаются в очереди, но просто делаются невидимыми, чтобы другие потребители не смогли получить и обработать одно и то же сообщение.
- После обработки сообщения потребитель отвечает за его удаление. Если потребитель не удалит сообщение, например, из-за того, что он сломался во время обработки, сообщение становится снова видимым после истечения таймаута видимости сообщения.
- Каждый раз, когда это происходит, количество полученных сообщений увеличивается. Когда это количество достигает настроенного предела, сообщение помещается в очередь «невостребованных писем» (на примере Amazon SQS)



НО! Это может привести к непредвиденным последствиям. См. «Используете Kafka с микросервисами? Скорее всего, вы неправильно обрабатываете повторные передачи» https://habr.com/ru/company/southbridge/blog/531838

Enterprise Integration Patterns Gregor Hohpe and Bobby Woolf ISBN 0321200683

Достоинства и недостатки

Достоинства менеджеров сообщений

- **Слабосвязанность** программных компонентов обеспечивается посредством единого интерфейса данных
- **Избыточность данных:** парадигма “put-get-delete” позволяет избежать потери данных, даже если они не были обработаны после их получения
- **Масштабируемость и эластичность:** считывать и обрабатывать заявки из очереди могут несколько независимых процессов одновременно. При этом, если один из таких процессов падает, любой другой может взять на себя задачу обработки сообщений
- **Очередность:** сообщения попадают в очередь одно за другим, и, соответственно, обрабатываются в порядке поступления.
- **Буферизация:** если время на обработку сообщения больше, чем время поступления новых сообщений, очередь буферизует новые сообщения и они не теряются.
- **Асинхронность взаимодействия:** время работы клиента и сервера не зависят друг от друга. Клиент может отправить сообщение и продолжить свою работу не дожидаясь ответа.
- **Высокая прозрачность:** вся коммуникация проходит через менеджер сообщений. Таким образом, можно в любой момент времени оценить, какие сообщения, от кого и кому поступали.

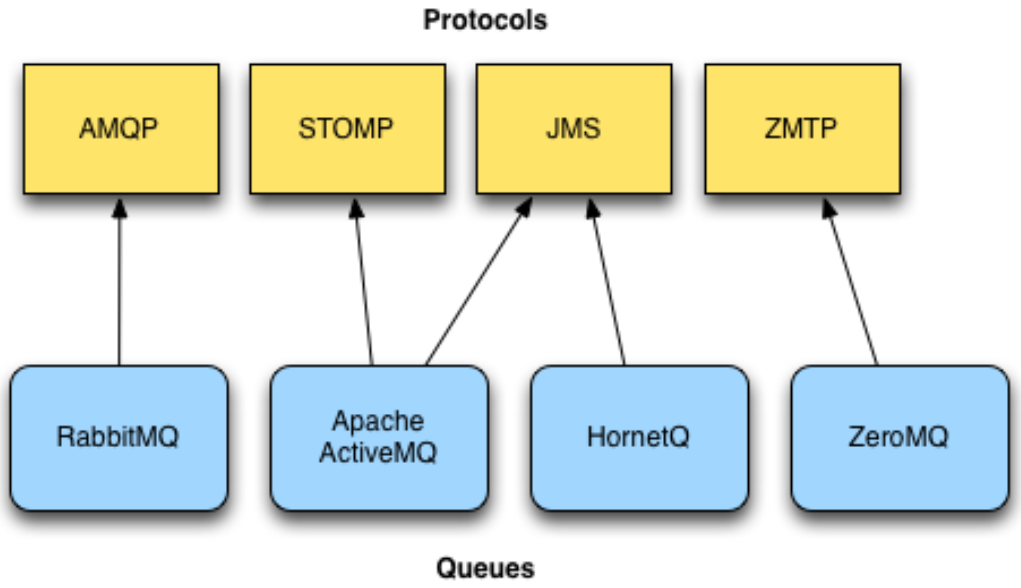
Недостатки менеджеров сообщений

- необходимость **явного** использования очередей распределенным приложением;
- сложность реализации **синхронного** обмена;
- определенные **накладные расходы** на использование менеджеров очередей;
- **сложность получения ответа:** передача ответа может потребовать отдельной очереди на каждый компонент, посылающий заявки.

Протокол AMQP

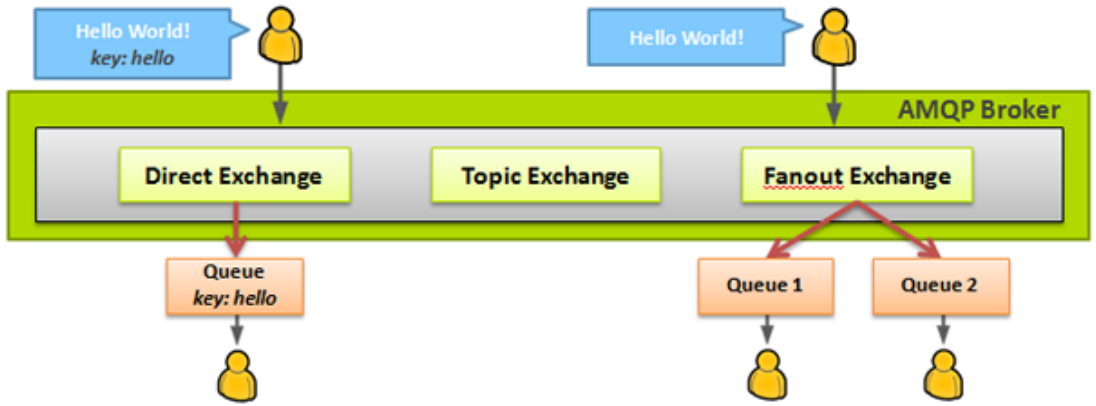
Службы очередей сообщений

- Службы очередей сообщений (Message Queue Services, MQS) используются в разработке начиная с 1980-х
- IBM WebSphere MQ (~8 000 $ на 100 процессоров).
- JMS – Java Message Service
- Microsoft Message Queuing (MSMQ - .NET)



Протокол AMQP

- AMQP (Advanced Message Queueing Protocol) – это открытый протокол прикладного уровня для передачи сообщений между компонентами вычислительной системы.
- Независимые приложения могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.



Понятия AMQP

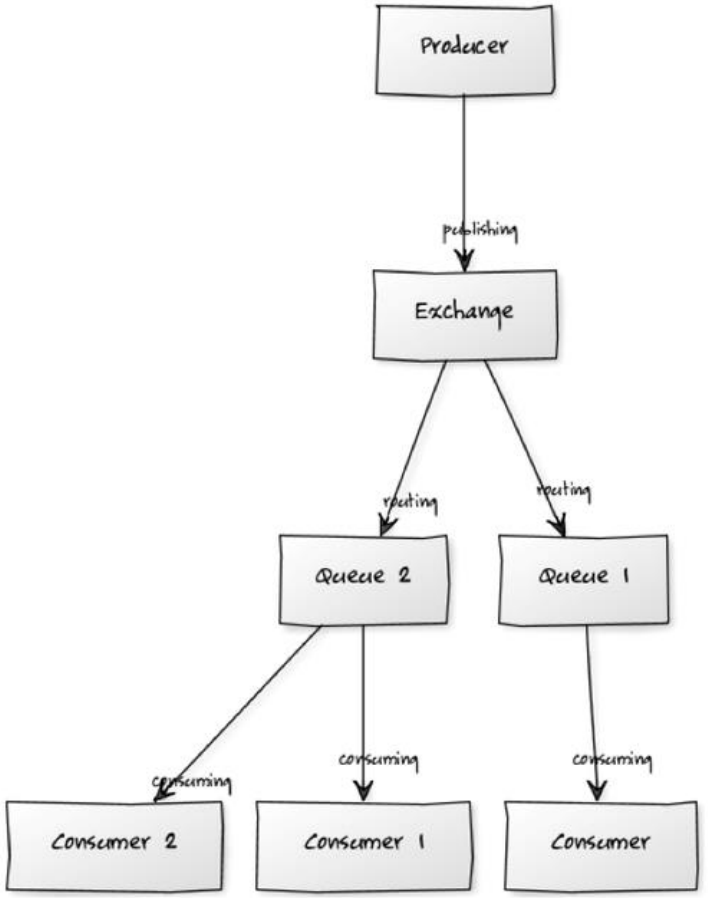
- **Сообщение (message)** — единица передаваемых данных, основная его часть (содержание) никак не интерпретируется сервером, к сообщению могут быть присоединены структурированные заголовки.
- **Точка обмена (exchange)** — в неё отправляются сообщения. Точка обмена распределяет сообщения в одну или несколько очередей. При этом в точке обмена сообщения не хранятся.
- **Ключ маршрутизации (routing key)** — маршрут, по которому передать сообщение (в какую очередь положить).
- **Очередь (queue)** — здесь хранятся сообщения до тех пор, пока не будут забраны клиентом. Клиент всегда забирает сообщения из одной или нескольких очередей.

Маршрутизация на точке обмена

- После получения сообщений от клиентов точки обмена обрабатывают и направляют их в одну или несколько очередей. Тип выполняемой маршрутизации зависит от типа точки обмена:
  - **direct** — сообщение передаётся в очередь с именем, совпадающим с ключом маршрутизации (routing key). Обычно используется в балансировке нагрузки round-robin.
  - **fanout** — сообщение передаётся во все привязанные к ней очереди. Используется для распространения сообщений нескольким клиентам (рассылки уведомлений и т.п.).
  - **topic** — нечто среднее между fanout и direct, сообщение передаётся в очереди, для которых совпадает маска на ключ маршрутизации, например:
    (work.* -> {work.manager, work.dev})

Схема работы AMQP

- Клиент (producer) посылает сообщение на точку обмена (exchange).
- Точка обмена кладёт сообщение в очереди, в зависимости от типа сообщений и ключей маршрутизации.
- Сообщения лежат в очереди, пока их не заберут клиенты, подписавшиеся на них.
- Клиенты получают сообщения и обрабатывают.



RabbitMQ vs ActiveMQ

- RabbitMQ (http://www.rabbitmq.com/)
  - Создан на основе Erlang
  - Работает на всех основных операционных системах
  - Поддерживает огромное количество платформ для разработчиков (чаще всего – Python, PHP, Ruby)
  - Конфигурация – на основе Erlang

- Apache ActiveMQ (http://activemq.apache.org/)
  - Построен на основе Java Message Service
  - Чаще всего используется совместно с Java-стеком (Java, Scala, Clojure и др).
  - Также поддерживает STOMP (Ruby, PHP, Python).
  - Конфигурация – на основе XML

Пример использования RabbitMQ

- RabbitMQ as a Service: http://www.cloudamqp.com/
- Пример приложения: https://github.com/cloudamqp/java-amqp-example
- Запускаем Sender (OneOffProcess.java), потом Receiver (WorkerProcess.java)

Sender:

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
String message = "Hello CloudAMQP!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
System.out.println(" [x] Sent '" + message + "'");
```

Receiver:

```
while (true) {
      QueueingConsumer.Delivery delivery = consumer.nextDelivery();
      String message = new String(delivery.getBody());
      System.out.println(" [x] Received '" + message + "'");
    }
```

Спасибо за внимание!

Готов ответить на
ваши вопросы!