

# **Web Services Best Practices**

**Sang Shin**  
**sang.shin@sun.com**  
**Technology Evangelist**  
Sun Microsystems, Inc.

# Agenda

- Web services Design Issues
- Layered Architecture
- Interface Design
- Choice of Endpoint (in Java world)
- SOAP message handlers
- SOAP message processing
- Web services technologies & Tools
- SOAP vs. REST
- Real-life Web services examples

# Web Services Design Issues

# Steps for designing a Web service

1. Decide on the interface for clients
2. Decide whether and how to publish this interface
3. Determine how to receive and preprocess requests
4. Determine how to delegate the request to business logic
5. Decide how to process the request
6. Determine how to formulate and send the response
7. Determine how to report problems

# Web Service Design Issues

- How will clients make use of your services?
  - > Consider what sort of calls clients may make and what might be the parameters of those calls.
- How will your Web service receive client requests?
  - > Consider what kind of endpoints you are going to use for your Web service.
- What kind of common preprocessing, such as transformations, translations, and logging, needs to be done?

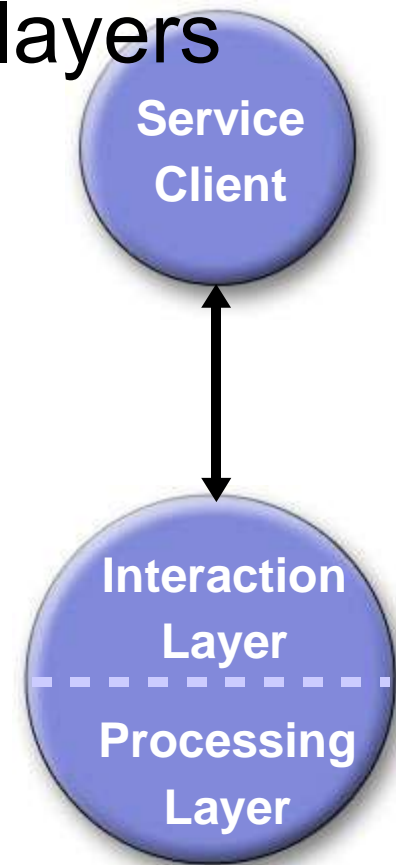
# Questions for Web Service Design

- How will the request be delegated to business logic?
- How will the response be formed and sent back?
- What kinds of exceptions will the service throw back to the clients, and when will this happen?
- How are you going to let clients know about your Web service? Are you going to publish your service in public registries, in private registries, or some way other than registries?

# Layered Architecture

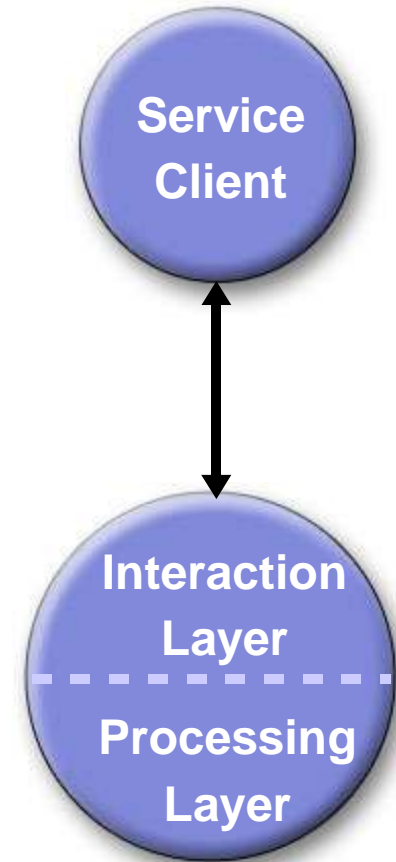
# Layered View

- Structure application in two layers
  - > Interaction Layer and Processing Layer
- Interaction layer
  - > Interface to clients
  - > Receive requests and perform required translations and transformations
  - > Delegate request to processing layer for processing
  - > Respond to clients



# Layered View (Cont.)

- Processing layer
  - > Process request
  - > Apply business logic
  - > Integrate with EIS
  - > Interact with peers



# Layered View (Cont.)

- Layered View helps to:
  - > Clearly divide responsibilities
  - > Decouple business logic completely
  - > Provide a common “place” for pre-processing and post-processing of requests and responses (like logging, translations, transformations, etc.)
  - > Expose a web service interface to existing business logic

# Interface Design

# Interface Design Considerations

## Development technique

- Two Choices
  - > JAVA™ to WSDL
  - > WSDL to JAVA™
- JAVA™ WSDL
  - > Easy—no need to know WSDL—rely on tools to create WSDL for you
  - > No finer control on the WSDL that is to be exposed to clients
  - > Might give rise to some semantic surprises

# Interface Design Considerations

## Development technique (Cont.)

- WSDL—Java™
  - > More control over what the service exposes
  - > Requires WSDL know-how
  - > Requires WS-I interoperability know-how to ensure interoperability
  - > Powerful but hard; chance of mistakes are more

# Choice of Endpoint

# Choice of Endpoint

- Two choices
  - > JAXRPC Service endpoint
  - > EJB™ Service endpoint
- Straightforward choices based on processing layer
  - > Web-centric architectures—  
JAXRPC Service endpoint
  - > EJB™-centric architectures—  
EJB™ Service endpoint

# Granularity of Service

- Coarse grained vs. fine grained
  - > More issues than you would face in case of remote EJB™ components
  - > Fine grained—almost never a good idea
  - > Coarse grained—efficient, minimizes overheads
  - > Prefer coarse grained services by consolidating logically related operations

# Interface and Parameters

- Prefer Java™ type parameters that have standard type mapping
- If passing XML documents
  - > Expect the documents to be wrapped and sent as `java.xml.transform.Source` Object
- Use of parameters without type mapping
  - > Extensible type mapping not standard
  - > Avoid as much as possible

# SOAP Message Handlers

# Handlers

- Handlers are specific to the port of a service
  - > All requests and responses go through the handlers
- From client's point of view—*handlers must be stateless*
  - > Storing port-specific state (like environment/QoS variables)
- Use Handlers for System-level services

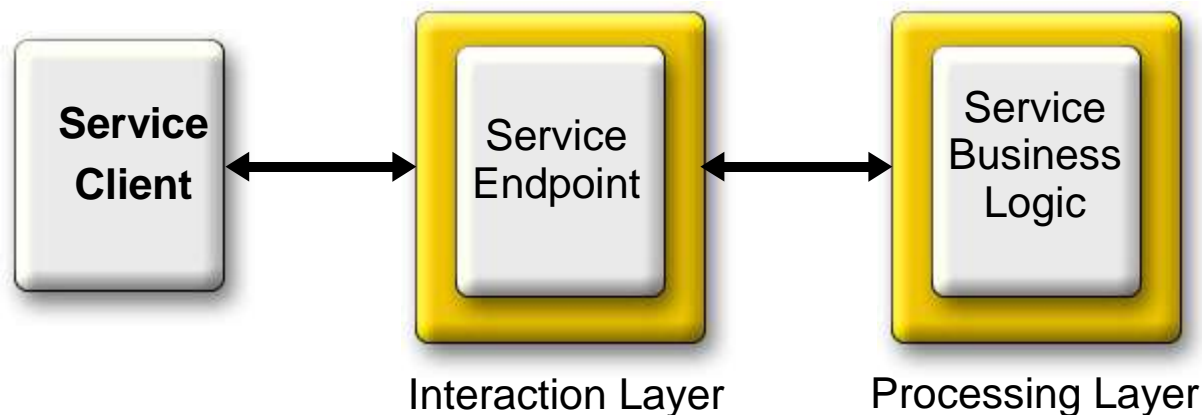
# SOAP Message Processing

# Requests and Response Processing

- Perform validation, transformation and other processing (like logging, etc.) as close to the endpoint as possible
  - > Helps catching errors early
  - > Helps support different client types with same business logic
  - > Helps in avoiding unnecessary round trips to the processing layer

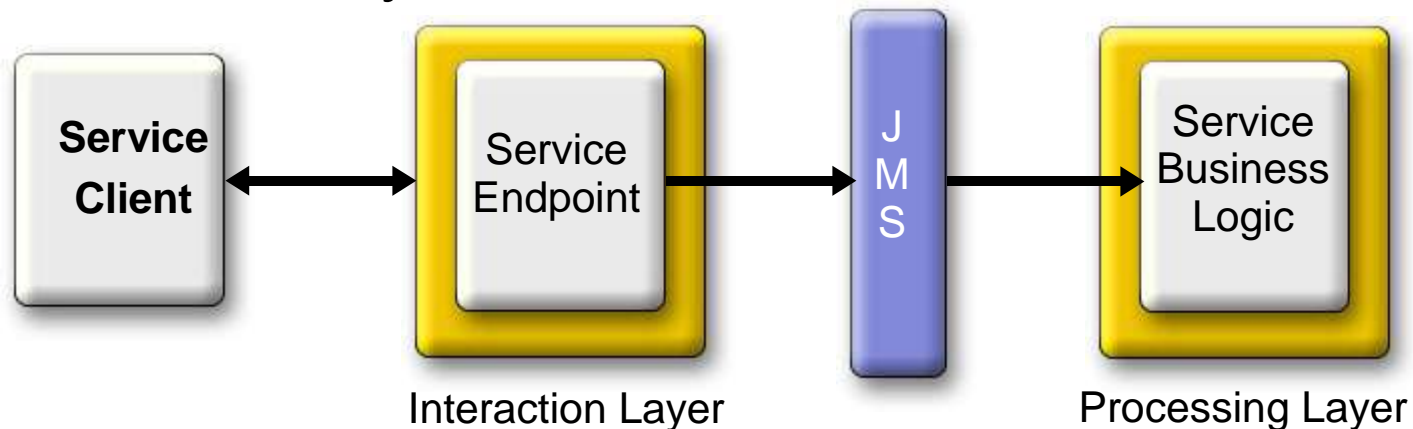
# Mapping Requests

- Two types of requests
  - > Short processing time—synchronous response
  - > Long processing time—asynchronous response
- Synchronous response
  - > Map requests directly to calls on business logic



# Mapping Requests (Cont.)

- Asynchronous response
  - > Map requests as messages on JMS
    - > No true asynchrony
- Endpoint may send an immediate reply with conversation ids
  - > Client may call back later for status



# Handling XML in Java Code

- Mapping to Java Objects
  - > JAXB
  - > Flexible Mapping with Java Objects
    - > Keep original document intact
    - > Map specific elements to Object methods
    - > Helps abstract document manipulation logic from Business logic
      - Helps in switching between different document manipulation mechanisms

# Web Services Technologies & Tools of Today

# Technologies You Can Use Today

- Mature standards
  - > SOAP 1.1, WSDL 1.1, UDDI, HTTPS, XML Schema
- Mature Java Web services technologies
  - > JAX-RPC 1.1, SAAJ, JAXR, JAXP, JAXB
  - > J2EE 1.4
- Recent standards
  - > Security standards: XML security, XML encryption, WSS (Oasis), SAML

# Implementations & Tools You Can Use Today

- Web services implementations
  - > J2EE 1.4 SDK
  - > Axis
- IDE's
  - > NetBeans (free), Eclipse (free), many more
  - > Java Studio Creator (free)
  - > SOA Editor (for building WSDL document, free)
  - > NetTool (For capturing SOAP traffic, free)

# Demo: Web Services Development using NetBeans

# Demo Scenario

- Building a simple HelloWorld Web service using NetBeans 4.1 or 5.0 beta
- Invoking the HelloWorld Web service from a Web application
- You can try this hands-on lab yourself
  - > [http://www.javapassion.com/handsonlabs/5310\\_netbeanswebservices.zip](http://www.javapassion.com/handsonlabs/5310_netbeanswebservices.zip) (unzip it and read index.html document to proceed)

# SOAP vs. REST

# REST

- Messages are represented in plain XML
- HTTP is used for the transfer protocol
- HTTP verbs are used for access/manipulation commands
- URIs are used to uniquely identify resources in message
- HTTP authentication provides security
- There is no formal method for expressing the interface contract

# SOAP

- Messages are represented in a standardized XML SOAP "envelope"
- Can be bound to various protocols including HTTP and SMTP
- Access to and manipulation of data are application specific
- XML schemas are used to define the contract between client and service

# Reasons for REST (over SOAP)

- REST is an attractive choice for more basic applications that involve high levels of interoperability between multiple platforms
- REST provides better performance than SOAP

# Reasons for SOAP (over REST)

- SOAP is very appropriate for larger, formal, sophisticated applications that require advanced capabilities between relatively homogenous systems
- Web services standards such as WS-\* make rich functionality easy to deliver quickly
- SOAP supports multiple protocols

# Web Services In the Context of Frameworks

# Remoting Support in Spring Framework

- Spring features integration classes for remoting support using various technologies
- The remoting support eases the development of remote-enabled services, implemented by your usual (Spring) POJOs

# Remoting Technologies Supported by Spring

- Remote Method Invocation (RMI)
- Spring's HTTP invoker
- Hessian
- Burlap
- JAX RPC

# Light-weight Web Services Framework

# Hessian/Burlap Binary Web Service Protocol

- Is a simple binary protocol for connecting web services
  - > Makes web services usable without requiring a large framework
  - > Can be used in smaller clients, like applets
- Well-suited to sending binary data without any need to extend the protocol with attachments
  - > Because it is a binary protocol

# When To Use and When Not to Use Hessian/Burlap

- Reasons for using it
  - > Simple to use
  - > Light-weight
  - > Higher performance
- Reasons for not using it
  - > Not for industrial-strength - No concurrent request handling

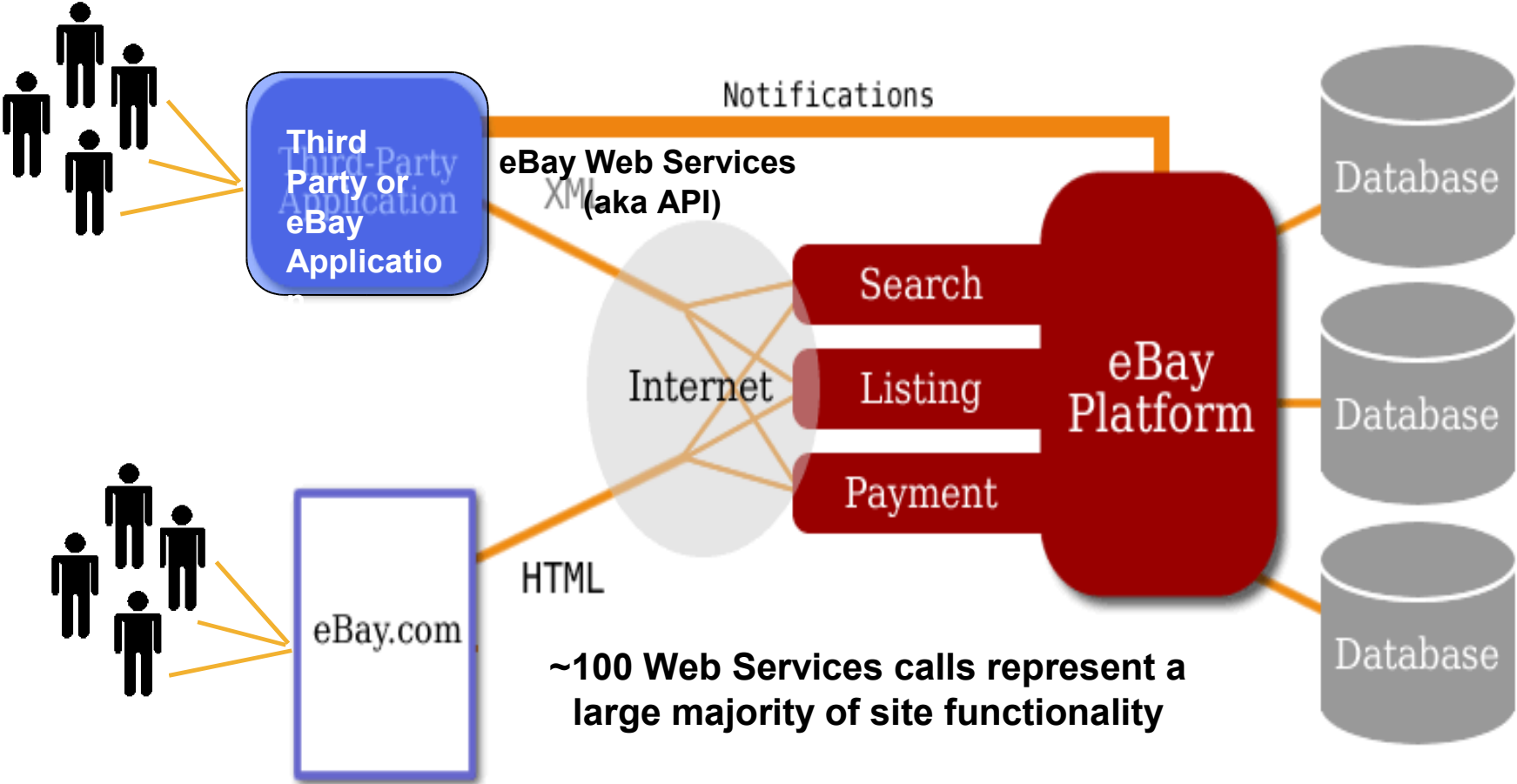
# Examples of Real-world Web Services

- eBay
- Amazon.com
- RouteOne

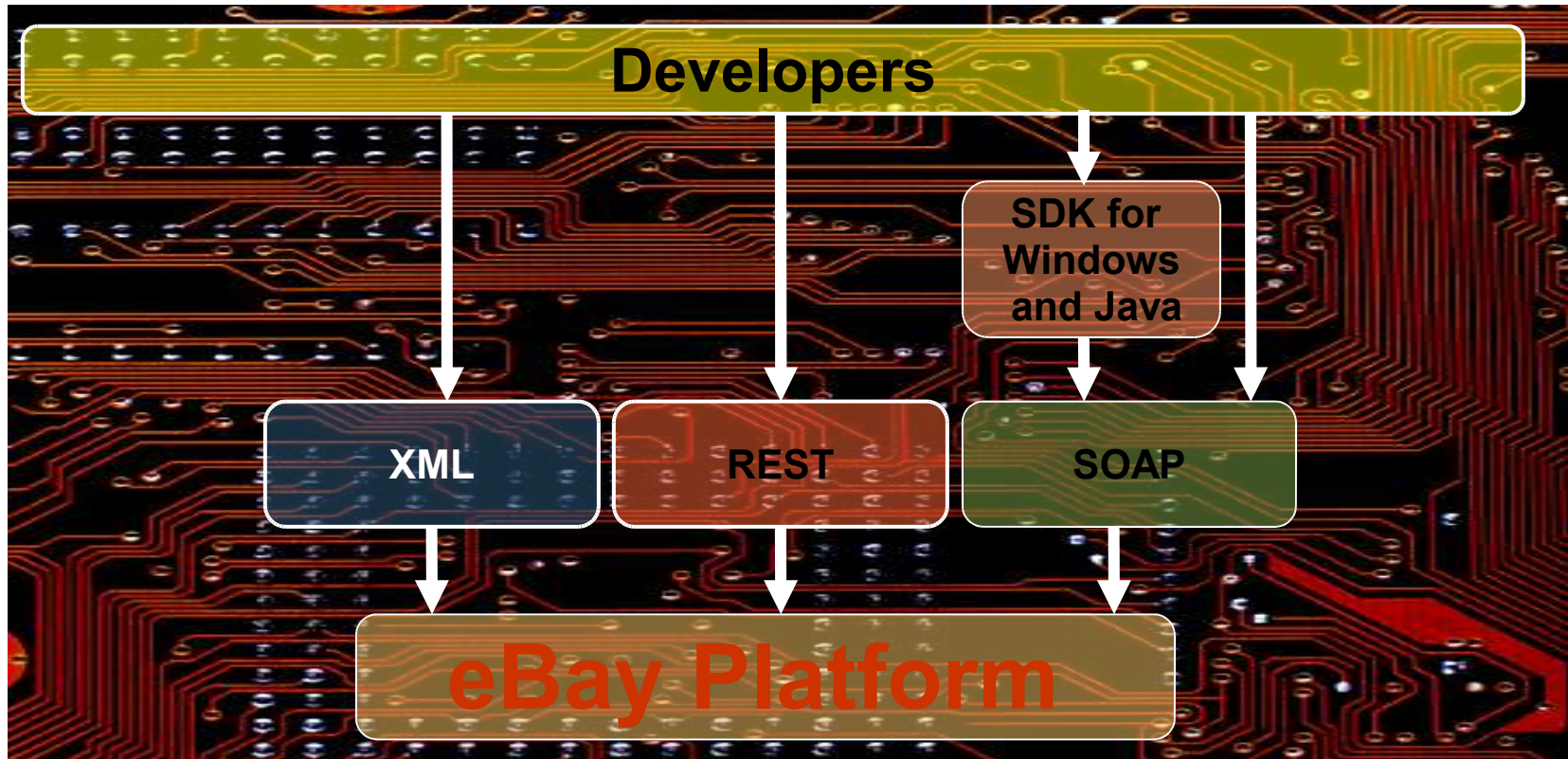
# eBay Ecosystem



# eBay's Technology Platform



# What Is eBay Web Services?



# What Is Amazon Web Services?

- Developer interface to Amazon.com
- APIs that give any developer outside of Amazon programmatic access to Amazon's data and technology:
  - > Product information
  - > Third-party seller information
  - > Customer-created content
  - > Amazon's shopping cart
  - > And much more...

# Why Does Amazon Offer AWS? (According to Amazon)

- Amazon has valuable data and technology!
  - > Partners (Associates, sellers) want our data and technology
  - > We don't want them to scrape our site
  - > Third-party developers extend Amazon's technology platform
  - > They implement features we never consider
- Lead ecommerce and Web development
  - > Driving fundamental shift in software development
  - > Leveraging others' innovation
  - > Empowering entrepreneurs

# Amazon Web Services Offering

- SOAP API
- REST API
- XSLT Transformation Service
- WSDL
- Schema

# Who Uses AWS?

- Over 100,000 developers
- Amazon Associates
- Amazon Marketplace Sellers
- Amazon Merchants
- General Developers
- Integrators and Solution Providers

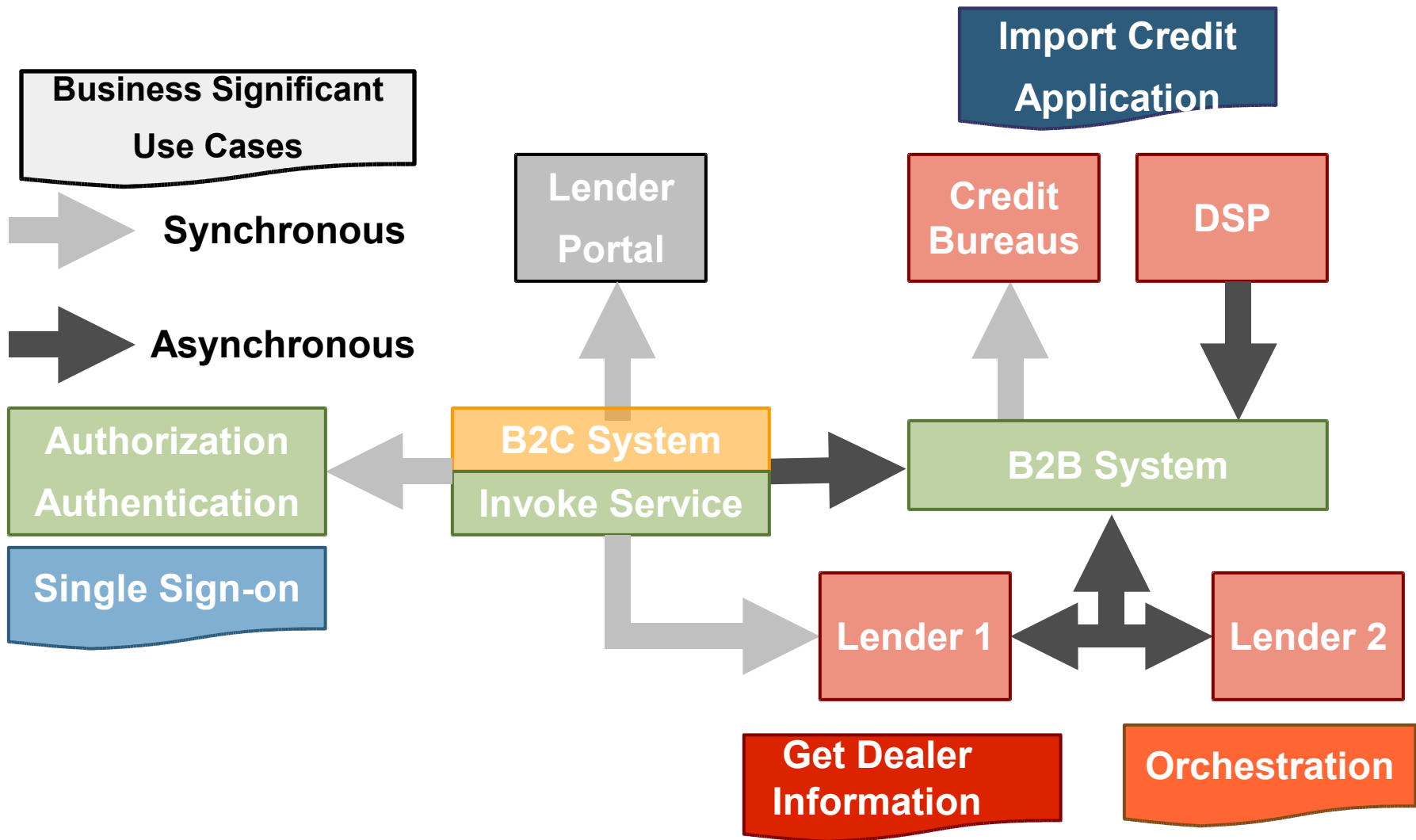
# Examples of Real-world Web Services

**-RouteOne**

# Requirements

- Single-Sign-On (SSO)
- B2B messaging
- Asynch. messaging
- Security
- Reliability

# Architecture



# Single Sign-on

- Using SAML and XML-DSig
- OASIS standard
- XML framework to exchange User Credentials
- Basis for the Liberty Alliance Project
- Supported by many vendors and products
- Standard SSO Profiles and Bindings
  - > Browser Artifact Profile, Browser POST Profile

Single Sign-on

# B2B Messaging

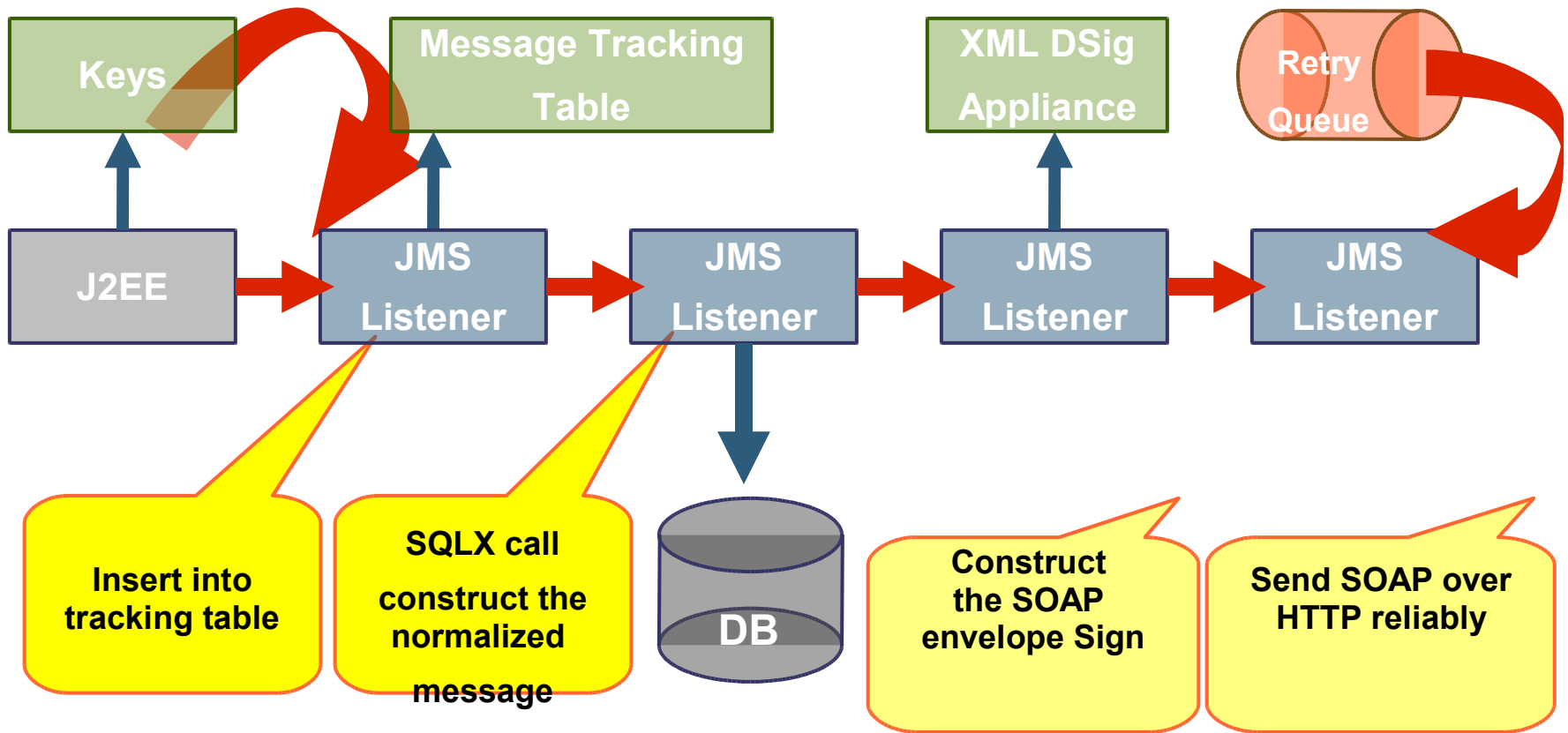
- WSDL Bindings
- Document and Binary Exchange Strategies
- Synchronous Messaging
  - > RPC/Literal, Custom Java binding
- Asynchronous Messaging
  - > Document/Literal
  - > SOAP over HTTPS and MQ
  - > STAR standard complaint payload (SOAP Body)
- Orchestration
  - > Performed using custom code
  - > SOAP Headers
  - > Includes Reliability, Auditing, TimeOut, Fault, etc.
  - > Messages digitally sealed using XML Dsig

Get Dealer  
Information

Import Credit  
Application

Orchestration

# Async Messaging Architecture



# Security—XML Signature

- XML Signature (XML-DSig)
  - > Digital signatures for XML documents
  - > W3C standard
- Provides
  - > Message Authenticity (Who sent this message?)
  - > Message Integrity (Is this what was sent?)
  - > Non Repudiation (Can the sender deny sending this message?)
- Supported by many vendors

# Reliable Message Exchanges

- Goals
  - > Guaranteed delivery—“at least once”
  - > Duplicate elimination—“at most once”
  - > Delivery in the right sequence
- Current Standards—ebXML, Java Message Service (JMS), MQ
- Competing standards
  - > WS—Reliability OASIS (Sun, Sonic, Oracle)
  - > WS—Reliable Messaging (IBM, Microsoft, BEA)
- WSRM—WS-Reliability 1.1 is an OASIS standard; released in November 2004

# Questions?