# Service Oriented Architecture & Web Services

Jan Jusko

FEE CTU

April 12, 2011

# Outline

# Corporate IT: Former State

- several independent software products - each department had its own set of applications
- client-server, mainframe based
- limited amount of data to process
- a lot of user effort
- associated costs (time, employee, business)

# Corporate IT: Transition

- business-processes optimization
  - time costs
  - employee costs
  - new business models
- a growing need for intra- and inter- company communication
  - mergers
  - outsourcing
- stronger business intelligence
- more extensive reporting

# Corporate IT: A Bank Example

- more than a dozen separate applications
- point in case: mortgages
  - verify line of credit
  - create a credit account
  - create a savings account
  - create a mortgage
  - interest rate change
- we can automate the process
- redesigning the mortgage process
- use of SOA

# SOA: The Definition

- **What it IS NOT:** Web Service(s)
- **What it IS:** a flexible set of design principles used during the phases of system development and integration
- SOA provides a way for consumers to find available SOA-based services

# Why SOA?

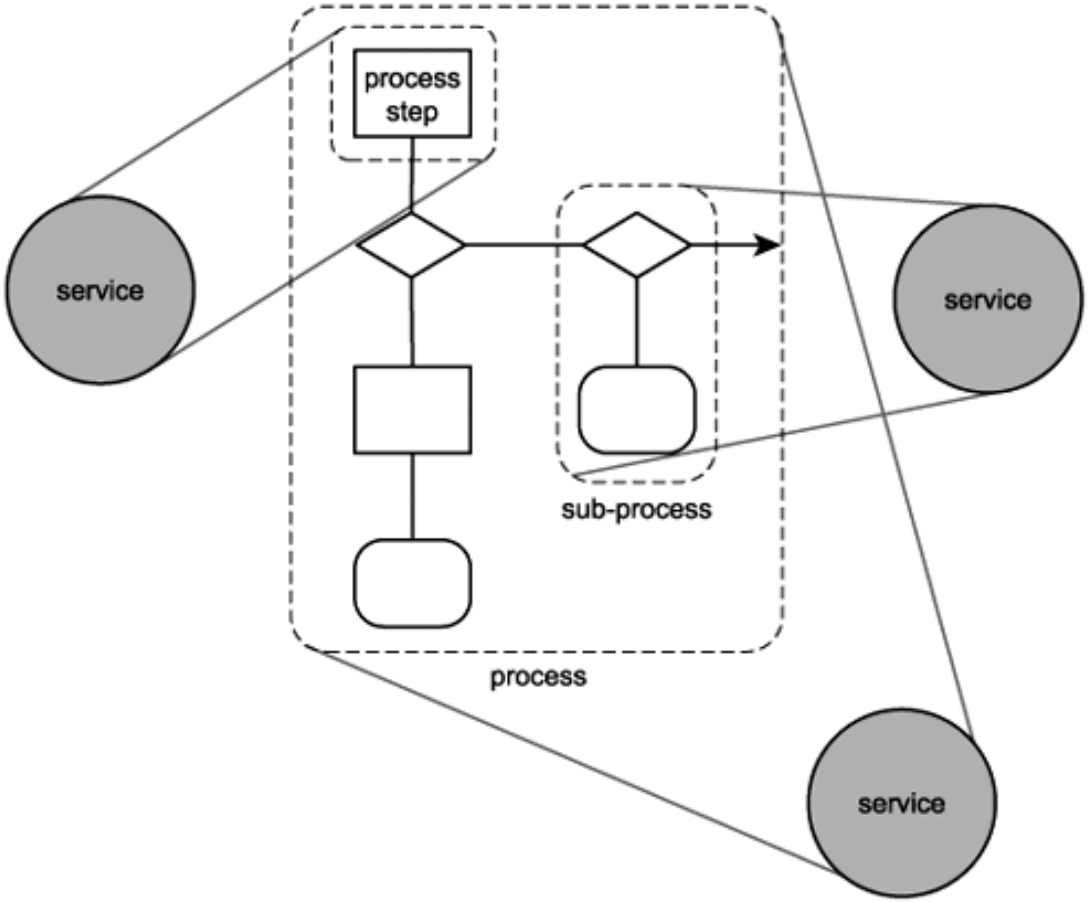
- Drivers
  - Large-scale Enterprise Systems
  - Internet-Scale services providers
  - reducing the cost of doing business
- Benefits
  - scalable, evolvable systems
  - encourage re-use of business functions

# Essence of SOA

- reflects the reality of ownership boundaries (in contrast to RMI and CORBA - those create transparent distributed systems)
- is task oriented, services are organized by function
- implementations rely on a mesh of software services

# More on SOA

- provides a loosely-integrated suite of services that can be used within multiple business domains
- defines how to integrate widely disparate applications for a world that is distributed and uses multiple implementation platforms
- defines the interface in terms of protocols and functionality
- requires loose coupling of services with operating system and other technologies that underlie applications
- separates functions into distinct units, or services, which developers make accessible over a network


process step
service
service
sub-process
process
service

# Services

- allow users to combine and reuse them in the production of applications
- services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services

# Services

- services comprise unassociated, loosely coupled units of functionality that have no calls to each other embedded in them
- each service implements one action
- instead of services embedding calls to each other in their source code they use defined protocols that describe how services pass and parse messages, using description metadata

# Services Principles

- reuse
  - a service once created can be reused for various application
- granularity
  - a question of the optimal choice of granularity
  - fine-grained means large overhead, coarse-grained means small reusability
- modularity
  - services can be deployed independently
- composability
  - services are recombinant and can be selected and assembled in various combinations
- componentization
  - services are simple units that can be easily combined
  - inspired by hardware manufacture
- interoperability
  - services are capable of working together

# Services Principles

- standards compliance
  - common & industry-specific
- services identification and categorization, provisioning
- service encapsulation
  - many services are consolidated for use under the SOA; often such services were not planned to be under SOA
- service loose coupling
  - services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other

# Services Principles

- service contract
  - services adhere to a communications agreement, as defined collectively by one or more service-description documents
  - header (name, version, responsible organization, type), functional (functionality, invocation) and non-functional (security, QoS) description
- service abstraction
  - beyond descriptions in the service contract, services hide logic from the outside world
- service autonomy
  - services have control over the logic they encapsulate

# Services constraints

- stateless service
  - each message that a consumer sends to a provider must contain all necessary information for the provider to process it. Each request can be treated as generic. There are no intermediate states to worry about, so recovery from partial failure is also relatively easy. This makes a service more reliable.
- stateful service
  - holds a session between a consumer and a provider. Stateful services require both the consumer and the provider to share the same context. It may reduce the overall scalability of the service as it needs to remember shared context for each consumer.
- idempotent requests
  - repeated execution has the same effect as a single execution

# SOA implementations

- **Web Services**
- RPC
- REST
- DCOM
- WCF
- ...

# Web Service Definition

- a software system designed to support interoperable machine-to-machine interaction over a network
- uses WSDL (Web Service Definition Language) to define an interface
  - a machine-processable format
- communicate with other services in a manner prescribed by its description using SOAP messages
- uses HTTP with XML serialization as a communication protocol
- uses UDDI for service discovery

# Web Service Types

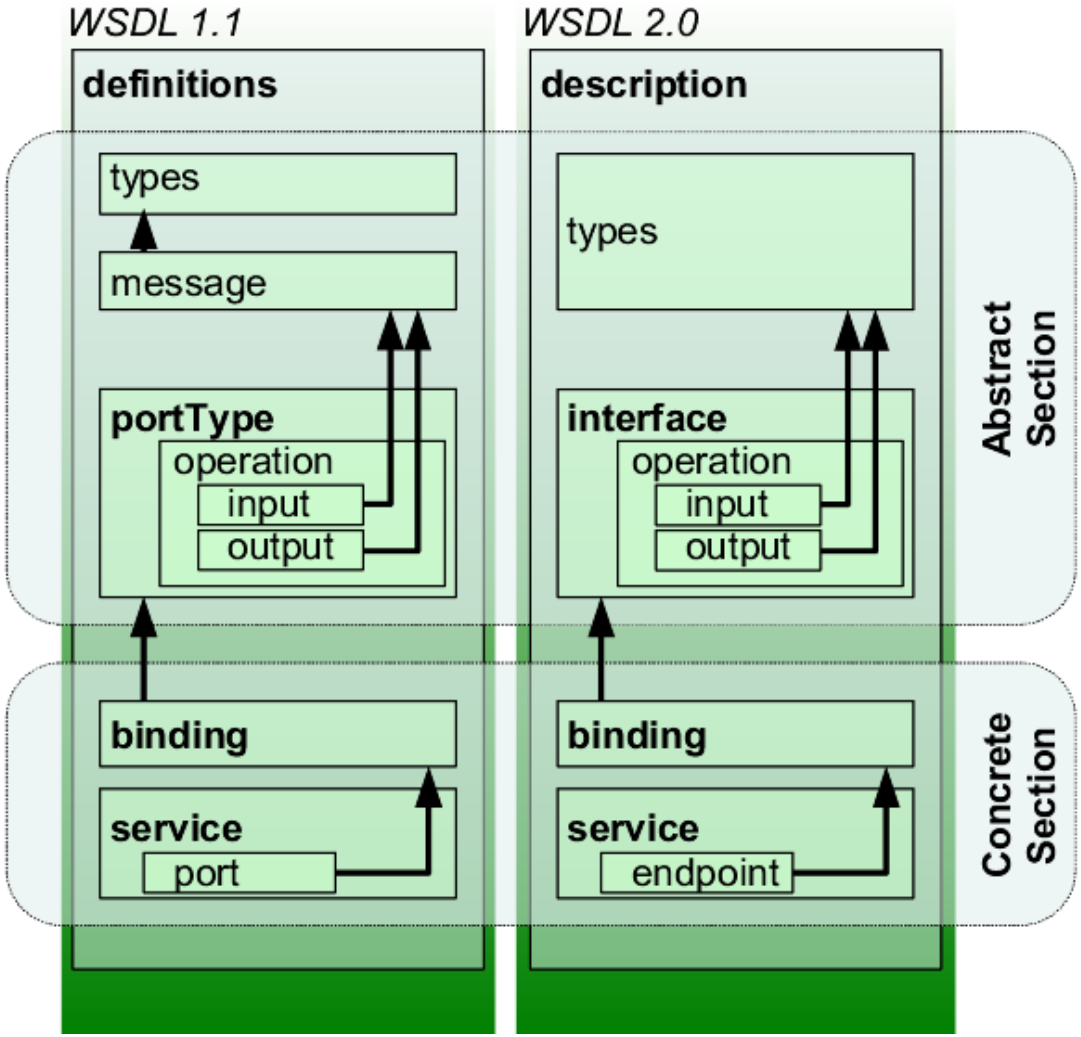
- RESTfull Web Services
  - Primary purpose is to manipulate XML representations of Web resources
  - Uniform set of stateless operations
- "Big" Web Services
  - expose arbitrary set of operations
  - (can be) stateful

# WSDL - Web Service Definition Language

- XML-based language that provides a model for describing Web services
- Defines services as collections of network endpoints, or ports associating a network address with a reusable bindings
- Describes the public interface of the web service

# WSDL Versions

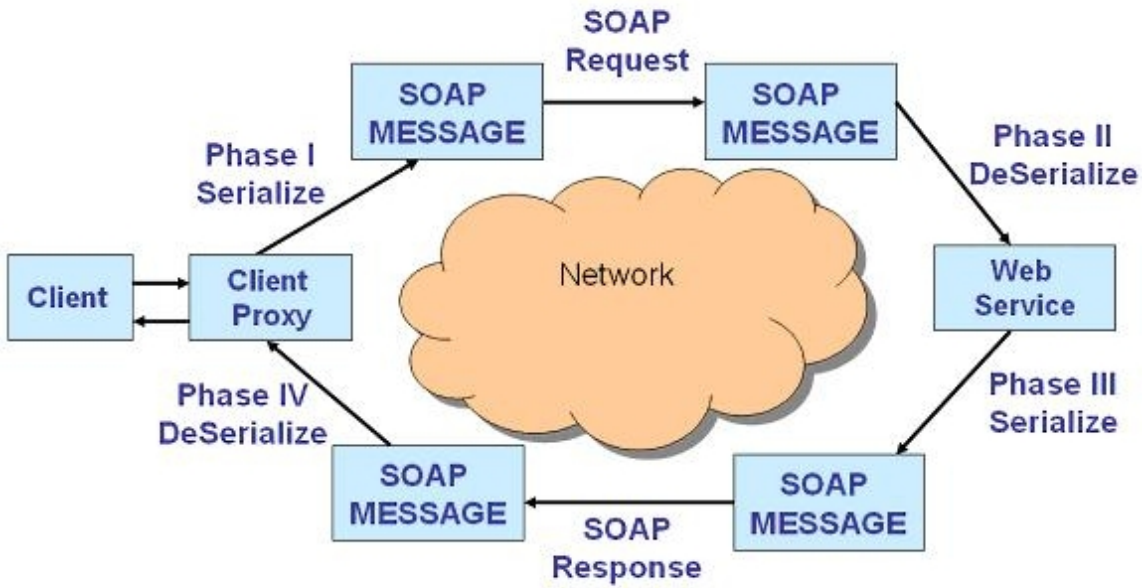
- WSDL 1.0 (2000), 1.1 (2001) – developed by IBM, Microsoft, and Ariba
- WSDL 1.2 (2003) – W3C draft; easier and more flexible for developers; attempts to remove noninteroperable features
- WSDL 2.0 (2007) – W3C recommendation


WSDL 1.1
definitions
types
message
portType
operation
input
output
binding
service
port
WSDL 2.0
description
types
interface
operation
input
output
binding
service
endpoint
Abstract Section
Concrete Section

# SOAP - Simple Object Access Protocol

- Protocol specification for exchanging structured information for WS
- Relies on XML for its message format
- Uses other application layer protocols (RPC, HTTP)
- Originally designed as an object-access protocol
- Underlying layer for Web Services, based on WSDL and UDDI

# SOAP Call


SOAP
Request
SOAP
MESSAGE
SOAP
MESSAGE
Phase I
Serialize
Phase II
DeSerialize
Client
Client
Proxy
Network
Web
Service
Phase IV
DeSerialize
Phase III
Serialize
SOAP
MESSAGE
SOAP
MESSAGE
SOAP
Response

# SOAP Nodes

- SOAP sender
- SOAP receiver
- SOAP message path
- Initial SOAP sender (Originator)
- SOAP intermediary
- Ultimate SOAP receiver

## SOAP Example

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/...">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

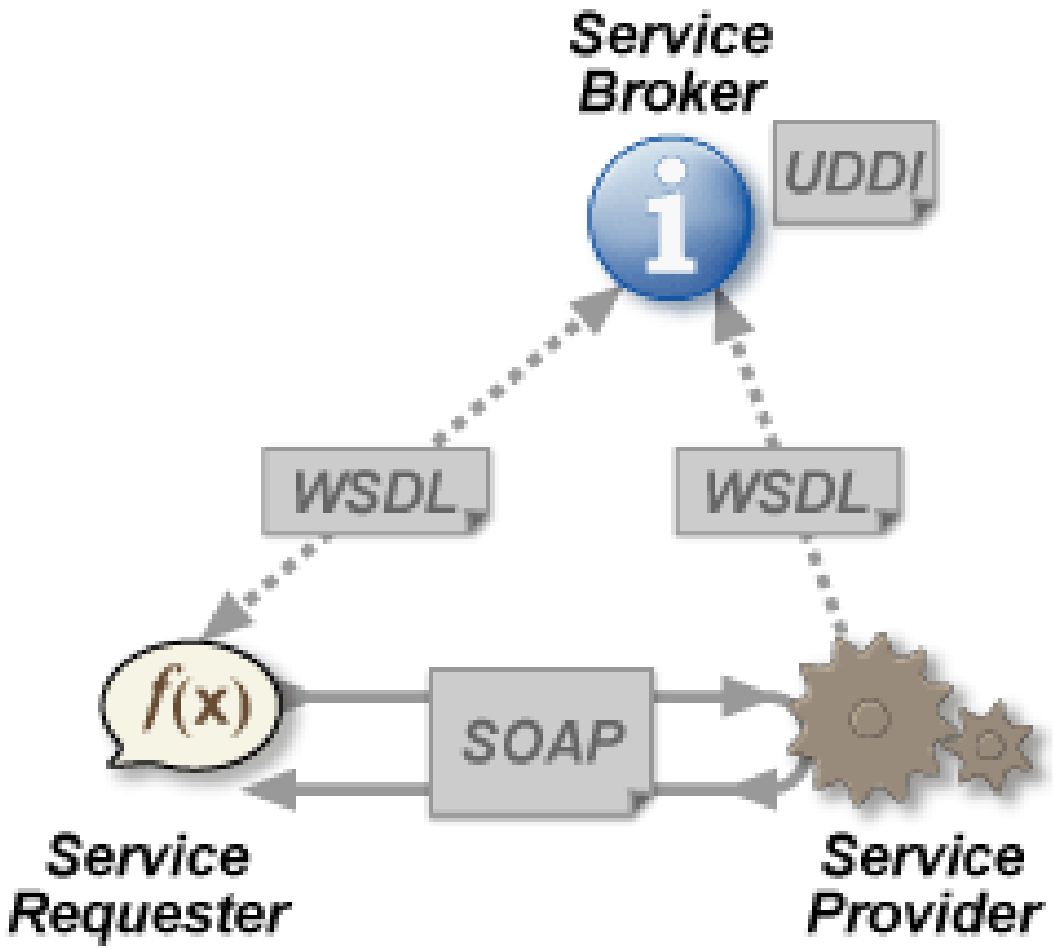
# UDDI

- Universal Description, Discovery and Integration
- Platform-independent XML-based registry for businesses worldwide to list themselves on the Internet
- Enables businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet
- brokerage system

# UDDI components

- White pages - address, contact, and known identifiers
- Yellow pages - industrial categorizations
- Green pages - technical information about services exposed by the business

# Web Service Architecture



# Web Service Design

- Bottom up
  - First write the implementing class, then generate WSDL
  - Considered simpler
  - Language/platform dependence/influence risk
- Top down - industrial categorizations
  - first write the WSDL document, then generate class skeleton
  - Considered more difficult
  - Produce cleaner designs

# References


- A4M33AOS materials by Jiri Vokrinek (http://cw.felk.cvut.cz/doku.php/courses/a4m33aos/start)
- Service-Oriented Architecture: Concepts, Technology, and Design by Thomas Erl