

SOA & Web Services

Trainer: Phong Le
PSE



Warm up - Introductions

- Your role
- Your background, experiences in the subject
- What do you want from this course

Agenda

- Service Oriented Architecture
- Web Services Overview
- XML Schema & Namespace
- XML Messaging – SOAP
- Service Description – WSDL
- Web Services in Action
- Web Services vs. Other Distributed Technologies

Course Audience and Prerequisite

- The course is for those who want to gain basic knowledge of Web Services.
- The following are prerequisites to SOA & Web Services:
 - XML
 - Java programming

Assessment Disciplines

- Class Participation: 70%
- Assignment: 30%

Course Duration

- Theory: 2.5 hours
- Practice: 1.5 hours

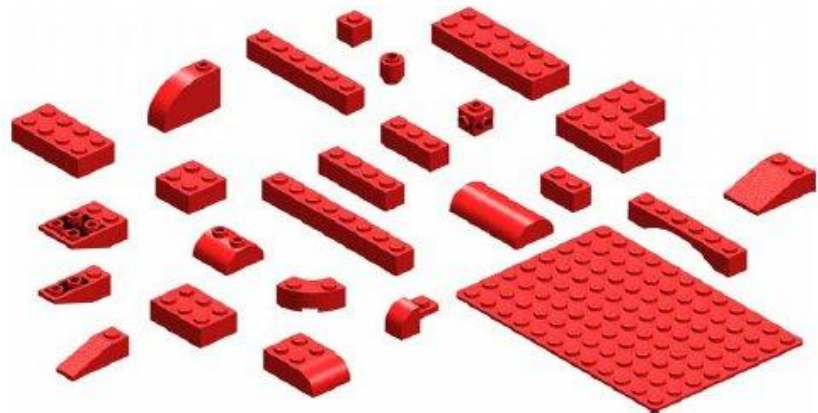
Service Oriented Architecture

Need for Decomposing a Monolith

Today



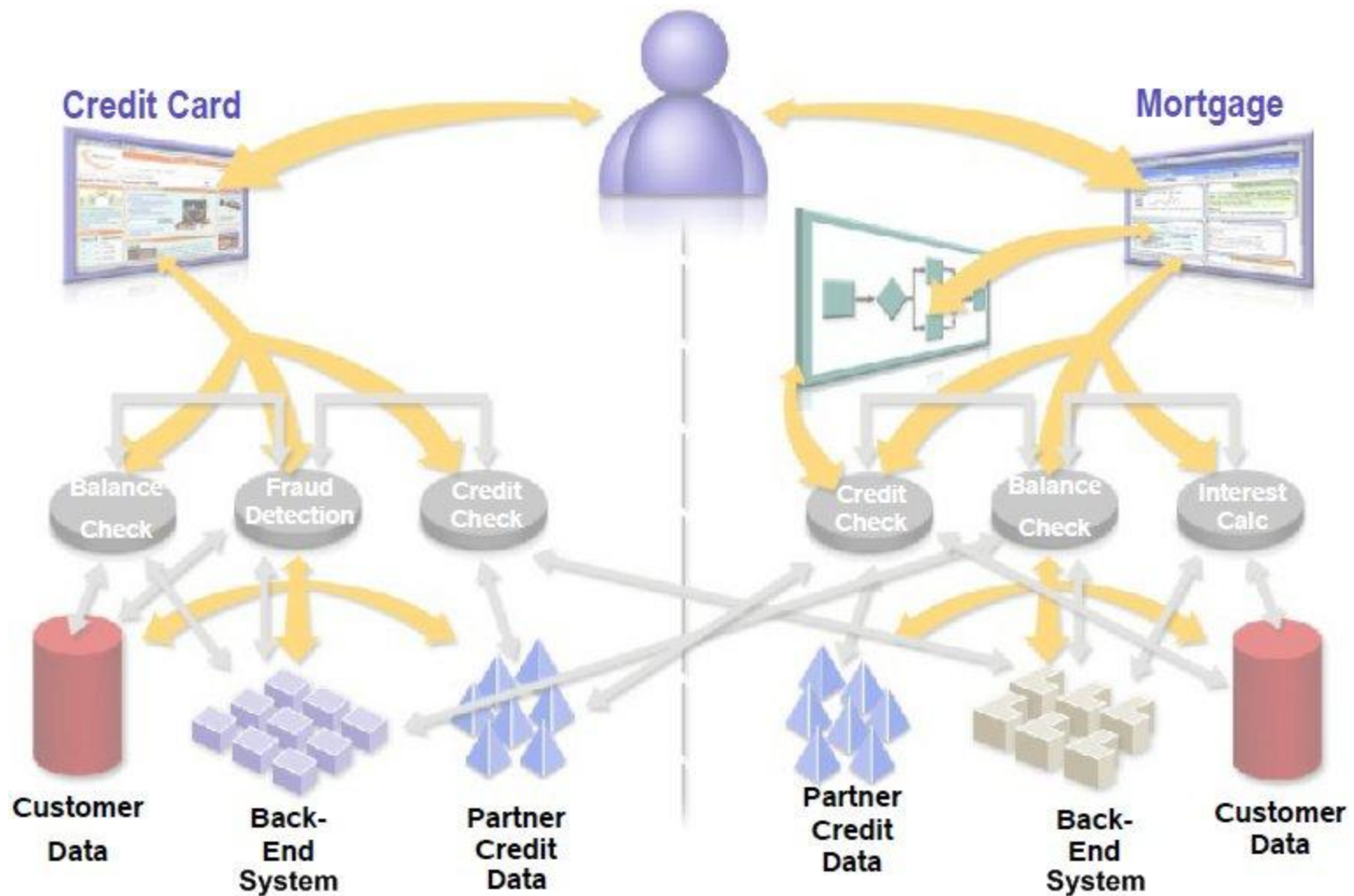
Tomorrow



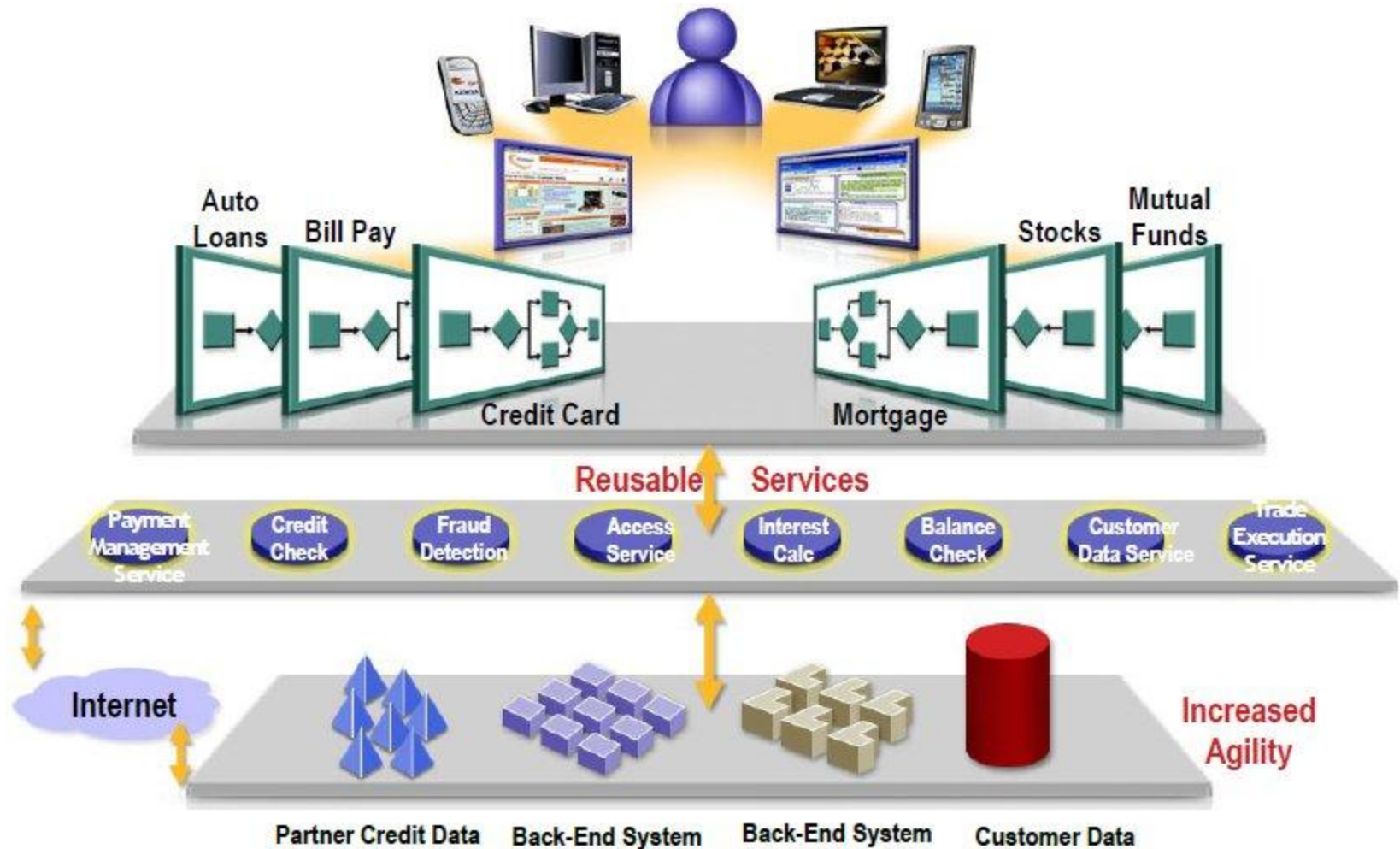
Future



Pre-SOA Scenario



SOA-Based Scenario



What is SOA?

A set of services that a business wants to expose to their customers and partners, or other portions of the organization.

Business Analyst



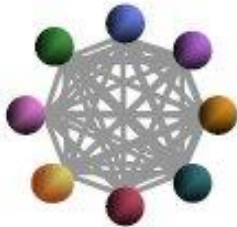
A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.

Software Architect

Evolution of SOA

Component-Based Development

Messaging Backbone



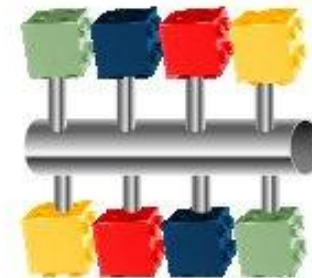
- Point-to-Point connection between applications
- Simple, basic connectivity

Enterprise Application Integration (EAI)



- EAI connects applications via a centralized hub
- Easier to manage larger number of connections

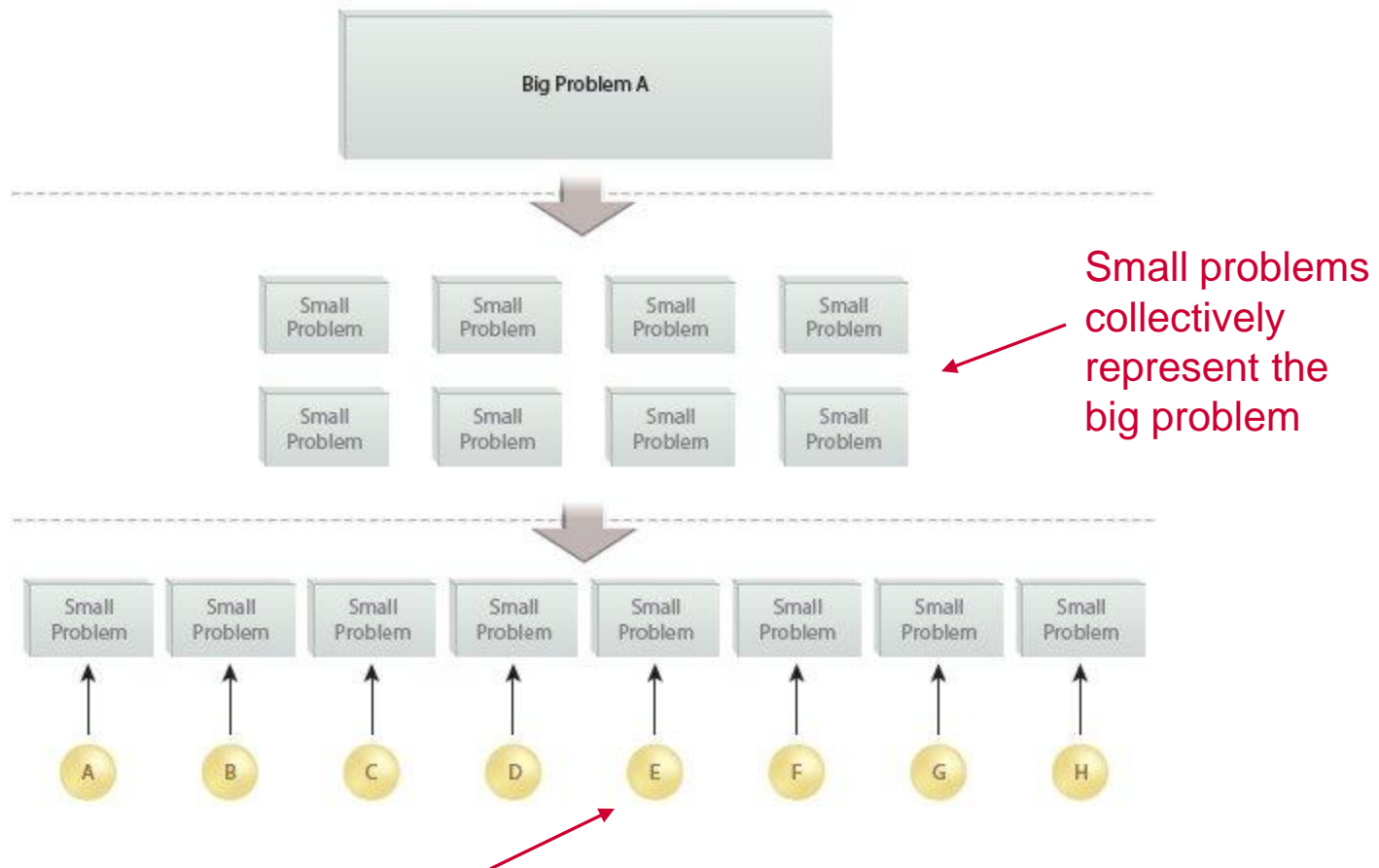
Service-Oriented Architecture



- Integration and choreography of services through an Enterprise Service Bus
- Flexible connections with well defined, standards-based interfaces

Flexibility

SOA Principles – Separation of Concerns

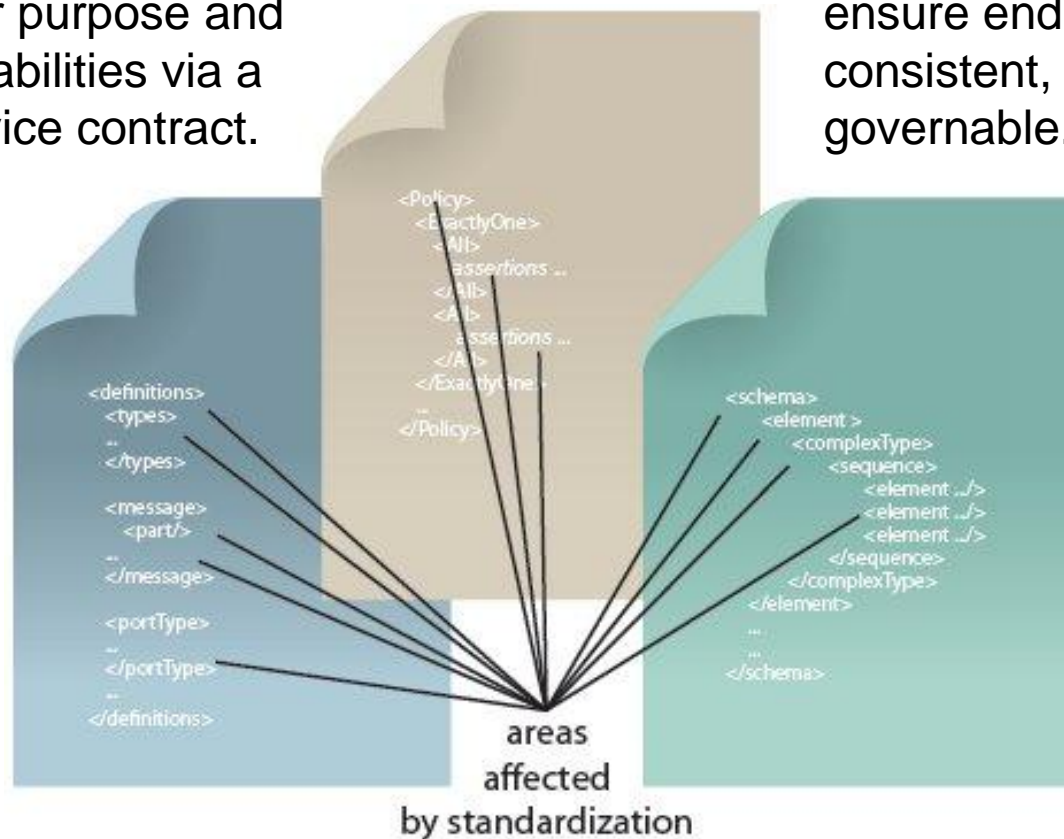


Units of solution logic that each address (solve) a small problem.

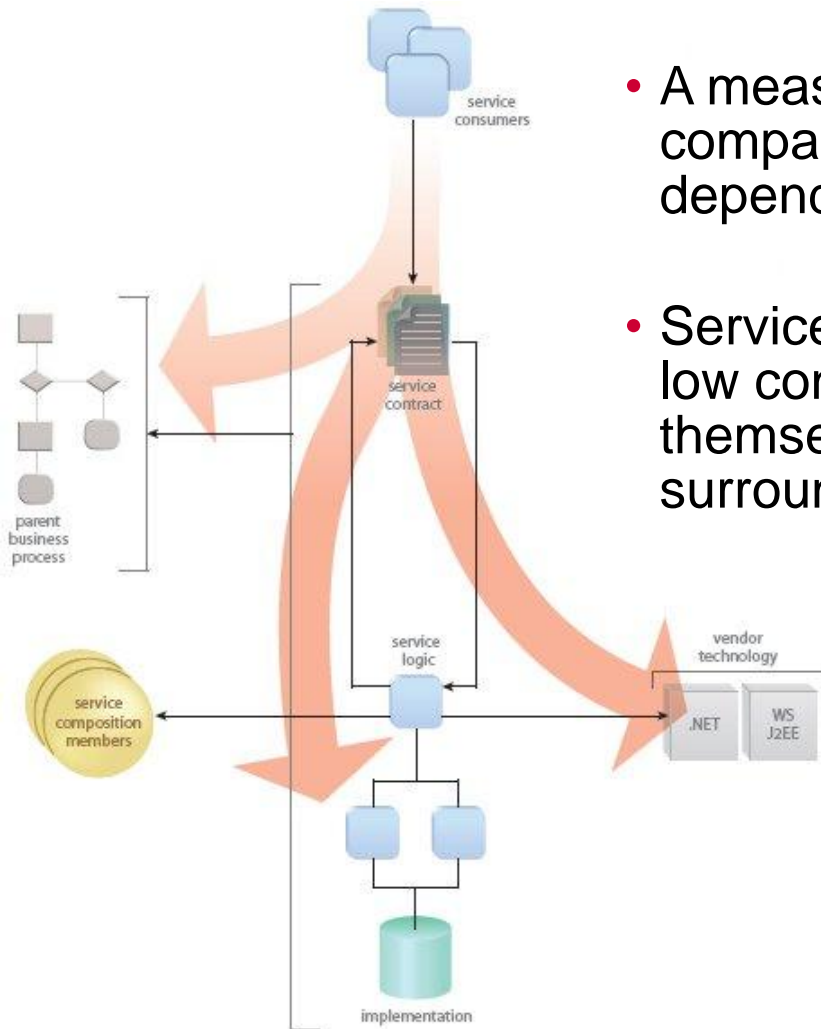
SOA Principles – Standardized Service Contracts

Services express their purpose and capabilities via a service contract.

Standardized contracts ensure endpoints are consistent, reliable, and governable.

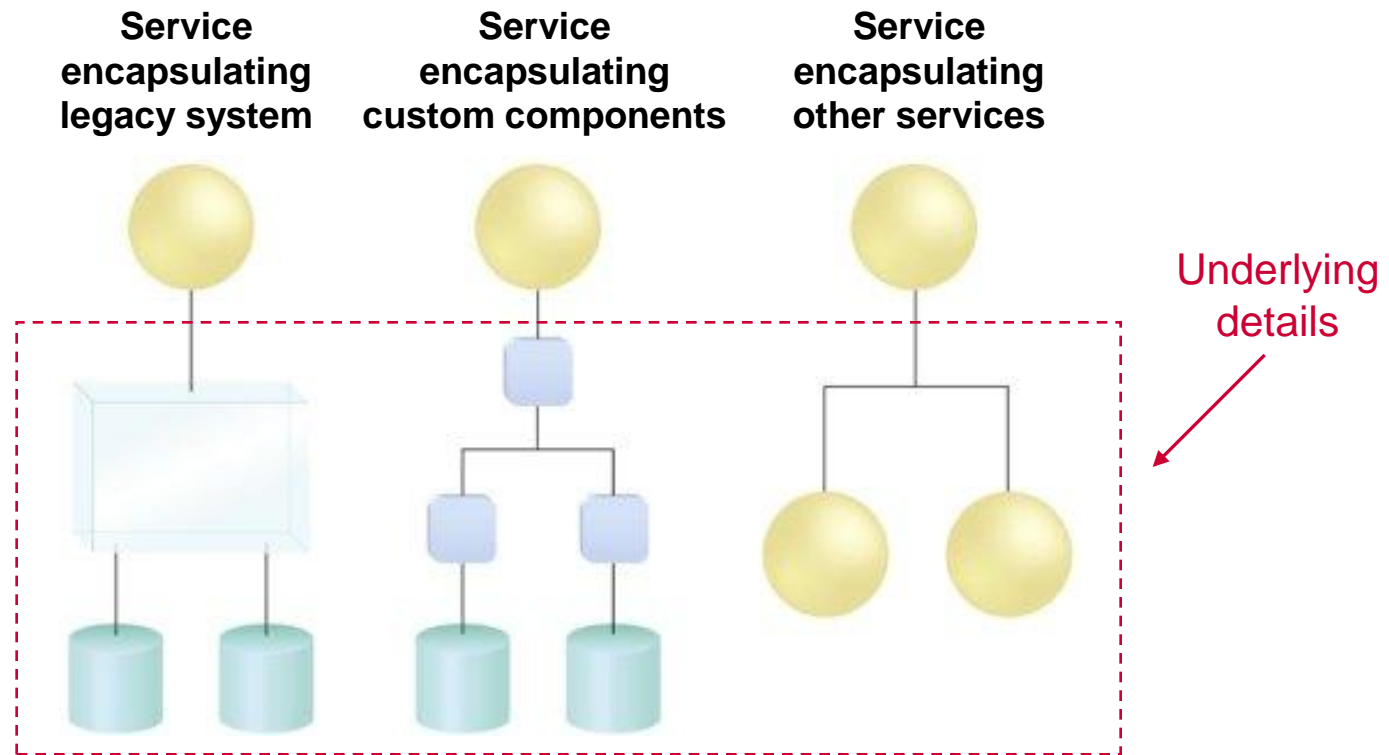


SOA Principles – Loose Coupling



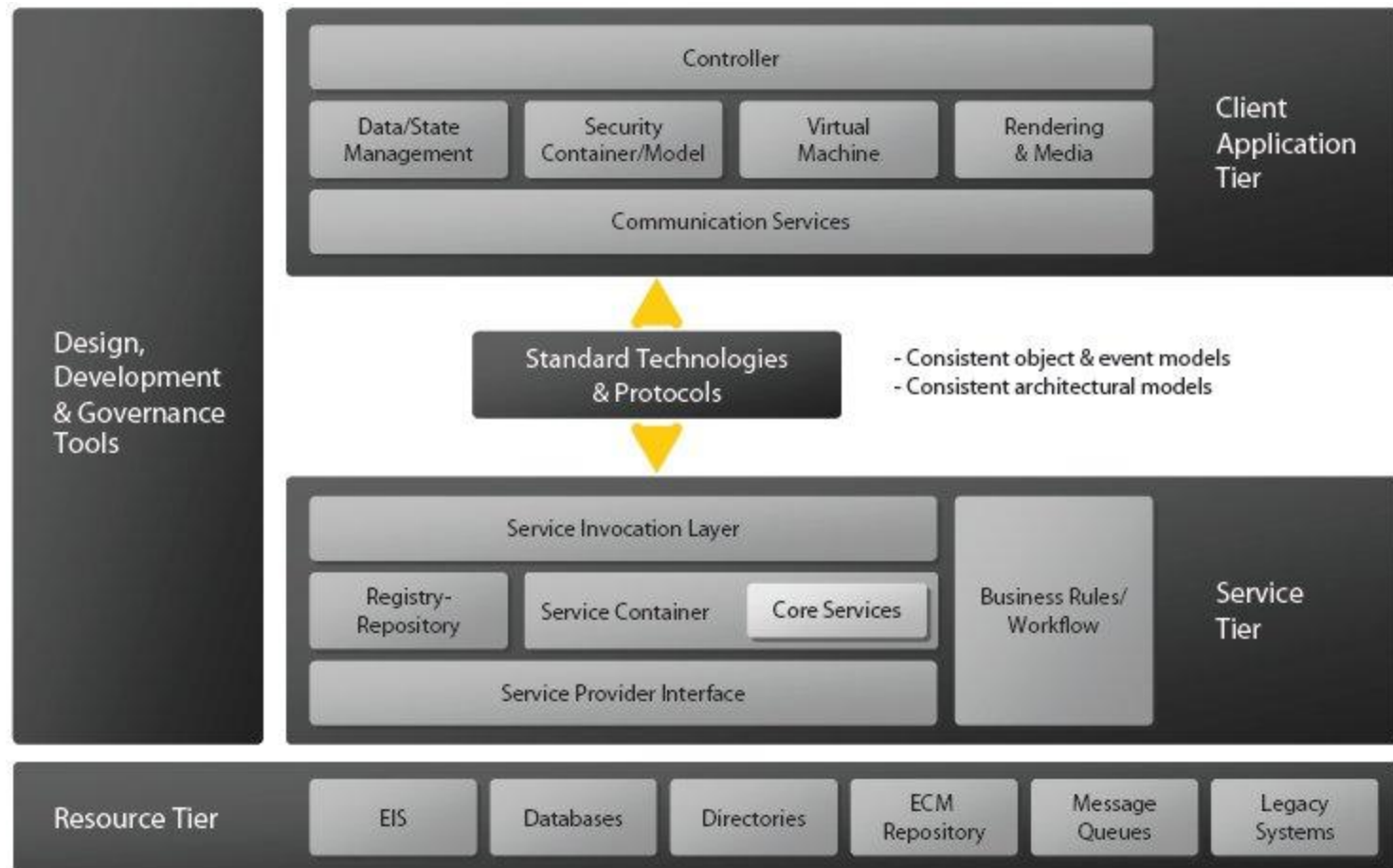
- A measure of coupling is comparable to a level of dependency.
- Service contracts need to impose low consumer coupling and are themselves decoupled from their surrounding environment.

SOA Principles – Service Abstraction



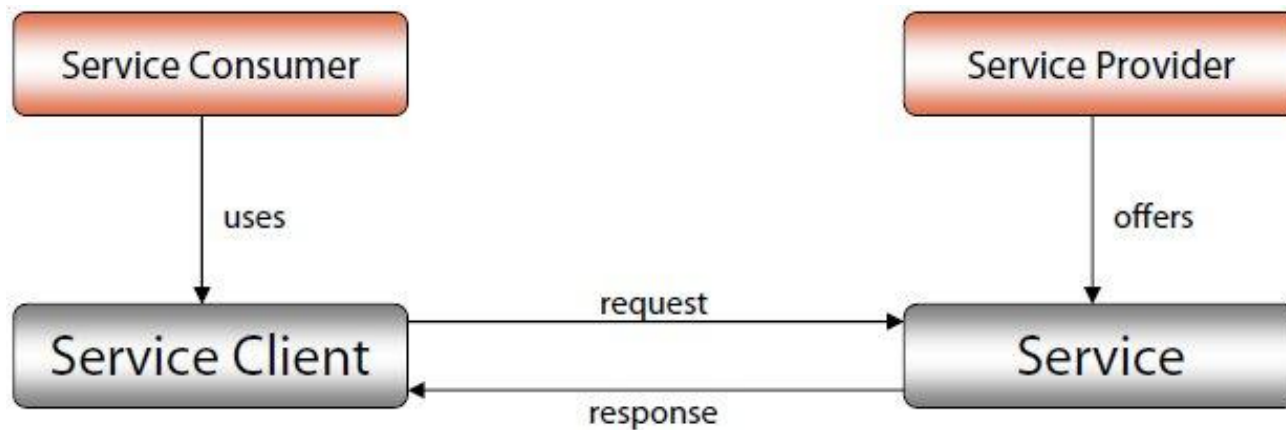
- This principle emphasizes the need to hide as much underlying details of a service as possible.
- Doing so directly enables and preserves the loosely coupled relationship.

Reference Architecture



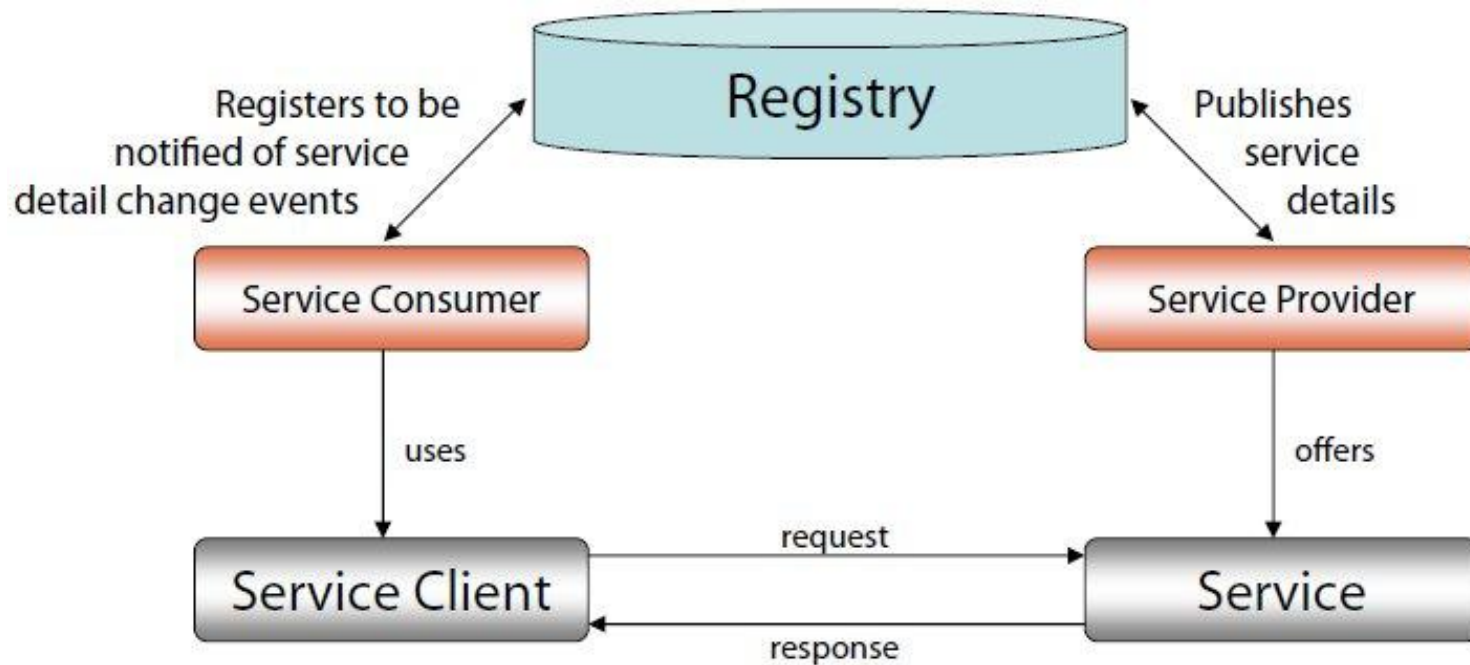
SOA Patterns

- Request-Response

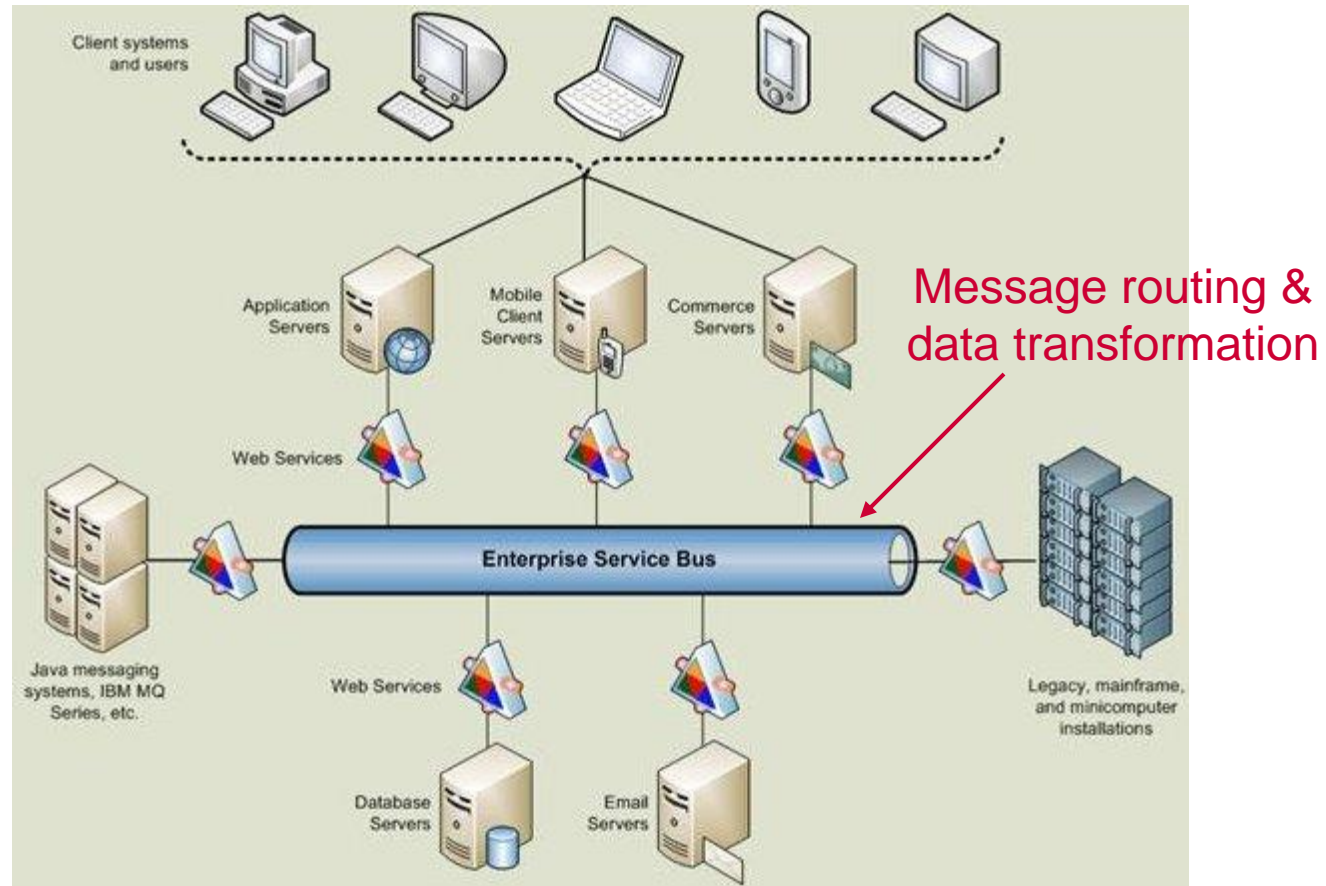


SOA Patterns (cont.)

- Request-Response via Service Registry



SOA Patterns – Enterprise Service Bus

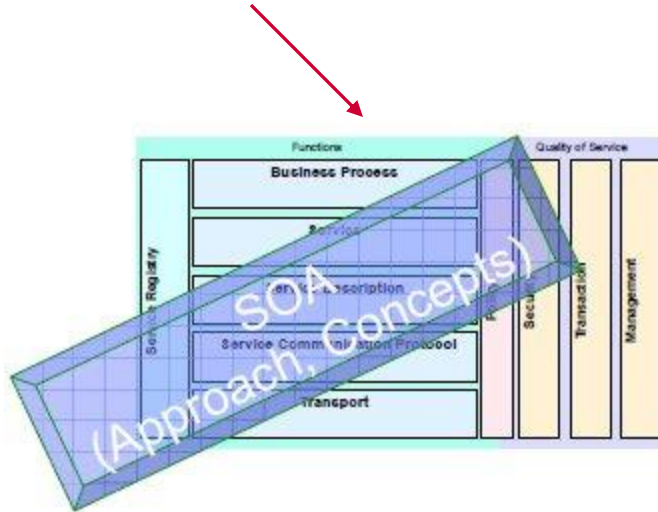


Business Benefits of SOA

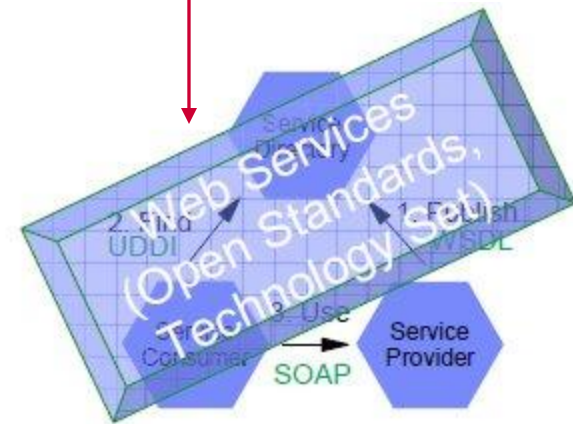
- More quickly adapt to business need changes.
- Leverage existing investments in technology.
- Reduce reliance on expensive custom development.
- Lower costs associated with the acquisition and maintenance.

Realizing SOA with Web Services

SOA is a conceptual architectural style.



Web services provide technology standards for SOA.



SOA may not need to use Web services, but Web services are the preferred way to realize SOA.

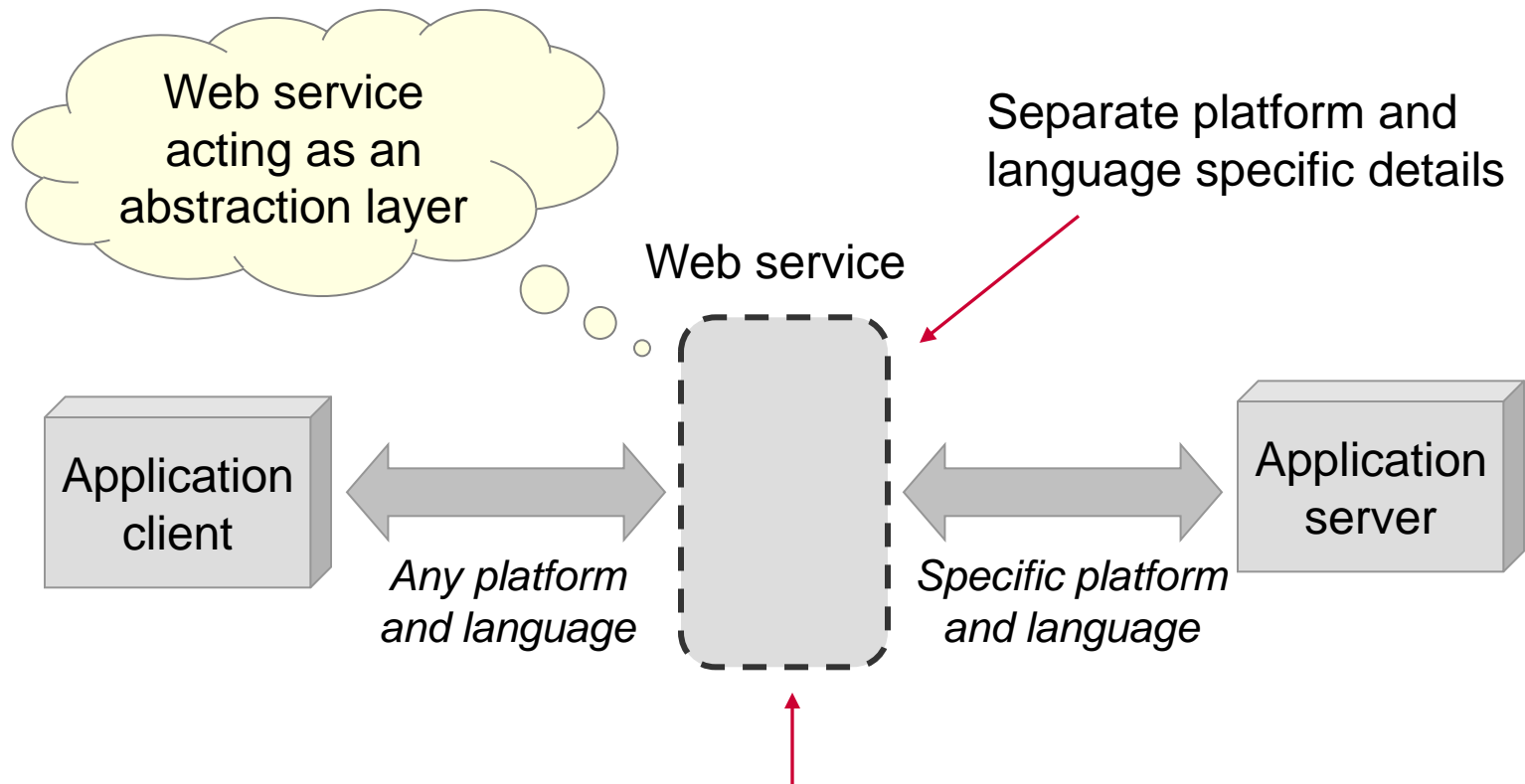
Web Services Overview

What is a Web Service?



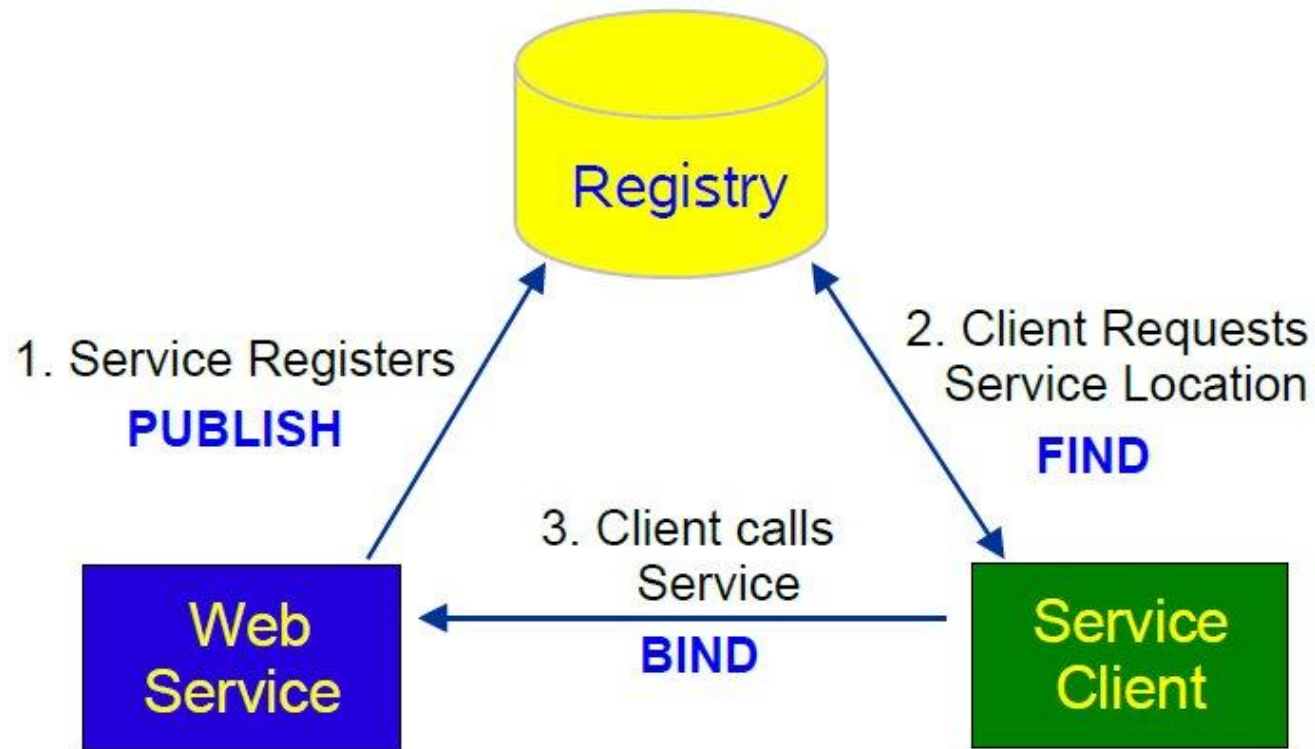
- A software system designed to support interoperable machine-to-machine interaction over a network.
- Other systems interact with the Web service in a manner prescribed by its description.
- Messages are typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web Service Fundamentals

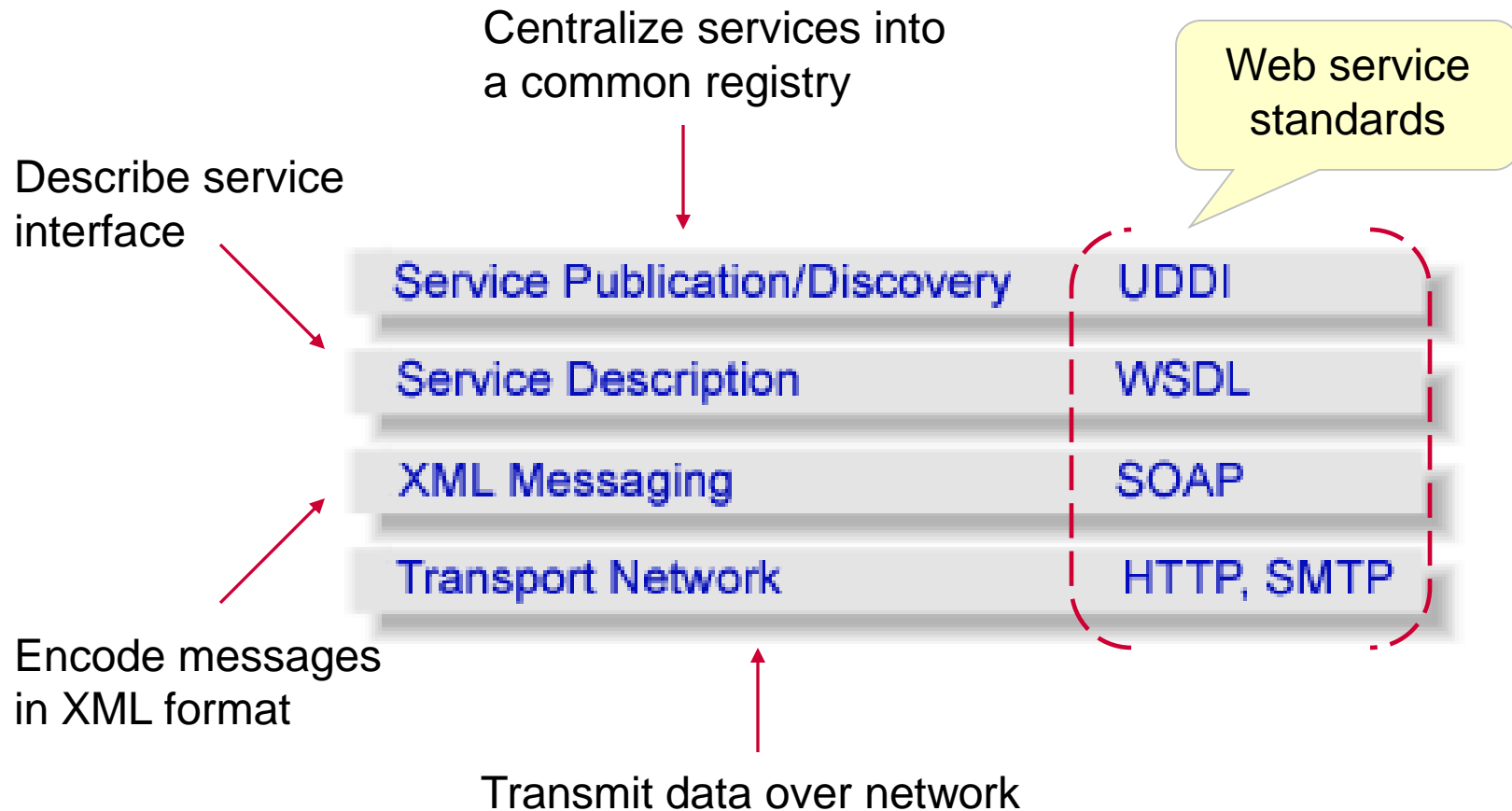


Through this standardized layer, any language that supports web services can access the application's functionality

Web Service Model



Web Service Stack

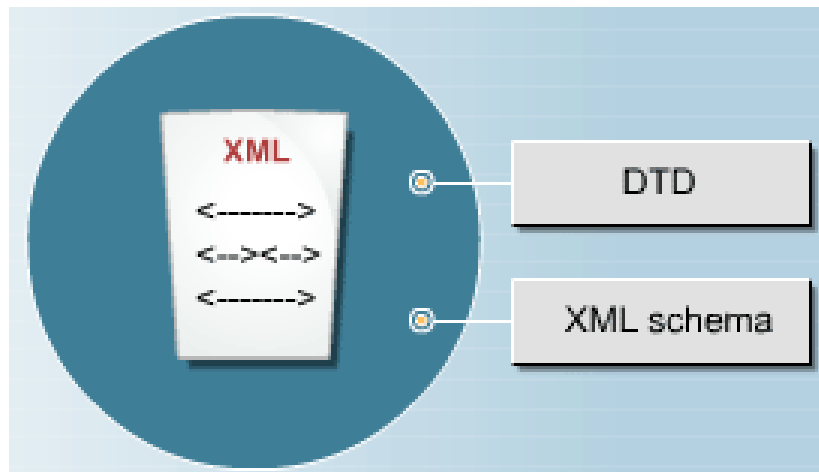


Characteristics of Web Services

- Loose coupling
- Dynamic binding
- Cross-platform interoperability
- Programming language independent
- Using neutral standards

XML Schema & Namespace

XML Schema



Describe the structure
of an XML document

- XML Schema has several advantages over DTD
 - XML Schemas are written in XML
 - XML Schemas allow to define custom data types
 - XML Schemas are extensible to future additions

XML Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="book_type"> ← Custom data type
    <sequence>
      <element name="title" type="string" />
      <element name="author" type="string" />
    </sequence>
  </complexType>

  <element name="book" type="tns:book_type" />
</schema>
```

Built-in data type

Root element

XML Document That Adheres to the Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:book xmlns:tns="http://www.example.org/BookSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/BookSchema
BookSchema.xsd">

  <tns:title>Love Story</tns:title>
  <tns:author>Erich Segal</tns:author>
</tns:book>
```


Motivation for Namespaces

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

← Carries HTML
table information


```
<table>
  <name>Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

} If these XML fragments
were added together,
there would be a name
conflict.

← Carries information
about a table (furniture)

Motivation for Namespaces (cont.)

Solving the name
conflict using a prefix



```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```


Namespace Declaration

Syntax: `xmlns:prefix="URI"`

```
<root xmlns:h="http://www.w3.org/TR/html4/"
      xmlns:f="http://www.w3schools.com/furniture">

  <h:table>
    <h:tr>...</h:tr>
  </h:table>

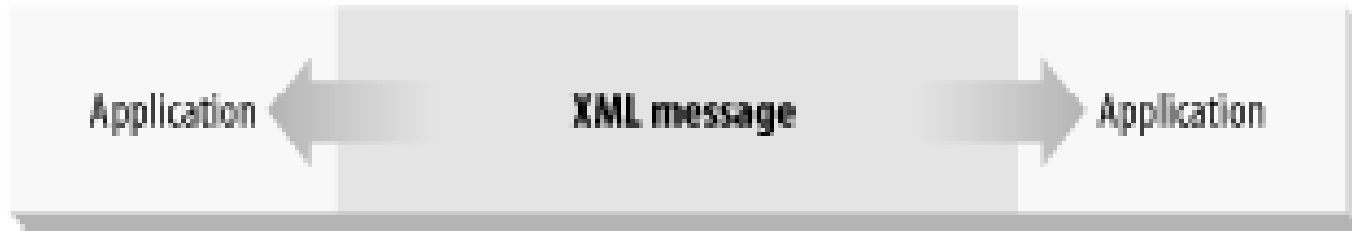
  <f:table>
    <f:name>...</f:name>
  </f:table>
</root>
```



Namespaces can be declared in a root element or at a lower level one

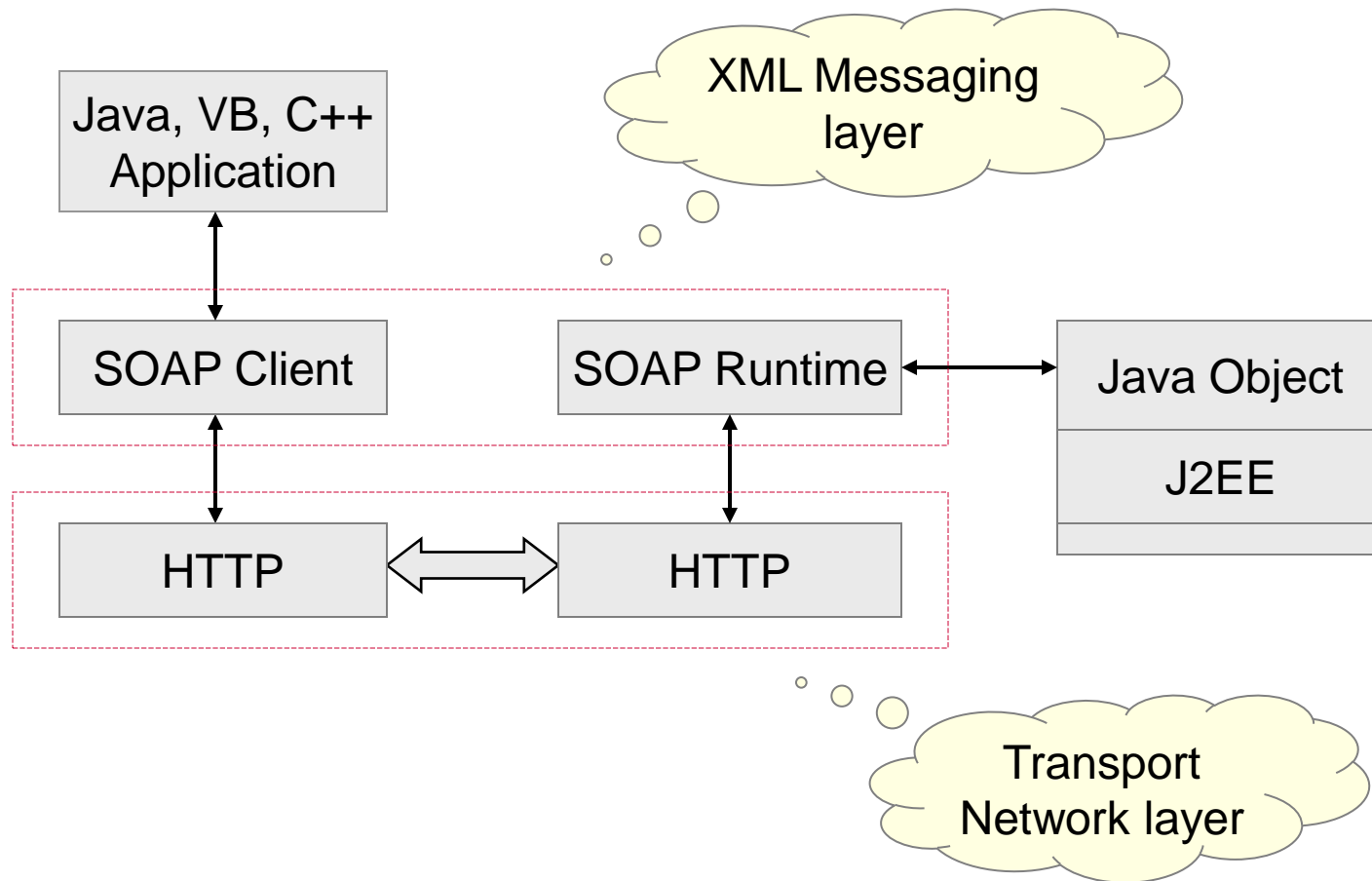
XML Messaging – SOAP

Introducing SOAP

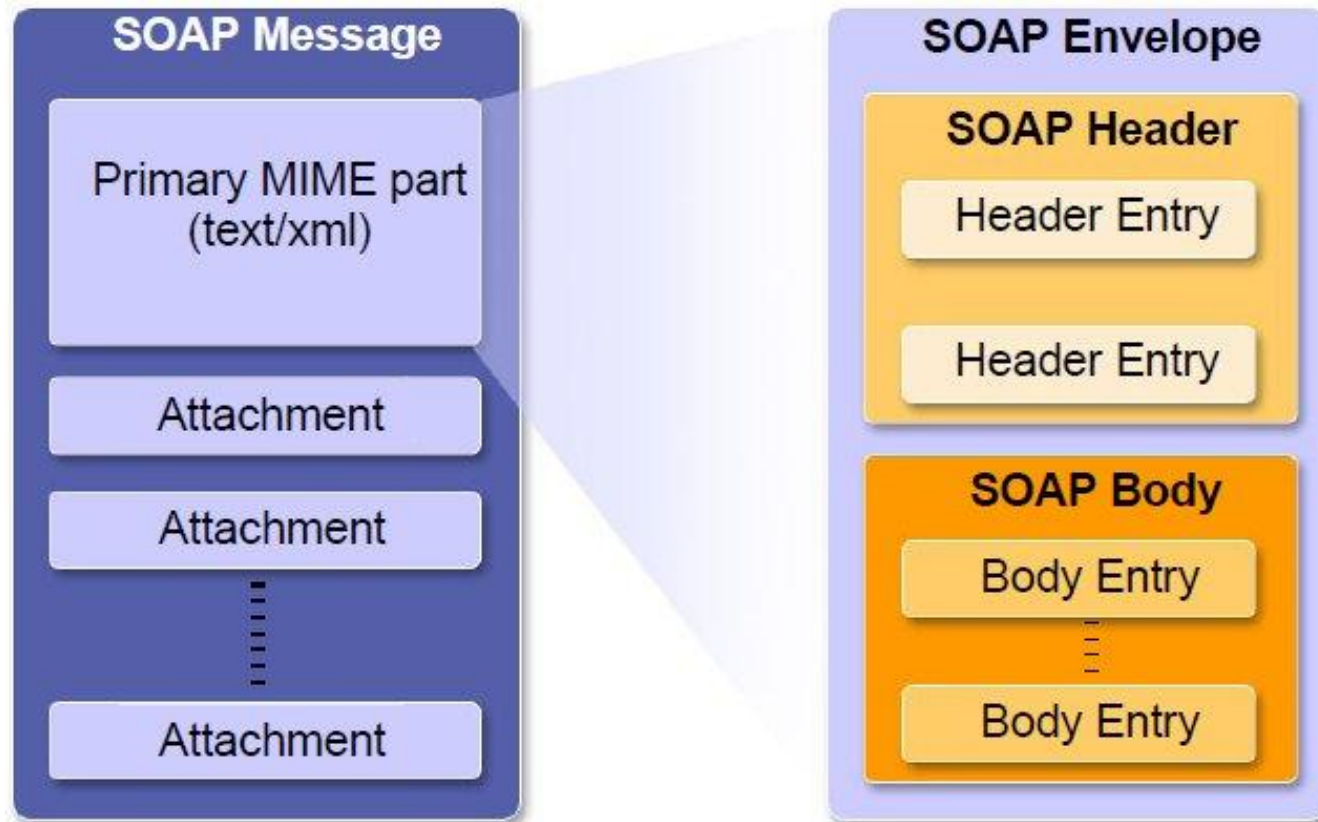


- **S**imple **O**bject **A**ccess **P**rotocol
- Lightweight distributed computing protocol
- Uses XML as its message format
- Transports over HTTP, SMTP, JMS, etc.

SOAP over HTTP



SOAP Message Structure



SOAP Envelope

- Encoding style
 - Defines how to express application data types in XML
- Header
 - Optional
 - Contains context knowledge (security, transaction, routing, etc.)
 - Can be handled by intermediaries
- Body
 - Mandatory
 - Contains application data
 - Handled only by ultimate receiver

SOAP Envelope Example

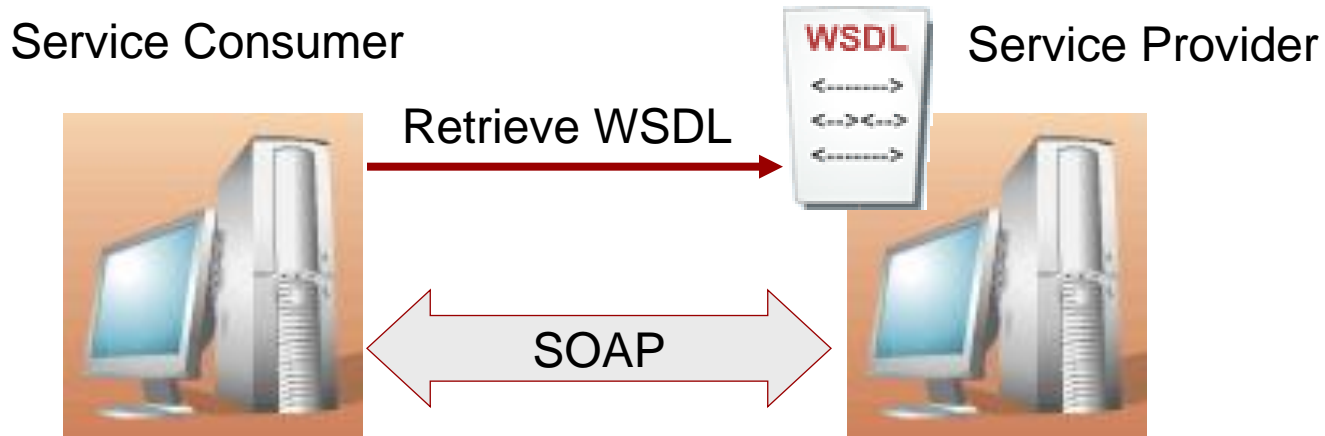
```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Header>
    <auth:authentication xmlns:auth="" soap:mustUnderstand="1">
      ...
    </auth:authentication >
  </soap:Header>

  <soap:Body xmlns:s="http://www.example.org/stock">
    <s:GetStockPrice>
      ...
    </s:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Service Description – WSDL

Introducing WSDL

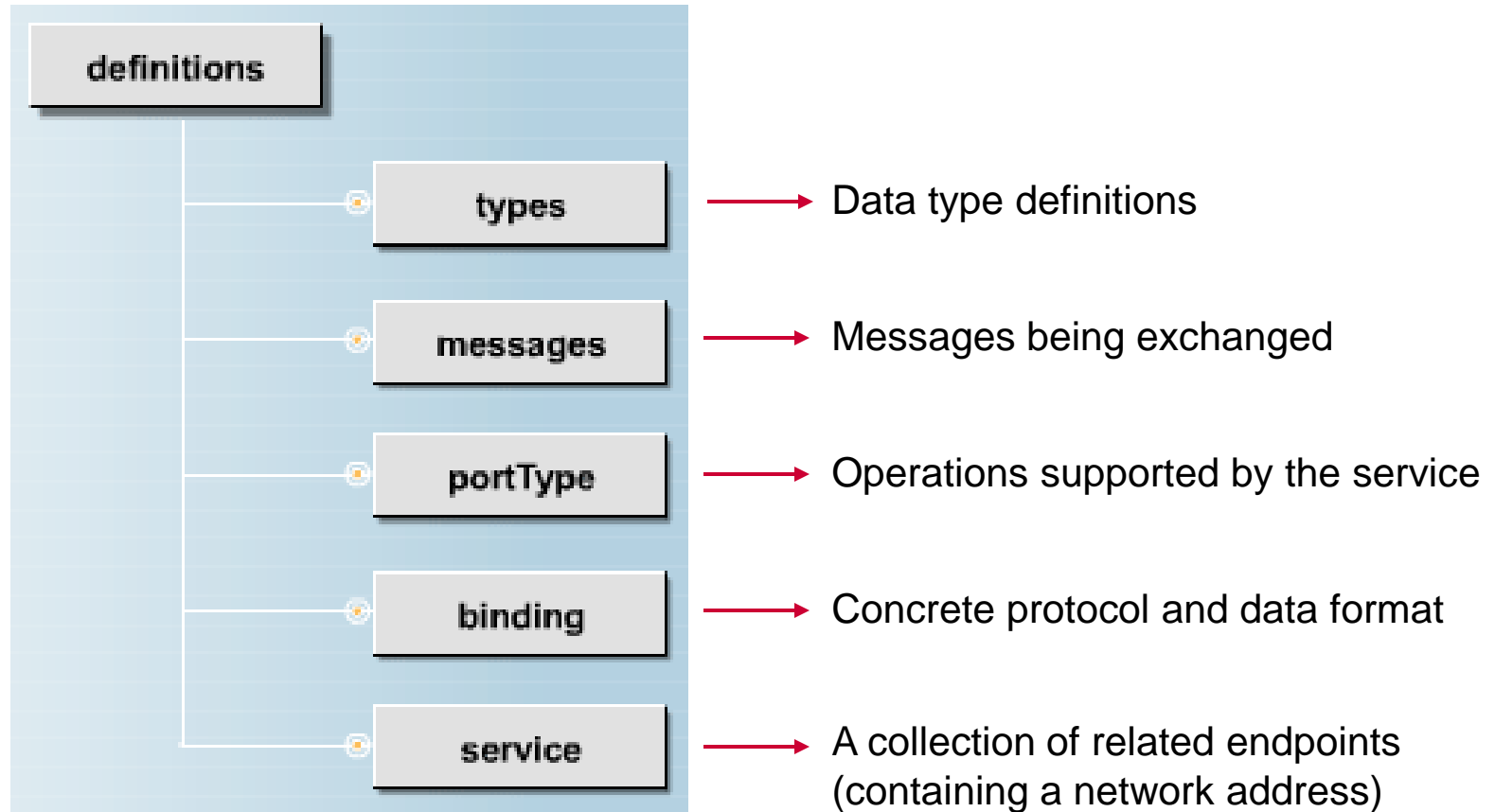


- **Web Services Description Language**
- XML-based language for describing web services

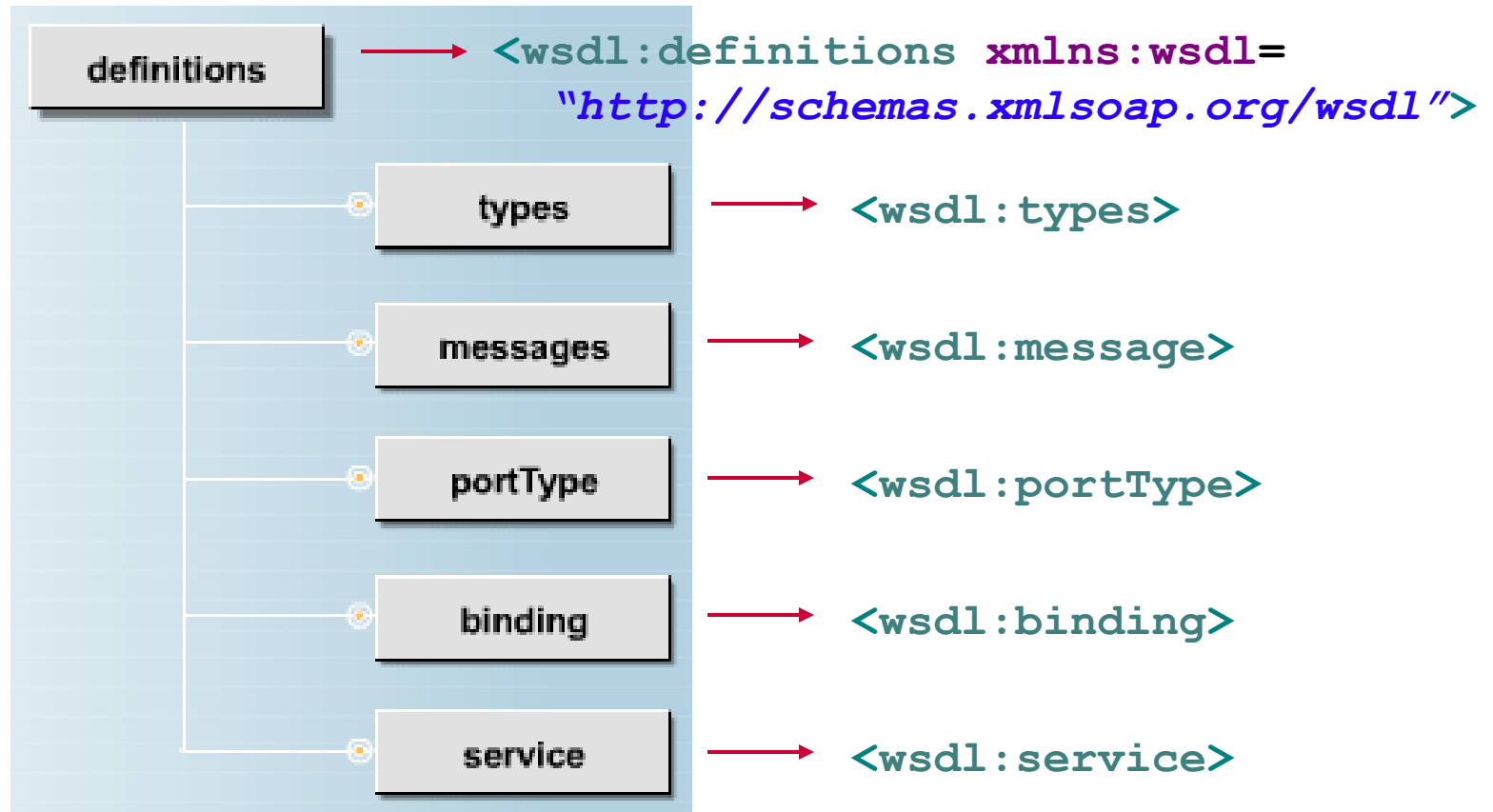
Why WSDL?

- WSDL enables clients to easily consume a service with little or no manual code.
- WSDL takes part in decoupling service consumers from service providers.

WSDL Structure



WSDL Syntax



Describing a Service Interface



Describing a Service Binding

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
  <soap:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="rpc" />
  <operation name="sayHello">
    <soap:operation soapAction="sayHello" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

Transport protocol

Data format (encoding)

Specifying Location of a Service

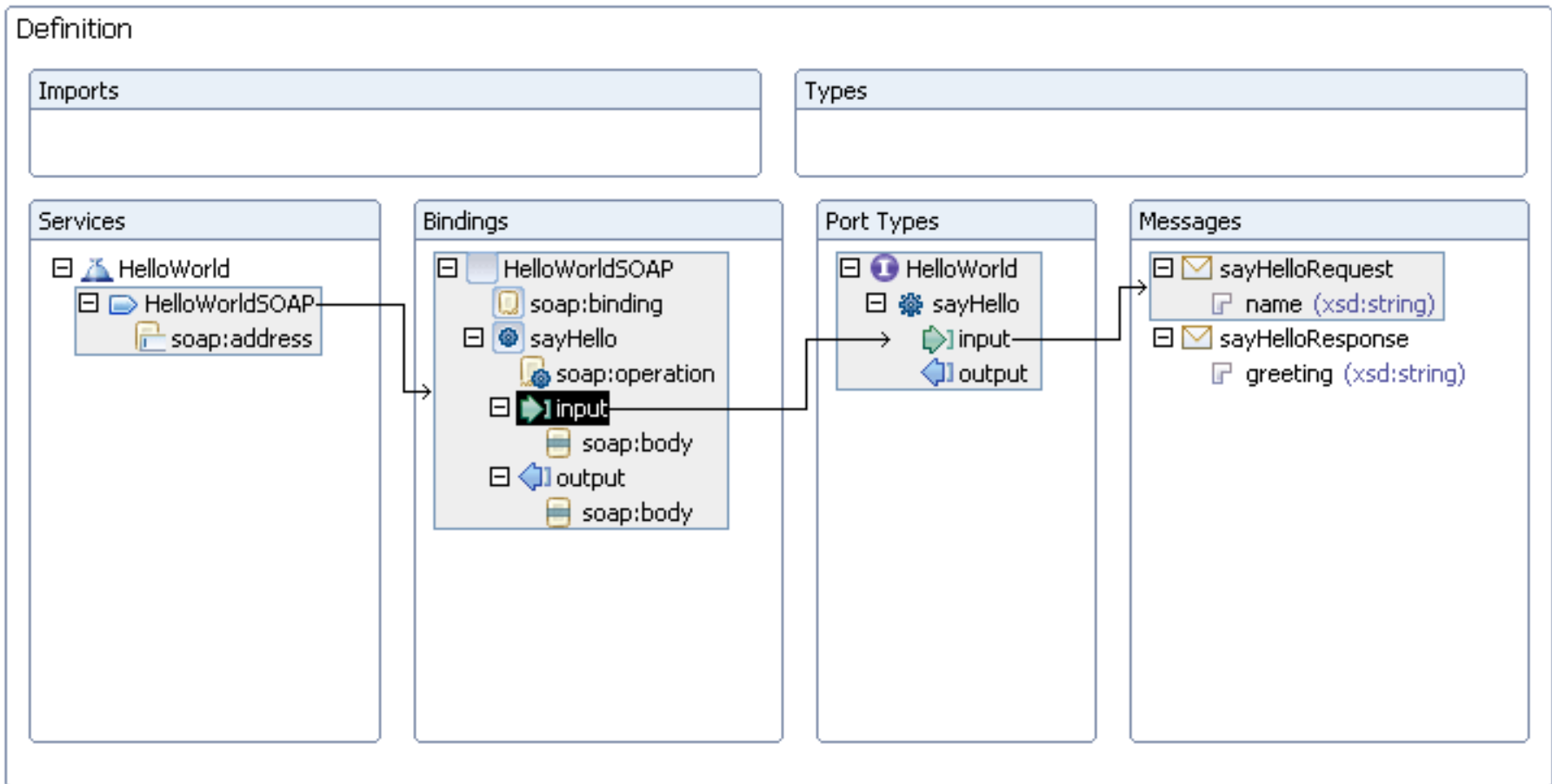
```
<service name="Hello_Service">
  <documentation>WSDL File for Hello
    Service</documentation>

  <port binding="tns:Hello_Binding" name="Hello_Port">
    <soap:address location=
      "http://localhost:8080/services/greeting" />
  </port>
</service>
```

Service endpoint
URL



WSDL Graph

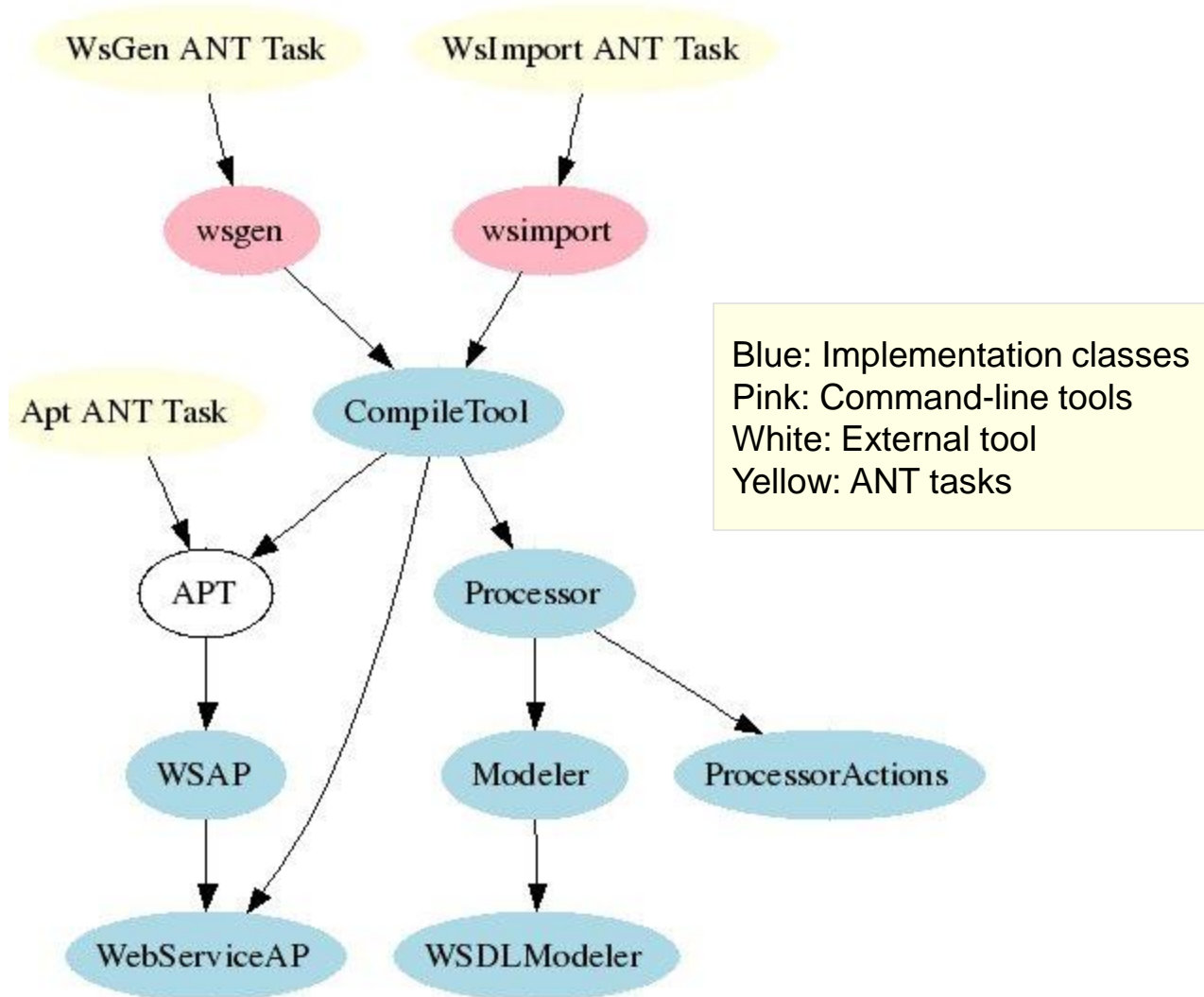


Web Services in Action

Introduction to JAX-WS

- **Java API for XML Web Services**
- Core of Metro project, developed by Sun Microsystems
- Embrace plain old Java object (POJO) concepts
- Descriptor-free programming
- Part of Java SE 6 and Java EE 5 platforms

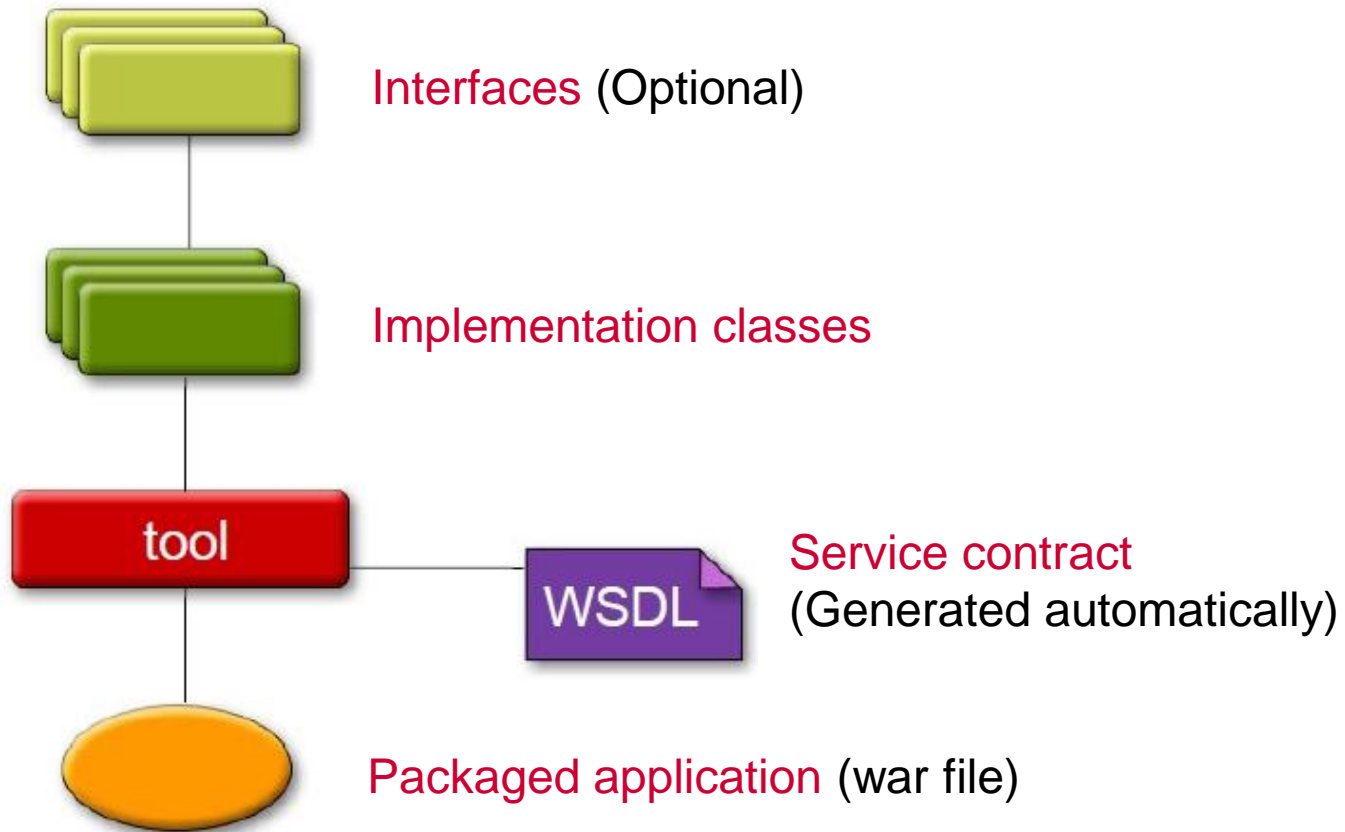
JAX-WS Tools



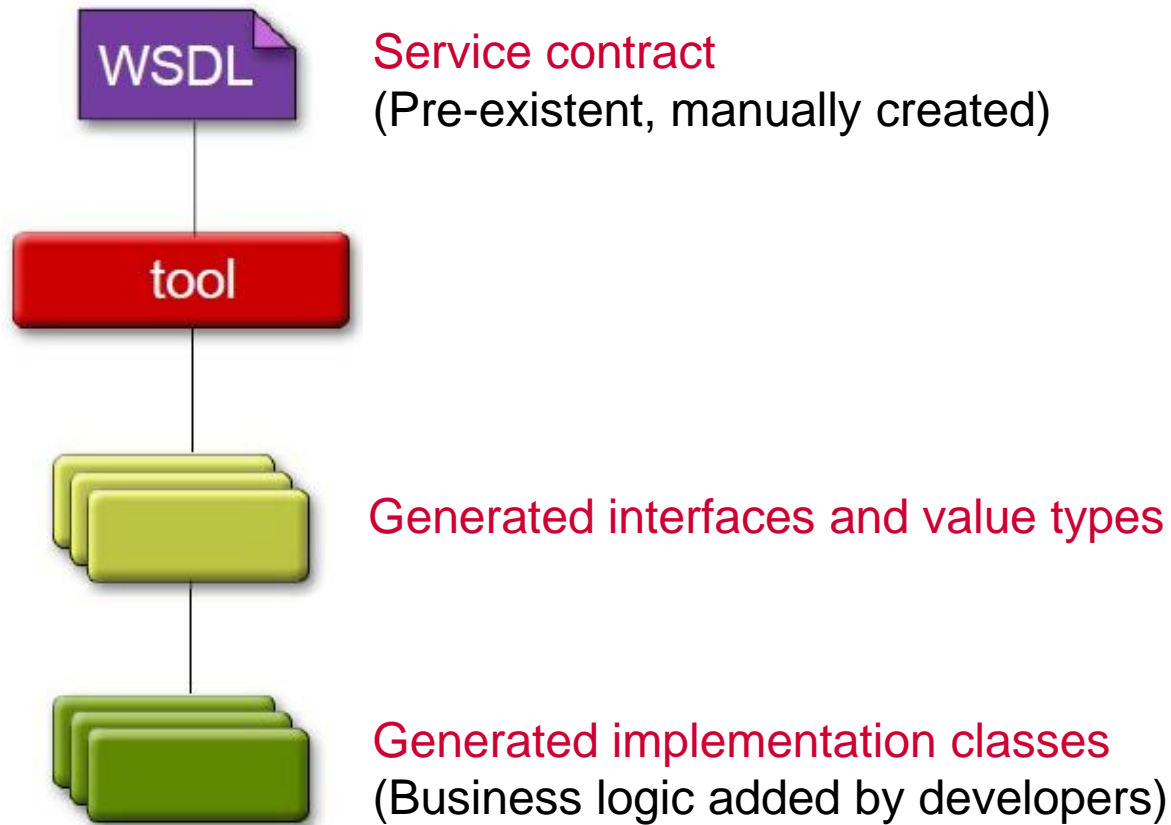
Developing a Web Service

- There are 2 approaches to developing a web service:
 - Starting from JAVA
 - Starting from WSDL

Starting from JAVA (bottom-up approach)



Starting from WSDL (top-down approach)



Service Implementation Class

```
package service.simple;

@WebService ← Indicates this POJO is a Web service
public class Greeting {
    public String sayHello(String arg) {
        return "Hello " + arg;
    }
}
```

Service Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>

<endpoints xmlns=
  "http://java.sun.com/xml/ns/jax-ws/ri/runtime">

  <endpoint
    name="Greeting"
    implementation="service.simple.Greeting"
    url-pattern="/services/greeting" />

</endpoints>
```

Generating Stubs from WSDL

- Command line

```
wsimport -d <directory> -p <package> <wsdl>
```

- Ant task

```
<wsimport destdir="directory"  
package="target package" wsdl="wsdl file"/>
```

Invoking a Service Using Stubs

```
package client.simple;

public class GreetingClient {
    public static void main(String[] args) throws
        Exception {
        GreetingService service = new GreetingService();
        Greeting proxy = service.getGreetingPort();
        String result = proxy.sayHello("WORLD");
    }
}
```



Practice

Required Software

- Java JDK
 - jdk 1.5+
- Container
 - JAX-WS can run on any servlet container.
 - Tomcat 5.5 is recommended.
- JAX-WS binary
 - <https://jax-ws.dev.java.net/2.1.7>

Exercise 1: Code First (Starting from Java)

- Code an implementation class.
- Compile the implementation class.
- Generate artifacts required to deploy the service.
- Package the files into a WAR file.
- Deploy the WAR file.
 - Tie classes are generated by the server during deployment.
 - WSDL file is generated automatically by the server and available at: `http://host:port/<url-pattern>?wsdl`

Exercise 2: Contract First (Starting from WSDL)

- Create a WSDL file using an IDE (Eclipse is recommended)
- Generate classes from the WSDL file using `wsimport`.
- Add business logic to the generated implementation class.
- Perform the steps 2-4 in the exercise 1.

Exercise 3: Invoking a Service

- Code a client class.
- Use `wsimport` to generate and compile stub files.
- Compile the client class.
- Run the client.



Web Services vs. Other Distributed Technologies

Distributed Component Technologies

- RMI – Remote Method Invocation
- DCOM – Distributed Component Object Model
- CORBA – Common Object Request Broker Architecture

Web Services vs. Distributed Component Technologies

Web Services

- Message exchange model
- Dynamic binding
- Loose coupling
- URL
- Text format
- Stateless

Distributed Component Technologies

- Object model
- Static binding
- Tight coupling
- Object reference
- Binary format
- Stateful

When to Use Which Technologies?

- CORBA is better suited for intra-domain communication
 - All distributed entities share a common object model
 - The deployment is more or less permanent
- Web services deal mostly with coarse-grained integration
 - Object architecture and modeling are hidden from the Web services.
 - Loose coupling is preferable, whereby domains can independently design and deploy their applications

Conclusion

- Web services and other distributed technologies are not exclusive but rather should be seen as complementary technologies.
- The choice of one over another is a matter of choosing “the right tool for the right job”.



One size do not fit all !

References

- W3C Specifications
 - <http://www.w3.org/TR/soap>
 - <http://www.w3.org/TR/wsdl>
- OASIS Specifications
 - <http://docs.oasis-open.org/soa-rm/v1.0>
 - <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf>
- SOA Principles
 - <http://www.soapprinciples.com>
- Java Passion
 - <http://www.javapassion.com/webservices/index.html>
- Java Web Services Tutorial
 - <http://java.sun.com/webservices/docs/2.0/tutorial/doc>

Points to Remember



Q&A



Thank You.

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
2 Feb 2010	1.0	Launched	Phuong H Nguyen	Luat Nguyen, Phong Le, Nguyet Pham
27 Apr 2010	1.1	Added Revision History	Nguyet Pham	