University of Tokyo Java Class
September 22-26, 2003
Intro to Web Services

Marc Hamilton
Director of Technology
Global Education and Research
Sun Microsystems, Inc

Primary Goal of the Presentation

Learn Web services architecture over J2EEô platform and how to create and deploy Web services using core Javaô APIs for Web services: JAX-RPC (focus of this session), JAXR, JAXM.

Agenda for Technical Session

- Web services architecture over J2EE™ platform
- JAX-RPC (Focus of this session)
- JAXM, JAXR
- Steps for building and deploying a Web service
- Steps for building a Web service Client
- Web services tools for J2EE™ platform

Web Services Architecture Over J2EE™ Platform

What Is a Web Service?

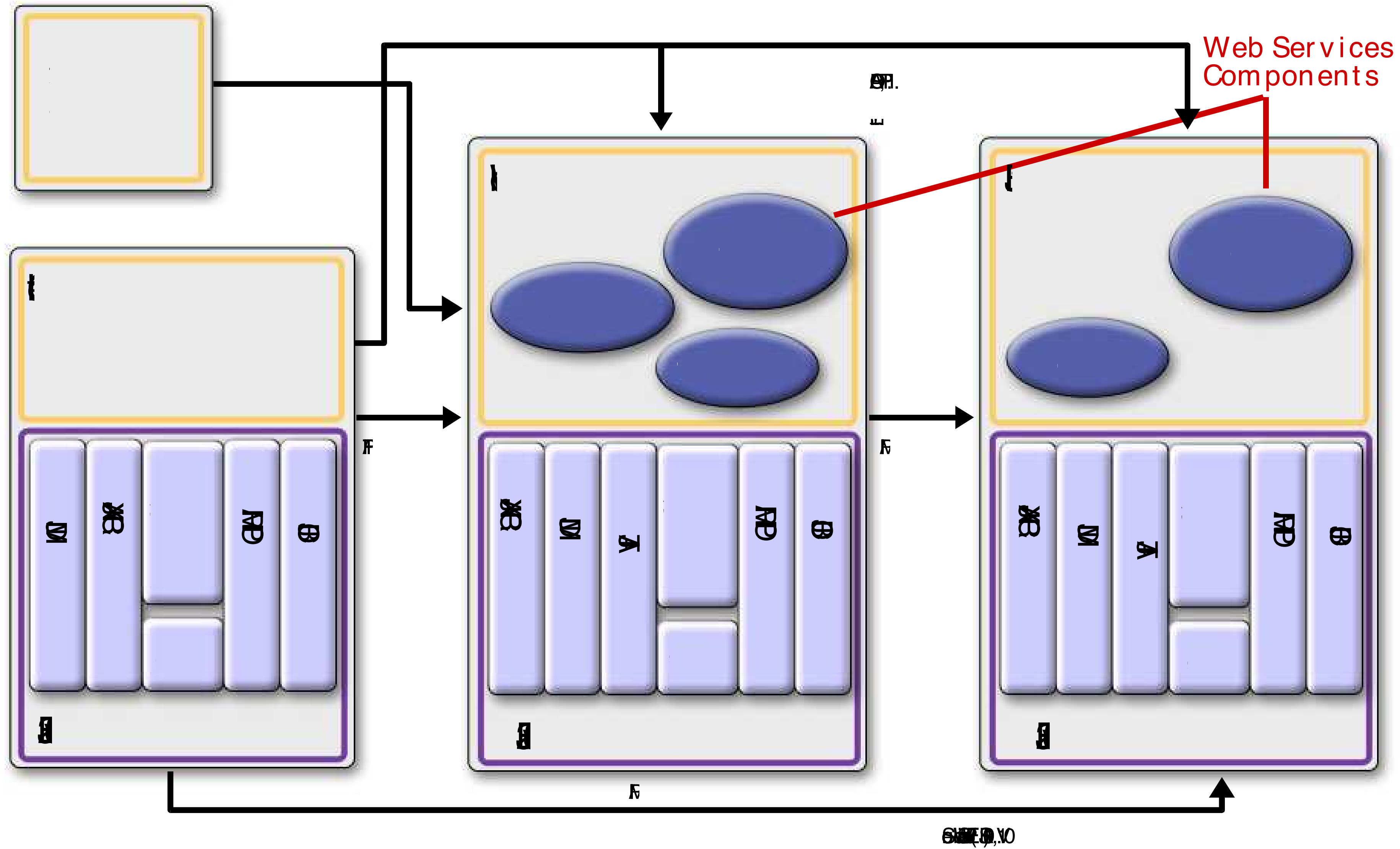
Sun microsystems

- A set of endpoints (ports) operating on messages
- Ports are operating within a container
  - Container provides runtime environment
  - Contract for runtime environment are specified in JAX-RPC, EJB™ 2.1 specification, JSR 109
- Service is described abstractly in WSDL document and published to a registry
  - WSDL specifies a contract between service provider and client

Web Service Component and Container

- Container vs. Component model
  - Web services components get executed within a container
  - Components are portable (under J2EE™ 1.4 container)
- Web service components
  - Web-tier (Servlet-based endpoint)
  - EJB™-tier (Stateless session bean-based endpoint)

Web Service Components



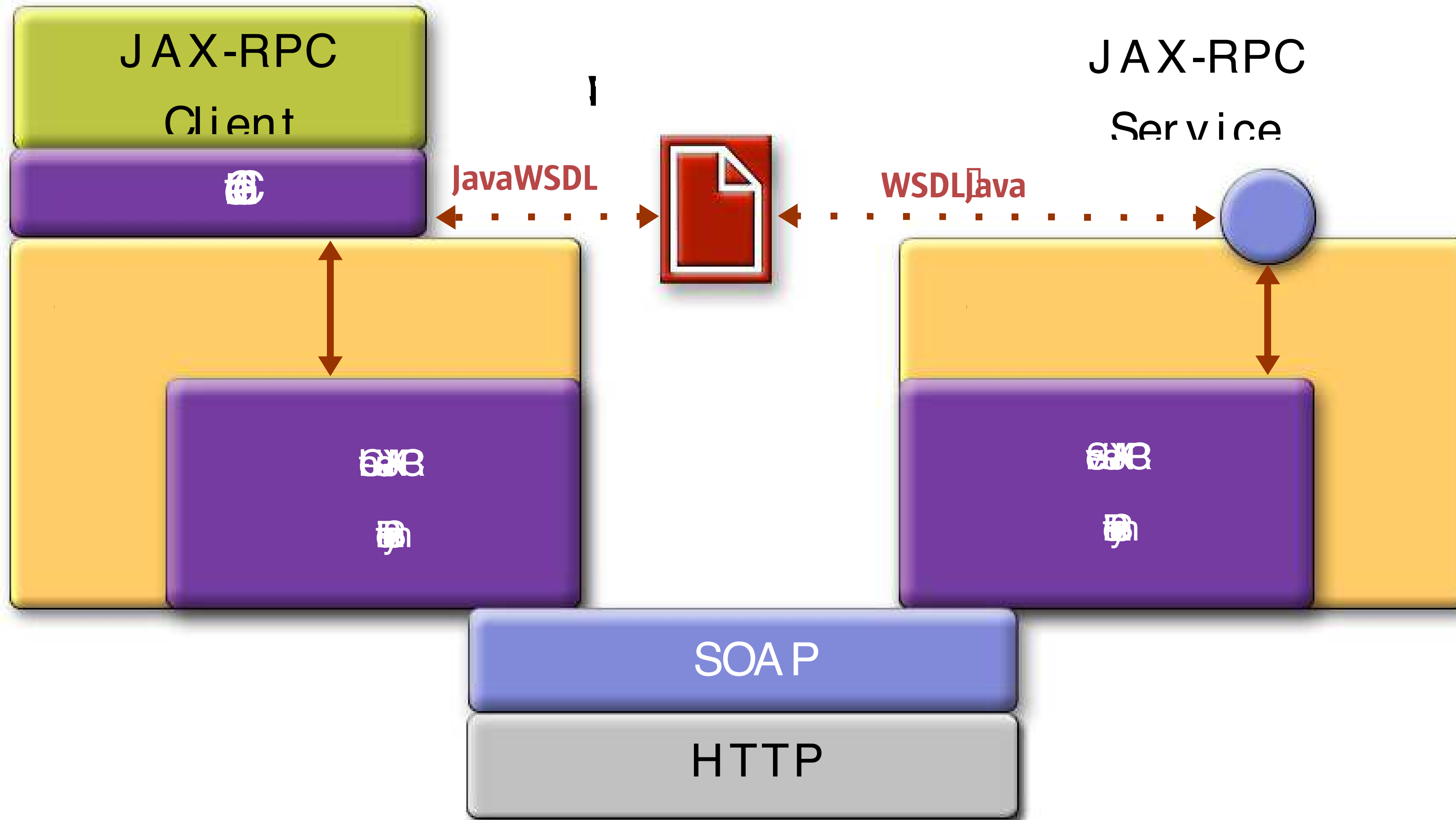
JAX-RPC

JAX-RPC

Sun microsystems

- Servlet-based Web service endpoint model
- WSDL to/from Java™ mapping specification
- XML data types to/from Java™ language types mapping (serialization) specification
- Extensible type mapping
- SOAP Message Handler framework
- Packaging
- Client Programming Models

JAX-RPC Architecture Diagram



Relationship to WSDL

JAX-RPC describes a Web Service as a collection of remote interfaces and methods

WSDL describes a Web Service as a collection of ports and operations

SOAP Message Handlers

Sun microsystems

Handlers let you access/modify SOAP request and response messages

- Typically used to process service contexts in SOAP header blocks
- Can be used to extend functionality of Web services runtime system
  - J2EE™ containers (which provide Web services runtime) are likely to use them internally to provide session/transaction propagation

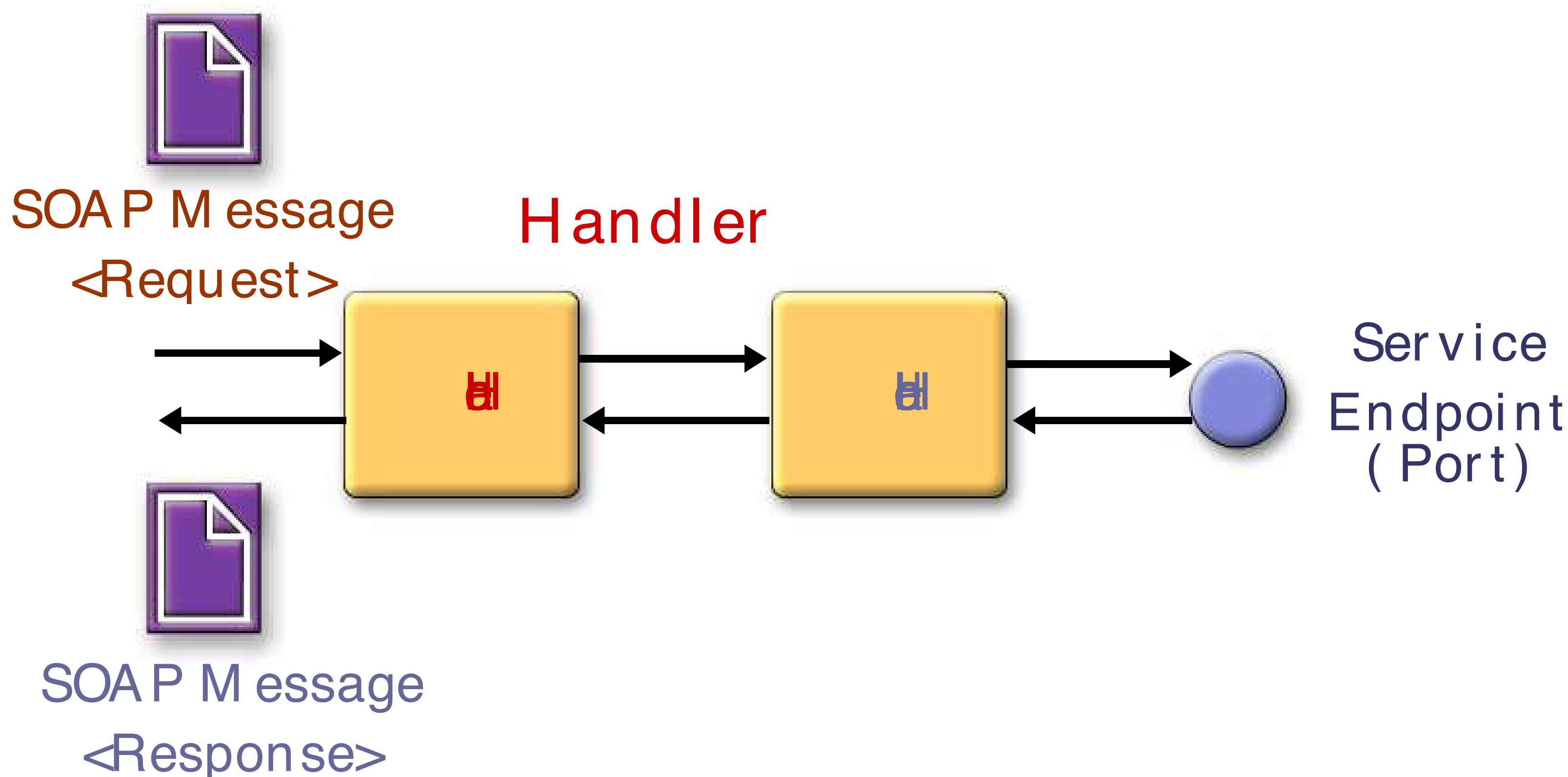
Example handlers:

- Encryption, decryption, authentication, authorization, logging, auditing, caching

SOAP Message Handlers

- Pluggable and chainable
  - Through standardized programming API
  - Portable across implementations
- Has its own lifecycle
  - JAX-RPC runtime system calls init(), destroy() of a handler
- Handler instances can be pooled
- MessageContext is used to share properties among handlers in a handler chain

SOAP Message Handlers



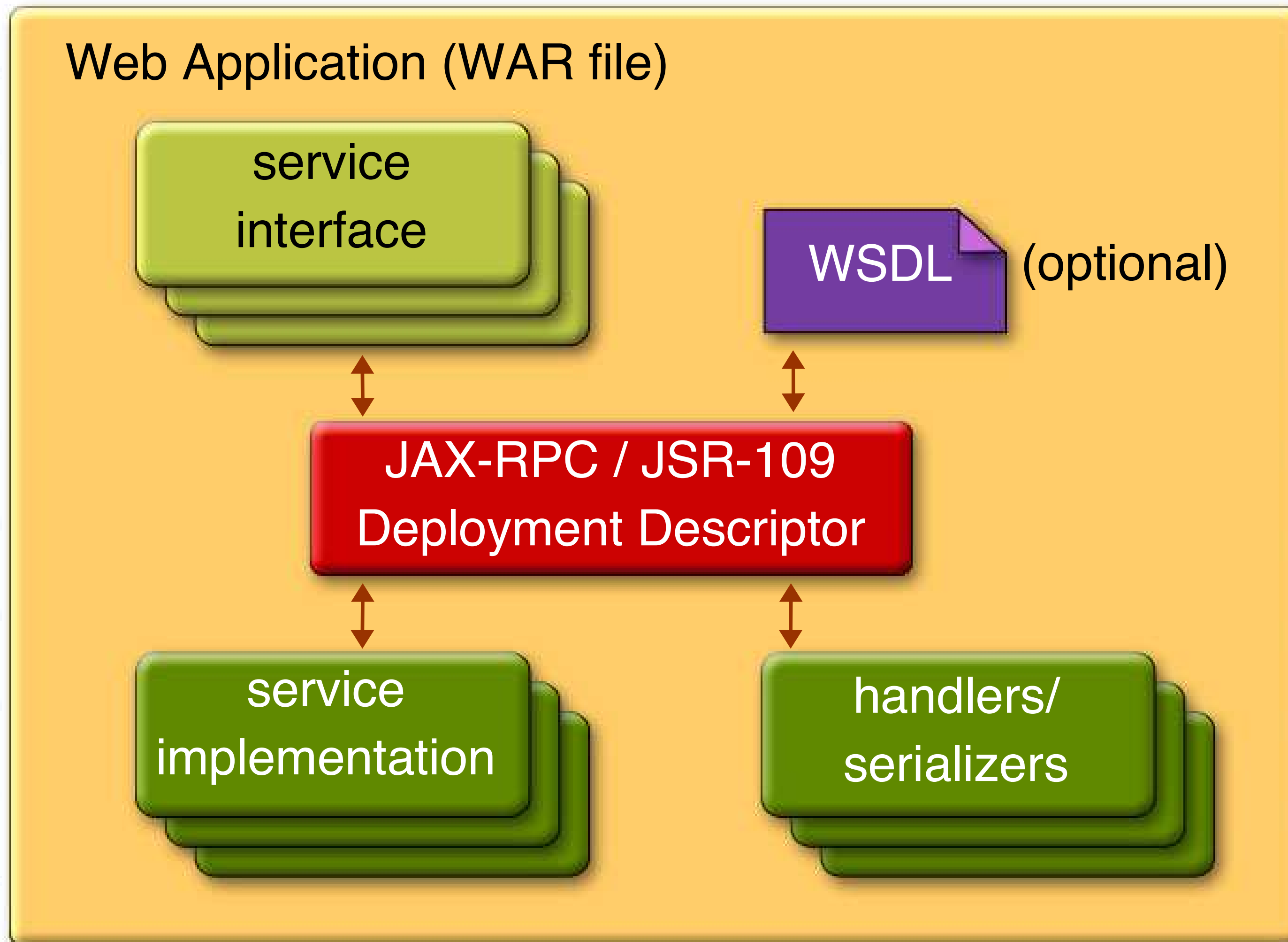
Example SOAP Message Handler

```
package com.example;
public class MySOAPMessageHandler implements javax.xml.rpc.handler.Handler {
      public MySOAPMessageHandler() { ... }
      public boolean handleRequest(MessageContext context, HandlerChain chain){
          try {
              SOAPMessageContext smc = (SOAPMessageContext)context;
              SOAPMessage msg = smc.getMessage();
              SOAPPart sp = msg.getSOAPPart();
              SOAPEnvelope se = sp.getEnvelope();
              SOAPHeader sh = se.getHeader();
              // Process one or more header blocks
              // ...
              // Next step based on the processing model for this handler
          }
          catch(Exception ex) {
              // throw exception
          }
      }
      // Other methods: handleResponse(), handleFault(), init(), destroy()
}
```

Session Management

- JAX-RPC runtime system manages session
  - Service client or service developer do not have to deal with session management
- Supported Session management schemes over HTTP
  - Cookie-based
  - URL rewriting
- SOAP Header-based session management scheme in the future

Packaging of JAX-RPC API-Based Applications



Client Programming Models

- Stub-based (least dynamic)
  - Both interface (WSDL) and implementaion (stub) created at compile time
- Dynamic proxy
  - Interface (WSDL) created at compile time
  - Implementation (dynamic proxy) created at runtime
- Dynamic invocation interface (DII)
  - Both interface (WSDL) and implementation created at runtime

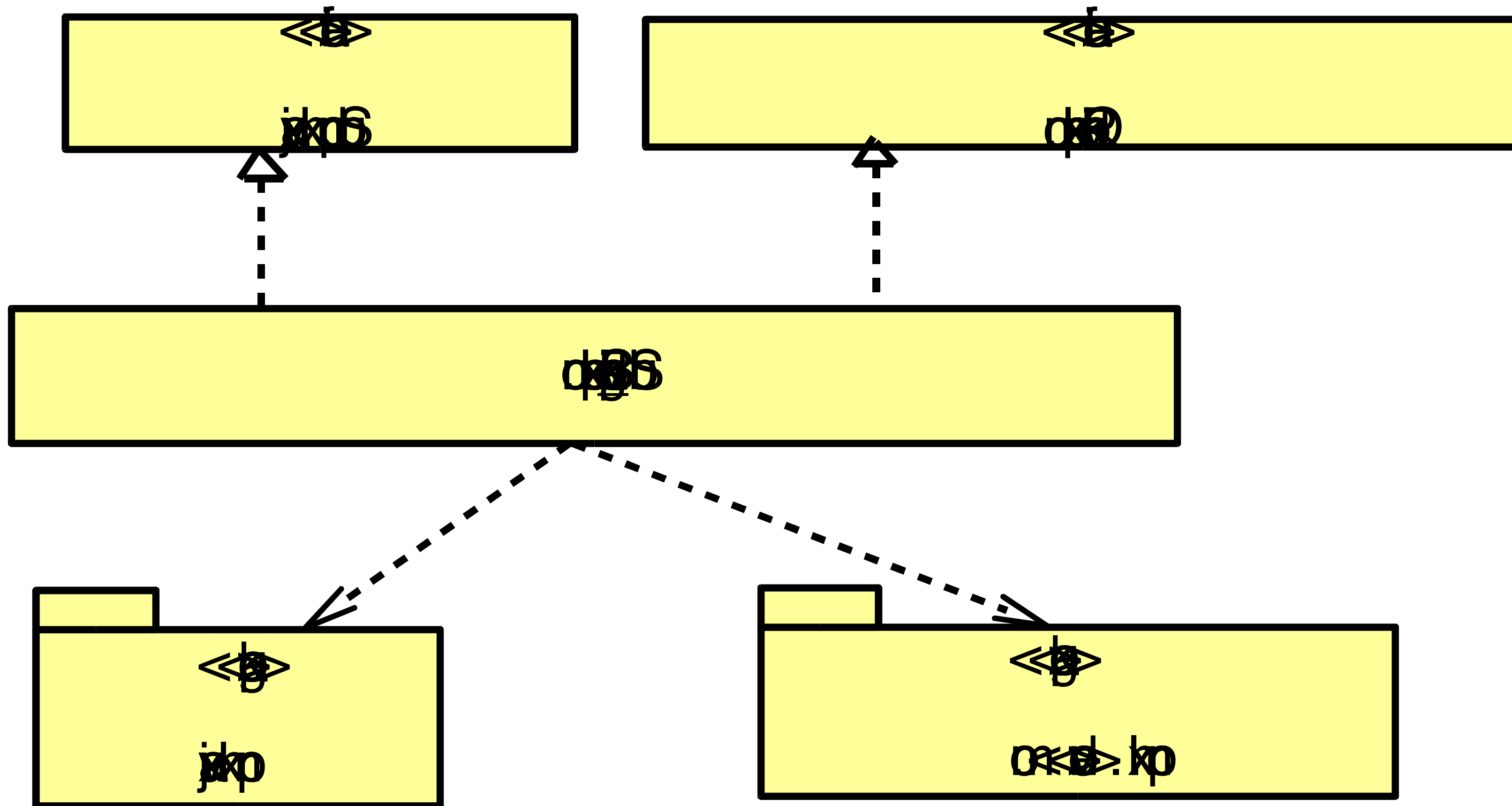
Stub-based Invocation Model

Sun microsystems

Stub class gets generated from WSDL at compile time

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
    - [illegible]
    - [illegible]

Stub Class Hierarchy



Dynamic Proxy-based Invocation Model

- Dynamic proxy is generated on the fly by JAX-RPC client runtime
- Application provides the Web service definition interface the dynamic proxy conforms to during runtime

Example: Dynamic Proxy Client

Sun microsystems

```
package proxy;
 import java.net.URL;
 import javax.xml.rpc.Service;
 import javax.xml.rpc.JAXRPCException;
 import javax.xml.namespace.QName;
 import javax.xml.rpc.ServiceFactory;
 public class HelloClient {

     public static void main(String[] args) {
         try {
             String UrlString = "http://localhost:8080/ProxyHelloWorld.wsdl";
             String nameSpaceUri = "http://proxy.org/wsdl";
             String serviceName = "HelloWorld";
             String portName = "HelloIFPort";

             URL helloWsdlUrl = new URL(UrlString);

             ServiceFactory serviceFactory = ServiceFactory.newInstance();

             Service helloService =
                 serviceFactory.createService(helloWsdlUrl,  new QName(nameSpaceUri, serviceName));

             HelloIF myProxy = (HelloIF) helloService.getPort(new QName(nameSpaceUri, portName),  proxy.HelloIF.class);

             System.out.println(myProxy.sayHello("Buzz"));
         } catch (Exception ex) {
             ex.printStackTrace();
         }
     }
 }
```

DII Invocation Model

- Gives complete control to client programmer
- Most dynamic
- Enables broker model
  - Client finds (through some search criteria) and invokes a service during runtime through a broker
  - Used when service definition interface is not known until runtime
  - You set operation and parameters during runtime
- Has to create Call object first

Example: DII Client

```
package dynamic;

import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;

public class HelloClient {

    private static String endpoint =
        "http://localhost:8080/dynamic-jaxrpc/dynamic";
    private static String qnameService = "Hello";
    private static String qnamePort = "HelloIF";

    private static String BODY_NAMESPACE_VALUE =
        "http://dynamic.org/wsdl";
    private static String ENCODING_STYLE_PROPERTY =
        "javax.xml.rpc.encodingstyle.namespace.uri";
    private static String NS_XSD =
        "http://www.w3.org/2001/XMLSchema";
    private static String URI_ENCODING =
        "http://schemas.xmlsoap.org/soap/encoding/";
```

Example: DII Client

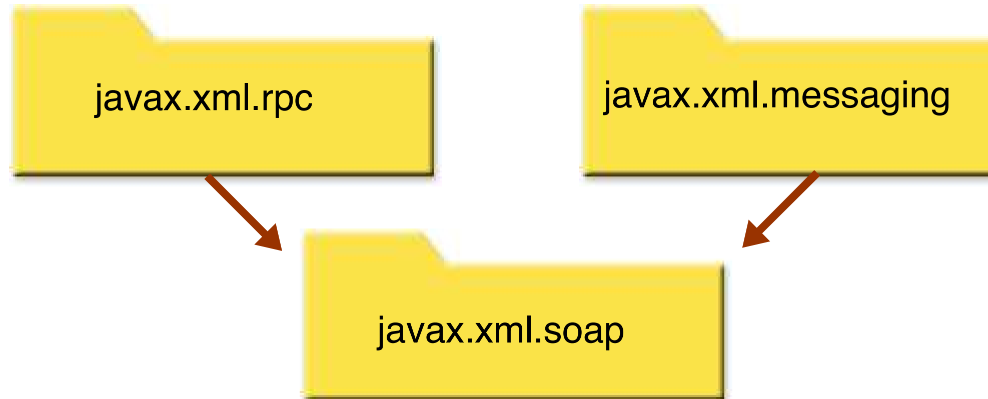
```
public static void main(String[] args) {
            try {
                ServiceFactory factory = ServiceFactory.newInstance();
                Service service =  factory.createService(new QName(qnameService));
                QName port = new QName(qnamePort);

                Call call = service.createCall(port);
                call.setTargetEndpointAddress(endpoint);

                call.setProperty(Call.SOAPACTION_USE_PROPERTY, new Boolean(true));
                call.setProperty(Call.SOAPACTION_URI_PROPERTY,"");
                call.setProperty(ENCODING_STYLE_PROPERTY, URI_ENCODING);
                QName QNAME_TYPE_STRING = new QName(NS_XSD, "string");
                call.setReturnType(QNAME_TYPE_STRING);
                call.setOperationName(new QName(BODY_NAMESPACE_VALUE "sayHello"));
                call.addParameter("String_1", QNAME_TYPE_STRING, ParameterMode.IN);
                String[] params = { "Duke!" };

                String result = (String)call.invoke(params);
                 System.out.println(result);

            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
```

JAXM

What Is JAXM?

- Javaô technology support for sending and receiving SOAP messages
  - Based on the SOAP 1.1 and the SOAP with Attachment specifications
- Supports higher-level semantics built on top of SOAP through a profile
  - ebXML Message Service profile

JAX-RPC Relationship With JAXM



JAXM Architectural Roles

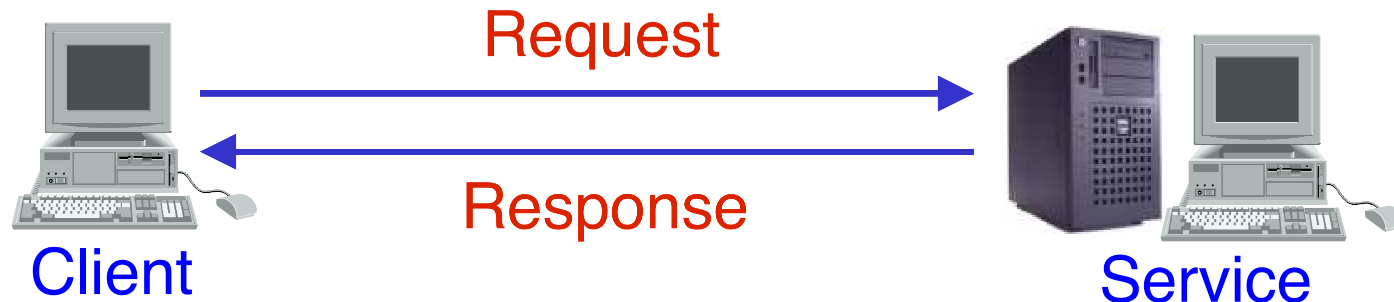
- JAXM Messaging Provider (JAXM Provider)
- JAXM Client (JAXM Application)
  - JAXM client that does not use JAXM provider (Standalone JAXM client)
  - JAXM client that uses JAXM provider

JAXM Messaging Provider

Sun microsystems

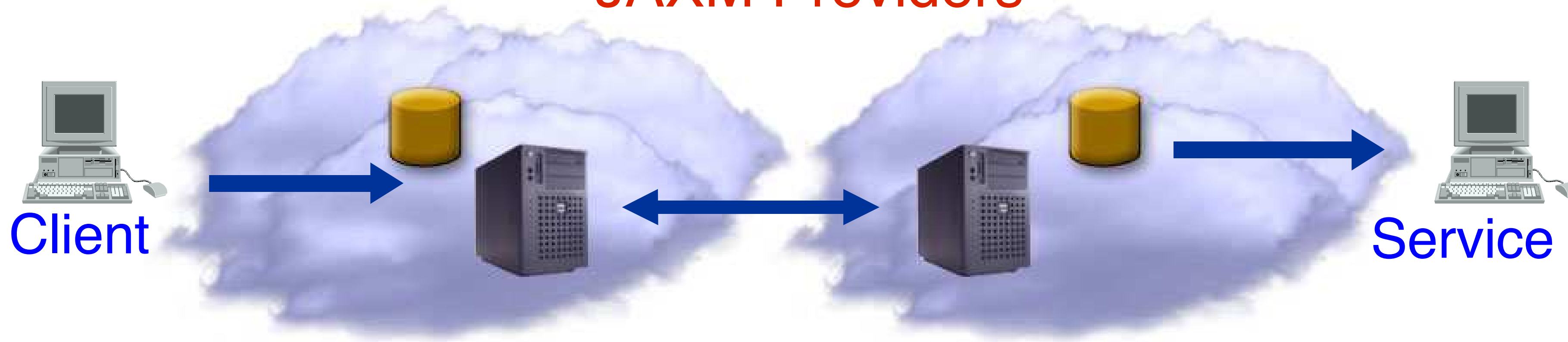
- Works behind the scene on behalf of the JAXM client
- Offers message routing and reliable messaging as all messages go through it
  - Assigning message identifiers and keeping track of messages
  - Persistently storing messages

JAXM Client That Does Not Use JAXM Provider (Standalone Client)



- Just a standalone J2SE™ application
- Point-to-point operation
  - Establishes a connection directly with the service (using a URL)
- Synchronous only
  - Request/response interaction
- In “CoffeeBreak” lab exercise, only standalone client is used

JAXM Client That Uses JAXM Provider



JAXR

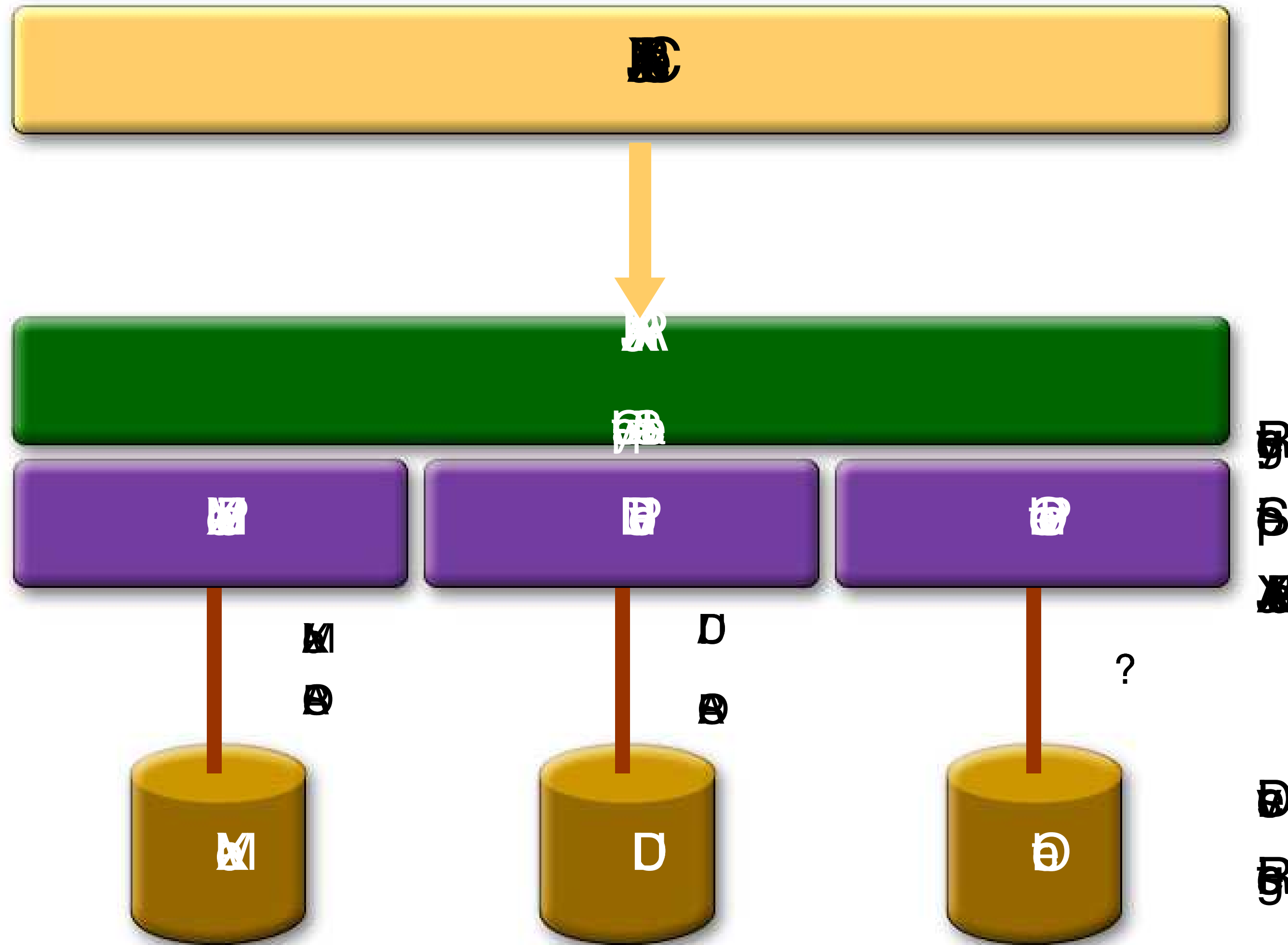
What Is JAXR?

Standard Java™ API for performing registry operations over diverse set of registry providers

- Service registration and discovery

A unified information model for describing business registry content

JAXR Architecture



Web Service Development Steps

Steps for Development and Deployment of Web Services

I. Define a Web service
II. Implement the Web service
III. Produce deployment ready package
IV. Deploy package over J2EE™ platform
V. Publish the Web service and binding information to a service registry
VI. Serve service requests from client

Step 1: Defining a Web Service

- Web service is defined in:
  - WSDL or
  - Web service endpoint interface (Java interface)
- Top-down
  - WSDL is created (or found) first before its implementation
- Bottom-up ("CoffeeBreak" lab exercise)
  - WSDL and Web service endpoint interface get generated from existing Javaô class

Web Service Endpoint Interface

Sun microsystems

- A Java™ interface type as specified in JAX-RPC
  - Extends java.rmi.Remote
- Describes remote Web services in an abstract fashion
- Could be generated from WSDL
  - In "CoffeeBreak" lab exercise, Sun™ ONE Studio generates Web services endpoint interface from WSDL

Example: "StockQuote" Web Service Endpoint Interface

```
public interface StockQuoteProvider
        extends java.rmi.Remote {

  public float getLastTradePrice(String tickerSymbol)
            throws java.rmi.RemoteException;
     ...
 }
}
```

Example: “SupplierService” Web Service Endpoint Interface

```
// This interface is generated by
// Java Studio from WSDL document, which have been in turn
// generated from existing Java class (Supplier.java).

public interface SupplierServiceRPC extends java.rmi.Remote {

   public beans.PriceListBean getPriceList()
         throws KomodoException,java.rmi.RemoteException;

   public beans.ConfirmationBean placeOrder(beans.OrderBean order)
         throws KomodoException,java.rmi.RemoteException;

}
```

Step 2: Implement Web Service

- Choose implementation form
  - Java™ class (for servlet-based endpoint)
  - Stateless session bean
- In "CoffeeBreak" lab exercise, we use bottom-up approach
  - Implementation class (Supplier.java) already exist
  - WSDL document and Web services end point interface are created from existing implementation class

Example: Implementation

```
public class StockQuoteProviderImpl
        implements StockQuoteProvider {

    public float getLastTradePrice(String tickerSymbol)
            throws java.rmi.RemoteException{
            // business logic for method
    }
}
```

Example: Implementation in "CoffeeBreak"

Sun microsystems

```
public class Supplier {

    // Business method to be exposed as a Web service
    public ConfirmationBean placeOrder(OrderBean order) {
        Date tomorrow = DateHelper.addDays(new Date(), 1);
        ConfirmationBean confirmation = new ConfirmationBean();
        confirmation.setOrderId(order.getId());
        confirmation.setShippingDate(tomorrow);
        return confirmation;
    }
    // Business method to be exposed as a Web service
    public PriceListBean getPriceList() {
      PriceListBean priceList = loadPrices();
      return priceList;
    }
    ...
}
```

Step 3: Create Deployable Package

- Ready-to-deployable package
  - WAR file (servlet-based)
  - EJB-JAR file (stateless session bean based)
- Standardization for portability
  - Package structure
  - Web Services Deployment descriptor

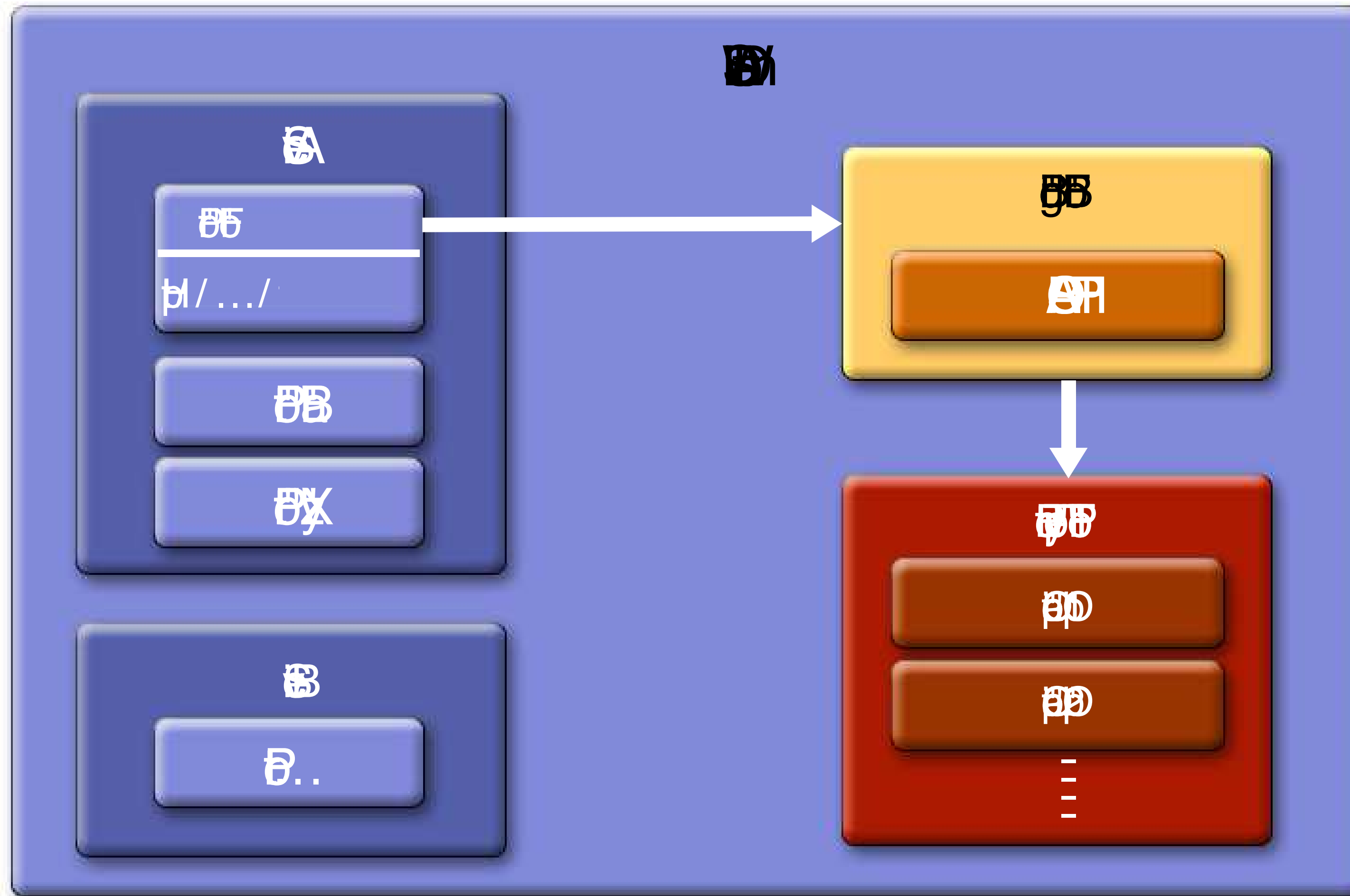
Step 4: Deploy Package

Sun microsystems

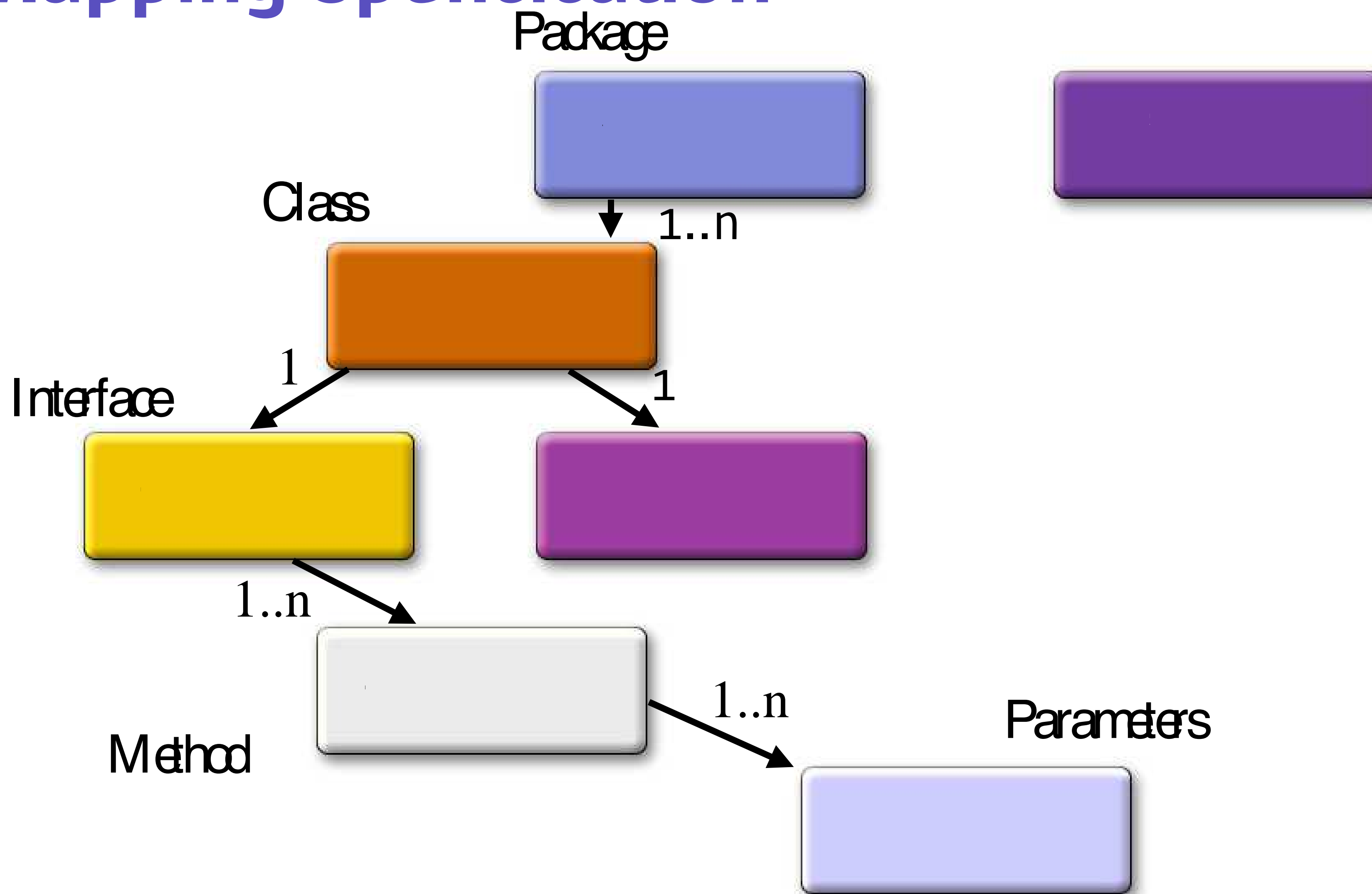
- Responsibility of Container (or deployment tool)
  - Validation of the package
  - Creation of runtime artifacts
  - Configuration of the server's SOAP request listeners for each port (binding to a port)
  - Generation of concrete WSDL document
  - Publication of Web services

Web Service Client Development Steps

WSDL View of a Web Service



WSDL Elements/Java™ Mapping Speficication



Web Services Client View

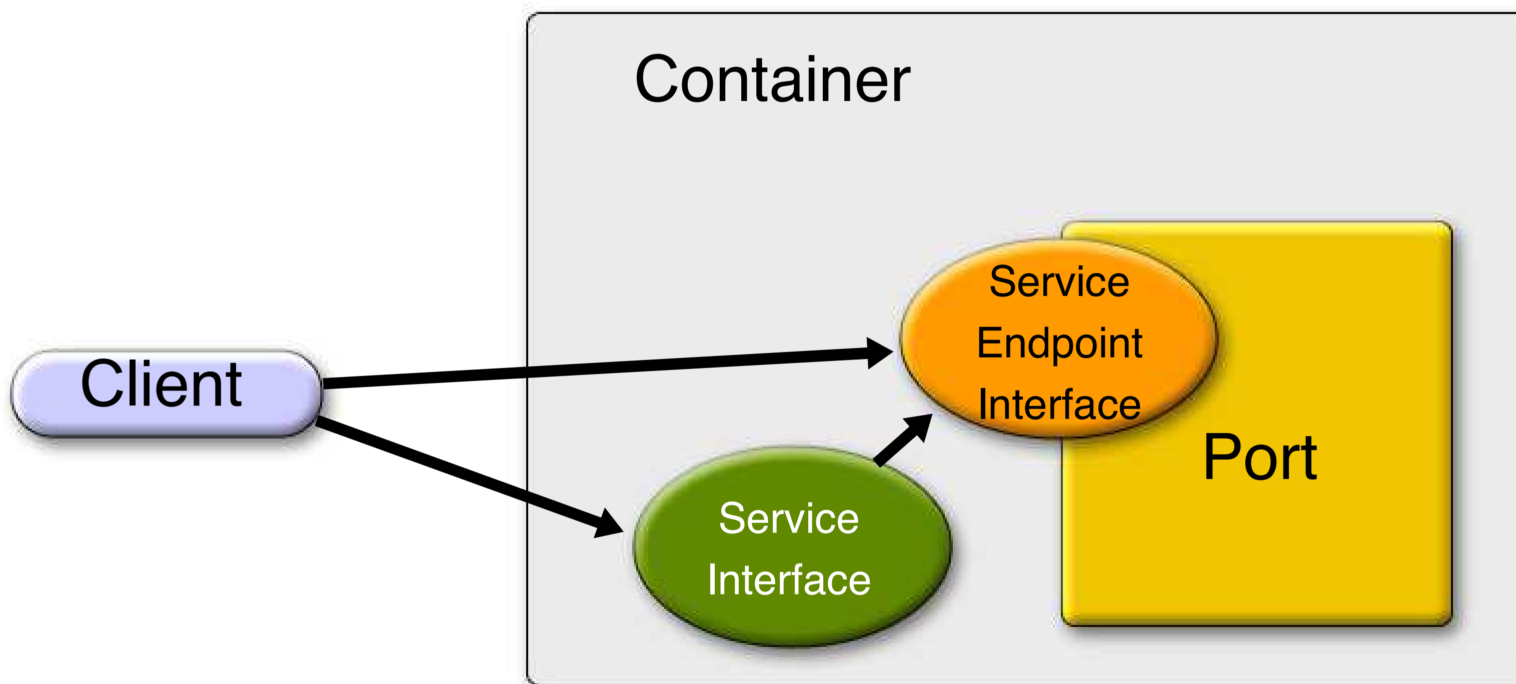
Abstract part of WSDL document (PortType) is represented by Service Endpoint Interface

- Container provides actual implementations of Service Endpoint Interface
  - Stub or dynamic proxy

Concrete part of WSDL document (Service, Port) is represented by Service Interface

- Container provides actual implementation of Service interface
  - Service object
- Service object is a factory class for stub or dynamic proxy

Web Services Client and Service



Development Steps for Web Service Client

- Discover WSDL description of a service
  - Identify endpoint address of the service
- Get client-side Web services code artifacts (i.e., Stub, dynamic proxy, DII)
  - Code artifacts are generated by container (or deployment tool)
- Send messages to service endpoints that provide service implementation (through stub or dynamic proxy)
- Receive back messages that contain results

Example: Web Service Client Using Stub

```
public class OrderCaller {
  // Local variable for service specific stub object
  private SupplierServiceRPC  supplier;

  public OrderCaller(String endpoint) {
    try {
      // Note: SupplierService_Impl is implementation-specific.
      // Get Stub object
      Stub stub = (Stub)
       (new SupplierService_Impl().getSupplierServiceRPCPort());
      // Set endpoint address through Stub interface
      stub._setProperty(ENDPOINT_ADDRESS_PROPERTY, endpoint);
      // Cast Stub object to service specific stub object
      supplier = (SupplierServiceRPC)stub;
    } catch (Exception ex) {ex.printStackTrace();}
  }
```

Example: Web Service Client Using Stub

```
  public ConfirmationBean placeOrder(OrderBean order) {
     ConfirmationBean result = null;
     try {
       // Invoke a method through stub object
       result = supplier.placeOrder(order);
     } catch (Exception ex) {
       System.out.println("Error in OrderCaller.placeOrder");
       ex.printStackTrace();
     }
     return result;
   }

} // class
```

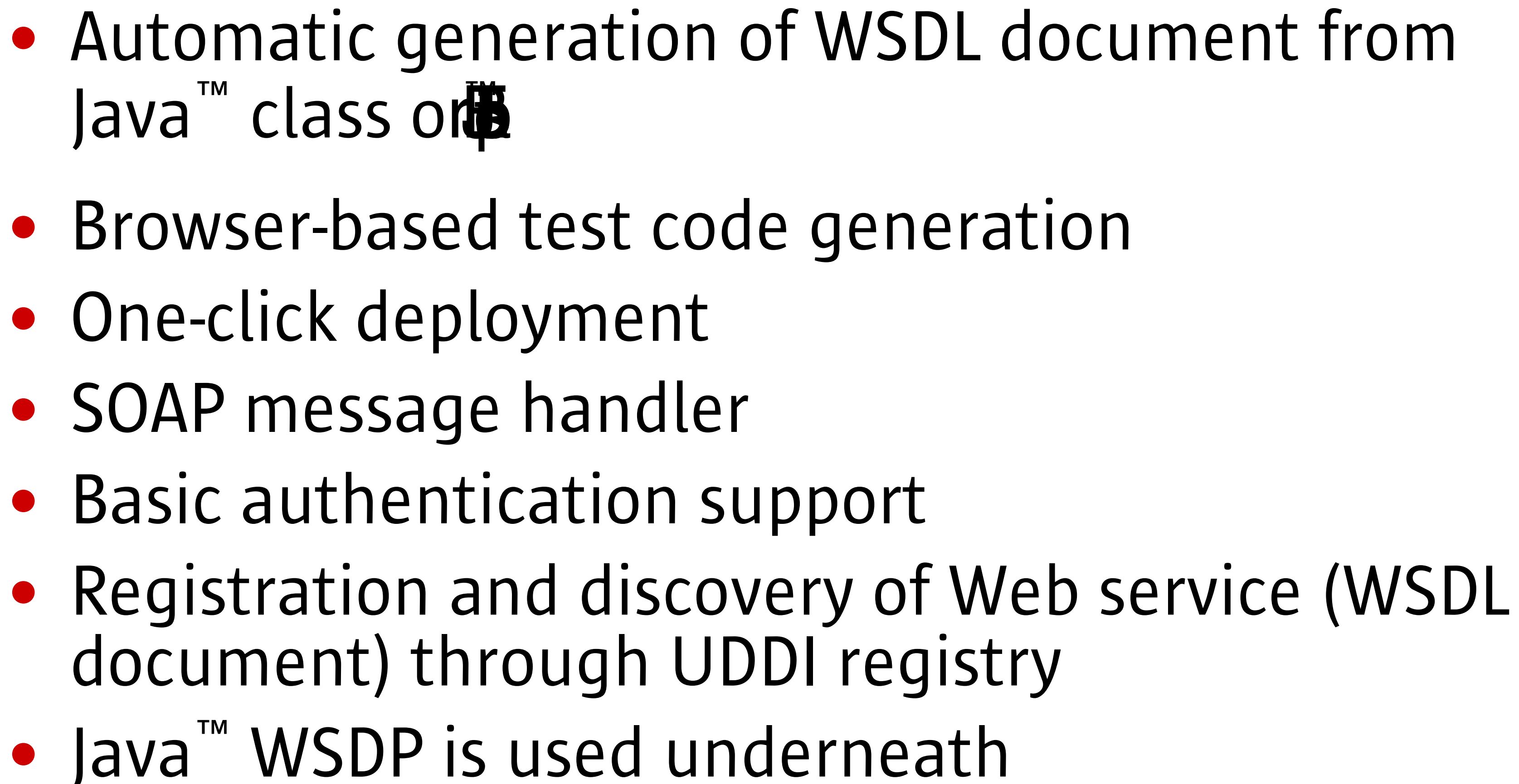
Web Service Tools for J2EE™ Applications

Java™ Web Services Developer Pack (Java WSDP)

Provides a convenient all-in-one package

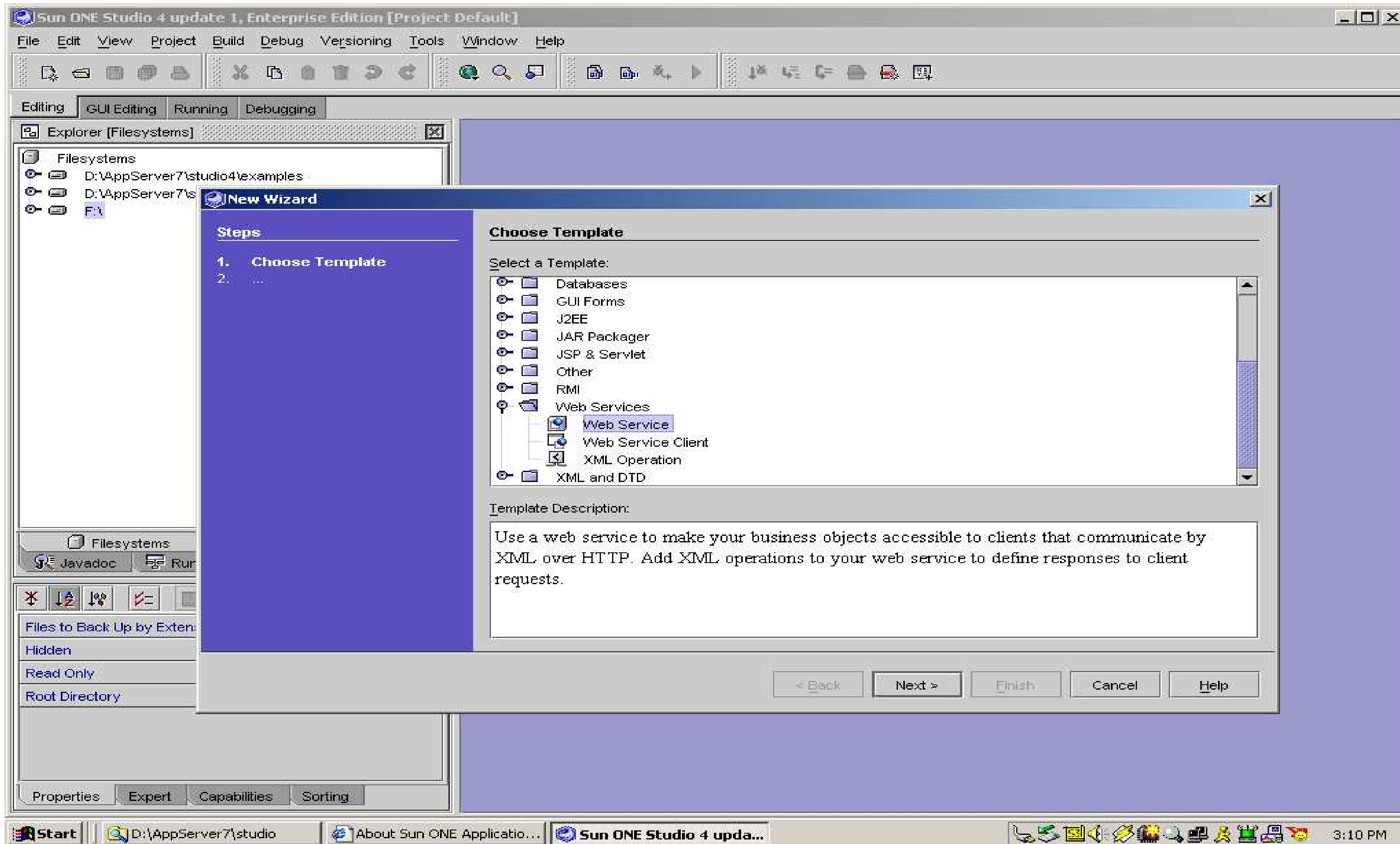
Java™ WSDP 1.2, June 2003,

Sun™ ONE Studio 5 SE (Web Services Support)

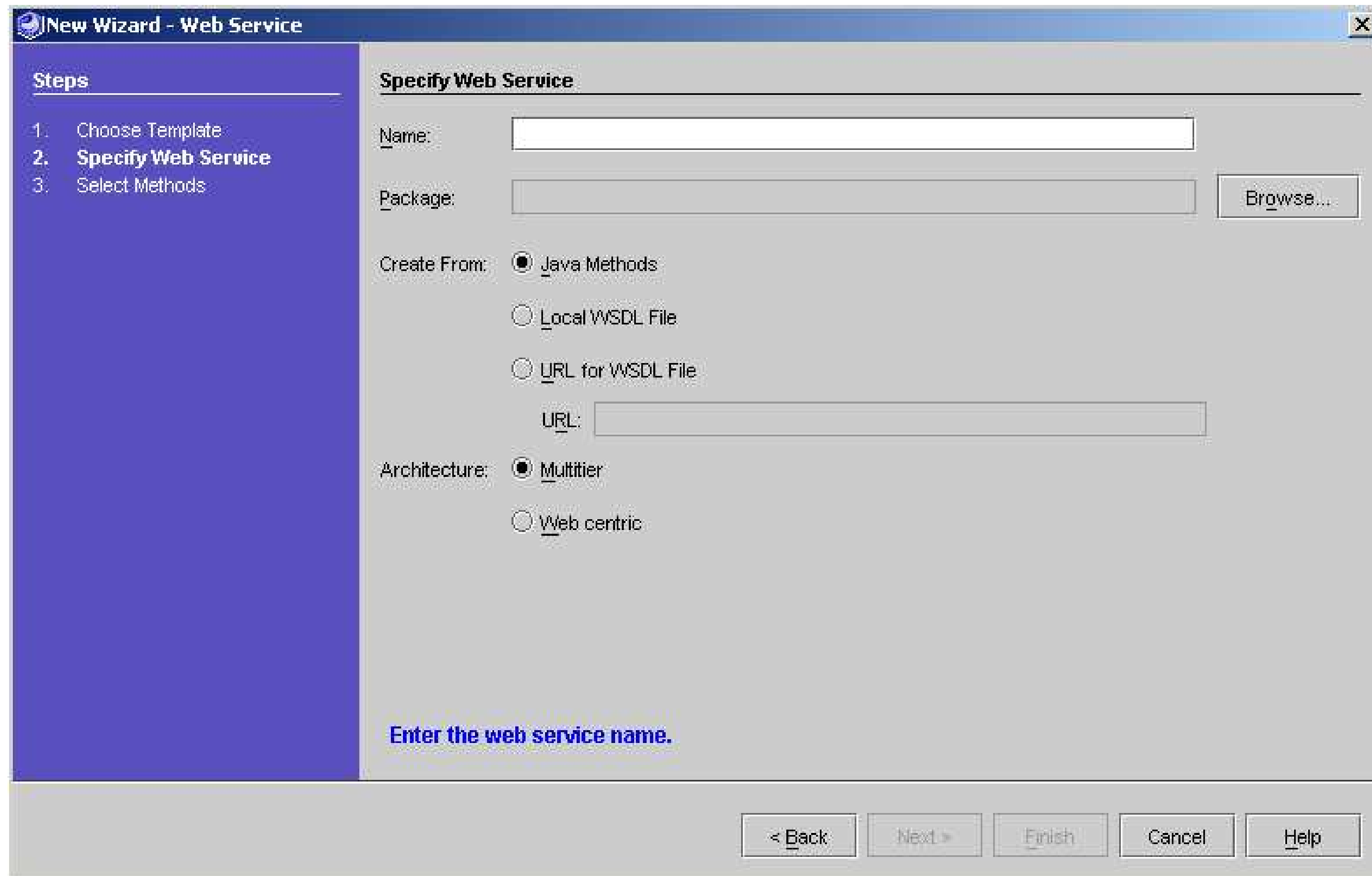


- Automatic generation of WSDL document from Java™ class or EJB™
- Browser-based test code generation
- One-click deployment
- SOAP message handler
- Basic authentication support
- Registration and discovery of Web service (WSDL document) through UDDI registry
- Java™ WSDP is used underneath

Creating a Web service in Java Studio

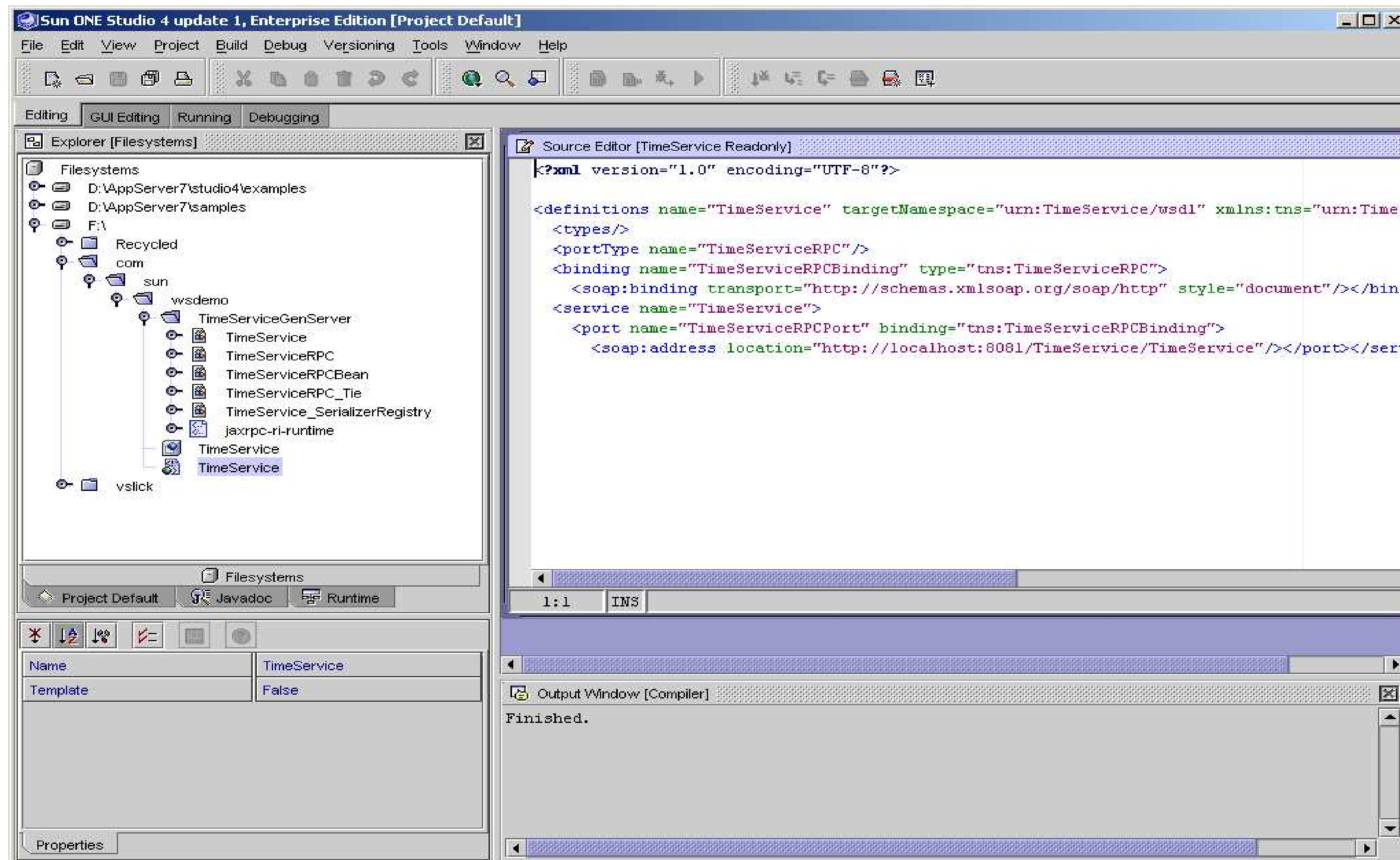


Creating a Web service in Java Studio

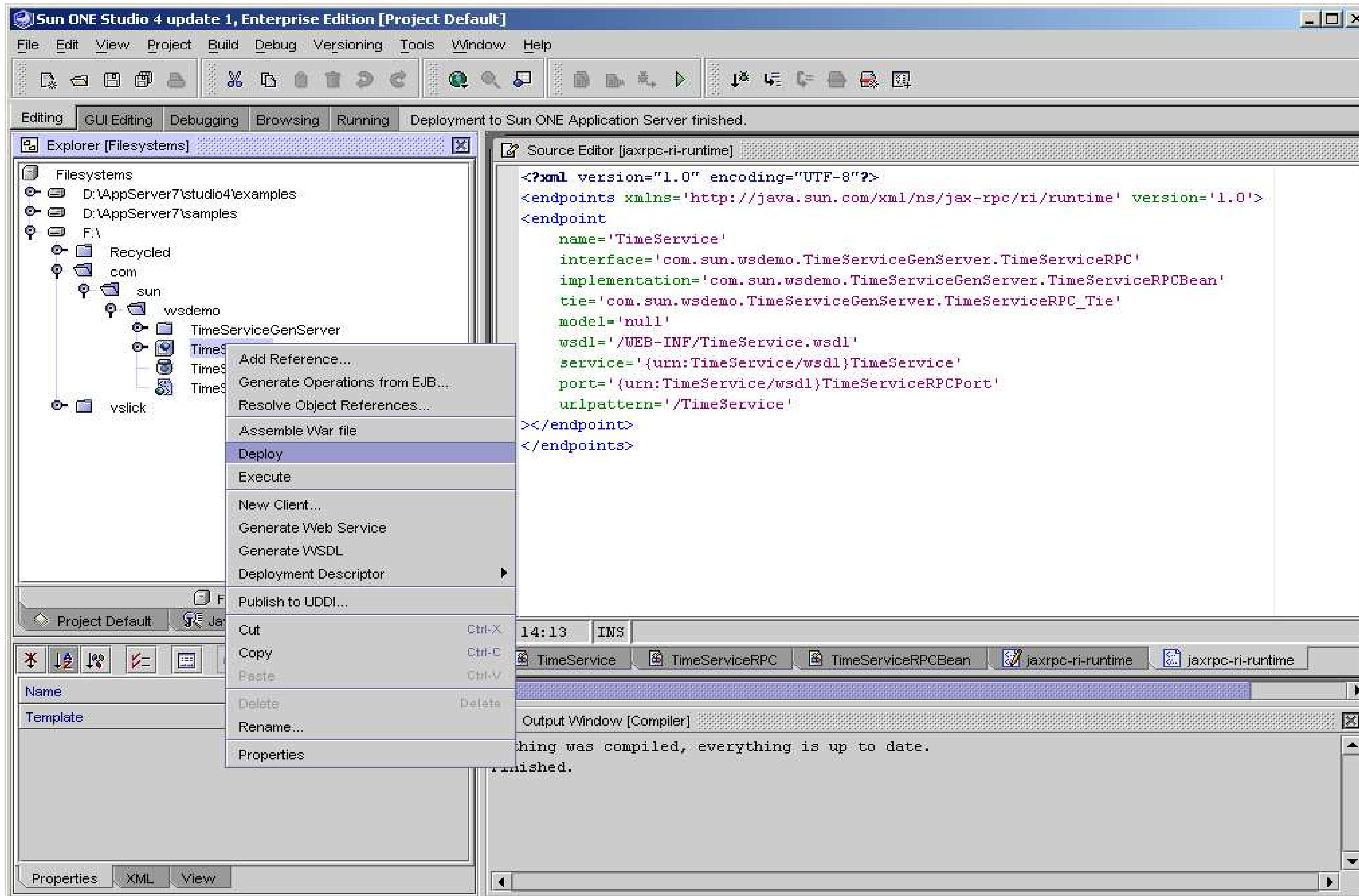


Sun microsystems

Creating a Web service in Java Studio



Creating a Web Service in Java Studio



Summary and Resources

Summary

J2EE™ platform is the platform of choice for the development and deployment of Web services

There are already comprehensive set of Java™ APIs for Web services

Tools are also available for development and deployment of Web services

J2EE™ 1.4 specification makes Web services component as 1st class citizen

Resources on J2EE™ Technology and Web Services

- J2EE™ technology Home Page
  java.sun.com/j2ee
- J2EE™ platform 1.4 Beta
  developer.java.sun.com/developer/earlyAccess/j2sdkee/
- J2EE™ 1.4 Blueprints Adventure Builder 1.0
  developer.java.sun.com/developer/releases/adventure/
- JAXM, JAX-RPC, JAXR
  java.sun.com/xml

Resources on Web Services Tools

- Java™ Web Services Developer Pack Download
  java.sun.com/webservices/downloads/webservicespack.html
- Java™ Web Services Developer Pack Tutorial
  java.sun.com/webservices/downloads/webservicestutorial.html
- Sun™ Application Server
  wwws.sun.com/software/products/appsrvr/home_appsrvr.html
- Java Studio
  wwws.sun.com/software/sundev/jde/buy/index.html

Web Services Tutorials With Sun™ App Server and Java Studio

- J2EE™ Application Tutorial for Sun™ ONE Platform
  - java.sun.com/j2ee/1.3/docs/tutorial/doc/index.html
- Developing Amazon.com Web service client
  - developer.java.sun.com/developer/technicalArticles/WebServices/amazonws/
- Building Web services with Sun™ ONE Application Server
  - sunonedev.sun.com/building/tech_articles/jaxrpc/
- Java Studio Web services tutorial
  - wwws.sun.com/software/sundev/jde/examples/index.html
- JAX-RPC on the Sun™ ONE Web Services Platform Developer Edition
  - sunonedev.sun.com/building/tech_articles/jaxrpcs1.html

Marc Hamilton
marc.hamilton@sun.com
Sun Microsystems, Inc.