# J2EE and .NET: Competition or Cooperation

## Integration using SOAP

Jeff Swisher

jeffs@dunnsolutions.com

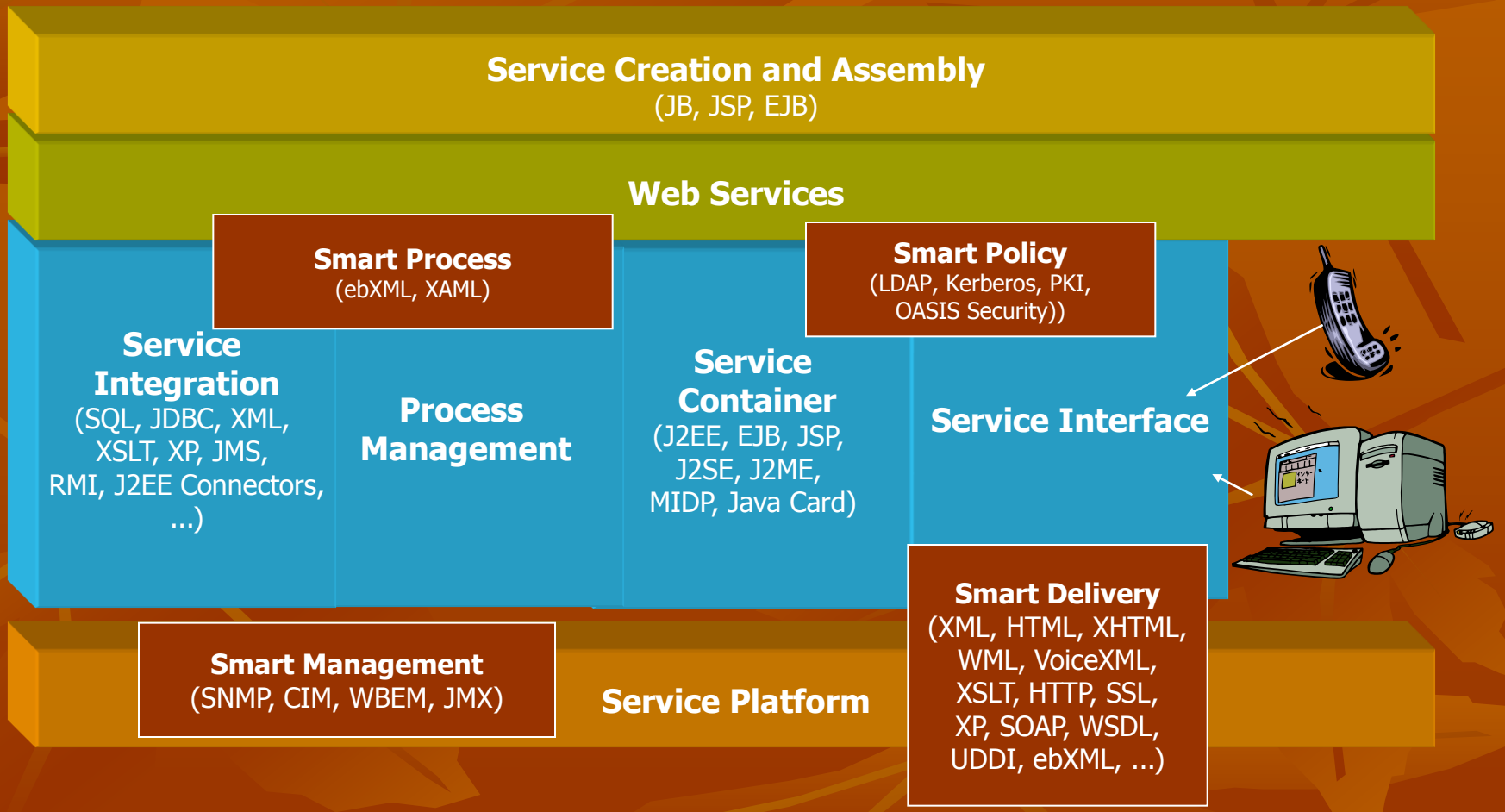
# Objectives

- History of J2EE and .NET
- Understand how J2EE and .NET implement web services
- Understand the key issues with integrating the two web services platforms
- Understand how the two platforms can be integrated

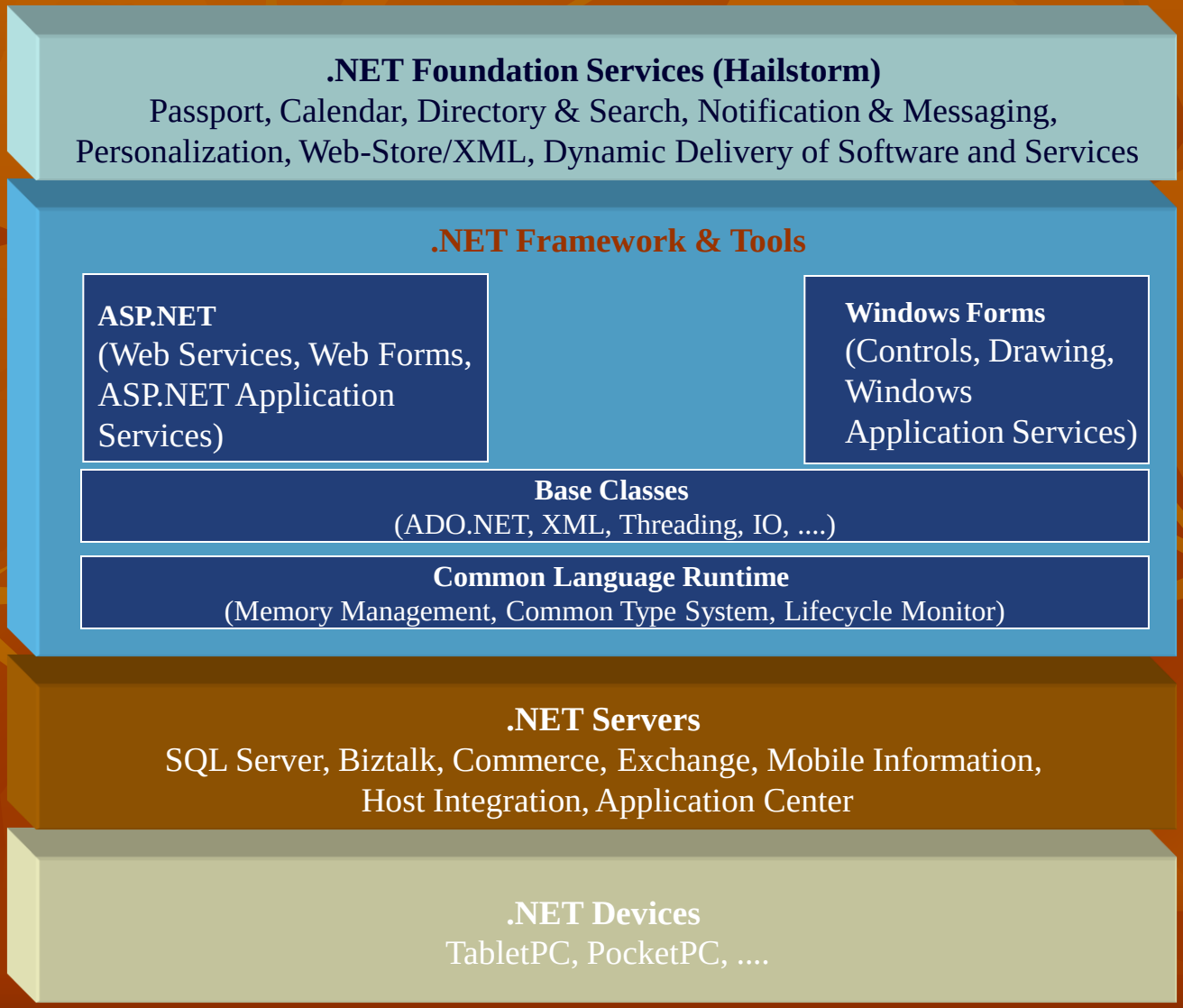
# Introduction

- Both J2EE and .NET web services exist and need to interoperate
- How do I demonstrate that they can interoperate?
  - test that a .NET client can call a J2EE web service
  - test that a J2EE client can call a .NET web service

# J2EE Way of Life



# .NET Way of Life

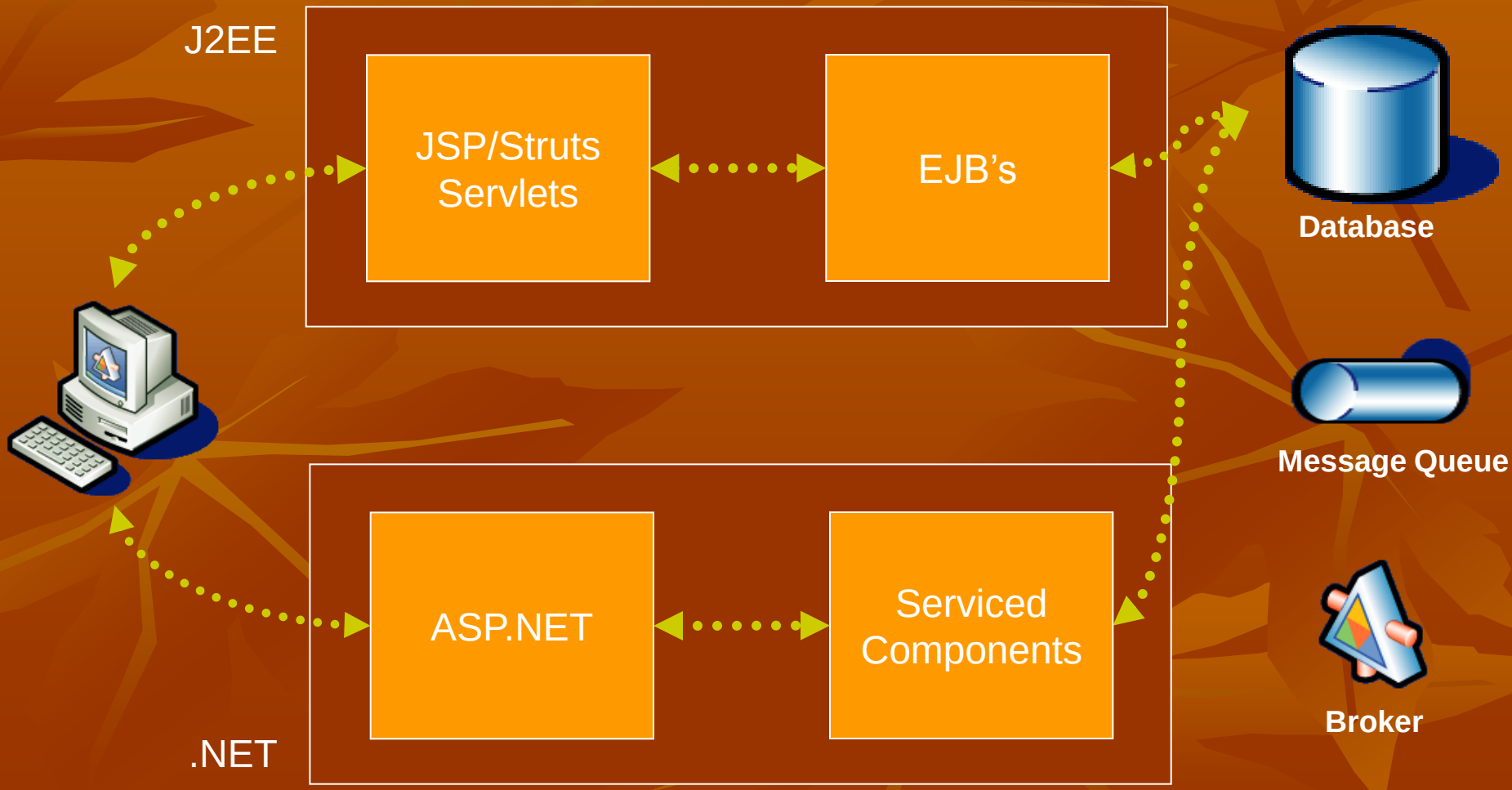


# Battle for the Enterprise Begins

- **K**eeping track of the ongoing rivalry between Microsoft and Sun Microsystems as they seek to control the enterprise application development market can puzzle even the most dedicated developer. For the software architect, the competing "standards" can be bewildering.

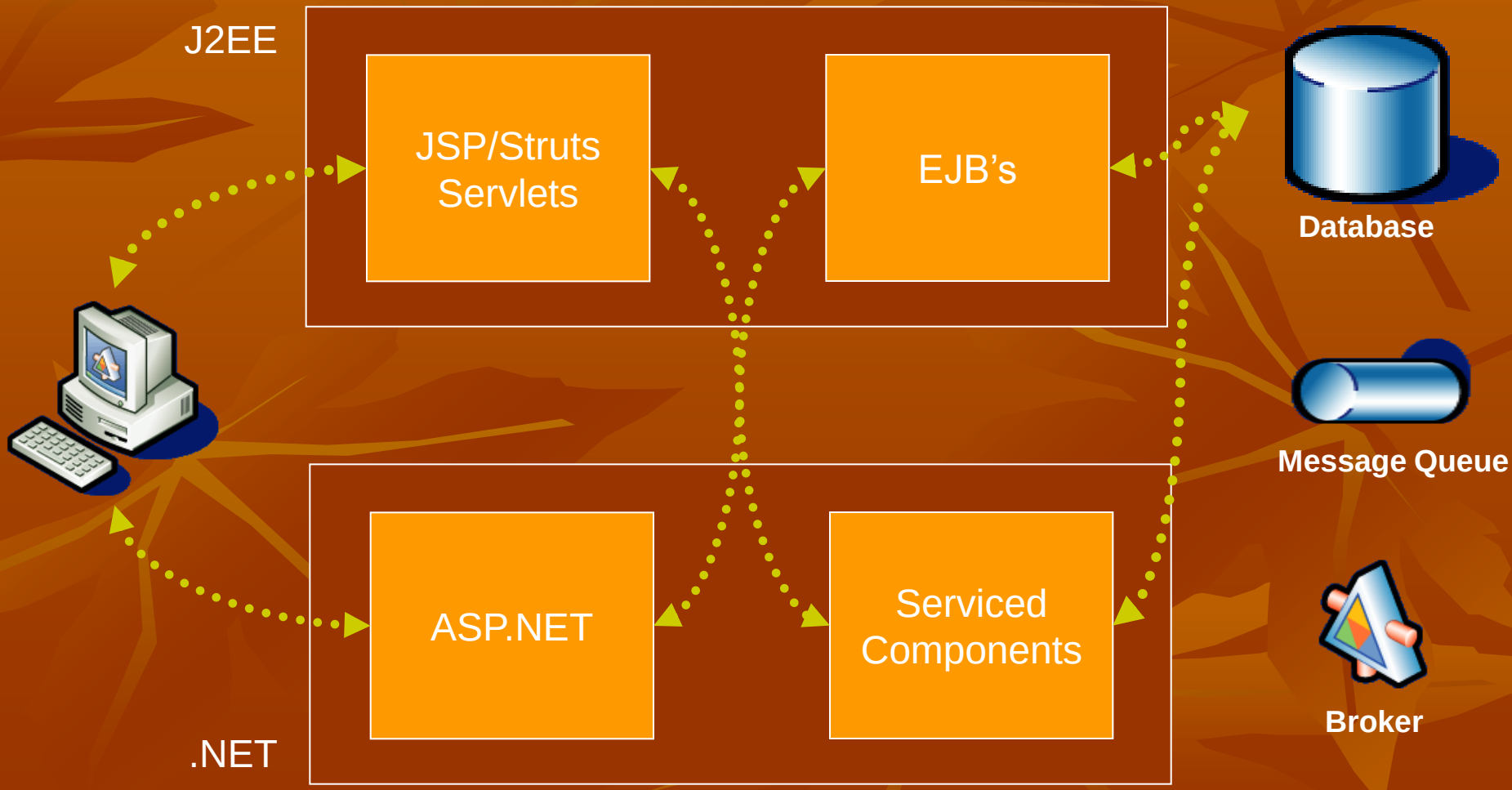
# Interoperability Scenarios

## Technology Map



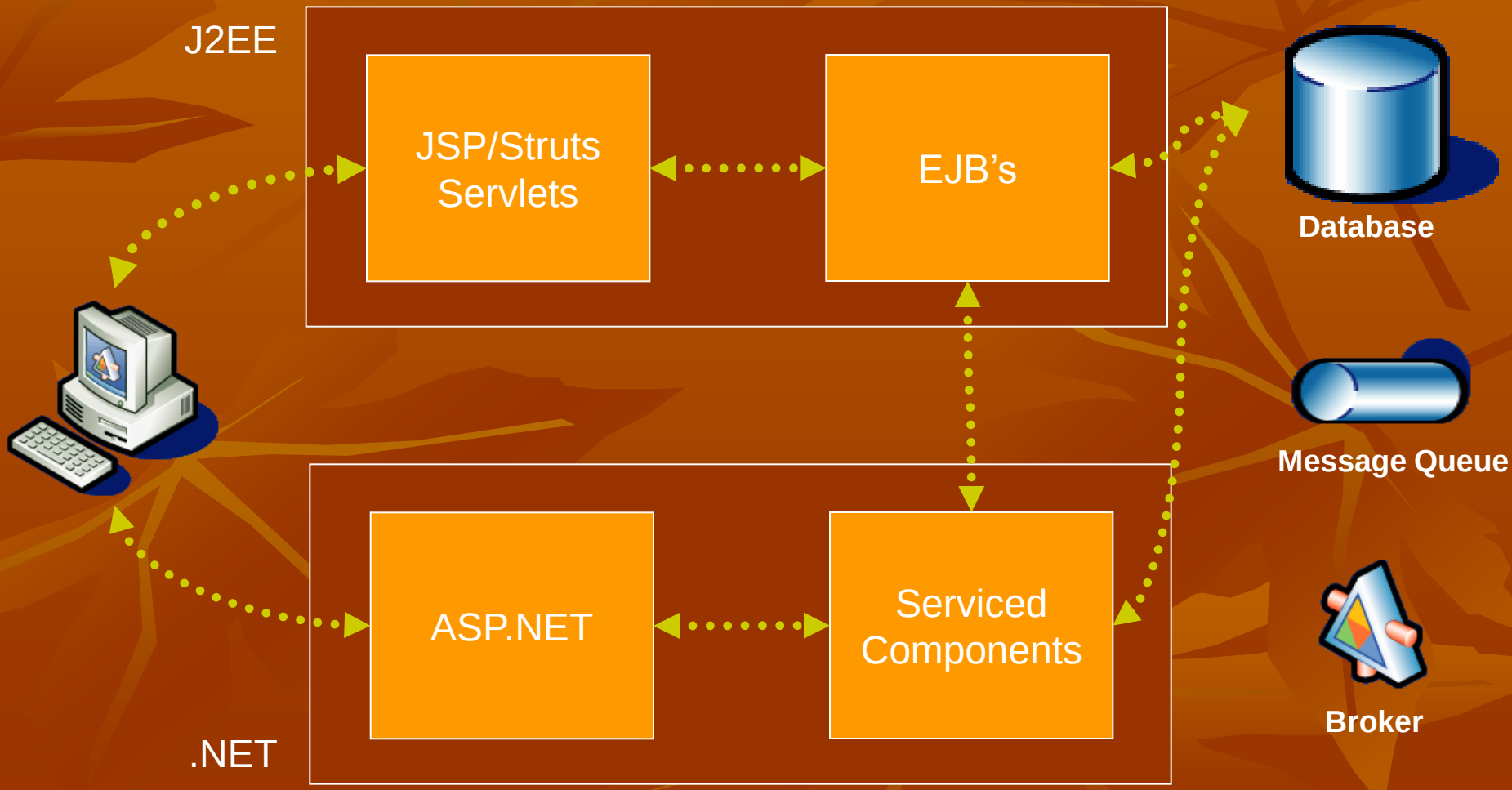
# Interoperability Scenarios

## Presentation Tier



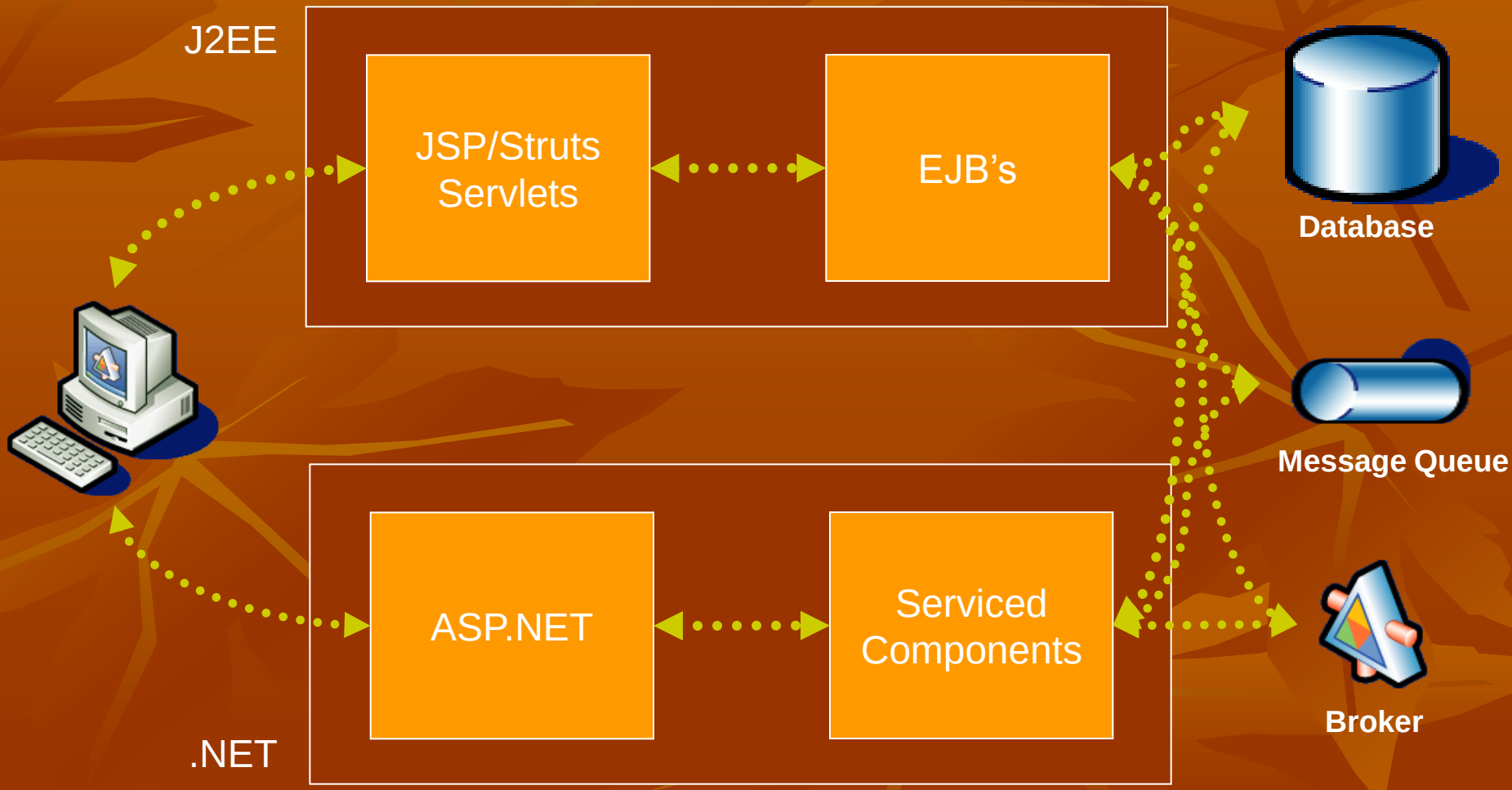
# Interoperability Scenarios

## Business Tier



# Interoperability Scenarios

## Shared Resources



# Integration Technologies

- Wire Level
- Message Queues
- Web Services

# Integration Technologies

## Wire Level

- Products
  - Janeva(Borland), Ja.NET (Intrinsyc), JNBridge Pro (JNBridge)
- Advantages
  - Wire Level, Binary Performance
  - Keep Alive
- Disadvantages
  - Tightly Coupled Interface
  - Vendor Specific

# Integration Technologies
## Message Queues

- Products
  - MSMQ, IBM WebSphere MQ
  - Host Integration Server 2000, BizTalk Server 2002
- Advantages
  - Loosely Coupled, N to N Scenarios, SOA
  - Transactions, Security, RM
- Disadvantages
  - Synchronous Operation is Limited
  - Possible Port / Firewall Issues
  - Message Queue between Organizations?

# Integration Technologies

## Web Services

- Products
  - Standards-based SOAP stacks
  - MS ASP.NET
  - Apache Axis, Glue etc.
- Advantages
  - Leverage component architectures & design patterns
  - Facilitate service-orientation
  - Loosely Coupled
- Disadvantages
  - Verbose

# Integration Complications

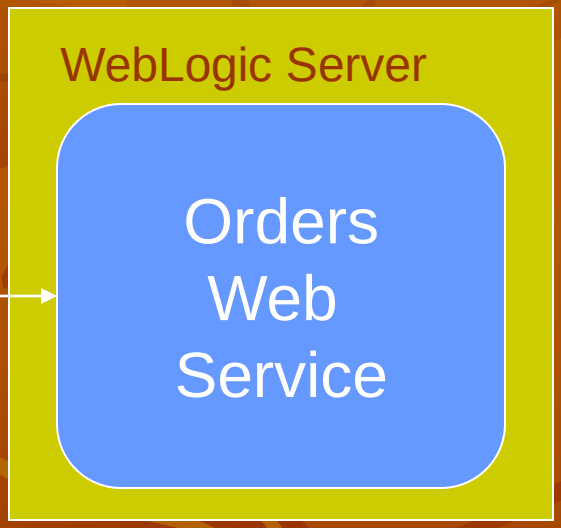
- Wire
  - Creates proxies of data types
  - Proxies Not Platform Friendly (e.g. No Data Binding)
- Message Queues
  - Already exchange messages!
  - Need a common format
- Web Services
  - Should be exchanging messages!
  - WSDL differences
    - RPC vs. document style interfaces
  - Not Platform Friendly (e.g. No Data Binding)

# WebServices Integration

Calling a J2EE Web Service from a .NET Client

# .NET Client for Orders

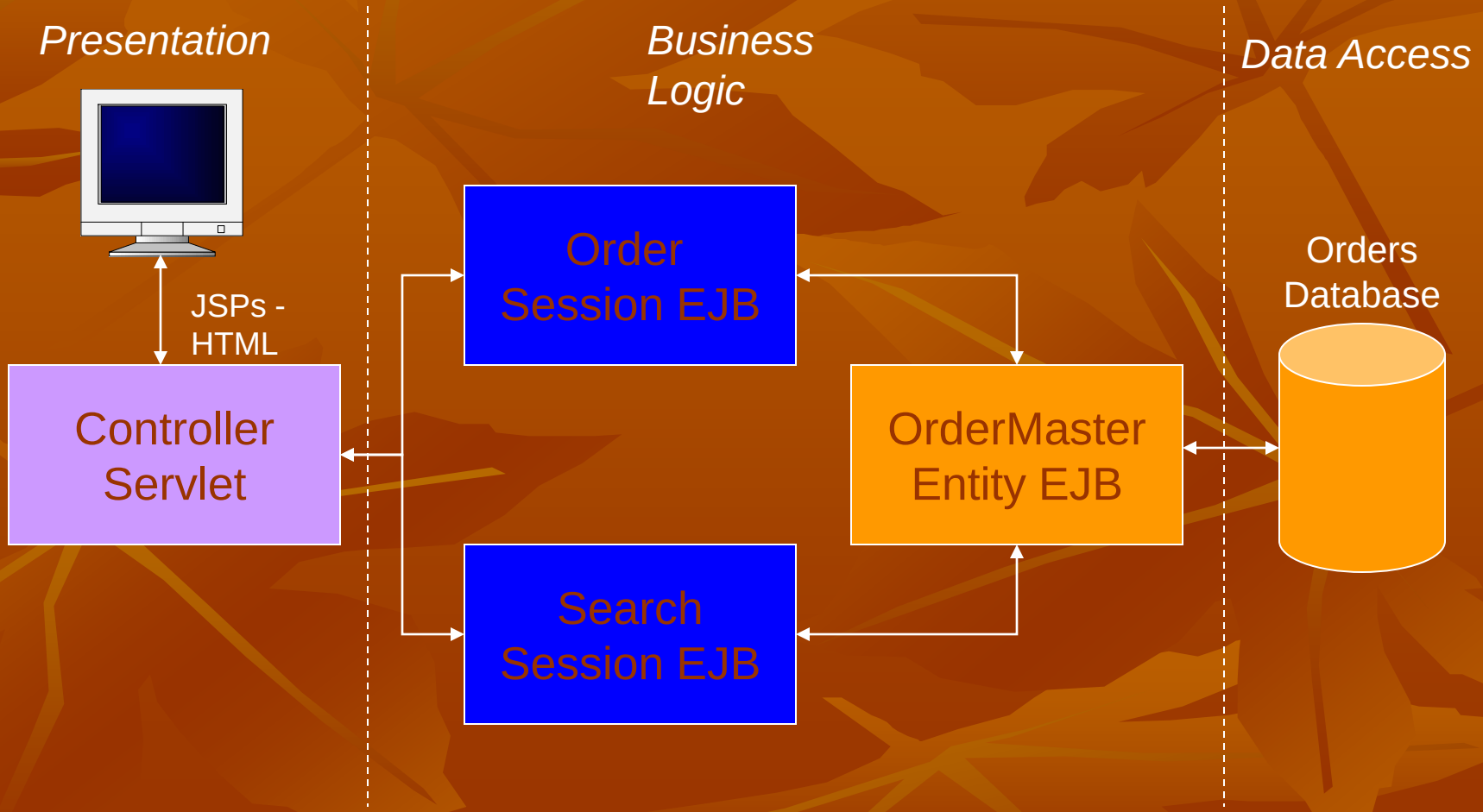





Redirect to the Order list page

Call the Orders web service and get the user's Order information to display

# J2EE Orders Application

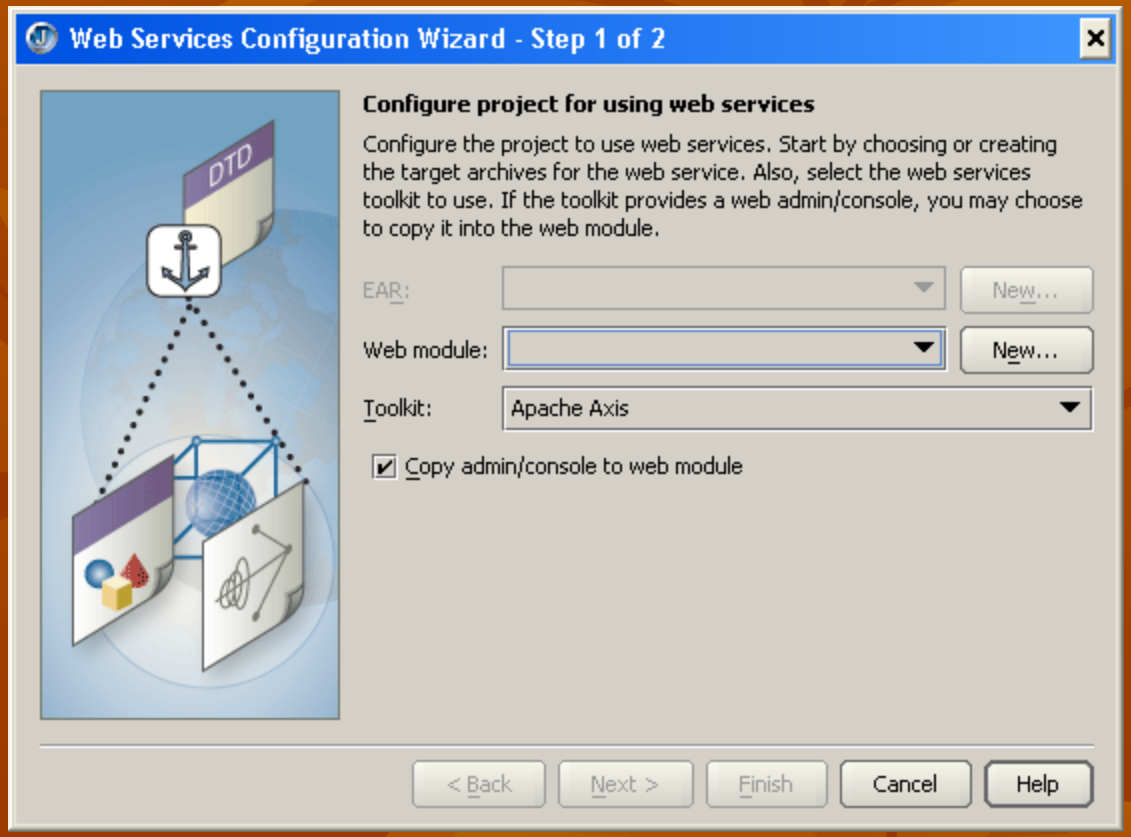


# Making Orders a J2EE Web Service

- reuse the EJBs to process the web service requests – you want to make sure you have a stateless interface
- create a web service that will expose the methods your EJBs provide
- add the EJBs to the web service using controls
- check to see if you need to change any of the data types being returned from the web service
- generate a WSDL file for clients

# Adding the EJBs to the web service

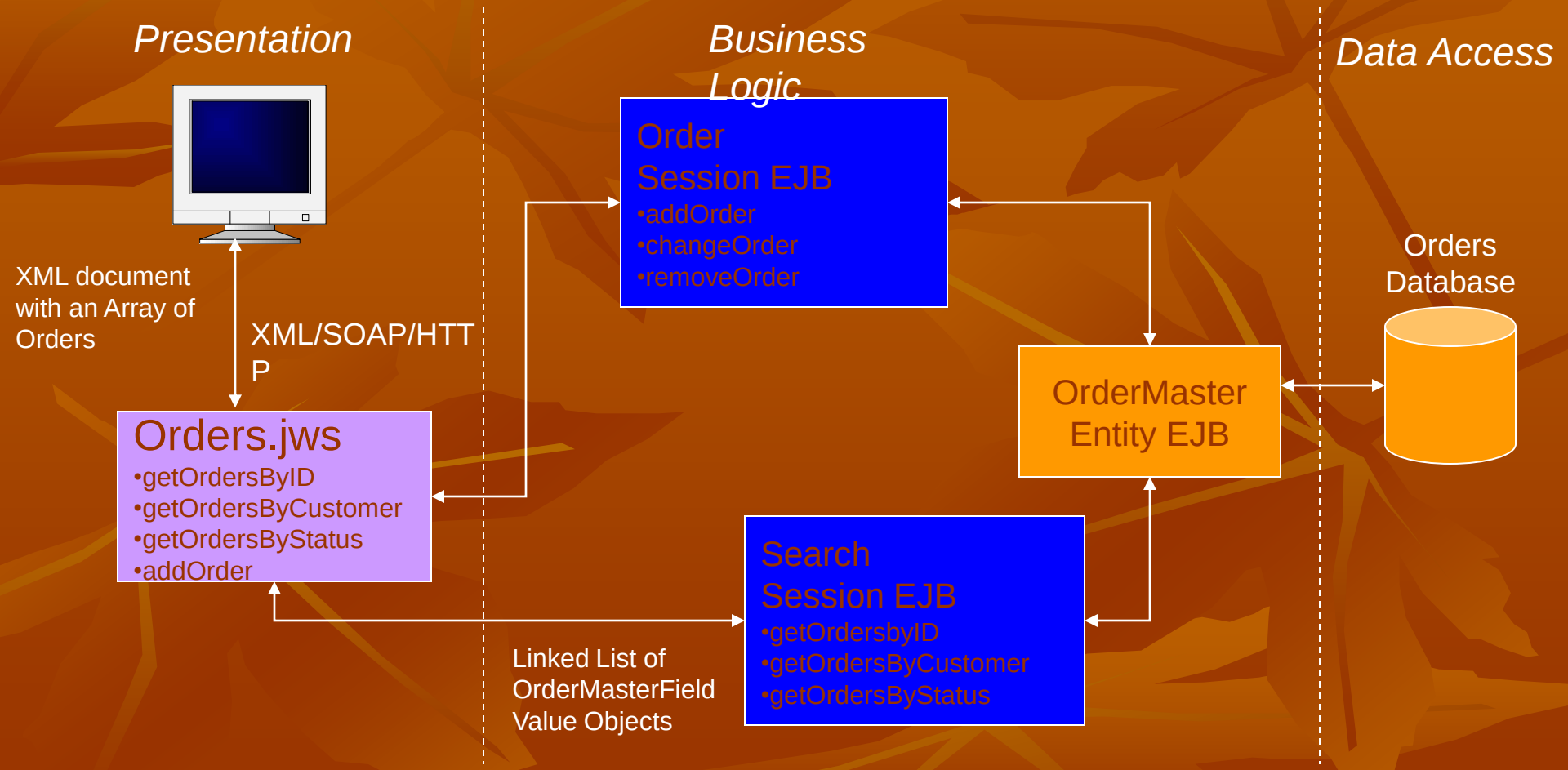
- add EJB controls
- creates an interface to the EJB so the web service can contact it



# Calling the EJBs from Orders.jws

```
/**
     * @jws:operation
     */
    public ordermasterfields[] getOrdersByID(String ID)
    {
         ordermasterfields[] orderArray = null;
        try{
            LinkedList orderList = SearchEJB.getOrdersbyID(ID);
             orderArray = getArray(orderList)
        }
        catch(Exception e){
            System.out.println("*** In getOrdersByID method *** \n
                 Exception is: " + e.getMessage());
        }
        return (orderArray);
    }
```

# J2EE Orders Web Service



# Request and Response Code

- the request

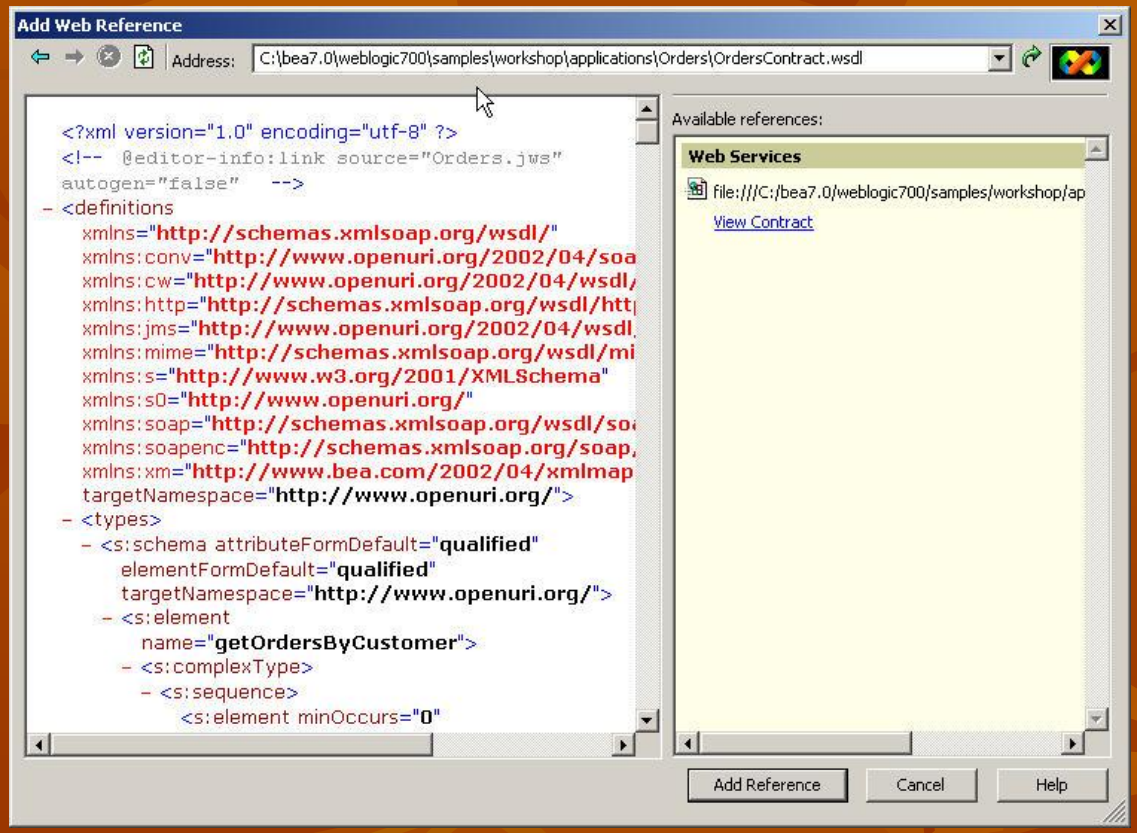
```
<getOrdersByID xmlns="http://www.openuri.org/">
  <ID>JANE</ID>
</getOrdersByID>
```

- the response

```
<ArrayOfordermasterfields xmlns="http://www.openuri.org/">
        <ordermasterfields>
                <orderNbr>1</orderNbr>
                <customerNbr>100</customerNbr>
                <employeeID>JANE</employeeID>
                <customerName>Johns Lock Shop</customerName>
                … …
        </ordermasterfields>
</ArrayOfordermasterfields>
```

# Creating a .NET Client for Orders

- add a web reference to the client
- automatically generates proxy code



# .NET Client Proxies

- defines all the methods and data types used to interact with the Orders web service

Reference.cs

```
public Orders() {
  this.Url = "http://HOBSONEVELYN:7001/Orders/Orders.jws"; }

public ordermasterfields[] getOrdersByID(string ID) {
  object[] results =
    this.Invoke("getOrdersByID", ne object[] {ID});
return ((ordermasterfields[])(results[0]));}
```

# Mapping Between the Client Proxy and the WSDL

**Reference.cs**

```
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.openuri.org/")]
    public class ordermasterfields {
          public long orderNbr;
          public string orderDate;
          public System.Single partPrice;
          … …
    }
```

Orders WSDL file

```
<s:complexType name="ordermasterfields">
   <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="orderNbr" type="s:long" />
          <s:element minOccurs="0" maxOccurs="1" name="orderDate" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="partPrice" type="s:float" />
          … …
   </s:sequence>
</s:complexType>
```

# Creating a .NET Client for Orders

- create an instance of orders

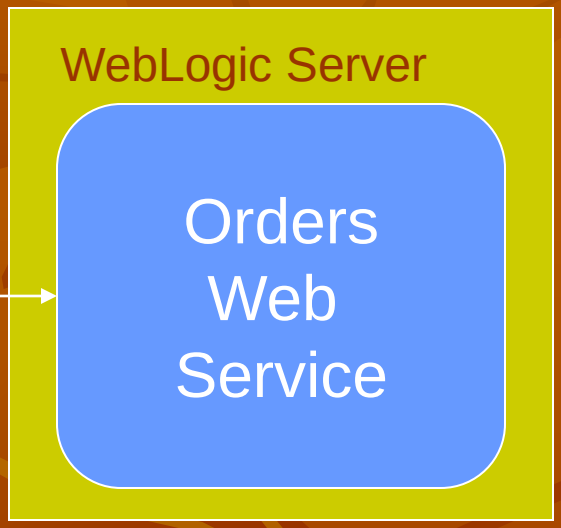
```
using OrderWebApplication.WebReference1;

WebReference1.Orders orders = new
WebReference1.Orders();
```

- call the web service method

```
WebReference1.ordermasterfields[] ordersResult =
orders.getOrdersByID(userName);
```

# .NET Client for Orders

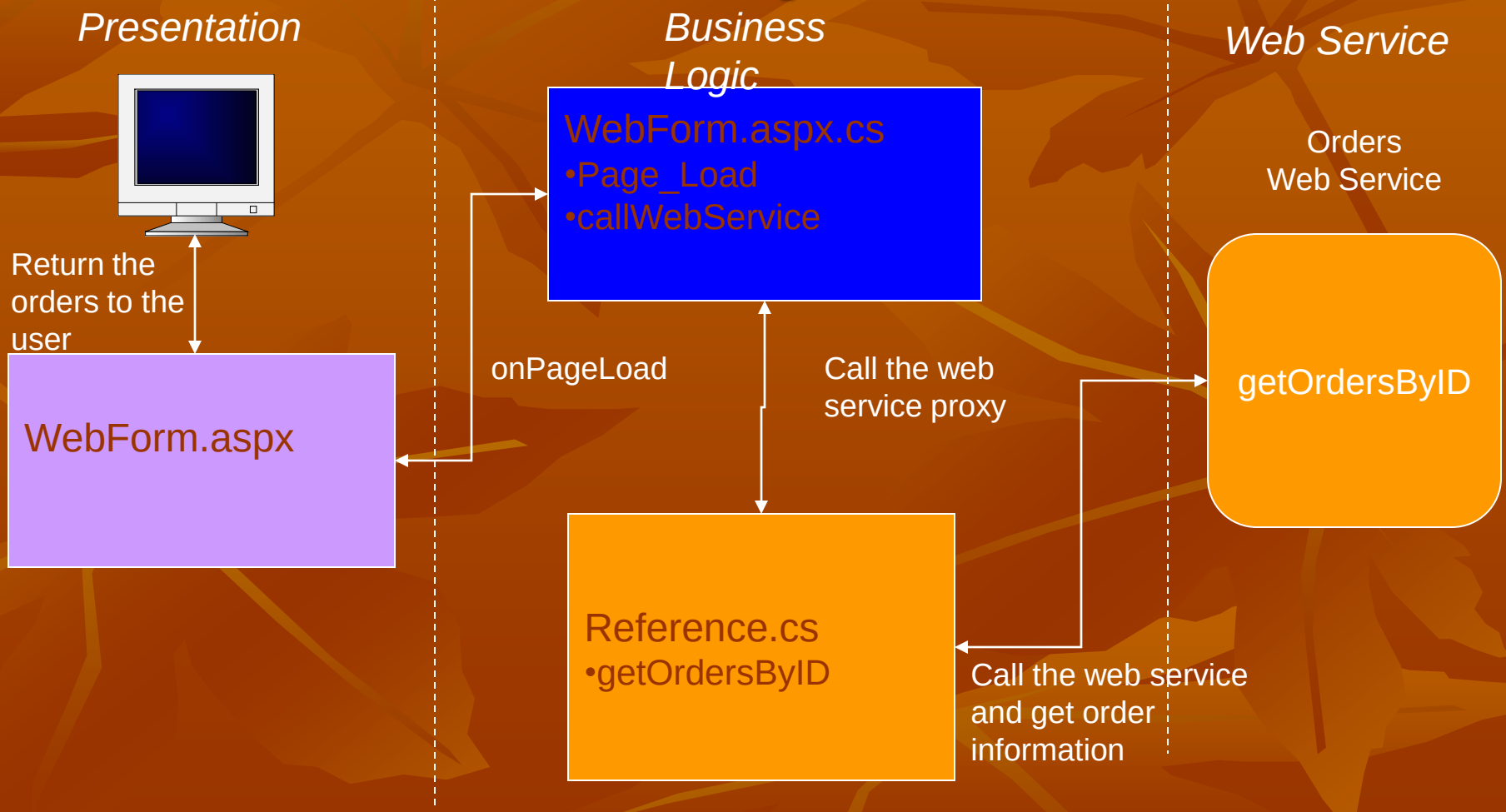





Redirect to the Order list page

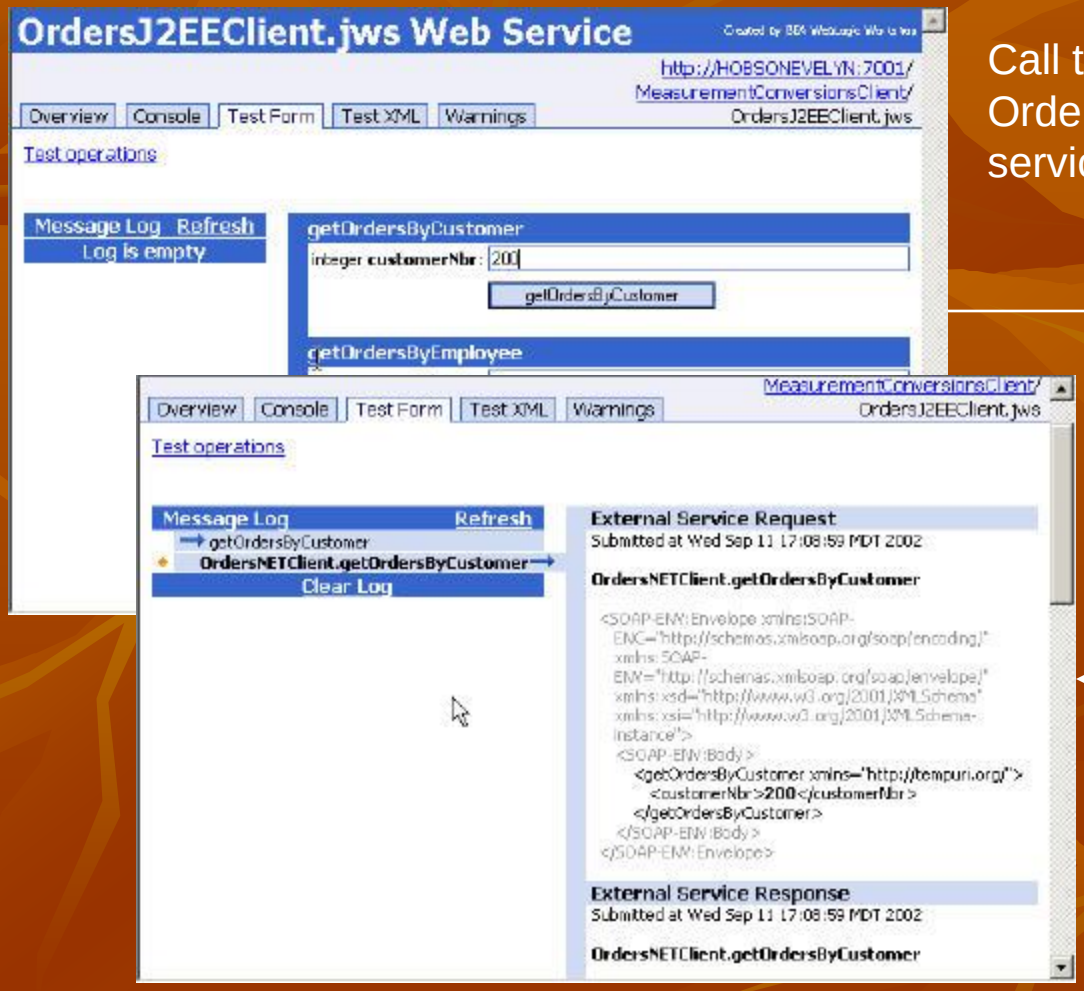
Call the Orders web service and get the user's Order information to display

# .NET Orders Client an example call



# Calling a .NET Web Service from a J2EE Client

# J2EE Client for Orders



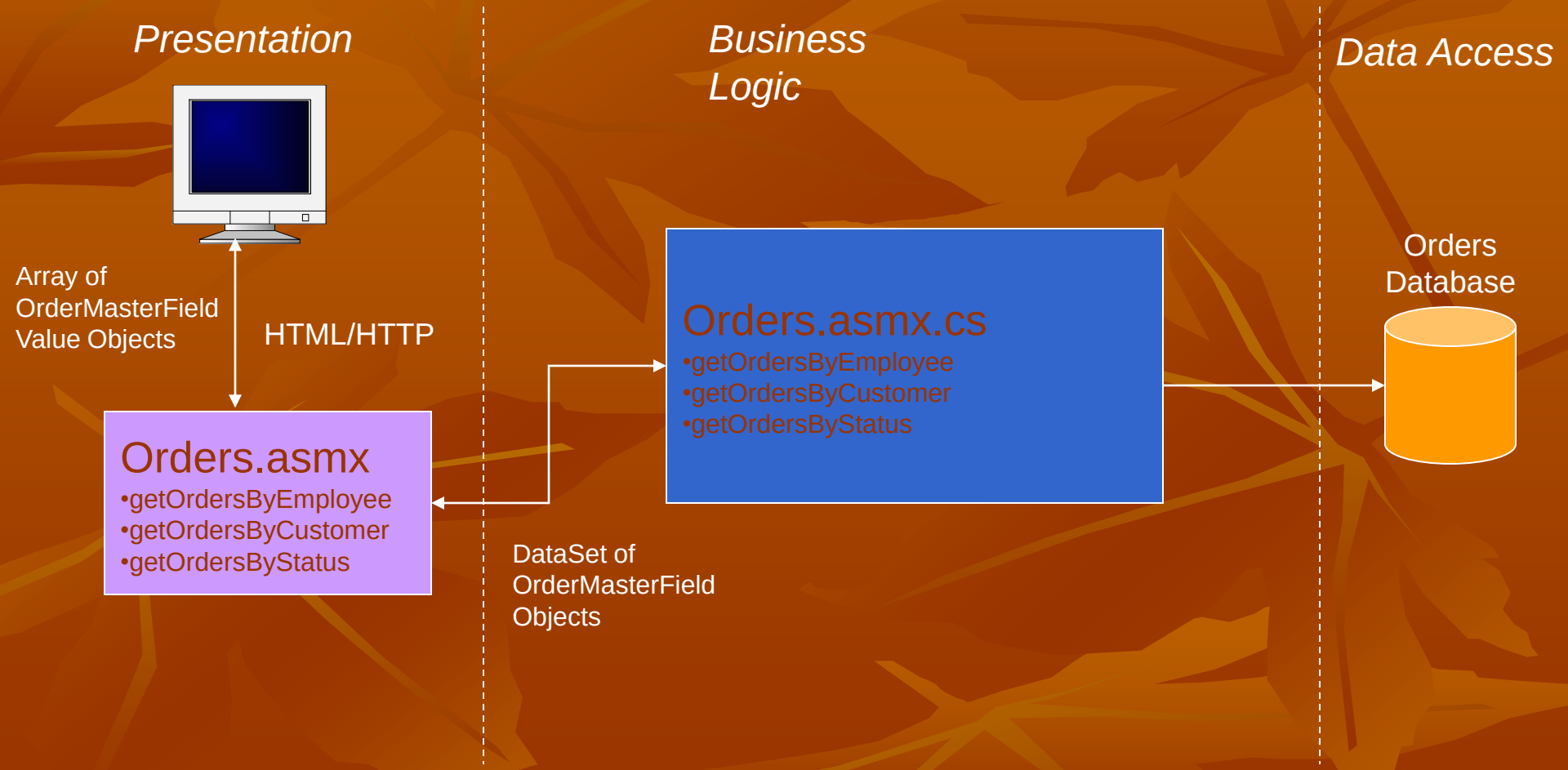

Call the Orders web service

IIS

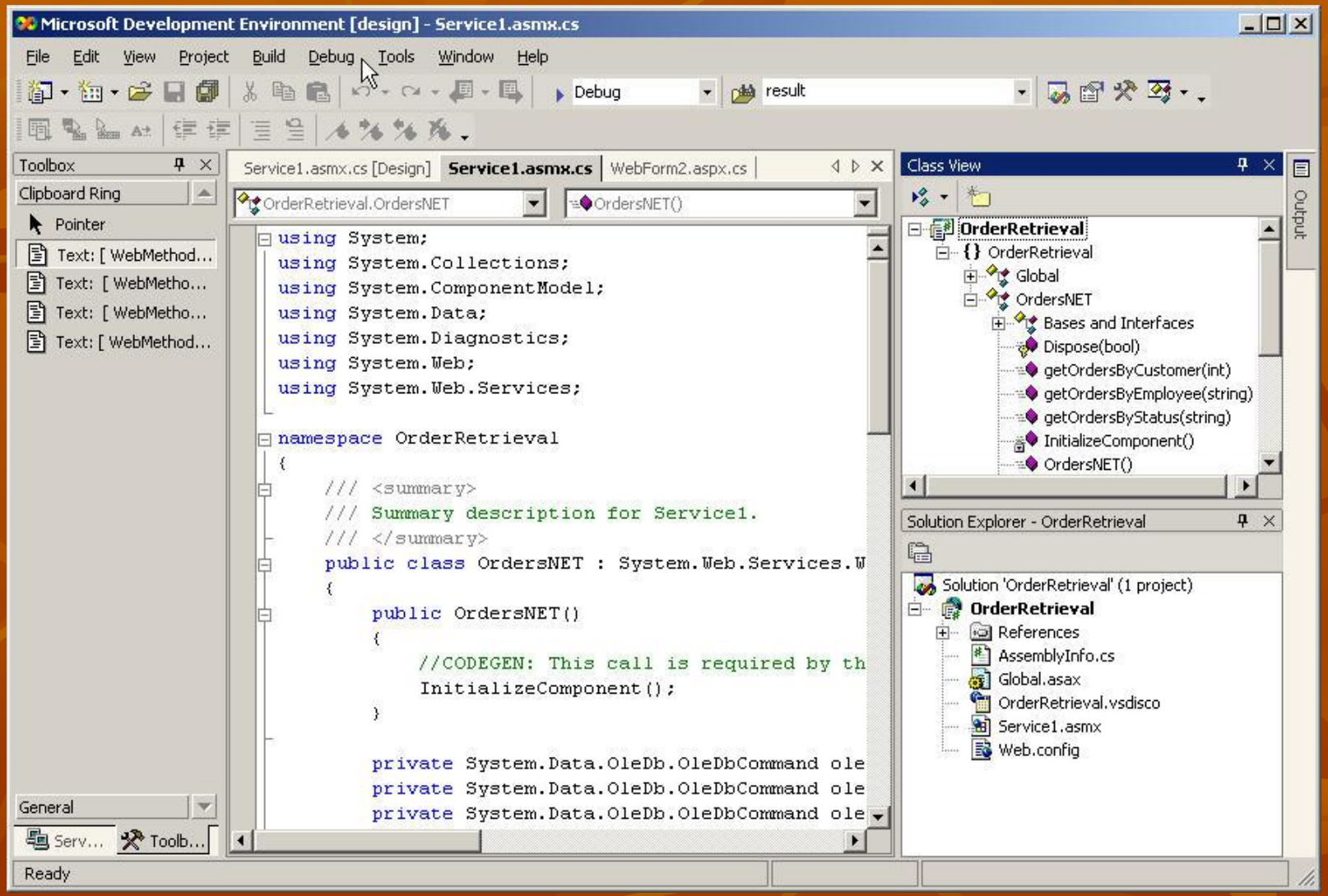
Orders Web Service

Return the user's Order information to display

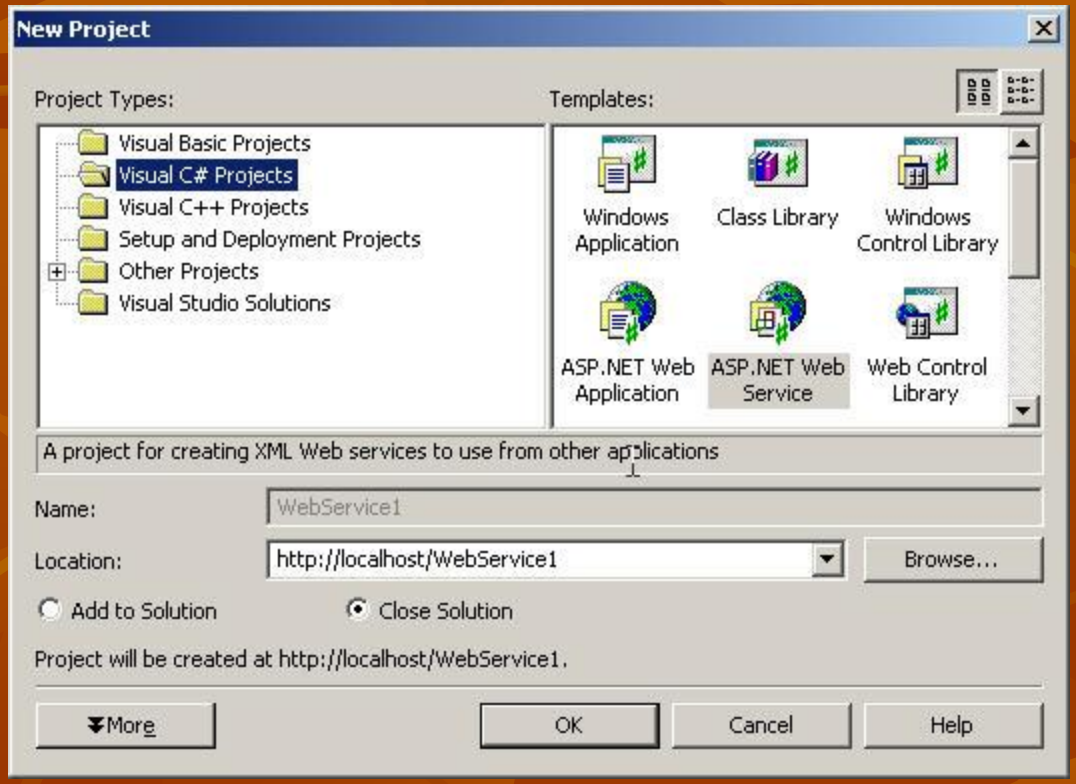
# .NET Orders Web Service

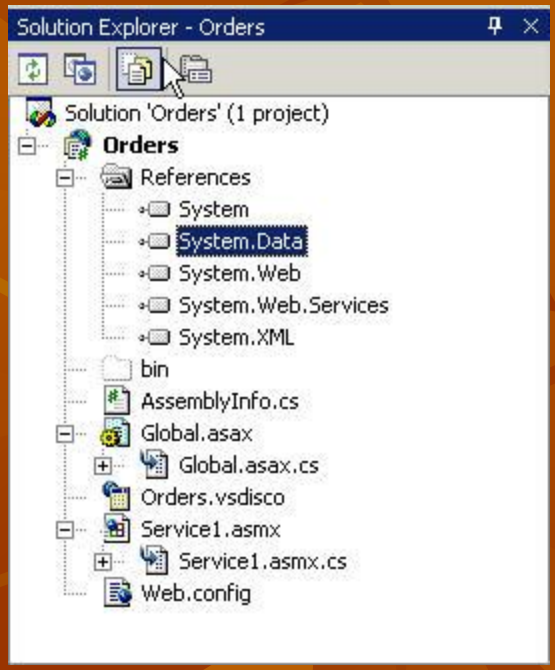


# Orders in VisualStudio



# Create a ASP.NET Web Service



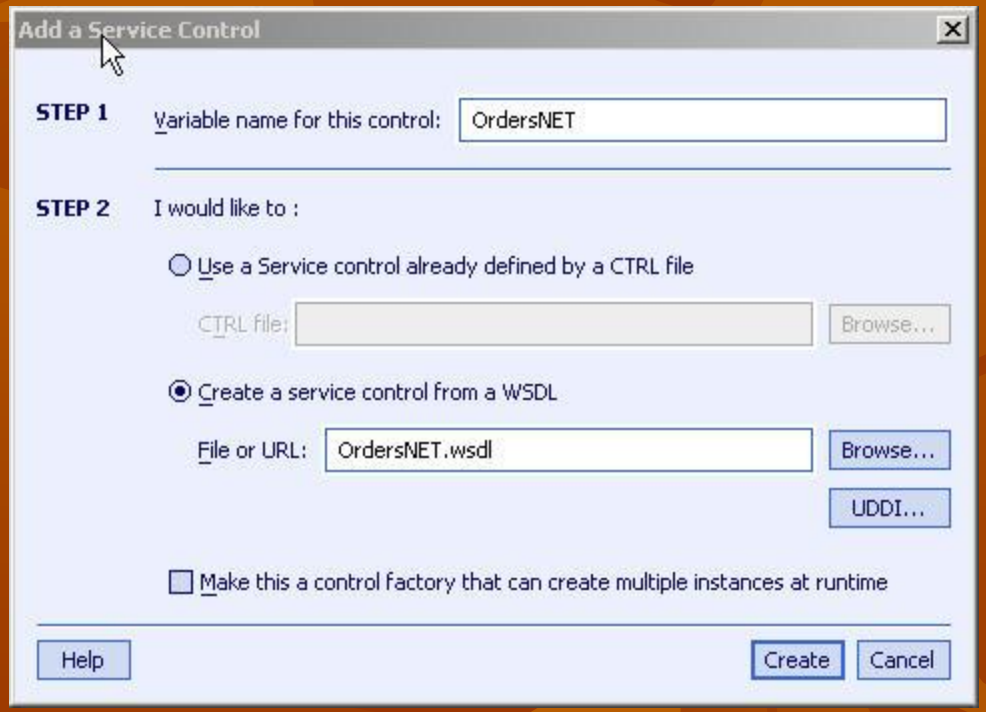



# Create Web Methods for the Web Service

```
[ WebMethod ]
public DataSet getOrdersByCustomer(int customerNbr)
{
    DataSet ds = new DataSet("OrderSet");
    oleDbDataAdapter1.SelectCommand.CommandText = "SELECT
    OrderNumber,CustomerName,ExpectedShipDate FROM OrderMaster WHERE
    CustomerNbr=" + customerNbr + " ORDER BY OrderNumber";
    oleDbDataAdapter1.Fill(ds,"Orders");
    return ds;
}
```

# Creating a J2EE Client for Orders

- Add the WSDL to the directory tree
- Create a service control for the .NET web service



# Creating a J2EE Client for Orders

- create an instance of the service control
- create methods that map to the web service methods
- call the web service method

```
private OrdersNETControl OrdersNETClient;

public Node getOrdersByCustomer(int customerNbr)
{
    return (Node)
    OrdersNETClient.getOrdersByCustomer(customerNbr);
}
```

# J2EE Client Proxies

OrdersNETControl.control

```
/** @editor-info:link source="OrdersNET.wsdl" autogen="true" */
import weblogic.jws.control.ServiceControl;

/**
 * @jws:location http-
   url="http://localhost/OrderRetrieval/Service1.asmx"
 * @jws:wsdl file="#OrdersNETWsdl"
 */
public interface OrdersNETControl extends ServiceControl
{
    public org.w3c.dom.Node getOrdersByEmployee
                (java.lang.String EmployeeID);
…

}
```

# Mapping Between the Client Proxy and the WSDL

**OrdersNETControl.ctrl**

```
public interface OrdersNETControl extends ServiceControl
{
    public org.w3c.dom.Node getOrdersByEmployee (java.lang.String EmployeeID);
    public org.w3c.dom.Node getOrdersByStatus (java.lang.String status);
    public org.w3c.dom.Node getOrdersByCustomer (int customerNbr);

}
```

OrdersNET WSDL file

```
<s:element name="DataSet" nillable="true">
  <s:complexType>
    <s:sequence>
      <s:element ref="s:schema" />
      <s:any />
    </s:sequence>
  </s:complexType>
</s:element>
```

# Request and Response Code

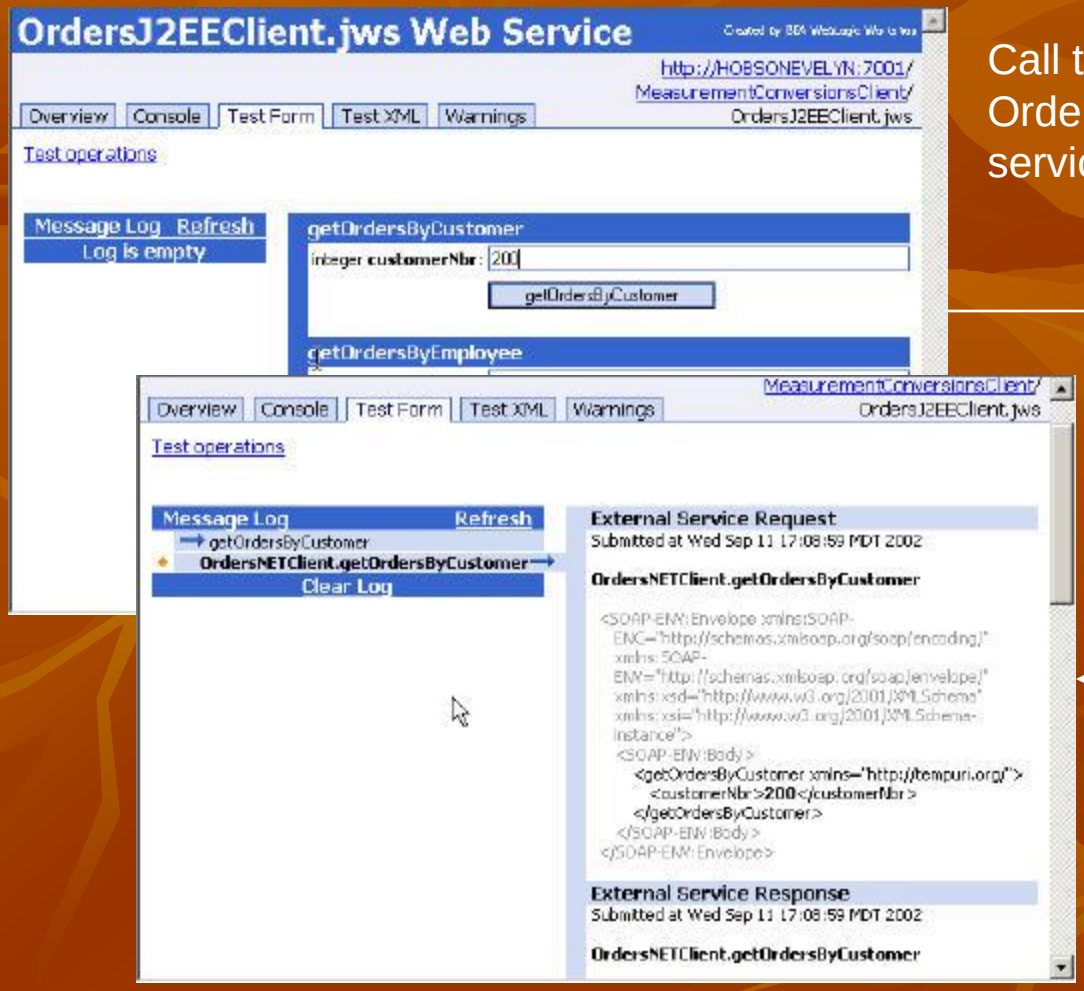
- the request

```
<getOrdersByCustomer xmlns="http://tempuri.org/">
    <customerNbr>200</customerNbr>
</getOrdersByCustomer>
```

- the response

```
<OrderSet xmlns="">
    <Orders diffgr:id="Orders1" msdata:rowOrder="0">
            <OrderNumber>2</OrderNumber>
            <CustomerName>Set in Place</CustomerName>
            <ExpectedShipDate>2002-10-02</ExpectedShipDate>
    </Orders>
</OrderSet>
```

# J2EE Client for Orders




Call the Orders web service

IIS

Orders Web Service

Return the user's Order information to display

# How Can .NET and J2EE Interoperate?

- simple web services interoperability - SOAP, WSDL
  - J2EE client can invoke a .NET web service
  - J2EE client can use a .NET WSDL document to create a stub
  - .NET client can invoke a J2EE web service
  - .NET client can use a J2EE WSDL document to create a stub
- adding support for all XML Schema types
  - XML Schema choice → Java type ?
- adding support for .NET types
  - ADO.NET DataSet → Java type ?
- adding support for .NET SOAP extensions

# Enhancing Interoperability

- SOAP interoperability issues still exist
- WSDL is at the heart of service interoperability
  - a common WSDL interface can increase reusability
  - WSDL also enables interoperability with SOAP clients
  - tools can automatically generate client-side proxies
- other best practices
  - stick with well-accepted standards (e.g., SOAP 1.2)
  - keep your data types simple
  - provide XML schema definitions for all your data types

# Integration Complications…

- .NET and Java Data Types do NOT map
  - .net DataSet = ????
  - Java Vector = ????
- We like to work with Native Objects
  - DataSets, Collections, etc.

# Integration Complications…

- Thoughts:
  - We Should be Thinking Service
  - We Should be Thinking Multiplatform
  - We Should NOT be Thinking RPC's
    - i.e. Method Calls
- We are Exchanging Messages
- XML is our Data Format
- XSD Should Define Message Structure

# Conclusions

- Web services can be written successfully in either .NET or J2EE
- Interoperability between J2EE and .NET web services is possible today and continued improvements are coming
- Interoperability has improved dramatically
- Use common data types that can be easily serialized
- Learning a new language or paradigm might be needed

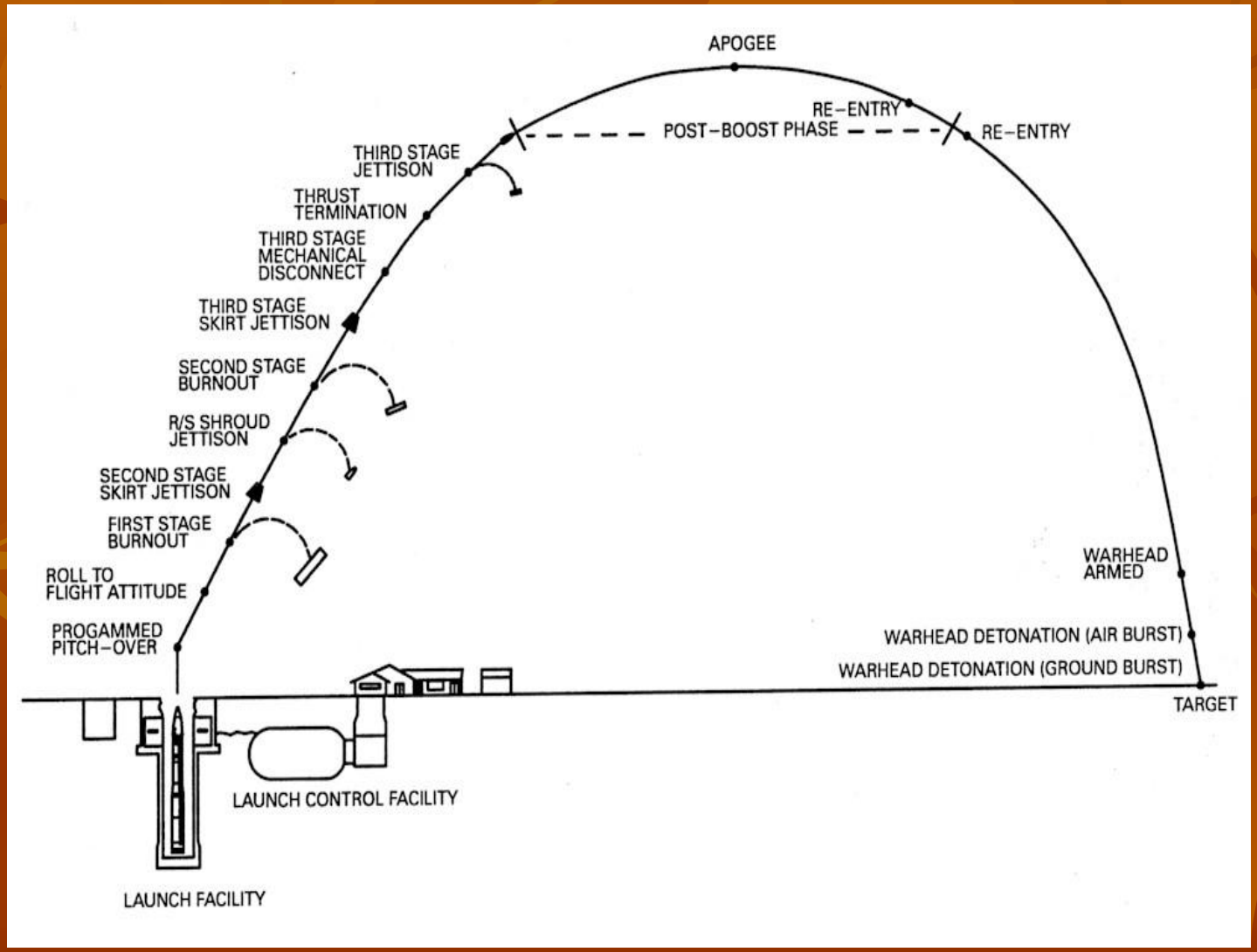
# Why Borland?

- Borland is uniquely positioned to leverage both web services platforms with our hardware offerings and partnerships with Microsoft and J2EE vendors
- Borland will extend its industry-leading development software to provide an end-to-end web services management solution for both J2EE and .NET

# Real Example

- United States Air force Space Command
  - Create a system to pull together (.NET, Fortran, Cobol) and other systems into one integration platform.

# Air Force Space Command…



# Questions

Jeff Swisher
Dunn Solutions Group
Email: jeffs@dunnsolutions.com
www.dunnsolutions.com