Katherine Sanders – Software Engineer, IBM Hursley
18 March 2010

Leveraging The Web Services Interface Of The Communications Enabled Applications Feature Pack

Agenda

- CEA Feature Pack overview
- Business Scenarios
- Interfaces to the communications services
- Web services overview
- Sample Web services application using Rational Application Developer
- External Web services support
- Summary
- Further information / Questions?

IBM® WebSphere® Application Server V7 Feature Pack for Communications Enabled Applications (CEA feature pack)

"A ***communications enabled application (CEA)*** *is a set of information technology (IT) components and communication technology components that are integrated using a particular service-oriented architecture (SOA) to increase the productivity of an organization and/or improve the quality of users' experiences."* - Wikipedia

- Free extension available to WebSphere Application Server V7.0 customers
- Simplifies the addition of communications capabilities to new and existing applications
  - Developers do not need to know about the underlying communications protocols
  - No client-side installation required
- Uses an SOA and Java™ based programming model
- Improves customer support experience through new methods of interaction
- Lowers costs through reuse of existing Java skills, applications and telephony infrastructure

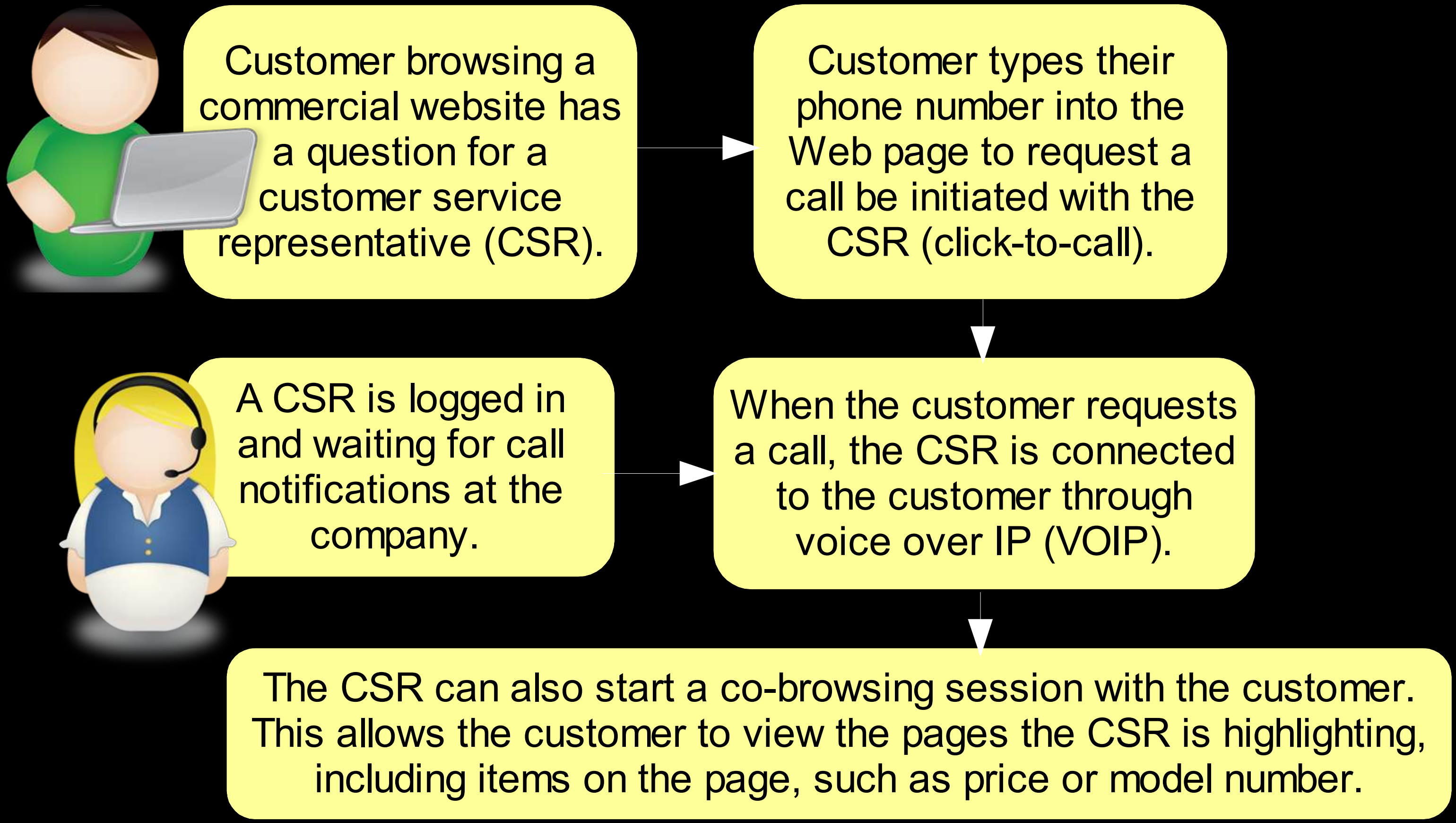
CEA feature pack version 1.0

- Shipped on top of WebSphere Application Server V7.0.0.5
- Enables multi-modal communications capabilities such as:
  - Click-to-call – customer requests a call from a company by entering their phone number on the company's website
  - Co-browsing – two Web users to share the same browsing session. One user controls the session; the other user has no control, but can view the activity of the other user.
  - Two-way synchronised forms – HTML forms in which two people can collaboratively edit and validate fields. Both parties can see the same form. The fields in the form change in response to input provided by either person.
- Utilises diverse developer skills with Web services and REST services-based APIs and Web 2.0 widgets
- Provides access to standards-based telephony infrastructure
- Supports the latest Session Initiation Protocol (SIP) Servlet 1.1 standard (JSR 289)
- Includes a unit test environment to prototype and test applications without the need to access the corporate telephony network

CEA feature pack version 1.0.0.1

- Shipped on top of WebSphere Application Server V7.0.0.7
- Adds clustering and failover support for Web collaboration and telephony services on distributed platforms
- Makes Web collaboration URIs more secure with a nonce to prevent session snooping
- Provides iWidget support for CEA widgets
  - iWidget is an IBM specification that defines a way to wrap Web content so that it can participate in a mashup environment.
- Widgets support new Dojo level 1.3.2 (previously 1.3.1)
  - This means base Dojo widgets will now support:
    - Internet Explorer V8
    - Firefox 3.5
    - Google Chrome 2

Business scenario 1: Click-to-call with co-browsing



Customer browsing a commercial website has a question for a customer service representative (CSR).

Customer types their phone number into the Web page to request a call be initiated with the CSR (click-to-call).

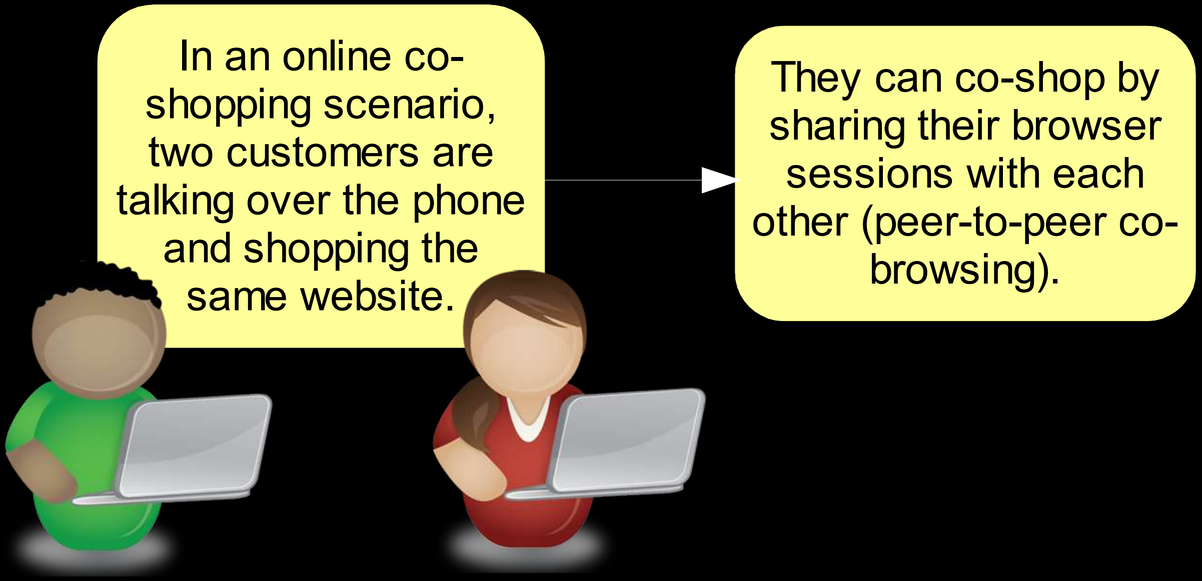
A CSR is logged in and waiting for call notifications at the company.

When the customer requests a call, the CSR is connected to the customer through voice over IP (VOIP).

The CSR can also start a co-browsing session with the customer. This allows the customer to view the pages the CSR is highlighting, including items on the page, such as price or model number.

- Registered customers, with preferences, might have additional features available.
  - E.g. The CSR could check the inventory of their preferred store location for the item
  - E.g. The customer can view the page in a different language than the CSR.

Business scenario 2: Shopping with a friend



In an online co-shopping scenario, two customers are talking over the phone and shopping the same website.

They can co-shop by sharing their browser sessions with each other (peer-to-peer co-browsing).

- Two-way collaboration is enabled, allowing each person to show pages to the other and highlighting items of interest.
- Each user, however, is able to maintain a separate session with their own shopping cart and preferences.

Business scenario 3: Tracking and reporting call statistics

A company wants to track the number of calls they receive in a day.

They have a business application that uses a Web service client to interact with the CEA Web service.

The application registers to be notified of incoming calls using the Web service client.

When calls are received, the telephone system notifies the CEA Web service, which sends a notification to the business application.

The business application receives the notification and increments a counter.

The data that the application gathers in its counter is logged and stored, to be used in a report that is generated at the end of the business day.

Interfaces to the communications services – REST

- Representational State Transfer (REST) is a network architecture that defines how resources on the Internet are accessed.
- Calls are similar to standard HTTP requests, except that they have a specific format that is recognised by the REST interface
- The CEA Web service has a servlet that is constantly listening for requests.
- The servlet parses each incoming request and returns a response to it using JSON or XML
- The REST APIs allow you to perform the following operations:
  - Make/End a call
  - Get call status
  - Register/Unregister for call notification
  - Get call notification information
  - Enable collaboration
  - Get collaboration status
  - Start/End a collaboration session with a peer
  - Send data to the collaboration peer
  - Retrieve event data (call status, collaboration status, collaboration data)

Interfaces to the communications services – widgets

- Dojo widgets are pre-packaged components of JavaScript and HTML code that add interactive features that work across platforms and browsers.
- The CEA feature pack comes with three ready-to-integrate widgets, and two extendable widgets that developers can customise to handle more advanced tasks.
- These widgets provide the following capabilities in Web applications:
  - Request phone calls (click-to-call)
  - Receive call notifications
  - Collaborate and Co-browse
  - Create and configure two-way forms
- The widgets can be embedded into any Web page by adding a JavaScript reference to the CEA Dojo toolkit and adding the tag definitions to the HTML files.
  - Just 2 imports and 1 line of HTML to add click-to-call on any Web page
- The widgets communicate with the back-end service using the REST APIs.
- The Plants By WebSphere sample application has been extended to include the click-to-call and call notification widgets for the feature pack as an example

Interfaces to the communications services – JSR 289

- The SIP Servlet Specification 1.1 (JSR 289) provides the Java API standards for Session Initiation Protocol (SIP)
- SIP is a signalling protocol used for creating, modifying, and terminating IP communication sessions such as telephony applications.
- SIP is not limited to voice communication and can mediate any kind of communication session, such as multimedia.
- SIP's rich media capabilities make Communications Enabled Applications (CEA) possible and extensible.
- The JSR 289 interface is for telephony applications that prefer to use the telephony APIs to build communication applications.

Interfaces to the communications services – Web services

- Web service calls can be made by business applications to access the communication system and:
  - Open a telephony session
  - Make a call
  - End a call
  - Close a telephony session
  - Get asynchronous call status updates using WS-Notification

Web services overview

- Web services enable interoperable machine-to-machine interaction over a network
- Typically Web service applications:
  - Communicate using XML messages that follow the SOAP standard over HTTP
  - Publish a machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL)
- Applications can be written in a variety of languages and environments
- The Java API for XML Web Services (JAX-WS) is a Java programming language API for creating Web services
  - Uses annotations to simplify development and deployment
  - Clients create a local proxy to represent a service, then invoke methods on the proxy
  - The JAX-WS run time converts API calls and matching responses to and from SOAP messages to shield the developer from that complexity
- The WS-Notification standard provides a standards-based framework through which Web service applications can participate in publish and subscribe messaging patterns

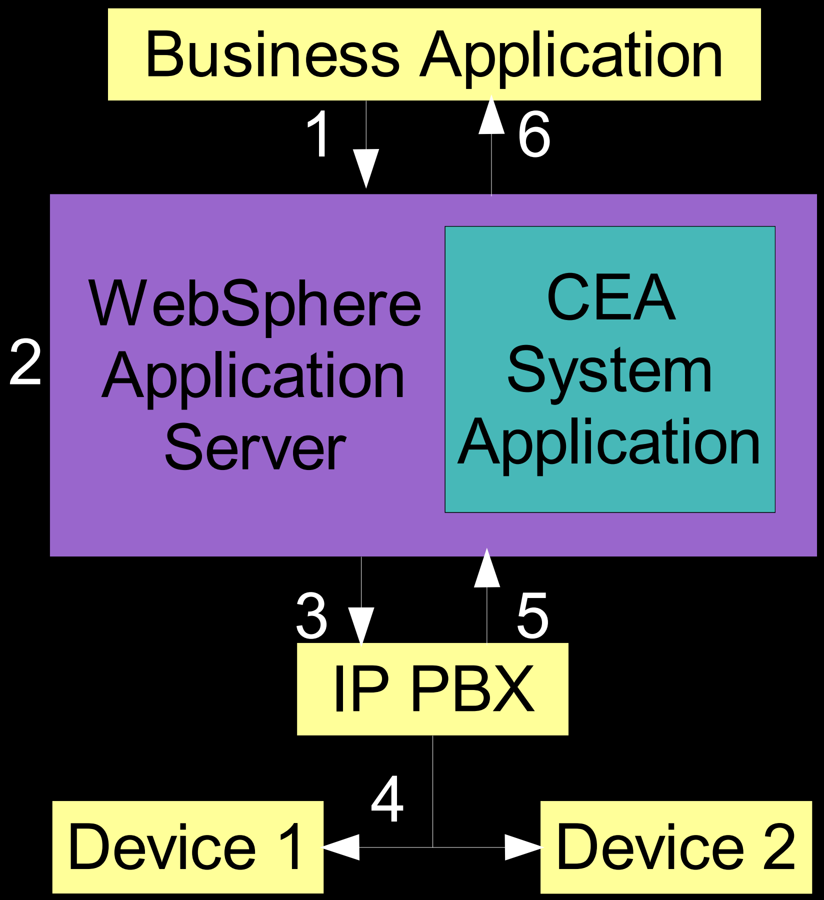
Web services tooling

- JAX-WS provides two main command line tools for developing Web services:
  - wsimport
    - Top-down development
    - Creates Java beans, service client, service endpoint interface, and wrappers from a provided WSDL
  - wsgen
    - Bottom-up development
    - Creates a WSDL document from Java code with the proper Web service annotations
- The Web services tooling in IBM Rational® Application Developer builds on the wsimport and wsgen commands and allows either a service client or a skeleton bean to be created using a wizard.

CEA feature pack WSDL documents

- ControllerService.wsdl
  - Contains the description of the operations offered by the CEA Web service
  - Used to generate the Web services client code needed to communicate with the CEA Web service
  - Generated Java classes include OpenSession, CloseSession, MakeCall, and EndCall.
  - Applications call the methods on the generated classes to manage telephone calls
- CeaNotificationConsumer.wsdl
  - Describes a WS-Notification consumer service
  - Used to generate a service implementation class
  - Implement the notify() method in that class to receive and process notification messages about changes to the call status

Web services call flow



1 The business application sends a call request to the CEA Web service

2 The WebSphere Application Server Web container routes the request to the CEA system application

3 The CEA system application sends the request to the Internet Protocol (IP) Private Branch Exchange (PBX) (business telephone system) in a SIP message

4 The IP PBX establishes a call between the devices

5 The IP PBX sends device events to the CEA system application

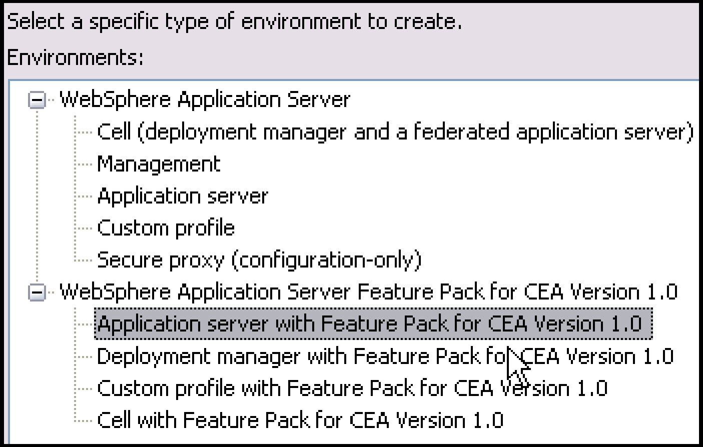
6 The CEA system application sends device events to the business application using WS-Notification

Sample application using Rational Application Developer

- This talk explains the steps necessary to develop the sample Web services application that is supplied with the CEA feature pack using Rational Application Developer 7.5
- To access telephony services with Web services:
  1 Create an application server profile for the CEA Feature Pack
  2 Obtain the WSDL and schema files for the CEA service from the application server
  3 Configure the application server to support the CEA Web service
  4 Install and configure the IP PBX
  5 Restart the server
  6 Develop and deploy the application

Create an application server profile for the CEA Feature Pack

- Create a new profile by selecting an environment under the WebSphere Application Server Feature Pack for CEA section in the Profile Management Tool:

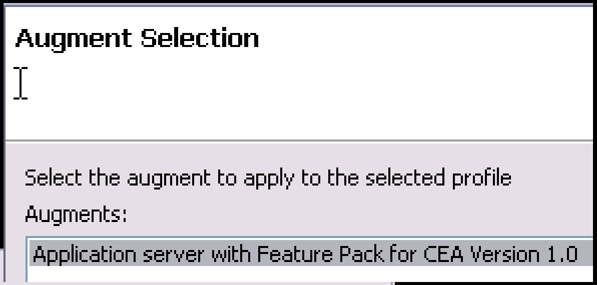


Select a specific type of environment to create.

Environments:

- WebSphere Application Server
  - Cell (deployment manager and a federated application server)
  - Management
  - Application server
  - Custom profile
  - Secure proxy (configuration-only)
- WebSphere Application Server Feature Pack for CEA Version 1.0
  - Application server with Feature Pack for CEA Version 1.0
  - Deployment manager with Feature Pack for CEA Version 1.0
  - Custom profile with Feature Pack for CEA Version 1.0
  - Cell with Feature Pack for CEA Version 1.0

- It is also possible to augment existing profiles that were not created for the CEA Feature Pack using the Profile Management Tool:

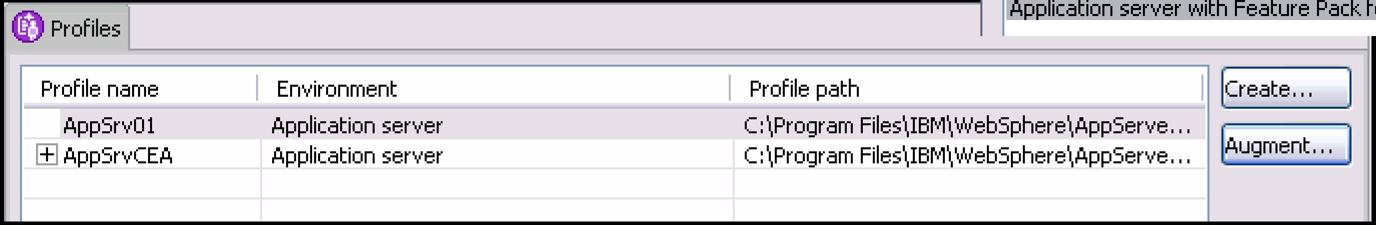


Augment Selection

Select the augment to apply to the selected profile

Augments:

Application server with Feature Pack for CEA Version 1.0



Profiles

Profile name	Environment	Profile path
AppSrv01	Application server	C:\Program Files\IBM\WebSphere\AppServe...
AppSrvCEA	Application server	C:\Program Files\IBM\WebSphere\AppServe...

Obtain the WSDL files and schema files (1)

1 Ensure the application server is started

2 Open the URL: http://host:port/commsvc.rest/ControllerService?wsdl

- In this URL, host is the IP address or host name on which the Web container is listening and port is the default http port.
- The WSDL file created during installation is read by the Web services infrastructure. The correct host and port are configured in the WSDL file sent to the browser.

```
<port binding="tns:ControllerPortBinding" name="ControllerPort">
  <soap:address location="http://localhost:9080/commsvc.rest/ControllerService"/>
</port>
```

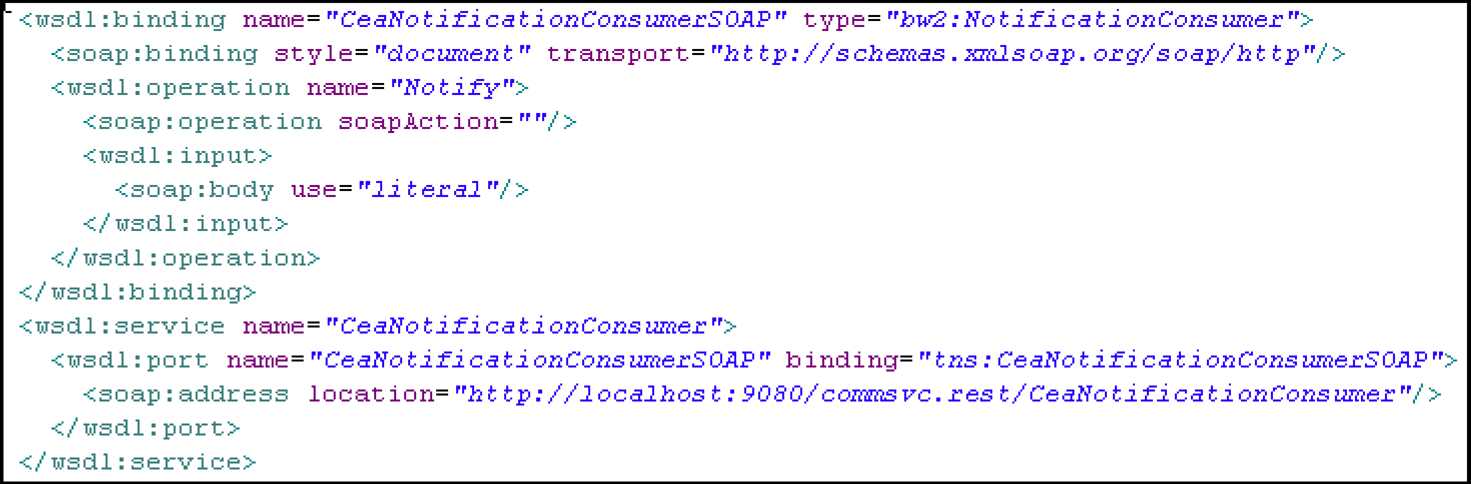
- The file contains the ControllerService service and the operations openSession, makeCall, endCall and closeSession.

```
<operation name="makeCall">
  <input message="tns:makeCall">
</input>
  <output message="tns:makeCallResponse">
</output>
  <fault message="tns:CTIControlException" name="CTIControlException">
</fault>
</operation>
```

3 Save the file as ControllerService.wsdl

Obtain the WSDL files and schema files (2)

4 Open the URL: http://host:port/commsvc.rest/CeaNotificationConsumer?wsdl
- The file contains the CeaNotificationConsumer service with the CeaNotificationConsumerSOAP binding to the Notify operation of the WS-Notification NotificationConsumer.



```
<wsdl:binding name="CeaNotificationConsumerSOAP" type="bw2:NotificationConsumer">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="Notify">
    <soap:operation soapAction=""/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="CeaNotificationConsumer">
  <wsdl:port name="CeaNotificationConsumerSOAP" binding="tns:CeaNotificationConsumerSOAP">
    <soap:address location="http://localhost:9080/commsvc.rest/CeaNotificationConsumer"/>
  </wsdl:port>
</wsdl:service>
```

5 Save the file as CeaNotificationConsumer.wsdl

6 Open the URL: http://host:port/commsvc.rest/ControllerService/WEB-INF/wsdl/ControllerService_schema1.xsd

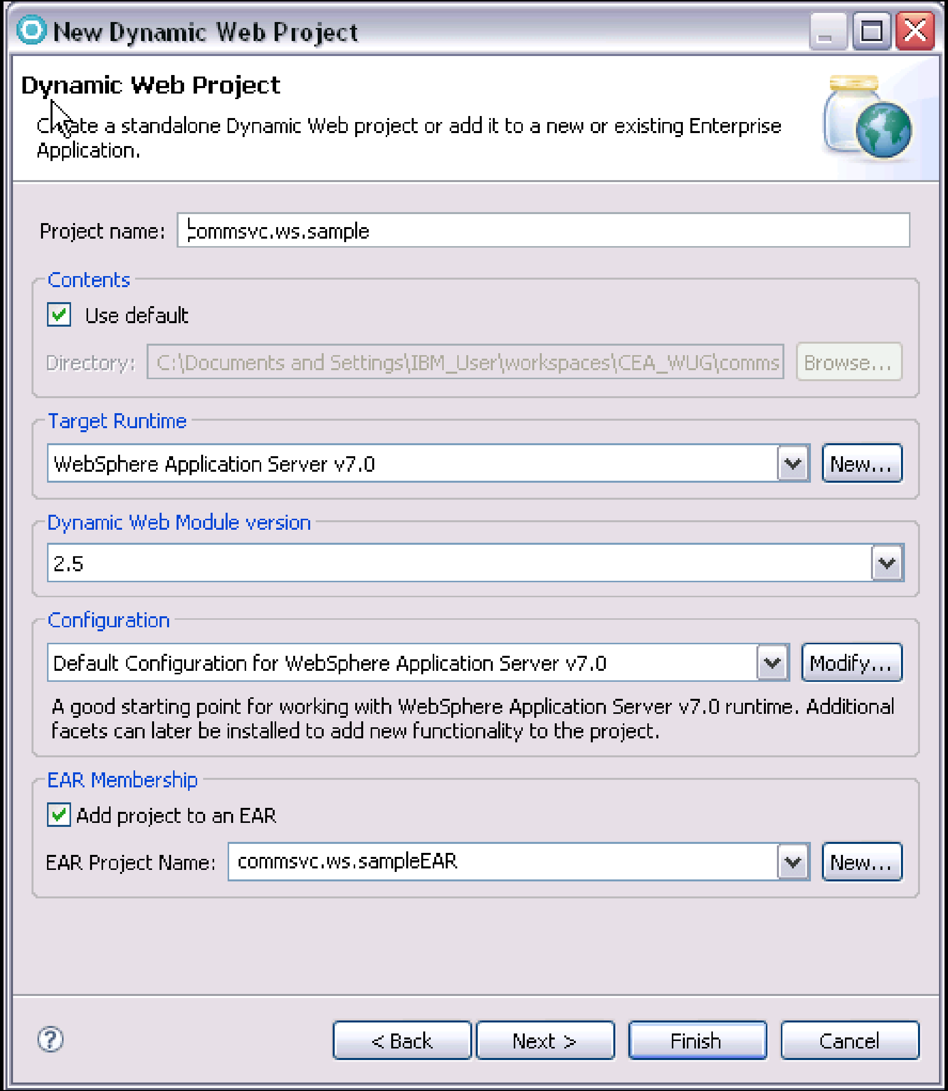
7 Save the file as ControllerServiceschema1.xsd

Create a new workspace and project

1 Open Rational Application Developer 7.5 with a new workspace

2 Create a new Dynamic Web Project called commsvc.ws.sample

3 Open the Java EE perspective


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: commsvc.ws.sample

Contents

✓ Use default

Directory: C:\Documents and Settings\IBM_User\workspaces\CEA_WUG\comms

Target Runtime

WebSphere Application Server v7.0

Dynamic Web Module version

2.5

Configuration

Default Configuration for WebSphere Application Server v7.0

A good starting point for working with WebSphere Application Server v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR Membership

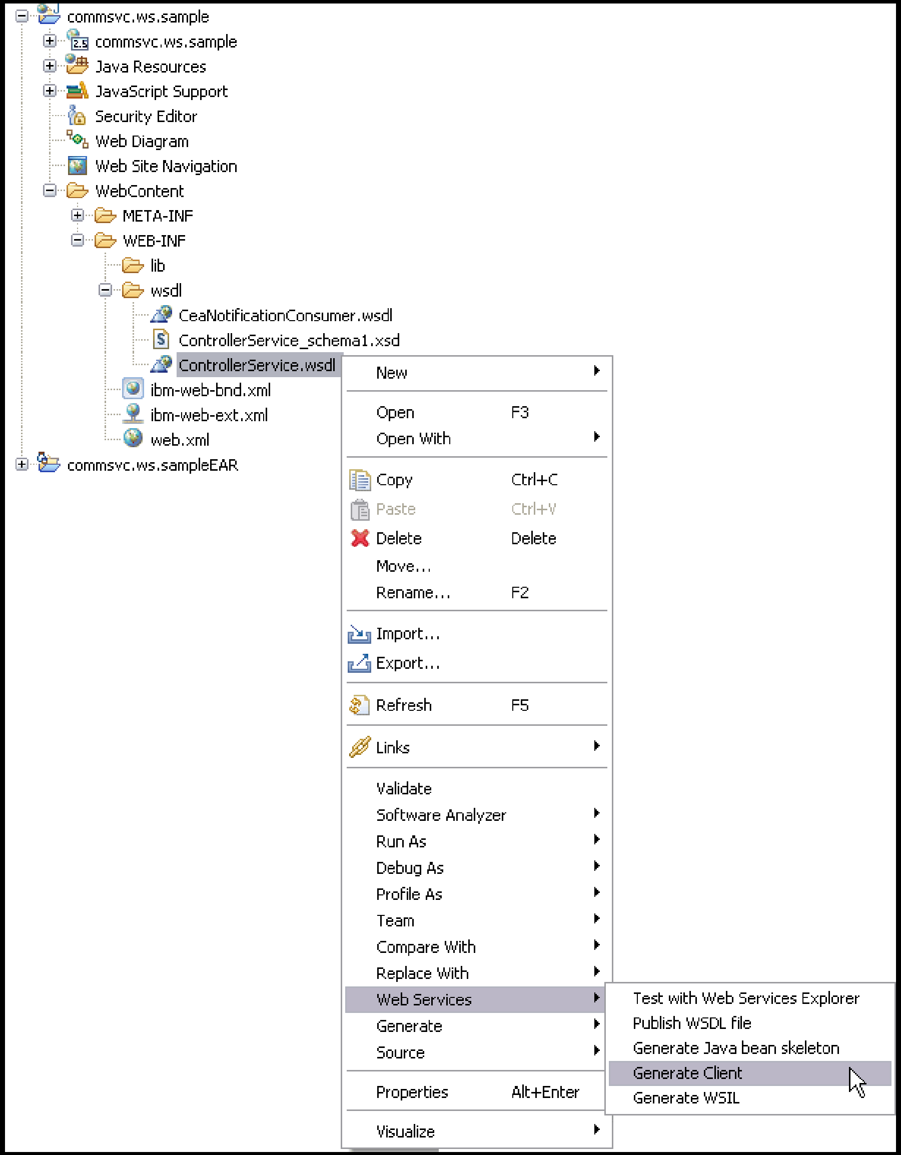
✓ Add project to an EAR

EAR Project Name: commsvc.ws.sampleEAR

Generate the Web services client code (1)

1 Create a new folder called wsdl under commsvc.ws.sample/WebContent/WEB-INF and import the WSDL and schema files into it.

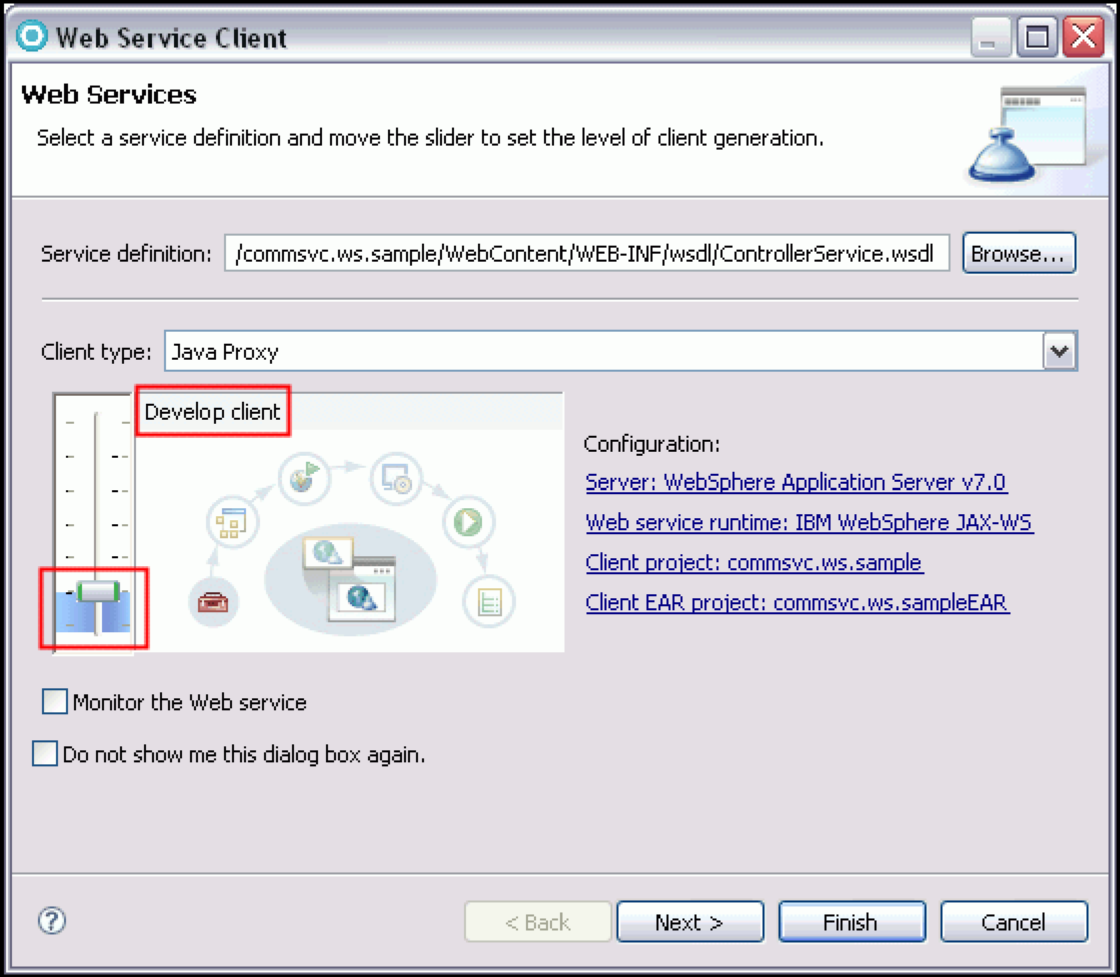
2 Right-click the ControllerService.wsdl file and select Web Services > Generate Client:



Generate the Web services client code (2)

3 On the Web Services page, set the configuration slider to Develop client
- This tells the wizard to create the WSDL definition and implementation of the Web service.
- This includes such tasks as creating the modules that contain the generated code, the WSDL files, the deployment descriptors, and the Java files when appropriate.

4 Click Finish.



Select a service definition and move the slider to set the level of client generation.

Service definition: /commsvc.ws.sample/WebContent/WEB-INF/wsdl/ControllerService.wsdl

Client type: Java Proxy

Configuration:
Server: WebSphere Application Server v7.0
Web service runtime: IBM WebSphere JAX-WS
Client project: commsvc.ws.sample
Client EAR project: commsvc.ws.sampleEAR

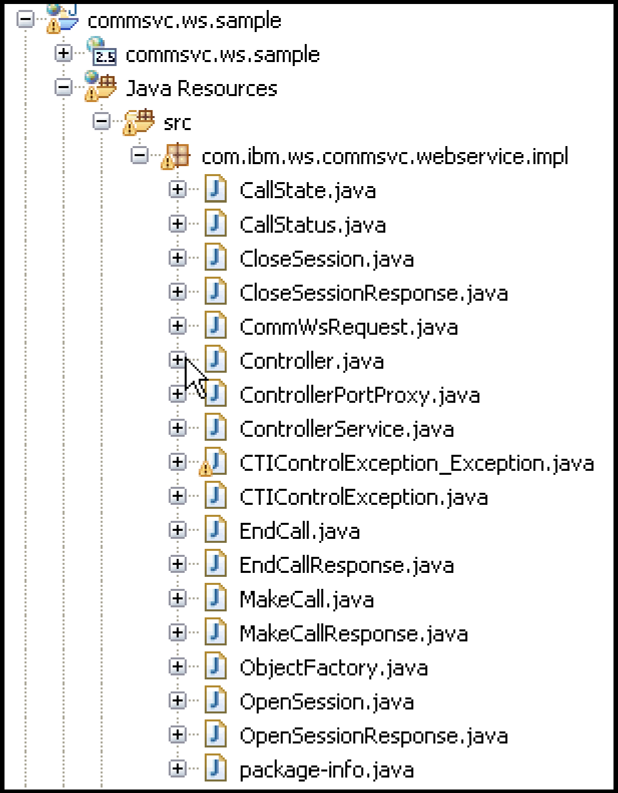
☐ Monitor the Web service

☐ Do not show me this dialog box again.

Generate the Web services client code (3)

5 Look through the newly generated files located in Java Resources/src/com.ibm.ws.commsvc.webservice.impl:

- Notice that classes have been generated for the OpenSession, CloseSession, MakeCall, and EndCall operations available on the CEA Web service

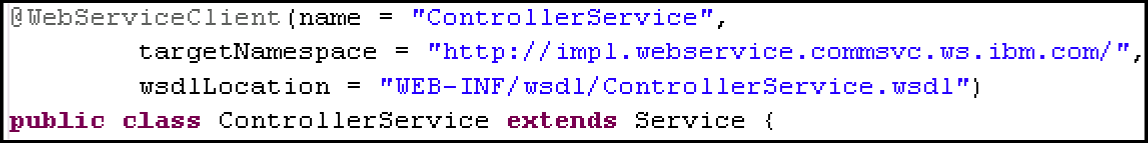


```
commsvc.ws.sample
  commsvc.ws.sample
  Java Resources
    src
      com.ibm.ws.commsvc.webservice.impl
        CallState.java
        CallStatus.java
        CloseSession.java
        CloseSessionResponse.java
        CommWsRequest.java
        Controller.java
        ControllerPortProxy.java
        ControllerService.java
        CTIControlException_Exception.java
        CTIControlException.java
        EndCall.java
        EndCallResponse.java
        MakeCall.java
        MakeCallResponse.java
        ObjectFactory.java
        OpenSession.java
        OpenSessionResponse.java
        package-info.java
```

Generate the Web services client code (4)

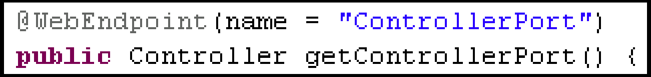
6 Open the ControllerService.java file to see the JAX-WS annotations:

- The @WebServiceClient is used to annotate a generated service interface. The information specified in this annotation is sufficient to uniquely identify a wsdl:service element inside a WSDL document. This wsdl:service element represents the Web service for which the generated service interface provides a client view:

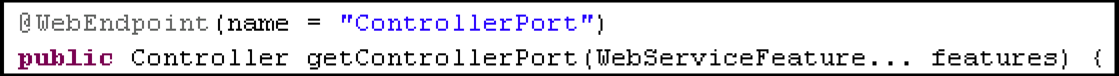


```
@WebServiceClient(name = "ControllerService",
        targetNamespace = "http://impl.webservice.commsvc.ws.ibm.com/",
        wsdlLocation = "WEB-INF/wsdl/ControllerService.wsdl")
public class ControllerService extends Service {
```

- The @WebEndpoint annotations are used to annotate the getPortName() methods of a generated service interface. The information specified in this annotation is sufficient to uniquely identify a wsdl:port element inside a wsdl:service:



```
@WebEndpoint(name = "ControllerPort")
public Controller getControllerPort() {
```



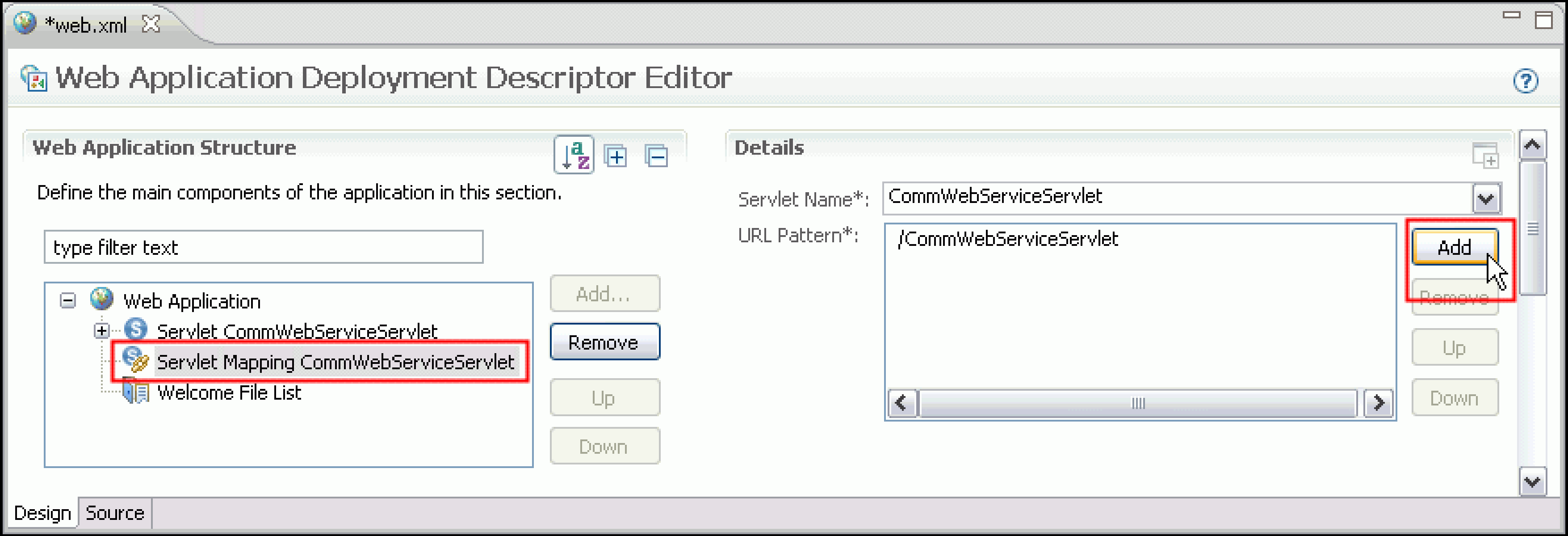
```
@WebEndpoint(name = "ControllerPort")
public Controller getControllerPort(WebServiceFeature... features) {
```

Create the front-end servlet (1)

1 Create a new servlet called CommWebServiceServlet in the commsvc.ws.sample project with the com.ibm.ws.commsvc.webservice Java package.

2 Open commsvc.ws.sample/commsvc.ws.sample/Servlets/CommWebServiceServlet.

3 In the Design view, on the left panel, select Servlet Mapping CommWebServiceServlet. In the Details section select Add.


*web.xml
Web Application Deployment Descriptor Editor
Web Application Structure
Details
Add...
Remove
Up
Down
Add
Remove
Up
Down
Design
Source

Define the main components of the application in this section.

type filter text

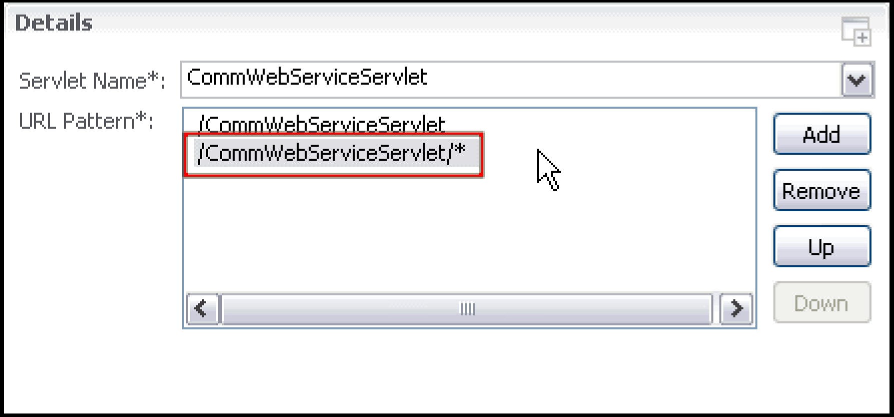
- Web Application
  - Servlet CommWebServiceServlet
  - Servlet Mapping CommWebServiceServlet
  - Welcome File List

Servlet Name*: CommWebServiceServlet

URL Pattern*: /CommWebServiceServlet

Create the front-end servlet (2)

4 Click the new item that shows in the URL Pattern box and enter /CommWebServiceServlet/*.



Servlet Name*: CommWebServiceServlet

URL Pattern*: /CommWebServiceServlet
/CommWebServiceServlet/*

5 On the left panel select Welcome File List and select Remove.

6 Save the file and close.

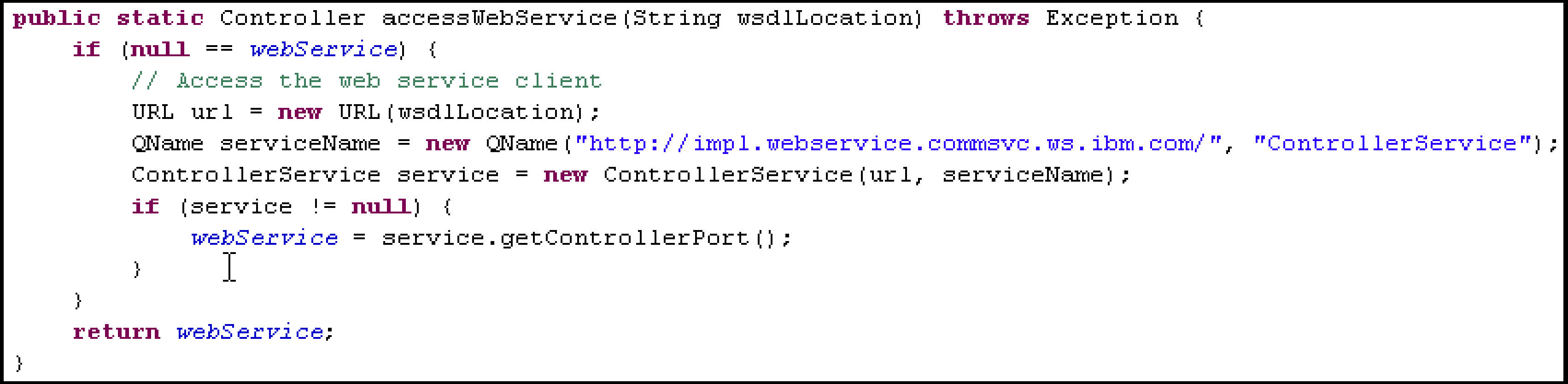
Create the front-end servlet (3)

7 Create two classes called CommWebServiceServlet.java and PhoneSession.java in commsvc.ws.sample\Java Resources\src\com.ibm.ws.commsvc.webservice and copy the sample code from the Redpaper referenced at the end of the talk.

8 CommWebServiceServlet.java outputs different HTML forms depending on the current call state. We will see screenshots of these forms later.

9 PhoneSession.java contains the code to control the state and actions taken related to a specific phone.

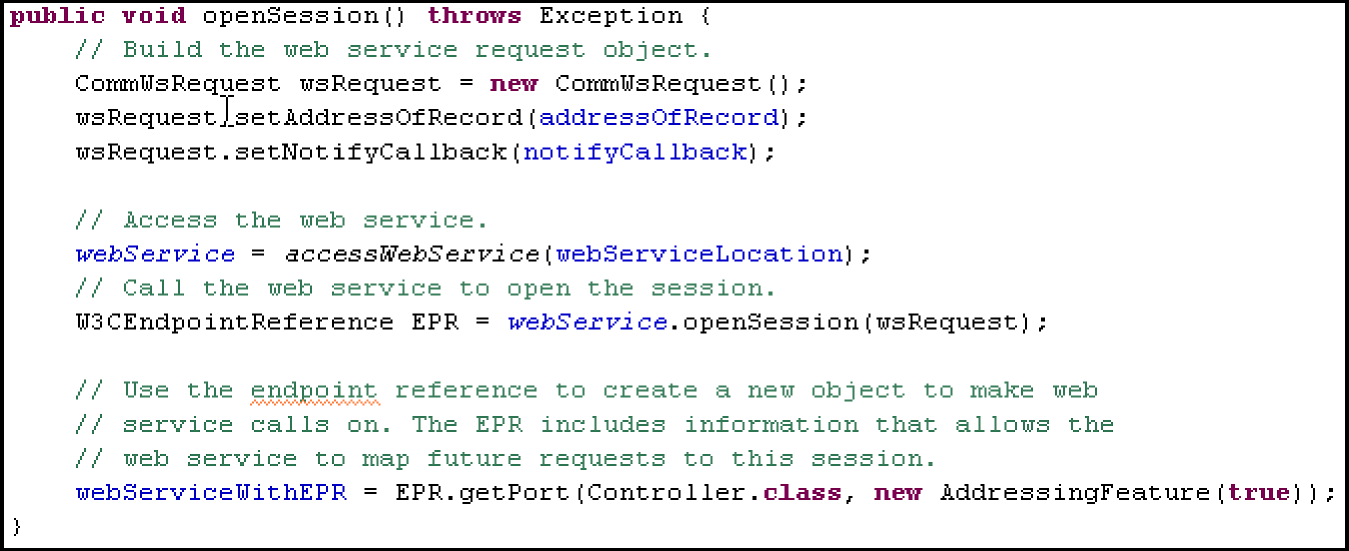
- The accessWebService() method gets access to the Web service client:



```
public static Controller accessWebService(String wsdlLocation) throws Exception {
    if (null == webService) {
        // Access the web service client
        URL url = new URL(wsdlLocation);
        QName serviceName = new QName("http://impl.webservice.commsvc.ws.ibm.com/", "ControllerService");
        ControllerService service = new ControllerService(url, serviceName);
        if (service != null) {
            webService = service.getControllerPort();
        }
    }
    return webService;
}
```

Create the front-end servlet (4)

- The openSession() method starts monitoring a telephone:

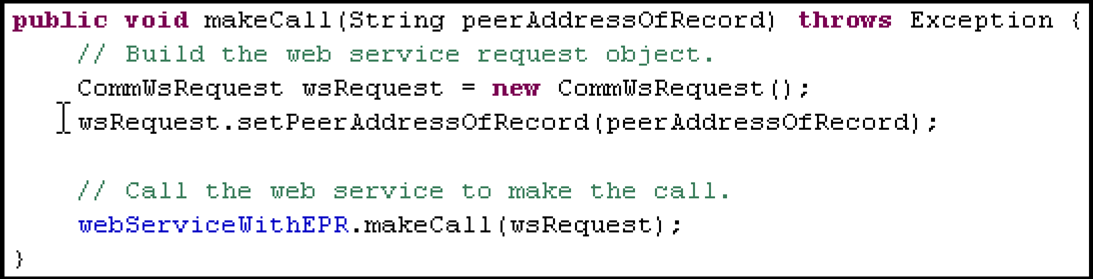


```
public void openSession() throws Exception {
    // Build the web service request object.
    CommWsRequest wsRequest = new CommWsRequest();
    wsRequest.setAddressOfRecord(addressOfRecord);
    wsRequest.setNotifyCallback(notifyCallback);

    // Access the web service.
    webService = accessWebService(webServiceLocation);
    // Call the web service to open the session.
    W3CEndpointReference EPR = webService.openSession(wsRequest);

    // Use the endpoint reference to create a new object to make web
    // service calls on. The EPR includes information that allows the
    // web service to map future requests to this session.
    webServiceWithEPR = EPR.getPort(Controller.class, new AddressingFeature(true));
}
```

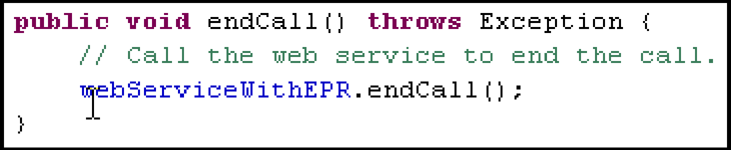
- The EPR must be used in all other API calls related to the session monitoring that phone. It has enough information for the Web service to track requests and to ensure follow-on requests go back to the same container in a clustered environment.
- Notice that notifyCallback is also set, this is the URL needed to contact to trigger a call notification (WS-Notification).

Create the front-end servlet (5)

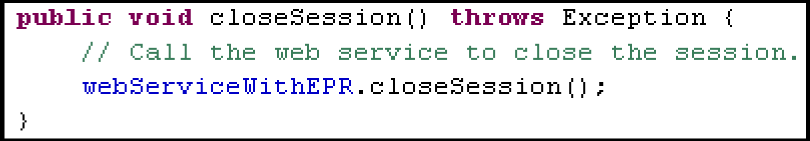
- There are also methods to make a call, end a call, and close a session. Note the use of the EPR in each of the methods:



```java
public void makeCall(String peerAddressOfRecord) throws Exception {
    // Build the web service request object.
    CommWsRequest wsRequest = new CommWsRequest();
    wsRequest.setPeerAddressOfRecord(peerAddressOfRecord);

    // Call the web service to make the call.
    webServiceWithEPR.makeCall(wsRequest);
}
```



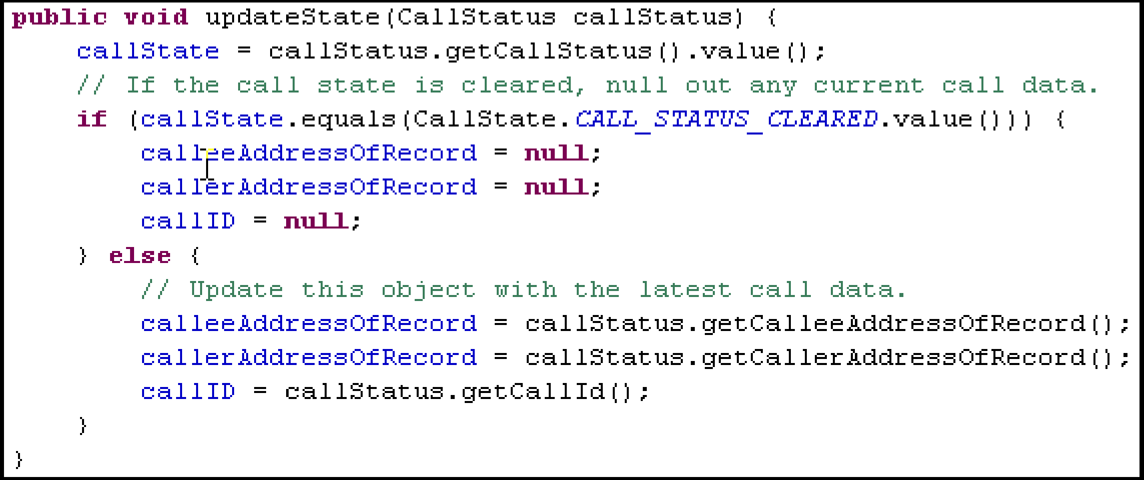
```java
public void endCall() throws Exception {
    // Call the web service to end the call.
    webServiceWithEPR.endCall();
}
```



```java
public void closeSession() throws Exception {
    // Call the web service to close the session.
    webServiceWithEPR.closeSession();
}
```

Create the front-end servlet (6)

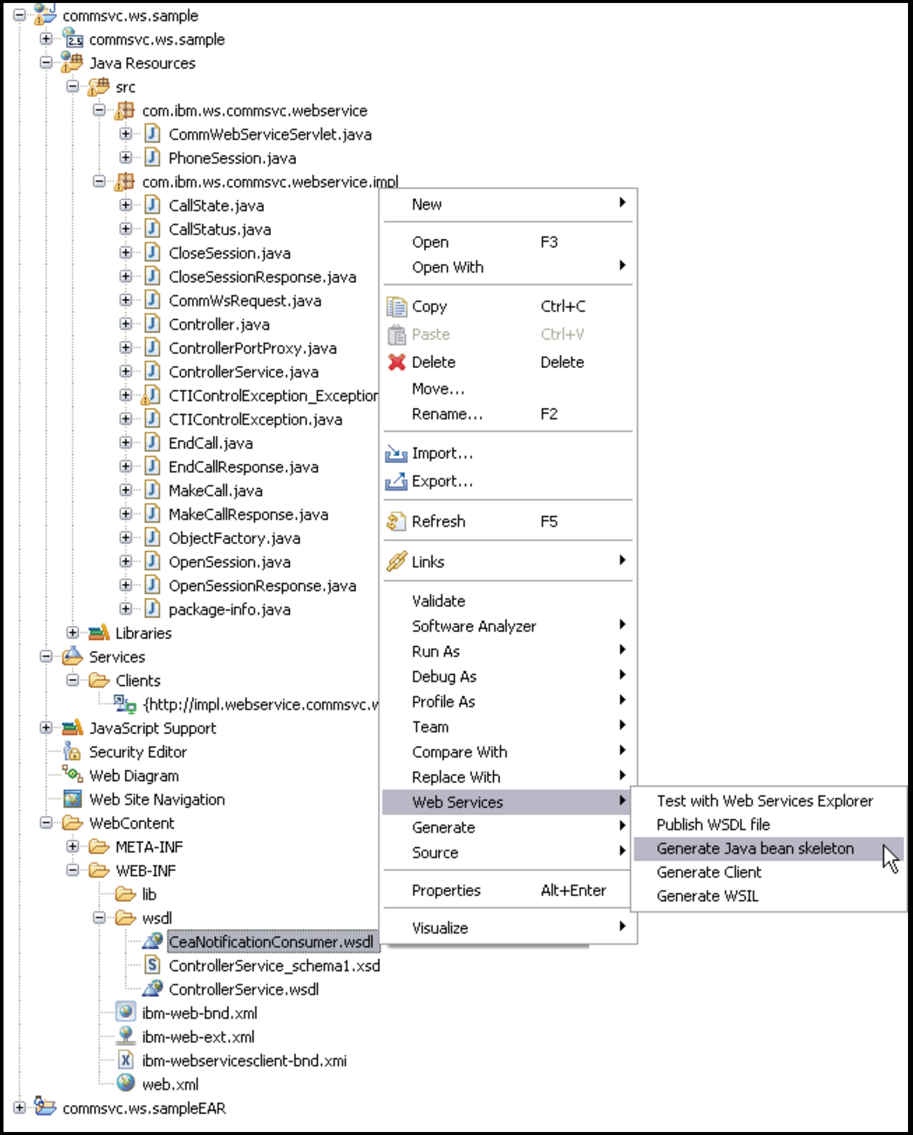
- The updateState() method will update the softphone session with the new call status information that arrived in a WS-Notification:



```
public void updateState(CallStatus callStatus) {
    callState = callStatus.getCallStatus().value();
    // If the call state is cleared, null out any current call data.
    if (callState.equals(CallState.CALL_STATUS_CLEARED.value())) {
        calleeAddressOfRecord = null;
        callerAddressOfRecord = null;
        callID = null;
    } else {
        // Update this object with the latest call data.
        calleeAddressOfRecord = callStatus.getCalleeAddressOfRecord();
        callerAddressOfRecord = callStatus.getCallerAddressOfRecord();
        callID = callStatus.getCallId();
    }
}
```

Generate the WS-Notification consumer service (1)

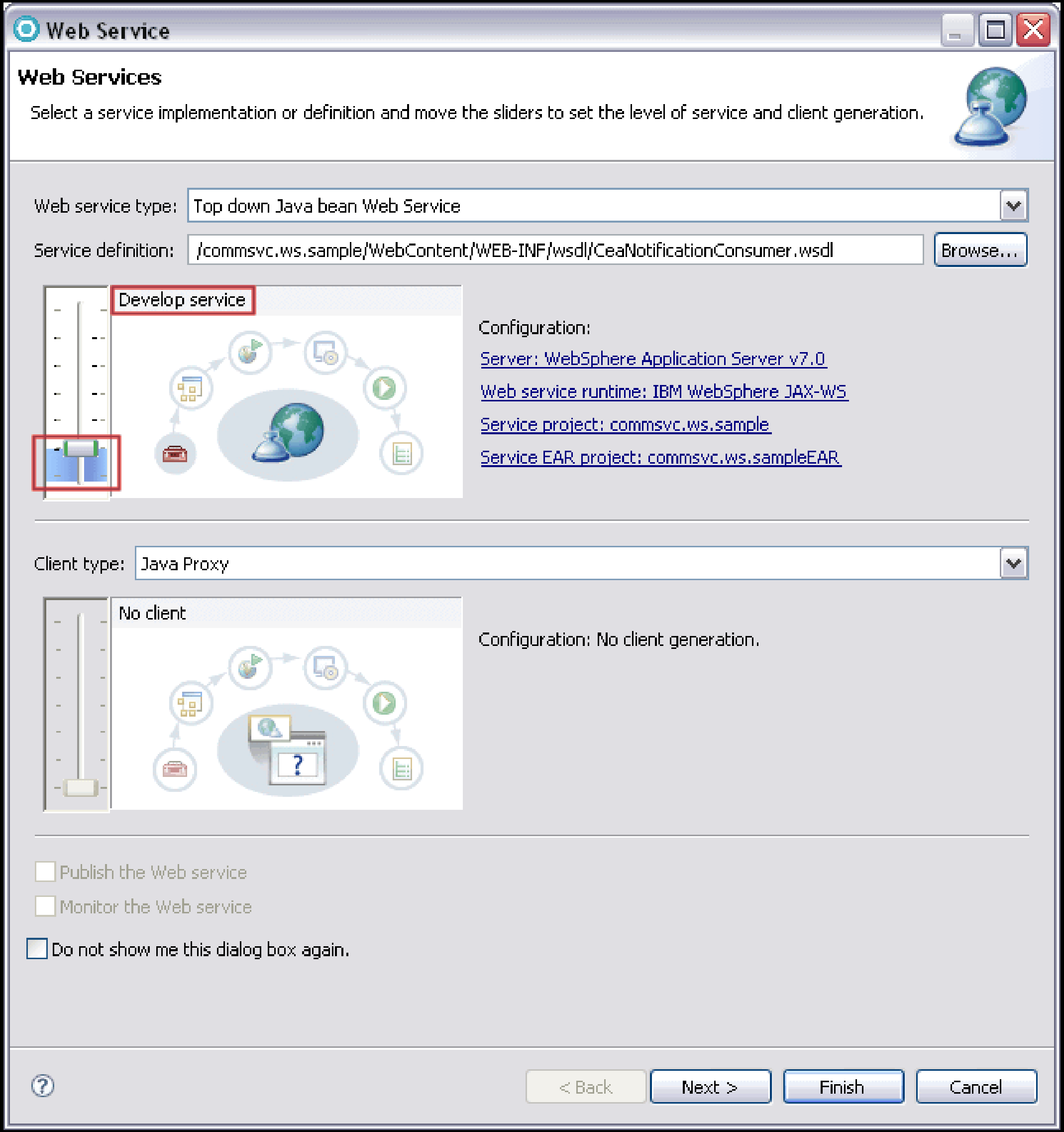
1 Right-click the CeaNotificationConsumer.wsdl file and select Web Services > Generate Java bean skeleton:



Generate the WS-Notification consumer service (2)

2 On the Web Services page set the configuration slider to Develop service

3 Click Finish



Web Service

Web Services

Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Top down Java bean Web Service

Service definition: /commsvc.ws.sample/WebContent/WEB-INF/wsdl/CeaNotificationConsumer.wsdl

Develop service

Configuration:
Server: WebSphere Application Server v7.0
Web service runtime: IBM WebSphere JAX-WS
Service project: commsvc.ws.sample
Service EAR project: commsvc.ws.sampleEAR

Client type: Java Proxy

No client

Configuration: No client generation.

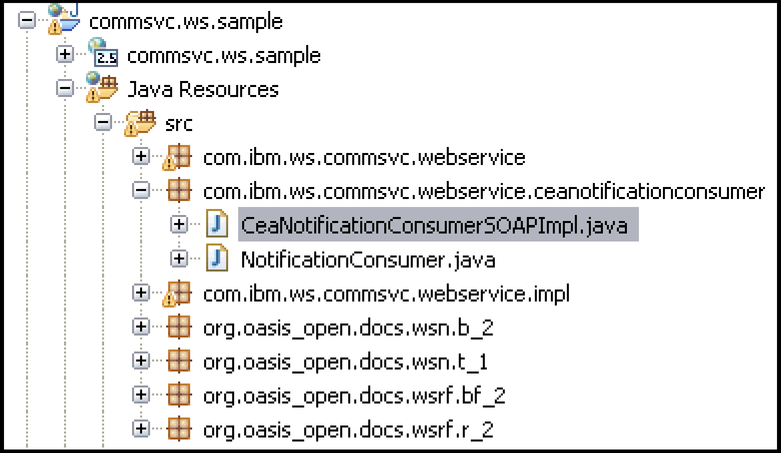
Publish the Web service

Monitor the Web service

Do not show me this dialog box again.

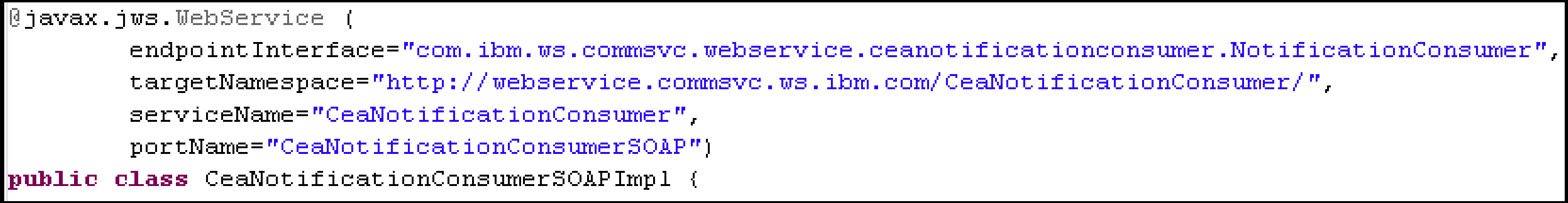
Generate the WS-Notification consumer service (3)

- The newly generated files include the Java service implementation class, CeaNotificationConsumerSOAPImpl.java, the service interface, NotificationConsumer.java, and the associated Oasis files:



```
commsvc.ws.sample
    commsvc.ws.sample
    Java Resources
        src
            com.ibm.ws.commsvc.webservice
            com.ibm.ws.commsvc.webservice.ceanotificationconsumer
                CeaNotificationConsumerSOAPImpl.java
                NotificationConsumer.java
            com.ibm.ws.commsvc.webservice.impl
            org.oasis_open.docs.wsn.b_2
            org.oasis_open.docs.wsn.t_1
            org.oasis_open.docs.wsrf.bf_2
            org.oasis_open.docs.wsrf.r_2
```

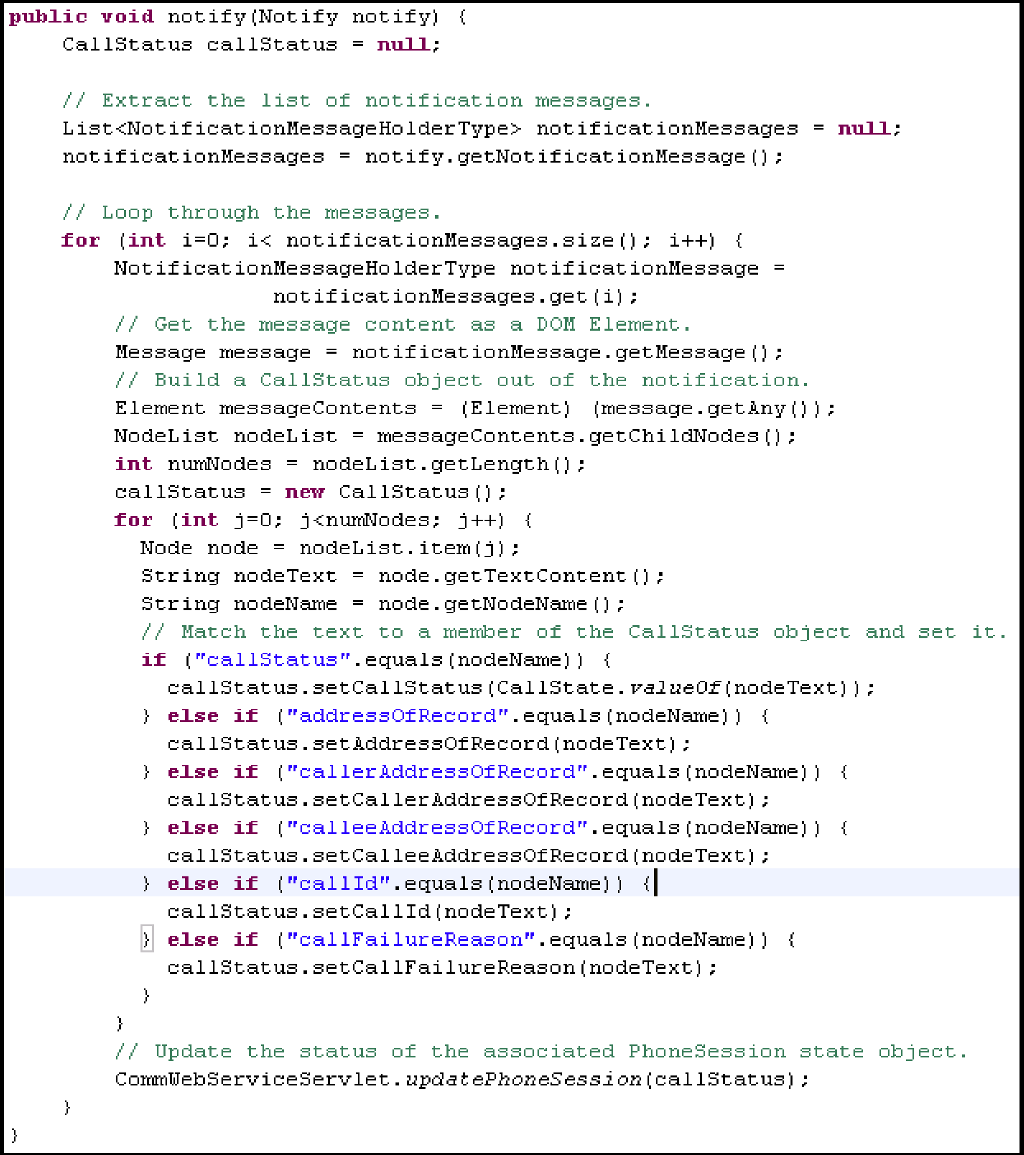
- Open the Java service implementation class, CeaNotificationConsumerSOAPImpl.java in Java Resources/src/com.ibm.ws.commsvc.webservice.ceanotificationconsumer. Notice the @WebService annotation. This is used to specify that the class is a Web service or that the interface defines a Web service:



```
@javax.jws.WebService (
        endpointInterface="com.ibm.ws.commsvc.webservice.ceanotificationconsumer.NotificationConsumer",
        targetNamespace="http://webservice.commsvc.ws.ibm.com/CeaNotificationConsumer/",
        serviceName="CeaNotificationConsumer",
        portName="CeaNotificationConsumerSOAP")
public class CeaNotificationConsumerSOAPImpl {
```

Generate the WS-Notification consumer service (4)

- Also notice that a notify() method is automatically generated.
  - This is the method that should be implemented to receive and process notification messages.
  - It is called automatically by the server's notification broker when a notification takes place.
- Replace the notify() method with this code. First it extracts the list of notification messages. Next, it loops through the messages and gets the message content as a DOM Element. Then it builds a CallStatus object out of the notification by looping through and matching the text to a member of the CallStatus object and setting it. Finally, it updates the status of the associated client state object.



```
public void notify(Notify notify) {
    CallStatus callStatus = null;

    // Extract the list of notification messages.
    List<NotificationMessageHolderType> notificationMessages = null;
    notificationMessages = notify.getNotificationMessage();

    // Loop through the messages.
    for (int i=0; i< notificationMessages.size(); i++) {
        NotificationMessageHolderType notificationMessage =
                    notificationMessages.get(i);
        // Get the message content as a DOM Element.
        Message message = notificationMessage.getMessage();
        // Build a CallStatus object out of the notification.
        Element messageContents = (Element) (message.getAny());
        NodeList nodeList = messageContents.getChildNodes();
        int numNodes = nodeList.getLength();
        callStatus = new CallStatus();
        for (int j=0; j<numNodes; j++) {
          Node node = nodeList.item(j);
          String nodeText = node.getTextContent();
          String nodeName = node.getNodeName();
          // Match the text to a member of the CallStatus object and set it.
          if ("callStatus".equals(nodeName)) {
            callStatus.setCallStatus(CallState.valueOf(nodeText));
          } else if ("addressOfRecord".equals(nodeName)) {
            callStatus.setAddressOfRecord(nodeText);
          } else if ("callerAddressOfRecord".equals(nodeName)) {
            callStatus.setCallerAddressOfRecord(nodeText);
          } else if ("calleeAddressOfRecord".equals(nodeName)) {
            callStatus.setCalleeAddressOfRecord(nodeText);
          } else if ("callId".equals(nodeName)) {
            callStatus.setCallId(nodeText);
          } else if ("callFailureReason".equals(nodeName)) {
            callStatus.setCallFailureReason(nodeText);
          }
        }
        // Update the status of the associated PhoneSession state object.
        CommWebServiceServlet.updatePhoneSession(callStatus);
    }
}
```

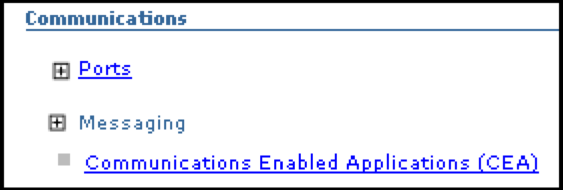
Configure the application server (1)

1 From Rational Application Developer, export the commsvc.ws.sampleEAR project as an EAR File. The EAR file will include the JAX-WS annotated classes, WSDL files, XSD schema, and client side code.

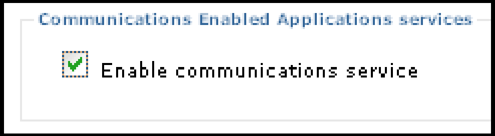
2 Ensure the application server is started.

3 Ensure the communications service is enabled using the WebSphere Application Server administrative console

– Navigate to Servers > Server Types > WebSphere application servers > <server_name> > select Communications Enabled Applications (CEA):



– Make sure the Enable communications service option is selected:


Communications Enabled Applications services

✓ Enable communications service

Configure the application server (2)

4 Run the script provided in the CEA installation to setup WS-Notification infrastructure in the base application server. The script creates a service integration bus, creates a WS-Notification service, creates a service point associated with the WS-Notification service, and starts the deployed service point enterprise application.
  - a Open a command prompt.
  - b Change directories to <WAS_HOME>\profiles\<CEA_PROFILE_NAME>\bin.
  - c Issue the following command:

```
wsadmin -f <WAS_HOME>\feature_packs\cea\scripts\CEA_WSN_JAXWS_Setup.py
```

  - d After the script has run successfully close the command prompt window.

5 Install the sample IP PBX (Applications > New application > New enterprise application > <WAS_HOME>\feature_packs\cea\samples\commsvc.pbx.ear > keep all defaults and Save)

6 Restart the application server

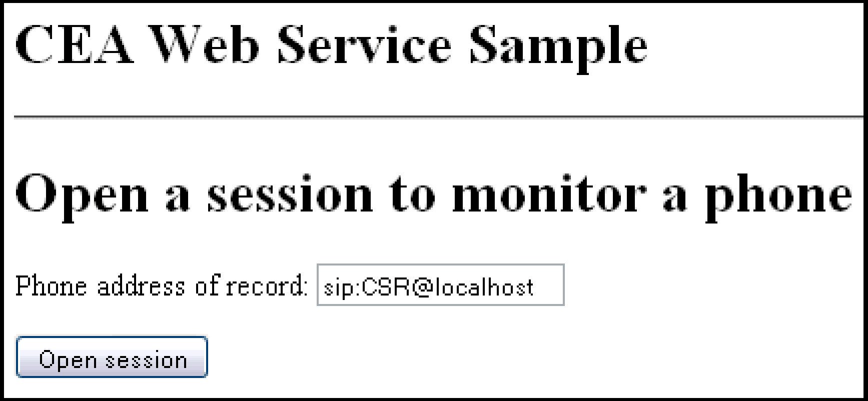
7 Install and start the application through the administrative console.

Test the application (1)

1 Start two softphones, for example: sip:CSR@localhost and sip:Customer@localhost.

2 Open a browser window and point it to: http://host:port/commsvc.ws.sample/CommWebServiceServlet
- The CEA Web Service Sample page displays. This page includes a form to Open a session to monitor a phone.
- In the Phone address of record text box, we'll enter the address of the softphone to be monitored. It must match the address of record that the softphone used to register with the PBX.

3 Enter sip:CSR@localhost in the “Phone address of record” text box.



CEA Web Service Sample

Open a session to monitor a phone

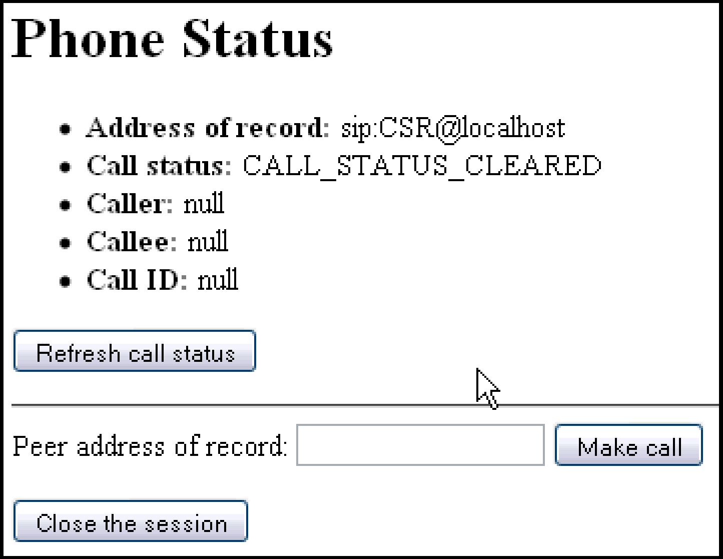
Phone address of record: sip:CSR@localhost

Test the application (2)

4 Click “Open session”.

- A request is sent to the sample's servlet, which calls the openSession Web service API.
- A “Phone Status” page is presented. Notice the address of record is set to what was entered on the first panel and that the call status is CALL_STATUS_CLEARED
- If a call is not active, the “Peer address of record” text box can be filled in with another phone's address. This address must also be registered with the PBX.
- Clicking “Make call”causes the monitored softphone to call the entered peer softphone with the makeCall Web service API.

5 Enter sip:Customer@localhost in the Peer address of record text box, and select Make a call.



Phone Status

- **Address of record:** sip:CSR@localhost
- **Call status:** CALL_STATUS_CLEARED
- **Caller:** null
- **Callee:** null
- **Call ID:** null

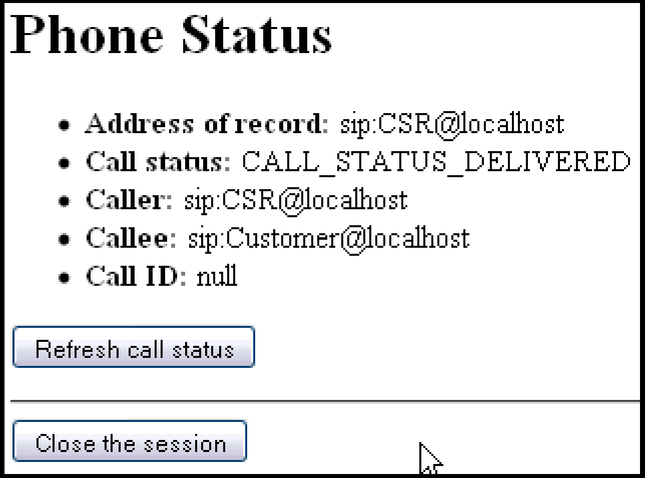
Test the application (3)

6 The CSR softphone rings. When the call gets delivered the status will update.

7 Answer both phones and select Refresh call status. This button fetches any status updates that the application servlet has received through WS-Notification. Notice how the status now states CALL_STATUS_ESTABLISHED.

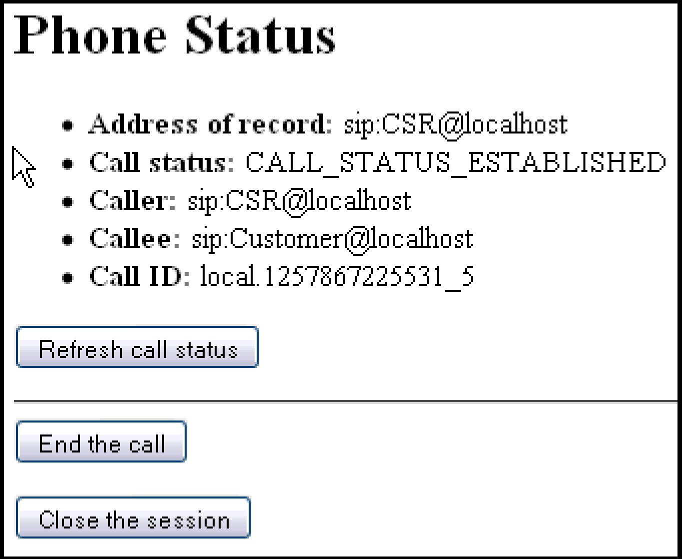
8 Select End the call. This triggers the endCall Web service API to end the call and the call status goes back to CALL_STATUS_CLEARED.

9 Select Close the session. This triggers the closeSession Web service API and terminates the monitoring session for the phone. The CEA Web Service Sample page displays again.



Phone Status

- **Address of record:** sip:CSR@localhost
- **Call status:** CALL_STATUS_DELIVERED
- **Caller:** sip:CSR@localhost
- **Callee:** sip:Customer@localhost
- **Call ID:** null



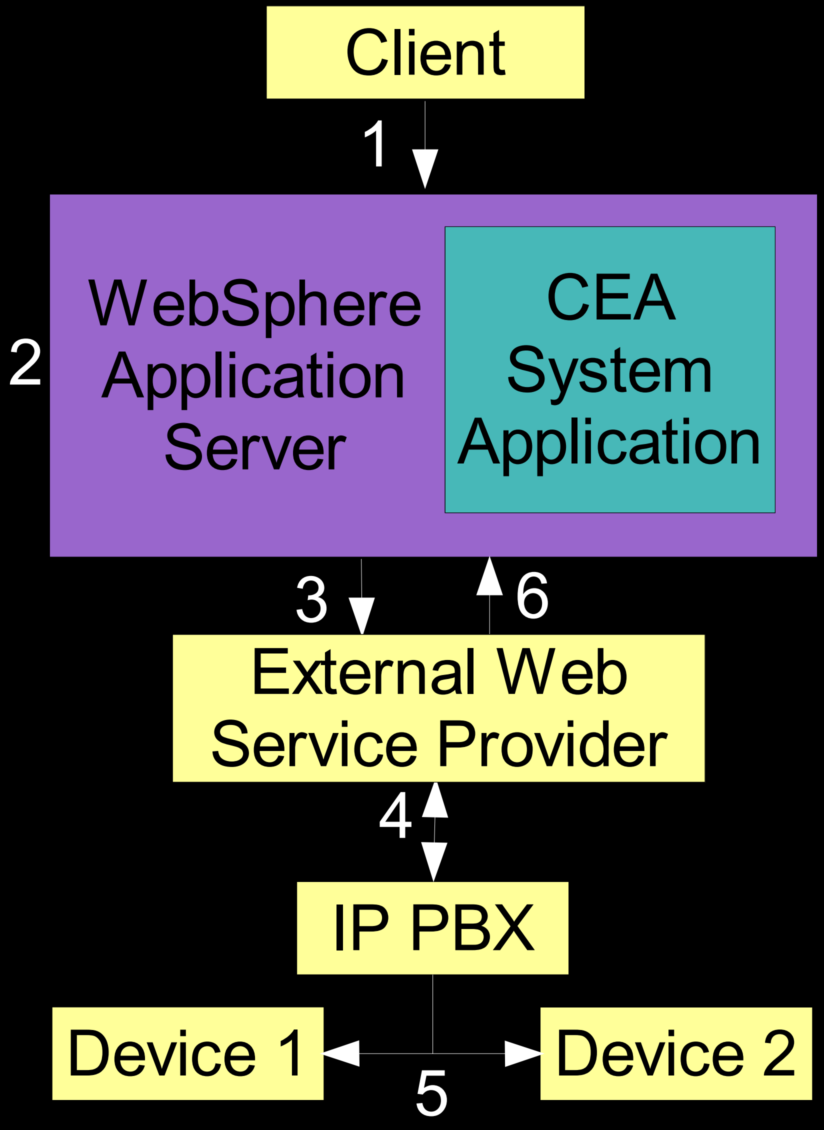
Phone Status

- **Address of record:** sip:CSR@localhost
- **Call status:** CALL_STATUS_ESTABLISHED
- **Caller:** sip:CSR@localhost
- **Callee:** sip:Customer@localhost
- **Call ID:** local.1257867225531_5

External Web services support

- When the CEA Web service is invoked, it interacts with an IP PBX to monitor and control phones.
- The CEA Web service interface is described by the ControllerService.wsdl file.
- If an external provider creates a Web service that implements the same WSDL, then CEA can be configured to use that provider instead of the CEA Web service.
- This allows vendors to customize interactions with their IP PBX.
- This configuration disables the existing CEA Web service, but the REST interface is still available.
- As REST requests are received, CEA uses a Web services client to communicate with the external Web service provider.
- The external Web service provider is responsible for all communication with the IP PBX to provide third party call control.
- The external Web service provider must be deployed and running on a server accessible from the CEA server and the location of the WSDL file for the external service must be known and accessible by using an HTTP request. An IP PBX must be started and configured as well.

External Web services call flow



1 The client sends an HTTP REST request.

2 The WebSphere Application Server Web container calls the CEA system application.
- The CEA servlet interprets the REST request.
- The CEA servlet uses a Web services client.

3 The CEA system application sends the Web service request to the external Web service provider.

4 The external Web service interacts with the IP PBX. Interaction can be proprietary.

5 The IP PBX establishes a call between devices.

6 The external Web service sends device events to CEA using WS-Notification.

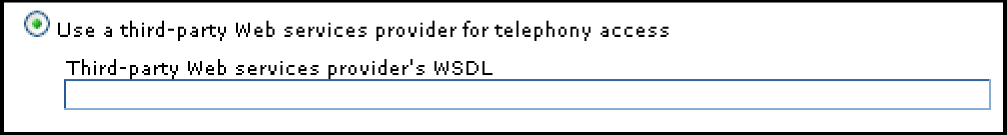
Implementation Overview

1 Enable the CEA system application in WebSphere Application Server.

2 Install and configure the IP PBX.

3 Install and configure the external Web service.

4 Configure the location of the third-party Web service WSDL:
- In the administrative console, click Servers > Server Types > WebSphere application
- servers > <server_name> > Communications Enabled Applications (CEA).
- Select the Use a third-party Web services provider for telephony access option and enter the HTTP URL of the third-party WSDL:


[✓ Use a third-party Web services provider for telephony access]

Third-party Web services provider's WSDL

- Save the settings and restart the server so that the new changes are applied to the run time.

5 Develop an application that calls the REST interface.

6 Install and start the new application.

Summary

- The CEA Feature Pack is a free extension available to WebSphere Application Server V7.0 customers that:
  - Simplifies the addition of communications capabilities to new and existing applications
  - Improves customer support experience through new methods of interaction such as click-to-call, co-browsing and synchronised two-way forms
  - Lowers costs through reuse of existing Java skills, applications and telephony infrastructure
- There are REST, Dojo widgets, JSR 289 and Web services interfaces available
- Web service calls can be made by application components to access the communication services to:
  - Open a telephony session
  - Make a call
  - End a call
  - Close a telephony session
  - Get asynchronous call status updates using WS-Notification
- It is possible to use an external Web service provider instead of the CEA Web service for proprietary interactions

Further information / Questions?

- Product information: http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/cea/
- V1.0.0.1 fixpack: http://www-01.ibm.com/support/docview.wss?uid=swg24024830
- Redpaper: http://www.redbooks.ibm.com/redpieces/abstracts/redp4613.html
- Information Center: http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.ceafep.multiplatform.doc/info/ae/ae/welcome_fepcea.html
- Blog: http://ibmcea.blogspot.com/
- YouTube: http://www.youtube.com/user/IBMcea

- Email: katherine_sanders@uk.ibm.com