IBM Software Group

IBM WebSphere Application Server Feature Pack for Communications Enabled Applications

Web services development

business on demand.

© 2009 IBM Corporation
Updated August 3, 2009

The IBM WebSphere Application Server Feature Pack for Communications Enabled Applications allows you to access telephony services with Web service clients. This presentation goes through the steps needed to create and deploy an application that can manage telephone calls using the Web services interface, including how to configure the application server. Formerly, this capability required building SIP servlets and a detailed understanding of the SIP specification. The functionality in this feature pack greatly reduces the amount of code required.

IBM

Agenda

- Manage telephone calls using a Web services client
- Configuration
- Development
- Web service sample
- Configuring external Web service providers to use the feature pack functionality

2

© 2009 IBM Corporation

This presentation covers managing telephone calls using the CEA Web service, configuring your server and system, and developing an application that uses the CEA Web service. It then covers the Web service sample that is included in the feature pack. The presentation ends with a brief overview of how to configure external Web service providers to use the functionality in the feature pack.

Section

Manage telephone calls using a Web services client

First is a brief overview of the CEA Web service and an overview of the steps needed to create and deploy an application that can manage telephone calls using the Web services interface.

CEA Web service

- IBM WebSphere Application Server Feature Pack for Communications Enabled Applications allows you to integrate telephony services into new and existing applications using its Web services interface

- The feature pack capability lets you:
  - ▸ Open a session to start monitoring a telephone
  - ▸ Get notifications about telephone activity
  - ▸ Make telephone calls between two phones
  - ▸ End an active telephone call
  - ▸ Close a session to stop monitoring a telephone

IBM WebSphere Application Server Feature Pack for Communications Enabled Applications allows you to integrate telephony services into new and existing applications using its Web services interface. This functionality lets you open a session to start monitoring a telephone and get notifications about telephone activity. You can also make telephone calls between two telephones, end an active telephone call, and close a session to stop monitoring a telephone.

CEA Web service

- The CEA Web service
  - ▸ Communicates over the HTTP protocol used on the Web
  - ▸ XML messages follow the SOAP standard
  - ▸ Description of operations offered by the service are written in Web Services Description Language (WSDL)

5

A Web service is designed to support interaction over a network and is frequently just Web application programming interfaces that can be accessed over a network, and run on a remote system hosting the requested services. A typical Web service application communicates over the HTTP protocol used on the Web. In the feature pack functionality, a Web service uses XML messages that follow the SOAP standard where there is a machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL). The WSDL file can be interpreted by Web service tools to generate the Web services client code needed to communicate with the Web service. As a result, an application developer need only call the correct set of Java APIs to manage telephone calls in an application.

Steps to accessing telephony services with WS

- Enable the CEA system application

- Configure your application server to support the CEA Web service

- Install IP-PBX

- Configure the IP-PBX location

- Restart your server

- Develop an application

- Install and start your application

6

Web services development   © 2009 IBM Corporation

In order to access telephony services with Web services you need to follow these steps. First, enable the CEA system application in your WebSphere Application Server. Second, configure your application server to support the CEA Web service. Third, install and configure your IP-PBX and restart your server. Then develop your application and install and start your application.

Section

Configuration

The next few slides will walk through the configuration steps that need to be done in order to access telephony services with Web services.

Enable CEA system application

- Update the configuration for each server running communications enabled applications
    - In the administrative console, click Servers > Server Types > WebSphere application servers > *server_name* > Communications Enabled Applications (CEA)
    - Ensure the check box labeled **Enable communications service** is checked

Communications Enabled Applications services

☑ Enable communications service

8
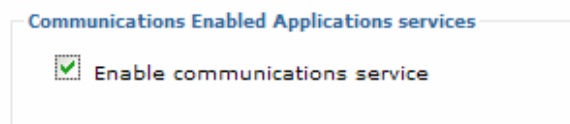
First, enable the CEA system application. For each server running communications enabled applications, update the configuration to ensure that communications service is enabled. In the administrative console, click Servers > Server Types > WebSphere application servers > *server_name* > Communications Enabled Applications (CEA). Ensure the check box labeled **Enable communications service** is checked as shown here.

Configure application server to support the CEA Web service

- Included in the feature pack is a script that applies these changes to the base application server configuration:
  - Creates a Service Integration Bus
  - Creates a WS-Notification service
  - Creates a service point associated with the previously-created service
  - Starts the deployed service point enterprise application

- Take the following actions to run this script:
  - Ensure the application server is started
  - Change directories to <CEA_Profile_Root>/bin
  - Issue the following command: wsadmin -f <WAS_HOME>/feature_packs/cea/scripts/CEA_WSN_JAXWS_Setup .py

9

Configure your application server to support the CEA Web service. The CEA Web service support is built on WS-Notification. Infrastructure in the base application server must be created to enable this support. To simplify this setup, a script is provided in the feature pack installation.

The script applies changes to the base application server configuration. It creates a Service Integration Bus, creates a WS-Notification service, creates a service point associated with the previously-created service, and starts the deployed service point enterprise application.

To run this script, first ensure the application server is started. Second, change directories to <CEA_Profile_Root>/bin. Finally, issue the wsadmin command shown here.

IP-PBX

- The CEA capability requires an IP private branch exchange (PBX) as part of your infrastructure

- A sample IP-PBX application is included in the application server installation
  - ▶ Install and start the SIP IP-PBX sample application
    - <WAS_HOME>/feature_packs/cea/samples/commsvc.pbx.ear

- Along with the sample IP-PBX, two soft phones are needed to test the application

10

The communications enabled applications capability requires an IP private branch exchange (PBX) as part of your infrastructure. An IP-PBX is a business telephone system designed to deliver voice over a data network and interoperate with the Public Switched Telephone Network (PSTN). A sample IP-PBX application is included in the application server installation. The sample IP-PBX is in the form of an application enterprise archive (EAR) file and is for test purposes only. The details of installing and configuring a vendor-specific IP-PBX are not provided. The IP-PBX must support the ECMA TR/87 protocol. Along with the sample IP-PBX, two soft phones are needed to test the application.

To set up a sample IP-PBX application that you can use for unit testing in the absence of an official PBX, start the application server where you deploy the sample IP-PBX application. Then install the SIP IP-PBX sample application. Then start the application.

Configure the IP-PBX location

- In the administrative console, click Servers > Server Types > WebSphere application servers > server_name > Communications Enabled Applications (CEA)
- Use the CEA settings page to select the Use SIP CTI (ECMA TR/87) gateway for telephony access option and configure the following fields:
  - ▸ Host name or IP address
  - ▸ Port
  - ▸ Protocol (TCP)
  - ▸ Superuser name
- Restart the application server

Telephony access method

⦿ Use SIP CTI (ECMA TR/87) gateway for telephony access

＊ Host name or IP address

`localhost`

＊ Port

`5060`

＊ Protocol

`TCP ▾`

☐ Extract user name from request

Superuser name

`ceauser`

After installing and starting the IP-PBX, you will then need to configure the IP-PBX location. In the administrative console for the server where the CEA system application is running, click Servers > Server Types > WebSphere application servers > *server_name* > Communications Enabled Applications (CEA). On the CEA settings page, select "Use SIP CTI (ECMA TR/87) gateway for telephony access" and configure the host name or IP address, port, protocol, and superuser name fields. Be sure to set the fields based on the server that is running the PBX application.

Use the host name field to provide the fully-qualified host name or IP address of the SIP CTI gateway that the CEA services connect to; the default is localhost. The port field specifies the port number of the SIP CTI gateway for connection communication services. For the TCP Protocol, the port is the SIP_DEFAULTHOST for the server that is running the PBX application; the default is 5060. If you are not using the default port then enter your port number here. In the protocol field, provide the protocol to use when connecting to the SIP CTI gateway; the default is TCP. The superuser name field specifies the name that is used when opening a new session to the gateway; the default is root. Be sure to restart the application server.

Section

Development

Now that you have your application server configured you can now integrate telephony services into new and existing applications.

IBM

Development process

- Obtain WSDL files and associated schema file

- Generate through JAX-WS a client using ControllerService.wsdl

- Write code to call methods against the Web service client

- Adding notification
  - ▶ Generate through JAX-WS a service implementation class using CeaNotificationConsumer.wsdl
  - ▶ Implement the notify method to receive and process notification messages

- The rest is handled by the runtime

Web services development | © 2009 IBM Corporation

The development process is outlined here. In communications enabled applications, a Web service uses XML messages that follow the SOAP standard where there is a machine-readable description of the operations offered by the service written in WSDL. The WSDL file can be interpreted by Web service tools to generate the Web services client code needed to communicate with the Web service. The feature pack includes two main machine-readable descriptions of the operations offered by the CEA Web service written in WSDL. You will first obtain the WSDL files ControllerService.wsdl and CeaNotification.wsdl and associated schema files that come with the feature pack. The ControllerService.wsdl file generates through JAX-WS the Web services client code needed to communicate with the Web service. Generated files include openSession, closeSession, makeCall, and endCall. As a result, you need only call the correct set of Java APIs to manage telephone calls in an application. Your code should include method calls to open a session, make a call, end a call, and close a session.

The CEA Web service support is built on WS-Notification. The CeaNotificationConsumer.wsdl follows WS-Notification allowing the CEA Web service to participate in publish and subscribe messaging patterns. The WSDL describes the consumer service. The CeaNotificationConsumer.wsdl file generates through JAX-WS a service implementation class, CeaNotificationConsumerSOAPImpl.java and a NotificationConsumer.java file. As a result, you just need to Implement the notify() method to receive and process notification messages, notifying you of the call status. The rest is handled by the runtime; neither the client code nor the provider code are required to write or parse SOAP messages.

Obtain the WSDL files

- Point your Web browser to
  - http://host:port/commsvc.rest/ControllerService?wsdl
  - http://host:port/commsvc.rest/CeaNotificationConsumer?wsdl
  - WSDL file created in the file system during installation is read by the Web services infrastructure

- Save the file and import it into your application

- Import associated schema file
  - http://host:port/commsvc.rest/ControllerService/WEB-INF/wsdl/ControllerService_schema1.xsd

The feature pack includes two main machine-readable descriptions of the operations offered by the CEA Web service written in WSDL. In order to obtain the WSDL files point your Web browser to the two URLs shown here.

In these URLs, "host" is the IP address or host name on which the Web container is listening, and "port" is the port number on which the Web container is listening.

When you do this, the WSDL file created in the file system during installation is read by the Web services infrastructure, and the correct host and port are configured in the WSDL file sent to the browser. Save the file and import it into your application. You will also need to import the associated schema file, which is the last URL listed on this slide.

ControllerService.wsdl
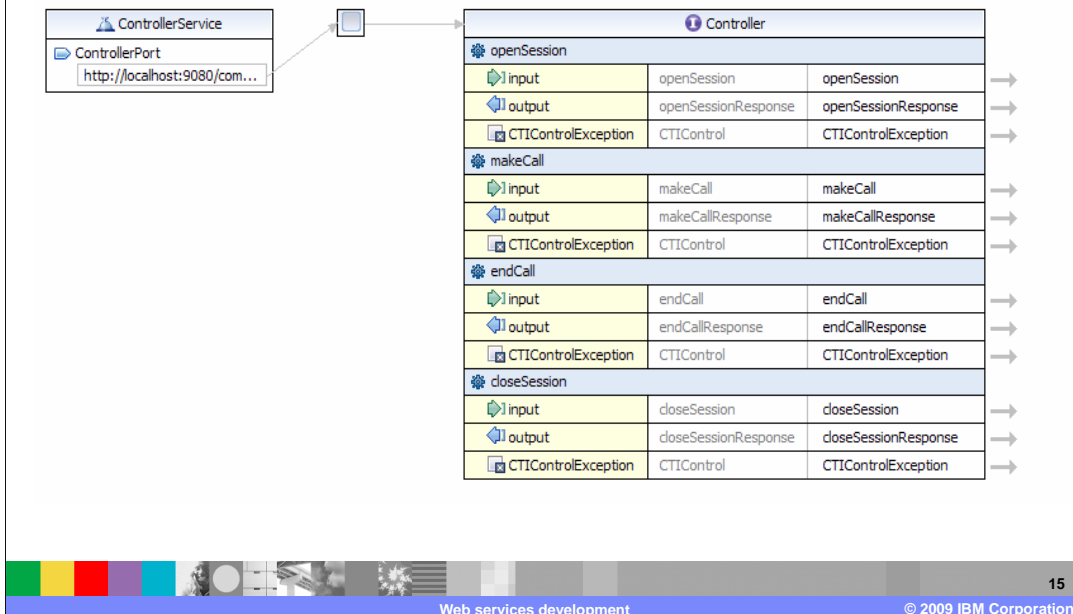
The ControllerService.wsdl file contains description of the operations offered by the service. Using Web service tools, you can generate the Web services client code needed to communicate with the Web service. Generated files include openSession, makeCall, endCall, and closeSession. On this slide is a visual display of the WSDL file.

CeaNotificationConsumer.wsdl
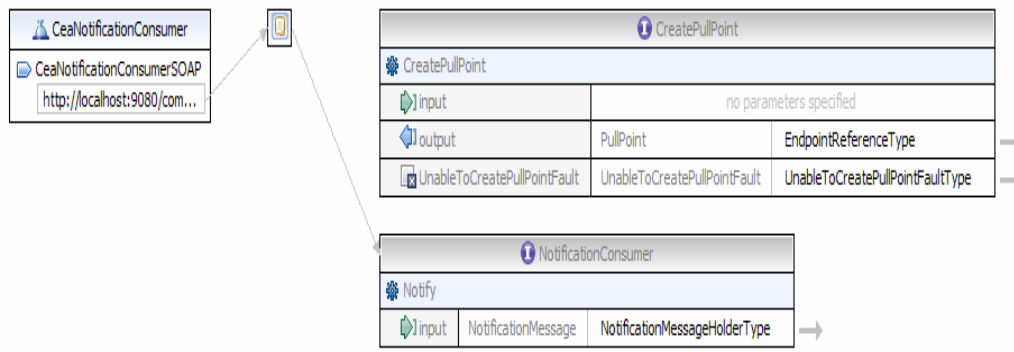
The CeaNotificationConsumer.wsdl follows WS-Notification, allowing the CEA Web service to participate in publish and subscribe messaging patterns. The WSDL describes the consumer service. The CeaNotificationConsumer.wsdl file generates through JAX-WS a service implementation class, CeaNotificationConsumerSOAPImpl.java and a NotificationConsumer.java file. As a result, you just need to Implement the notify() method to receive and process notification messages, notifying you of the call status. On this slide is a partial visual display of the WSDL file.

Generating Java artifacts

- JAX-WS tools support generating Java artifacts when starting with a WSDL file
  - ▶ Create a service client from a WSDL
  - ▶ Create a skeleton bean from a WSDL

- Rational Application Developer has the Web services tools that use the WebSphere JAX-WS runtime environment

- Command line tools available

The Java API for XML Web Services (JAX-WS) is a Java programming language API for creating Web services. JAX-WS uses annotations to simplify the development and deployment of Web service clients and endpoints. JAX-WS represents remote procedure calls or messages using XML-based protocols such as SOAP, but hides SOAP's innate complexity behind a Java-based API. Developers use this API to define methods, then code one or more classes to implement those methods and leave the communication details to the underlying JAX-WS API. Clients create a local proxy to represent a service, then invoke methods on the proxy. The JAX-WS runtime system converts API calls and matching replies to and from SOAP messages.

JAX-WS tools support generating Java artifacts when starting with a WSDL file. You can create a service client from a WSDL file. Web service clients are created from a WSDL document which describes where the Web service is deployed and what operations this service provides. You can also create a skeleton bean from a WSDL file. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.

Rational Application Developer has the Web services tools that use the WebSphere JAX-WS runtime environment. There are also command line tools available.

Rational Application Developer Web service wizard

- Switch to the Java EE perspective
- Right click your WSDL file
  - choose Web Services > Generate client or
  - Generate Java bean skeleton
- Select the stages of Web service client to Develop, select your server, runtime, and service project
- For Generate Java bean skeleton: select Top down Java bean Web service as your Web service type
- Files are generated

Using the WebSphere JAX-WS runtime environment Rational Application Developer has the Web services tools that allow you to either create a service client from a WSDL file or create a skeleton bean from a WSDL file. In order to use the Web service wizard in Rational Application Developer follow these steps. First switch to the Java EE perspective (Window > Open Perspective > Java EE). Import your WSDL file and associated schema file. Right click your WSDL file and choose Web Services > Generate client or Generate Java bean skeleton. For ControllerService.wsdl choose generate client and for CeaNotificationConsumer.wsdl choose generate Java bean skeleton.

On the Web services page select the stages of Web service development that you want to complete using the slider. This will set several default values on the remaining wizard panels. You will want to set the slider to develop, which will develop the WSDL definition and implementation of the Web service. This includes such tasks as creating the modules that will contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate. Select your server, runtime, and service project.

If creating a Java bean skeleton from a WSDL be sure to select Top down Java bean Web service as your Web service type. Top-down Web services development involves creating a Web service from a WSDL file. Go through the rest of wizard selecting other details and click Finish. Your Java files will then be generated.

Command line

- wsimport (top down) generates:
  - Service endpoint interface (SEI)
  - Service class
  - Exception class that is mapped from the wsdl:fault class (if any)
  - Java Architecture for XML Binding (JAXB) generated type values
- <WAS_HOME>\bin\wsimport -keep -b <WAS_HOME>\util\ibm-wsn-jaxb.xml -wsdllocation <WSDL_LOC> <WSDL_FILE>
  - Example:
    - C:\opt7\ibm\websphere\appserver\bin\wsimport -keep -b C:\opt7\ibm\websphere\appserver\util\ibm-wsn-jaxb.xml -wsdllocation "WEB-INF/wsdl/CeaNotificationConsumer.wsdl" CeaNotificationConsumer.wsdl

There are two main command line tools for working with JAX-WS to develop Web services. WSImport is a top down development tool that will create the necessary beans, service client, service endpoint interface, and wrappers from a provided WSDL file. The WSGen command will create a WSDL document and wrappers when needed from Java code with the proper Web service annotations.

Use wsimport, to process a WSDL file and generate portable Java artifacts that are used to create a Web service client. Using the wsimport tool you can create Service endpoint interface (SEI), and Service class. You also create the exception class that is mapped from the wsdl:fault class (if any), and Java Architecture for XML Binding (JAXB) generated type values which are Java classes mapped from XML schema types. Use the **-verbose** option to see a list of generated files when you run the command. Use the **-keep** option to keep generated Java files. Use the **-wsdlLocation** option to specify the location of the WSDL file. Use the **-b** option if you are using WSDL or schema customizations to specify external binding files that contain your customizations. You can customize the bindings in your WSDL file to enable asynchronous mappings or attachments.
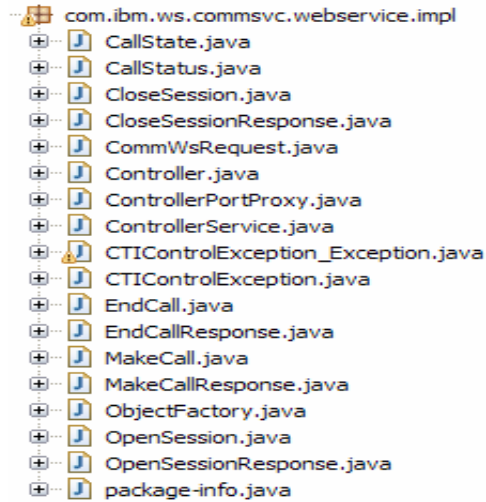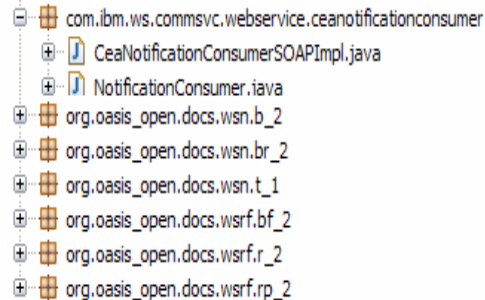
This picture shows the Web services client code that was generated from the ControllerService.wsdl file and the Java service implementation class and other oasis files that were generated from the CeaNotificationConsumer.wsdl file.

ControllerService.java

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

JAX-WS uses annotations to simplify the development and deployment of Web service clients and endpoints. Here is some of the code from the generated Java file ControllerService.java. This file was generated from ControllerService.wsdl. Notice the 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. The @WebEndpoint is used to annotate the get*PortName*() 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.

What to code

- Write client side code to call methods against the Web service client

- For notification
  - Implement the notify method to receive and process notification messages
    - Notify method is in CeaNotificationConsumerSOAPImpl.java
  - Add client side code to set the NotifyCallback in the Web service request object
  - Create a method to update the telephone session with call status information that arrived in a WS-Notification

Web services development   © 2009 IBM Corporation

After generating your Java code from the WSDL files you will write client side code to call methods against the Web service client. You will want to write client code to open a session, make a call, end a call, and close a session. For notification you will need to implement the notify method to receive and process notification messages. The notify() method is in the CeaNotificationConsumerSOAPImpl.java file. After coding the notify method you will want to add client side code to set the NotifyCallback in the Web service request object. You will also want to create a method to update the telephone session with call status information that arrived in a WS-Notification.

Sample code

```
public accessWebService () {
    // Access the Web services client
    controllerService = ControllerService();
    if (controllerService!= null) {
    controllerPort = controllerService.getControllerPort();
    } … }
// Open a session to monitor/control a phone
  public void openSession(String addressOfRecord, String notifyCallback) {
    // Build a request object
    CommWsRequest wsRequest = new CommWsRequest();
    wsRequest.setAddressOfRecord(addressOfRecord);
    wsRequest.setNotifyCallback(notifyCallback);
    W3CEndpointReference EPR = controllerPort.openSession(wsRequest);
    controllerPortWithEPR = EPR.getPort(Controller.class, new AddressFeature(true));
  }
```

Here is some sample client code. This code shows a sample accessWebService() method that gets access to the Web service client. Also, shown here is a sample openSession() method. openSession() is called in order to start monitoring a telephone. In this method you will first build the Web service request object, then access the Web service, and call the Web service to open the session. You will use the endpoint reference (EPR) to create a new object to make Web service calls on. The EPR includes information that allows the Web service to map requests to this session. The EPR must be used in all other APIs called related to the session monitoring that telephone. The EPR is critical for multiple reasons. First, it allows for Web service interface to be simpler eliminating the need to pass a state object as a parameter in all follow up requests. The EPR itself has enough information for the Web service to track it. It is also used in a clustered environment to ensure follow on requests go back to the same container monitoring the telephone.

Notice that notifyCallback is set. This is the URL needed to contact in order to trigger a call notification (WS-Notification). For the URL the host and port must be where the Web service client resides. The context root much match that of the WAR including this Web services client, and the name at the end must match the service name in the CeaNotificationConsumer.wsdl.

Sample code continued

```
// Make a call
public void makeCall(String calleeAddressOfRecord) {
  // Build a request object
    CommWsRequest wsRequest = new CommWsRequest();
    wsRequest.setPeerAddressOfrecord(calleeAddressOfRecord);
    // Make the call, using the EPR returned in openSession()
    controllerPortWithEPR.makeCall(wsRequest);
  }
  // End an active call
  public void endCall() {
    // End the call, using the EPR returned in openSession()
    controllerPortWithEPR.endCall(wsRequest);
  }
  // Close the session monitoring the phone
  public void closeSession() {
    // Close the session, using the EPR returned in openSession()
    controllerPortWithEPR.closeSession();
  }
```

24

© 2009 IBM Corporation

This is sample code to make a call, end a call, and close a session. In this code, the callee is the person being called. Note the use of the EPR in each of the methods.

Implement notify method

- Extract the list of notification messages

- Loop through the messages

- Get the message content as a DOM Element

- Build a CallStatus object out of the notification

- Loop through and match the text to a member of the CallStatus object and set it

- Update the status of the associated client state object

25

For notification you will need to implement the notify() method to receive and process notification messages. The notify() method is in the CeaNotificationConsumerSOAPImpl.java file. These steps list what a sample notify method should contain. First you extract the list of notification messages. Next you loop through the messages and get the message content as a DOM Element. You then build 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, you update the status of the associated client state object. The notify() method is called automatically by the server's notification broker when a notification takes place.

New application

- Bundle up your application that includes JAX-WS "annotated" classes, WSDL, XSD schema, and any client side code you created
- Install your sample
- Take the application-specific action that triggers the call to the Web service API
- In order to open a session and monitor a telephone for activity, you need to provide an address of record for your telephone
  - This can be a URI (uniform resource indicator) of a telephone
    - A SIP URI, for example, has the format of sip:username@serviceprovider, which represents the address of your telephone on the Internet
- Answer the source telephone when it rings
- Answer the destination telephone when it rings

After developing your application, bundle up your application that includes JAX-WS "annotated" classes, WSDL, XSD schema, and any client side code you created. Install your sample in your application server. Take the application-specific action that triggers the call to the Web service API. In order to open a session and monitor a telephone for activity, you need to provide an address of record for your telephone. This can be a URI (uniform resource indicator) of a telephone. A SIP URI, for example, has the format of sip:username@serviceprovider, which represents the address of your telephone on the Internet. Answer the source telephone when it rings and answer the destination telephone when it rings.

Section

Web service sample

The next two slides show the Web service sample that is included in the feature pack.

Web services sample

- Included in the feature pack is a Web service sample
  - Install the EAR file:
    <WAS_HOME>\feature_packs\cea\samples\webservice.sample\commsvc.ws.sample.ear
  - Visit:
    http://host:port/commsvc.ws.sample/CommWebServiceServlet

CEA Web Service Sample

Open a session to monitor a phone

Phone address of record: sip:Customer@localhos

[Open session]

The feature pack includes a Web services sample. You can view the code and see the generated Java files and sample client side code. In order to run the Web service sample, you will need to install the sample. The sample is included in the feature pack and located in the .ear file shown on this slide. After installing and starting the application, browse to the URL shown here. In order to open a session and monitor a telephone for activity, you need to provide an address of record for your telephone. This can be a URI of a telephone. Enter the URI value in the telephone address of record field and select Open session. Remember to complete the configuration steps earlier in the presentation before running this sample.

Making a call

1 **Phone Status**

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

Refresh call status

Peer address of record: sip:CSR@localhost Make call

Close the session

2 **Phone Status**

- **Address of record:** sip:Customer@localhost
- **Call status:** CALL_STATUS_DELIVERED
- **Caller:** sip:Customer@localhost
- **Callee:** sip:CSR@localhost
- **Call ID:** local.1245717055562_80

Refresh call status

Close the session

3 **Phone Status**

- **Address of record:** sip:Customer@localhost
- **Call status:** CALL_STATUS_ESTABLISHED
- **Caller:** sip:Customer@localhost
- **Callee:** sip:CSR@localhost
- **Call ID:** local.1245717055562_80

Refresh call status

End the call

Close the session

4 **Phone Status**

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

Refresh call status

Peer address of record: Make call

Close the session

Web services development   © 2009 IBM Corporation

29

Starting in the left corner you will see that there is not a call going on yet, but your address of record is filled in. Next enter a URI for the callee - the number that you want to call - and click "Make call". At the top right you will now see that the status is updated showing the caller, callee and the call status stating that the call was delivered. On the bottom left after both the phones are answered and after refreshing the status the call status will now state that the call is established. Finally, if you click "End the call," you will see on the bottom right that the call status is cleared.

Section

Configuring external Web service providers

This section shows the steps needed to configure an external Web service provider.

Using IP-PBX

- The feature pack for CEA provides Web telephony services in the form of REST APIs in addition to a Web service

- When invoked, a common core technology interacts with an IP-PBX to monitor and control phones
  - ▶ This core technology can be substituted with an external Web service to manage all communications with the IP-PBX

The feature pack provides Web telephony services in the form of REST APIs in addition to a Web service. When invoked, a common core technology interacts with an IP-PBX to monitor and control telephones. This core technology can be substituted with an external Web service to manage all communications with the IP-PBX.

External Web service provider

- If external provider creates a Web service that supports the CEA WSDL, application can be configured to use that provider
  - ▶ External Web service provider must be deployed and running on a server accessible from the application server
    - WSDL file for the external service must be known and accessible using an HTTP request
  - ▶ This configuration replaces the need for the existing CEA Web service, but the existing service can be used for REST requests

The CEA Web service interface is described by a WSDL file. If an external provider creates a Web service that supports this WSDL, then the communications enabled application can be configured to use that provider. To use an external Web service provider, it must be deployed and running on a server accessible from the application server. The location of the WSDL file for the external service must be known and accessible by using an HTTP request. Like the setup required when using the Web service provided by the feature pack, you must start and configure an IP private branch exchange (PBX) as well.

This configuration replaces the need for the existing CEA Web service, but the existing service can be used for REST requests. As REST requests are received, the application server uses a Web services client to communicate with the external Web service provider. The external Web service provider manages all communications with the IP-PBX.

Configure the external Web service

- Install and configure the external Web service
- 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)
  - ‣ Under Telephony access method, select the Use a third-party Web services provider for telephony access option
  - ‣ Enter the HTTP URL of the third-party WSDL
  - ‣ Save the settings and restart the server so that the new changes are applied to the run time

◉ Use a third-party Web services provider for telephony access

Third-party Web services provider's WSDL

Web services development   © 2009 IBM Corporation

You will need to install and configure the external Web service. For example, if the external Web service is delivered as an application that is deployed on WebSphere Application Server, you must install and configure the service on the local server. 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). Under Telephony access method, select "Use a third-party Web services provider for telephony access". Enter the URL of the third-party WSDL in the "Third-party Web services provider's WSDL" field. Save the settings and restart the server so that the new changes are applied to the run time.

Section

Summary and references

This section provides a summary and references.

IBM

Summary

- IBM WebSphere Application Server Feature Pack for Communications Enabled Applications lets you integrate telephony services into new and existing applications using the Web services interfaceObtain WSDL files and associated schema file
  - ▸ Generate client code using ControllerService.wsdl
  - ▸ Write code to call methods against the Web service client
  - ▸ Add Notification
- Web services sample
- Configure external Web service providers

35

Web services development

© 2009 IBM Corporation

This presentation talked about managing telephone calls using the CEA Web service, configuring your server and system, and developing an application that uses the CEA Web service. When developing an application, the main steps are to obtain the WSDL files and associated schema files, then generate the Java code. With the generated Java code you can just write client code to call methods against the Web service client. The CEA Web service is based on WS-Notification. After you generate the code needed for notification, you have to code the notify() method. The feature pack includes a Web service sample; be sure to view the code and run the sample. You can also use an external Web service provider to use communications enabled applications. The external Web service provider's role is to manage all communications with the IP-PBX.

References

- Web service
  - http://www.w3.org/TR/ws-arch/
- SOAP
  - http://www.w3.org/TR/soap/
- WSDL
  - http://www.w3.org/TR/wsdl
- WS-Notification
  - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
- JAX-WS
  - http://www.jcp.org/en/jsr/detail?id=224
- OASIS
  - http://www.oasis-open.org/

36

© 2009 IBM Corporation

This slide lists some useful references.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?

- Did it help you solve a problem or answer a question?

- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_CEAFP_WebServicesDevelopment.ppt

This module is also available in PDF format at: ../CEAFP_WebServicesDevelopment.pdf

37

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2009. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.