WebSphere. software

IBM

IBM Sales Center

With Computer Telephony Integration

- Integrate JTAPI technology into IBM Sales Center
- JTAPI and CTI examples
- IBM Sales Center and JTAPI use cases

Rufus Credle
Rajesh Adukkadukkath
Ramya Rajendiran
Ravindra Pratap Singh

Archived

ibm.com/redbooks

Redpaper

Archived

International Technical Support Organization

IBM Sales Center with Computer Telephony Integration

July 2007

Archived

REDP-4315-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

Archived

First Edition (July 2007)

This edition applies to Microsoft Windows XP, Microsoft Windows 2000 Server, Microsoft Windows 2003 Server, IBM Rational Application Developer V6.0.1.1, WebSphere Application Server Test Environment V6.0.2.5, IBM Sales Center for WebSphere Commerce V6.0, IBM WebSphere Commerce Developer V6.0, WebSphere Commerce Enterprise V6.0, DB2 Universal Database V8.2.3, IBM HTTP Server V6.0, WebSphere Application Server Network Deployment V6.0

© Copyright International Business Machines Corporation 2007. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices v
Trademarks vi

Preface vii
The team that wrote this paper vii
Become a published author viii
Comments welcome ix

Chapter 1. CTI overview 1

Chapter 2. Understanding Java Telephony Application Programming Interface 5

Chapter 3. JTAPI and CTI environment 7
3.1 Demonstration using the JTAPI-based sample code 8
3.1.1 Acquiring two telephone numbers from the CTI environment 8
3.1.2 Installing JTAPI-based sample code into the local machine 8
3.1.3 Running the sample code 9
3.2 JTAPI-based sample code 9
3.2.1 Receivecall.java 10
3.2.2 Receiver.java 12
3.2.3 Running the receivecall.java program 13

Chapter 4. How to integrate JTAPI into IBM Sales Center 17
4.1 The reason WebSphere MQ Everyplace must be used 18
4.2 WebSphere MQ Everyplace terms and concepts 18
4.3 Communication flow 19
4.4 Code snippets 20
4.5 Defining and configuring a Queue Manager 20
4.6 Creating a queue 21
4.7 Configuring and starting a Listener 22
4.8 Creating a connection 22
4.9 Putting and getting messages from the remote queue 23
4.10 Unconfiguring a Queue Manager 23
4.11 Enabling the Receiver to work with WebSphere MQ Everyplace 24
4.12 Processing messages from IBM Sales Center 24
4.13 Scope of the model used 26

Chapter 5. IBM Sales Center and JTAPI production environment 27
5.1 Use cases 28
5.1.1 The login use case 28
5.1.2 The non-monitored call use case 28
5.1.3 The direct monitored call use case 29
5.1.4 The forwarded call use case 29
5.1.5 The queued call use case 29
5.1.6 Queue call with Interactive Voice Response interaction use case 29

Appendix A. Additional material 31
Locating the Web material 31
Using the Web material 31
How to use the Web material 31

© Copyright IBM Corp. 2007. All rights reserved.

Related publications 33
IBM Redbooks 33
Online resources 33
How to get Redbooks 33
Help from IBM 33

Archived

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2007. All rights reserved.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

developerWorks®
eServer™
xSeries®
AFS®
BladeCenter®
Everyplace®
IBM®
Lotus Notes®
Lotus®
Notes®
Redbooks®
Redbooks (logo) ®
System x™
WebSphere®

The following terms are trademarks of other companies:

Java, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Archived

Preface

This IBM® Redpaper discusses how to integrate IBM Sales Center for WebSphere Commerce with a Computer Telephony Integration system that uses Java™ Telephony Application Programming Interface (JTAPI). This technology enables the Sales Center to identify a caller based on their phone number, and access customer-related information. This Redpaper also provides a demonstration using Cisco[1] JTAPI sample code and provides necessary information to design, develop, deploy, and customize applications.

This Redpaper is a derivative of the IBM Redbooks® publication, *Deploying and Customizing IBM Sales Center for WebSphere Commerce V6*, SG24-7249.

The team that wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Rufus Credle is a Certified Consulting IT Specialist at the ITSO, Raleigh Center. In his role as Project Leader, he conducts residencies and develops IBM Redbooks on network operating systems, enterprise resource planning (ERP) solutions, voice technology, high availability, and clustering solutions, Web application servers, pervasive computing, IBM and OEM e-business applications, IBM System x™, IBM eServer™ xSeries®, and IBM BladeCenter®. The various positions he has held during the course of his career at IBM for the past 27 years include assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He holds a bachelor's degree in Business Management from Saint Augustine's College.

Rajesh Adukkadukkath is a Staff Software Engineer in India Software Labs, Bangalore, India. He has six years of experience in software design and development of e-business, network management systems, and client-server technologies. He holds a master's degree in Computer Applications from Bharathiar University, Coimbatore, India. He has worked extensively on IBM Sales Center development for WebSphere® Commerce and his areas of expertise include Java and Java 2 Platform, Enterprise Edition (J2EE™) technologies, including plug-in development on the Eclipse framework.

Ramya Rajendiran is a Software Engineer in IBM India Software Labs, Bangalore. She has one year of experience in the software field and in WebSphere Commerce. She is currently working in WebSphere Commerce Content and Catalog Management. She holds a bachelor's degree in Engineering in Computer Science from the College Of Engineering, Guindy, Anna University.

Ravindra Pratap Singh is a Software Engineer with IBM India. He has over three years of experience in the WebSphere Commerce field. He holds a master's degree of Computer Applications from Jawaharlal Nehru University, New Delhi, India. His area of expertise is WebSphere Commerce Analyzer. He has written several articles and tutorials for IBM developerWorks® about WebSphere Commerce and WebSphere Commerce Analyzer.

[1] These materials have been reproduced by IBM, Inc. with the permission of Cisco Systems Inc. COPYRIGHT © 2007 CISCO SYSTEMS, INC. ALL RIGHTS RESERVED.

© Copyright IBM Corp. 2007. All rights reserved.

This Redpaper was derived from the Redbooks residency that produced *Deploying and Customizing IBM Sales Center for WebSphere Commerce V6*, SG24-7249. The residency team included Amit Jain, Lorilee Jarosinski, Mojca Spazzapan, and Dagmara Ulanowski.

Thanks to the following people for their contributions to this project:
Carolyn Sneed, Tamikia Barrow
ITSO, Poughkeepsie Center

Brian Nolan, IT Architect, WebSphere Business Integration Services Planning
IBM Research Triangle Park

Bill MacIver, WebSphere Commerce Suite Sr Development Manager
IBM Markham, Canada

Carl Kaplan, Worldwide e-Commerce Sales
IBM Waltham

Anthony Tjong, Manager, WebSphere Commerce Development
IBM Markham, Canada

Michael Au, Manager, WebSphere Commerce Foundation Development
IBM Markham, Canada

Peter Swithinbank, ITSO Project Leader
IBM Hursley, UK

Andy Kovacs, Support, Quality, and Measurements
IBM Markham, Canada

Tack Tong, Markham Lab
IBM Markham, Canada

Judy Chan, WebSphere Commerce Business-to-Business Solutions
IBM Markham, Canada

Brian Thomson, STSM Performance, Scalability, Availability
IBM Markham, Canada

Glenn Jones, SWG VoIP Infrastructure, Backup AFS® Cell Admin
IBM Markham, Canada

Wai-Kong Ho, Senior IT Specialist
IBM, Australia

Jegathasan Thambipillai, End User Support
IBM Toronto

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Archived

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Archived

Archived

1

CTI overview

CTI is a technology that allows the integration of interactions on a telephone and a computer. CTI includes the integration of all customer contact channels, such as voice, e-mail, Web, fax, and so on, with computer systems.

CTI involves using a computer to manage the incoming and outgoing telephone calls. A typical telephone switch allows users to receive and make calls. But CTI and its associated applications deliver a more productive environment because they integrate the computer infrastructure with the voice network. The benefits are most significant in a call center environment. With CTI, customer service representatives (CSRs) can receive calls and data and information associated with the calls. The data can originate from a telephone switch (such as the number that is called and the caller's number) or from a backend host (such as the account information of the caller) resulting from a database search by the CTI application.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

CTI enables several efficient functions, including intelligent call routing, screen-based telephony, intelligent dialing, and automated display of information, based on caller-provided information from an Interactive Voice Response (IVR) or other interface, and the coordinated transfer of data coupled with a telephone transfer. Figure 1-1 illustrates a CTI system.

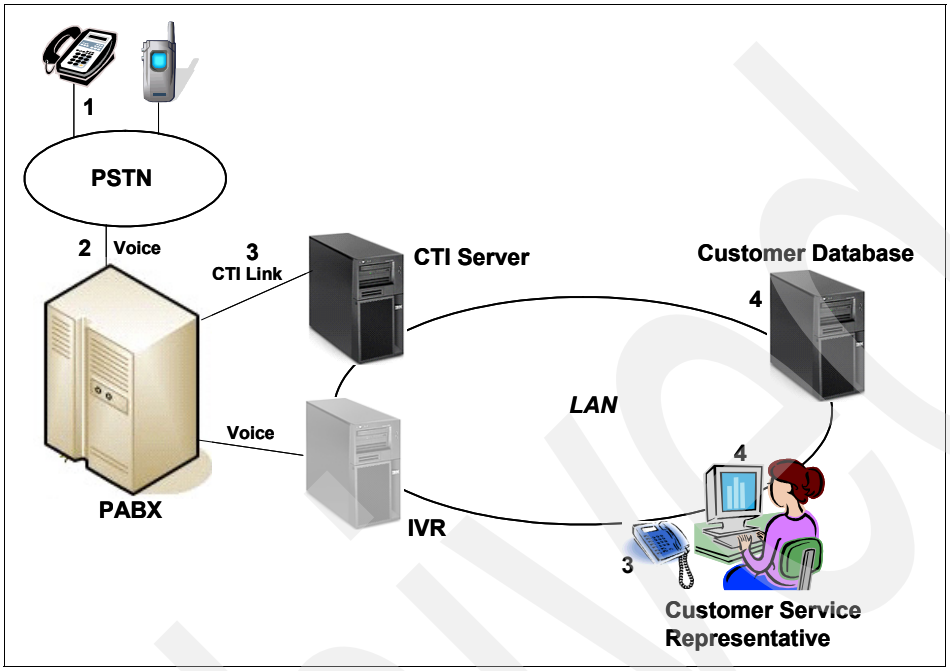

Figure 1-1 Computer Telephony Integration system

CTI technology allows the data collected from a telephone system to be used as input data to query databases with customer information, and populate that data instantaneously in a CSR's desktop window. The result is that the CSR has the required information on the desktop even before speaking with the customer.

There are several standards that are opened for private automatic branch exchange (PABX)/Automatic Call Distributor (ACD) vendors. These interfaces improve the call handling efficiency by enabling telephones and computers to work together.

The following are the popular computer-to-telephony interfaces:

- Computer Supported Telecommunications Applications (CSTA) standard
- JTAPI, the Java Telephony API promoted by Sun
- Telephony Server Application Programming Interface (TSAPI), promoted by Novell
- Telephony Application Programming Interface (TAPI), promoted by Microsoft®

The CTI applications can be written to perform many call center functions, including the following:

- Intelligent call transfer, where data is transferred with the voice call
- Providing IVR options to callers
- Providing speech recognition services
- Making outbound calls
- Providing call center statistics

The following are the benefits of CTI:

- A reduction in call time

- Customer information or request can be captured once and used many times
- Customer information or request can be presented with call arrival (pop-up window)
- Ability to establish consultant-to-consultant communications
- A more personal call answering service
- CTI call statistics

Figure 1-1 depicts the key components of a CTI system. When a user calls the call center from a public switched telephone network (PSTN), the following is the sequence of events that occur:

1. A caller calls through a PSTN (1) and the call arrives at the PABX (2). The called and calling information is passed from the PSTN to the PABX.
2. The PABX alerts the CSR handset (3) that a call is arriving. At the same time, the PABX sends incoming messages to the CTI and call information to the CTI server (3).

Note: The term CTI server can represent a number of servers that collectively perform CTI functions such as connecting the PABX to a computer, collecting call statistics, routing calls to a predefined group of CSRs based on their skill level and availability.

The switch vendors, for example, Avaya, Cisco, and Nortel, have their own CTI product suites that work well with their own products. However, you must always determine which CTI product suite supports the different brands of PABXs.

The Genesys CTI product suite supports PABXs from different vendors. It supports CTI on a multivendor PABX call center environment.

3. The CTI server sends CTI incoming call messages to the CSR's workstation (4). The application on the desktop then treats the messages accordingly. In this scenario, it alerts the CSR's desktop that a call has arrived.

   It is also possible that the desktop application will fetch the caller's database information from the backend host (4) using the calling and called number information provided.

   If an IVR is installed behind the PABX, it can be programmed to ask for the caller's account number and personal identification number (PIN), authenticate this information, and pass the information to the CTI server. The account information is sent to the CSR's desktop application for processing. The result is then presented to the CSR's desktop with the voice call.

The CSR answers the call and the CSR's desktop is populated with the relevant data related to the caller.

Archived

2

Understanding Java Telephony Application Programming Interface

The CTI message flow between the PABX and the CTI server are switch-specific. The CTI server can support a number of switch-specific protocols.

The message flow between the CTI server and the desktop applications can use one of the following common protocols:

- JTAPI
- TAPI
- TSAPI
- Call Control Extensible Markup Language (CCXML)

The basic function of these protocols allows the desktop applications to enable telephony services. For example, the JTAPI protocol has a suite of API, which allows an application to enable telephony services, such as making a call, having a conference call, transferring a call, and disconnecting a call.

JTAPI is defined as a portable, object-oriented API for computer telephony-integrated call control.

© Copyright IBM Corp. 2007. All rights reserved.

A JTAPI application, such as a desktop softphone, can be embedded on another application, such as Lotus® Notes® (Figure 2-1). The softphone emulates a telephone handset on the desktop. Each feature button on the softphone is coded in JTAPI.

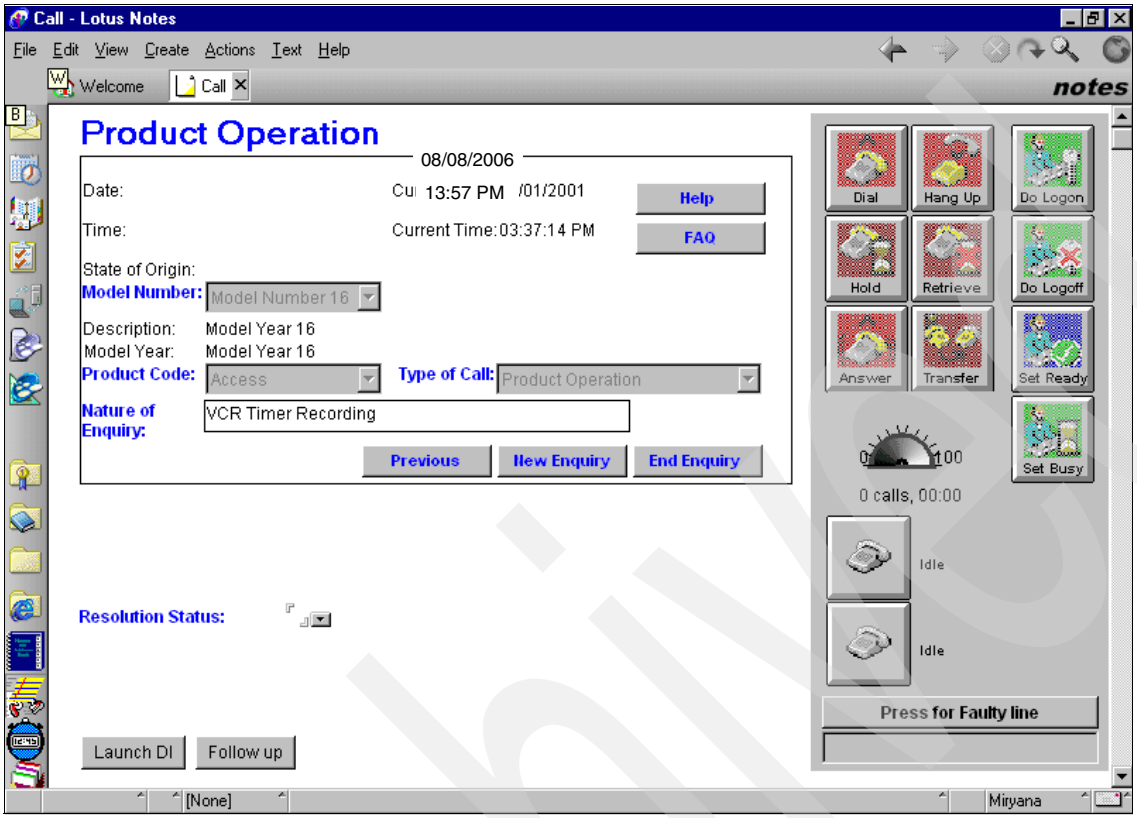

Figure 2-1 Desktop softphone embedded into Lotus Notes

For more information about using JTAPI, refer to the following Web site:

http://www.zurich.ibm.com/csc/distribsys/j323/jtapi-tutorial.pdf

Archive

3

JTAPI and CTI environment

This chapter discusses and demonstrates the use of a JTAPI application serving as an interface between IBM Sales Center and our virtual Cisco CTI laboratory environment.

Note: This section is intended for telephony software developers who are developing IP telephony applications that require JTAPI. This document assumes that the programmer is familiar with both the Java language and the Sun JTAPI V1.4 specification.

In our example, the CTI environment controls the calling information and provides a communication link with the CSR, while the CSR is logged into the IBM Sales Center. In our case, we used two Cisco softphone extensions, one designated for customer use and the other as an IBM Sales Center CSR. When the customer calls from one extension, attempting to reach a CSR, the CTI environment controls that call and redirects it to the registered CSR extension. The CSR is notified of this call through a pop-up message displayed within IBM Sales Center. In a production environment, the pop-up message can provide you with options or display customer account information if a procedure has been written to present such information. At this point, the CSR can communicate with the customer, and place or modify a customer order.

Our virtual CTI environment consisted of the following components:

- Two xSeries x342s
- Microsoft Windows® 2000
- Cisco Call Manager
- Ethernet network connections
- Gateways/Trunks

The virtual CTI architecture has three fundamental types of endpoints, telephone sets, virtual devices (media termination points and route points), and gateways. Among these endpoints, only telephones and media termination points are exposed through the JTAPI implementation.

The CallManager, which exists within the Telephony Server (Figure 3-1), allows users to configure telephones to have one or more lines, dialed numbers, which may be shared among multiple telephones simultaneously or configured for exclusive use by only one telephone at a time. Each line on a telephone is capable of terminating two calls

© Copyright IBM Corp. 2007. All rights reserved.

simultaneously, one of which must be on hold. This is similar to the *call waiting* feature on home telephones.

3.1 Demonstration using the JTAPI-based sample code

To perform our demonstration, the following prerequisites are required:

- A CTI environment
- Download JTAPI-based sample programs and a JTAPI.jar file
- Distribution of telephone extensions

The following are the tasks for establishing communications between the CTI system and IBM Sales Center using JTAPI:

1. Implementing a telephony environment, for example, Cisco
2. Registering CSR telephone numbers into the telephony environment
3. Installing and executing JTAPI-based sample code (In our example, we demonstrate the use of receivecall.java and Receiver.java.)
4. Customizing IBM Sales Center to create a listener for any customer calls being received

The following sections describe the way we constructed our environment and tested two JTAPI-based sample codes.

Note: The JTAPI-based sample codes were acquired from the Cisco Web site:

`http://www.cisco.com/cgi-bin/dev_support/access_level/product_support`

If you choose to test and use another vendor's JTAPI-based sample code, you must acquire the code from that vendor's site.

3.1.1 Acquiring two telephone numbers from the CTI environment

To begin our setup and the use of the JTAPI-based sample code, we asked our CTI administrator to gain access to the CTI environment. We were provided with a host name, a login ID, and a password. We were then provided with two telephone extensions, for example, 2262-customer and 2263-CSR. These extensions are registered in the directory of the CTI environment. As stated earlier, the CallManager within our CTI environment controls the calls from the 2262 extension to the 2263 extension and vice versa. Whenever, a call comes from a customer to the 2262 extension, the CallManager forwards this call to the CSR in the 2263 extension.

3.1.2 Installing JTAPI-based sample code into the local machine

After acquiring access the CTI environment, we requested for the JTAPI-based sample code, which will allow us to connect to the CTI environment. In our example, we used only the receivecall.java and the Receiver.java programs.

Note: We used all the sample code to execute the receivecall.java program. Remember that other Java programs are being executed in the background. The receivecall.java program is executed to connect to the CTI environment and listen for calls. The Receiver.java program is customized and executed to invoke any commands executed within IBM Sales Center.

3.1.3 Running the sample code

We executed the receivecall.java program from a directory in our desktop by providing the argument host name, login, password (this is described in “Receivecall.java” on page 10). This Java program communicates to the CTI environment, indicating that it is ready to receive the calls made to the 2263 telephone extension.

Now, when the customer (with the 2262 extension) calls the 2263 extension, the CallManager is notified within the CTI environment, and the call is routed to connect to 2263. After the customer and the CSR are connected, an action can be invoked to take place within IBM Sales Center.

Figure 3-1 shows the hardware stack and software stack that allow the network computer to communicate with the Telephony Server.

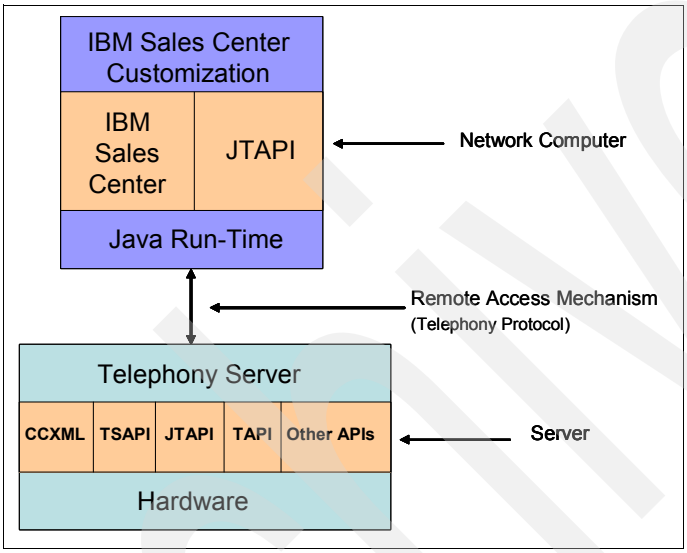

Figure 3-1 IBM Sales Center and JTAPI laboratory environment

3.2 JTAPI-based sample code

This section discusses the Cisco JTAPI-based sample code mentioned earlier in this chapter. This sample code can be used to test the JTAPI installation. Although a mention is made of the list of sample code that was available to us, only the use of the receivecall.java and Receiver.java programs used in our example are discussed. Note the receivecall.java program.

The Java program Makecall.java was developed to make calls to itself again and again, based on a defined time frame. We modified this Java program and named it as receivecall.java to place calls manually. After executing the receivecall.java program, a call can be placed from the telephone number 2262 to another telephone number 2263. This activity is shown in the trace log window (Figure 3-3 on page 15).

The following are the JTAPI-based sample codes:

- Makecall.java (in our example, we modified it to receivecall.java)
- Actor.java
- Originator.java
- Receiver.java

Archived

- StopSignal.java
- Trace.java
- TraceWindow.java

3.2.1 Receivecall.java

After a CSR logs in to IBM Sales Center successfully, receivecall.java must be running to receive a customer's call. This Java program is modified to obtain CallManager access using arguments, and facilitates the process of making calls from one telephone extension to another. This program executes a test that connects IBM Sales Center to the CTI environment.

Perform the following tasks after ensuring that you have the sample code and the jtapi.jar file, to prepare the local environment to run the sample code (b):

1. Install the sample code on the local system. In our example, we installed the sample code in the *C:\temp\makecall* directory.
2. Take a backup of the makecall.java program and rename it to receivecall.java.
3. Edit the receivecall.java program and make changes, as shown in Example 3-1.
4. Make a backup of Receiver.java program.
5. Edit the Receiver.java program to make changes, as shown in Example 3-1. Locate the jtapi.jar file path in the local system because this path is required during the classpath setting.

Example 3-1 Receivecall.java program

```
import java.util.*;
import javax.telephony.*;
import javax.telephony.events.*;
import com.cisco.cti.util.Condition;

public class receivecall extends TraceWindow implements ProviderObserver
{
   Vectoractors = new Vector ();
   ConditionconditionInService = new Condition ();
   Providerprovider;

   public receivecall ( String [] args ) {

      super ( "receivecall" + ": "+ new CiscoJtapiVersion());
      try {

         println ( "Initializing Jtapi" );
         int curArg = 0;
         String providerName = args[curArg++];
         String login = args[curArg++];
         String passwd = args[curArg++];
         //int actionDelayMillis = Integer.parseInt ( args[curArg++] );
         String src = null;
         String dest = null;

         JtapiPeer peer = JtapiPeerFactory.getJtapiPeer ( null );
         if ( curArg < args.length ) {
```

```
            String providerString = providerName + ";login=" + login + ";passwd=" + passwd;
            println ( "Opening " + providerString + "...\n" );
            provider = peer.getProvider ( providerString );
            provider.addObserver ( this );
            conditionInService.waitTrue ();

            println ( "Constructing actors" );

            for ( ; curArg < args.length; curArg++ ) {
               if ( src == null ) {
                  src = args[curArg];
               }
               else {
                  dest = args[curArg];
                  Originator originator = new Originator ( provider.getAddress ( src ), dest,
this );
                  actors.addElement ( originator );
                  actors.addElement (
                     new Receiver ( provider.getAddress ( dest ), this, originator )
                     );
                  src = null;
                  dest = null;
               }
            }
            if ( src != null ) {
               println ( "Skipping last originating address \"" + src + "\"; no destination
specified" );
            }

         }

         Enumeration e = actors.elements ();
         while ( e.hasMoreElements () ) {
            Actor actor = (Actor) e.nextElement ();
            actor.initialize ();
         }


      }
      catch ( Exception e ) {
         println ( "Caught exception " + e );
      }
   }

   public void dispose () {
      println ( "Stopping actors" );
      Enumeration e = actors.elements ();
      while ( e.hasMoreElements () ) {
         Actor actor = (Actor) e.nextElement ();
         actor.dispose ();
      }
   }

   public static void main ( String [] args )
   {
```

Archived

```
      new receivecall ( args );
   }

   public void providerChangedEvent ( ProvEv [] eventList ) {
      if ( eventList != null ) {
         for ( int i = 0; i < eventList.length; i++ )
         {
            if ( eventList[i] instanceof ProvInServiceEv ) {
               conditionInService.set ();
            }
         }
      }
   }
}
```

3.2.2 Receiver.java

This is the JTAPI-based sample code (Example 3-2) that gets notified by the CTI CallManager when a customer call comes in. After the customer call comes in, IBM Sales Center gets notified and then a CSR takes the appropriate action for this customer call.

Example 3-2 Receiver.java program

```
import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.events.*;

public class Receiver extends Actor
{
   Addressaddress;
   StopSignalstopSignal;
   Originatororiginator;

   public Receiver ( Address address, Trace trace, Originator originator ) {
      super ( trace, address);
      this.address = address;
      this.originator = originator;
   }

   protected final void metaEvent ( CallEv [] eventList ) {
      for ( int i = 0; i < eventList.length; i++ ) {
         TerminalConnection tc = null;
         try {
            CallEv curEv = eventList[i];

            if ( curEv instanceof CallCtlTermConnRingingEv ) {
               tc = ((CallCtlTermConnRingingEv)


curEv).getTerminalConnection ();
               delay ( "answering" );
               bufPrintln ( "Answering TerminalConnection " +


tc );
```


Archived

```
                tc.answer ();
                //stopSignal.canStop ();
             }
             if( curEv instanceof CallCtlTermConnTalkingEv ){
                tc = ((CallCtlTermConnTalkingEv)

curEv).getTerminalConnection();
                bufPrintln ("Getting call from " + tc);
                tc.answer();
             }
          }
          catch ( Exception e ) {
             bufPrintln ( "Caught exception " + e );
             bufPrintln ( "tc = " + tc );
          }
          finally {
             flush ();
          }
       }
    }
       protected final void onStart () {
       stopSignal = new StopSignal ();
    }

    protected final void onStop () {
       stopSignal.stop ();
       Connection[] connections = address.getConnections ();
       try {
          if ( connections != null ) {
             for (int i=0; i< connections.length; i++ ) {
                connections[i].disconnect ();
             }
          }
       }catch ( Exception e ) {
          println (" Caught Exception " + e);
       }
    }
    protected final void fireStateChanged () {
       originator.setReceiverState ( state );
    }
}
```

3.2.3 Running the receivecall.java program

To run the receivecall.java program, perform the following tasks:

1. Ensure that jtapi.jar is set in the Windows system class path. Set the class path for jtapi.jar by issuing the following command from the Windows command line:

   ```
   set classpath=%CLASSPATH%;<locate-jtapi.jar>
   ```

   In our example, we issued the following command:

   ```
   set classpath=%CLASSPATH%;C:\temp\makecall\jtapi.jar
   ```

2. In the same Windows command line, navigate to the directory that contains the receivecall.java program and execute the following commands:

Archived

– To compile the Java program:

```
javac receivecall.java
```

– To execute the Java program:

```
java receivecall <server name> <login> <password> <device 1> <device 2>
```

In our example, we executed the following command from the Windows command prompt:

```
java receivecall labcm1.torolab.ibm.com user001 us3r001 2262 2263
```

Figure 3-2 shows the steps mentioned previously.

Figure 3-2 Setting classpath with jtapi.jar, compiling and running the Java program

The following is the explanation of the arguments:

- <*server name*> is the host name or IP address of your CallManager (in our example, we used labcm1.torolab.ibm.com)
- <*device 1*> and <*device 2*> are directory numbers of IP phones. The phones must be a part of the associated devices of a given user, as administered in the Cisco CallManager's directory administration Web page.
- <*login*> and <*password*> are similar to that administered in the directory.

If the receivecall.java program is executed successfully, you will see a trace log window, as shown in Figure 3-3, which explains the adding and registering of the two numbers that are passed as arguments during Java program execution. The receivecall.java program is connected to the CTI environment. When a a call is made from extension 2262 to extension 2263, the CallManager controls this call and directs it to extension 2263.

Archived

In the trace log window shown in Figure 3-3, the last two lines display the message about receiving the call from extension 2262, and the last line about answering the call from extension 2263.

Figure 3-3 Trace log window displaying the calling information

Archived

Archived

4

How to integrate JTAPI into IBM Sales Center

The telephony integration is done using the IBM WebSphere MQ Everyplace® (MQe) messaging system. It exchanges messages with various applications, providing once-and-once-only assured delivery.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

4.1 The reason WebSphere MQ Everyplace must be used

We used the Cisco Call Manager for setting up the environment for call management. During our initial research, we found that the JTAPI libraries provided with Cisco had limitations working on the IBM Java Virtual Machine (JVM™), but worked fine on Sun JVM. We therefore decided to go with an intermediate agent (running on the same PC), which can receive calls and notify IBM Sales Center on WebSphere Everyplace Deployment for Windows and Linux®.

Moreover, WebSphere MQ Everyplace is designed to meet the messaging requirements of lightweight devices such as sensors, phones, personal digital assistants (PDAs), and mobile computers.

4.2 WebSphere MQ Everyplace terms and concepts

This section discusses the MQe terms and concepts to help you get familiar with the items used in our integrated environment.

WebSphere MQ Everyplace terms

The following are the MQe terms used:

- Messages
- Queues
- Queue Managers
- Connections
- Listeners

Messages

Messages are the means for communication in this integration. They are objects that are subclasses of the MQeMsgObject. Messages contain the information that has to be communicated. This information is contained in fields. Each field has a name-value pair.

Queues

When an application wants to transfer data to another application, it puts the data into messages, and then puts the messages into a queue. The messages then reach the destination application in a First-In-First-Out (FIFO) order. A GET action on the queue gets the message. This is the first message that enters the queue. A PUT action puts the message at the tail end of the queue. A queue can either be remote or local.

Queue Managers

An application must interact with the Queue Manager rather than directly with the queues.

Queue Managers are responsible for managing the queues and the details pertaining to sending and receiving messages.

There can be only one Queue Manager per JVM. However, the Queue Manager can manage many queues.

Archived

Connections

With the help of a connection, two Queue Managers can communicate with each other through messages. A connection identifies the target Queue Manager, its IP address, and the port number of the listener of the target Queue Manager.

Listeners

A listener basically listens for incoming messages at a particular port. It identifies the port at which it listens. This port number must match the port number that is specified by the connection to that Queue Manager.

4.3 Communication flow

Figure 4-1 shows two boxes, one denoting the agent that is receiving the call and the other the IBM Sales Center. Both the agent and IBM Sales Center run on the same computer.

As soon as the receiver receives a call, it communicates to the IBM Sales Center with the appropriate information, indicating that a call is coming in.

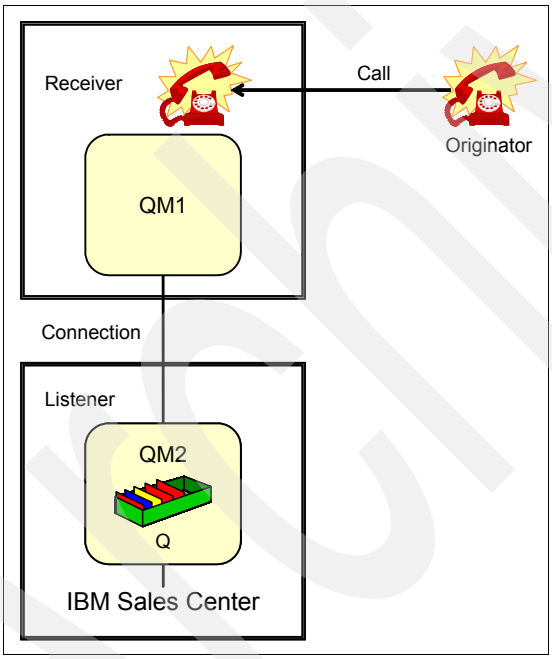

Figure 4-1 Communication flow

The terms used in the diagram are:

- QM1

  This is the Queue Manager maintained on the Call Receiver side.

- QM2

  This is the Queue Manager maintained on the Sales Center side.

- Q

  QM2 manages queue Q, which contains all the messages representing the incoming calls.

What follows is a brief description of our implementation. Whenever a call is made to the Receiver, a message representing the call is put into the queue. IBM Sales Center then

processes these messages in the FIFO order again through the Queue Manager QM2 and initiates the required actions on the call.

MQe contributes to the message flow as follows:

1. Queue Manager QM1 is used for sending messages to the queue Q. It does this by communicating to the Queue Manager QM2 to put or get messages from the queue Q.
2. Queue Q can be a remote queue or a local queue to QM1.
3. Queue Manager QM2 manages Q. It provides communication with QM1 and handles all the GET and PUT request to Q. It also handles all the process instructions from the IBM Sales Center side.
4. A listener listening at a particular port at QM2 for incoming messages to be obtained is required.
5. For communication between QM1 and QM2, a connection is established by QM1 with QM2.

   The following list shows how communication is established:

   – The adapter used for communication
   – The IP address of the Target QM, for example, QM2
   – The port number at which the QM2 listener is listening for incoming messages (For our example, we used ports 8087, 8082, and so on)
6. After the listener and the connection is established, QM1 can put and get messages from Q by invoking the appropriate methods on QM2.

4.4 Code snippets

The code snippets found in the following sections explain how to set up queue managers and how communication can be established between them by sending and receiving messages through a queue. For more information about MQe, refer to the following Web site:

http://publib.boulder.ibm.com/infocenter/iwedhelp/v6r0/index.jsp?topic=/com.ibm.mqe.doc/ovr50010.html

4.5 Defining and configuring a Queue Manager

Example 4-1 shows the way we defined and configured our Queue Managers. This can be used for both QM1 and QM2.

Example 4-1 Defining and configuring a Queue Manager

```
// Create all the configuration information needed to construct the
 // queue manager in memory.
MQeFields config = new MQeFields();

  // Construct the queue manager section parameters.
MQeFields queueManagerSection = new MQeFields();

queueManagerSection.putAscii(MQeQueueManager.Name, "QueueManagerName");
config.putFields(MQeQueueManager.QueueManager, queueManagerSection);

  // Construct the registry section parameters.
  // In this examples, we use a public registry.
```

```
MQeFields registrySection = new MQeFields();
 registrySection.putAscii(MQeRegistry.Adapter,"AdapterName");

registrySection.putAscii(MQeRegistry.DirName, "BaseDirectoryPath" + "/Registry");
config.putFields("Registry", registrySection);

// Construct a queue manager configuration utility object.
MQeQueueManagerConfigure configurator = new MQeQueueManagerConfigure(config,
"AdapterName" + ":" + "BaseDirectoryPath"  + "/Queues");

// Define a queue manager.
 configurator.defineQueueManager();

// Define some queues on the queue manager.
configurator.defineDefaultAdminQueue();
configurator.defineDefaultAdminReplyQueue();
configurator.defineDefaultDeadLetterQueue();
configurator.defineDefaultSystemQueue();

// Close the queue manager configuration utility class.
configurator.close();

myQueueManager = new MQeQueueManager();
myQueueManager.activate(config);
```

4.6 Creating a queue

Example 4-2 demonstrates how to create a queue. In our example, we used this to create queue Q in QM2.

Example 4-2 Creating a queue

```
/* Create an empty queue admin message and parameters field */
MQeQueueAdminMsg msg = new MQeQueueAdminMsg();

MQeFields parms = new MQeFields();

/*  Set name of queue to manage */
msg.setName( "QueueManagerName", "QueueName" );

/*Add any characteristics of queue here, to change default behaviour */
parms.putUnicode( MQeQueueAdminMsg.Queue_Description,"Description of Queue");

parms.putInt(MQeQueueAdminMsg.Queue_MaxQSize,200);
parms.putInt(MQeQueueAdminMsg. Queue_Expiry, 20000);

/* Set the admin action to create a new queue */
msg.create( parms );
```

Archived

4.7 Configuring and starting a Listener

Example 4-3 shows you how to configure and start the listener. This can be used to create the listener in QM2 shown in Figure 4-1 on page 19.

Example 4-3 Configuring and starting a listener

```
MQeCommunicationsListenerAdminMsg createMessage =   new
MQeCommunicationsListenerAdminMsg();

//Set Name of Listener
createMessage.setName("ListenerName");

// Set the Target QueueManager Name
createMessage.setTargetQMgr("TargetQueueManagerName");

//Create the Admin message which creates the listener
createMessage.create("NameofAdapter",PortNumbertoListenat, Timeoutinmilliseconds,
maxNumberOfChannels);
createMessage.putInt(MQe.Msg_Style, MQe.Msg_Style_Request);
createMessage.putAscii(MQe.Msg_ReplyToQ, MQe.Admin_Reply_Queue_Name);
createMessage.putAscii(MQe.Msg_ReplyToQMgr, queueManagerName);
match = "Msg" + System.currentTimeMillis();
createMessage.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());

//Put the admin message into the QueueManager
myQueueManager.putMessage("QueueManagerName", MQe.Admin_Queue_Name,
createMessage, null, 0);

//start the listener
MQeCommunicationsListenerAdminMsg startMessage =
        new MQeCommunicationsListenerAdminMsg();
startMessage.start();

//Get the response Message from the Admin Queue
MQeFields filter = new MQeFields();
filter.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());
MQeAdminMsg response = (MQeAdminMsg)
myQueueManager.waitForMessage(queueManagerName,
MQe.Admin_Reply_Queue_Name, filter,null, 0, 3000);
```

4.8 Creating a connection

Example 4-4 shows how to create a connection. This can be used to create a connection from QM1 to QM2.

Example 4-4 Creating a connection

```
//Create a admin message to create the connection
MQeConnectionAdminMsg connectionMessage = new MQeConnectionAdminMsg();
connectionMessage.setName("NameOfConnection");
connectionMessage.create("AdapterName:IPofTargetQueueManager:portof
theTargetQueueManagerListener", null, null, "Default Channel", "Example
connection");
```

```
//set the TargetQueueManager
connectionMessage.setTargetQMgr("TargetQueueManagerName");

//set Characteristics of the connection to differ from default
connectionMessage.putInt(MQe.Msg_Style, MQe.Msg_Style_Request);
connectionMessage.putAscii(MQe.Msg_ReplyToQ, MQe.Admin_Reply_Queue_Name);
connectionMessage.putAscii(MQe.Msg_ReplyToQMgr, queueManagerName);
String match = "Msg" + System.currentTimeMillis();
connectionMessage.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());

//Put the message to the local queue manager
myQueueManager.putMessage(queueManagerName, MQe.Admin_Queue_Name,
 connectionMessage, null, 0);
 MQeFields filter = new MQeFields();
 filter.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());

//Get the response from the Admin Queue
 MQeAdminMsg response = (MQeAdminMsg)
 myQueueManager.waitForMessage(queueManagerName,
MQe.Admin_Reply_Queue_Name,filter, null, 0, 3000);
```

4.9 Putting and getting messages from the remote queue

Example 4-5 shows how to put and get a message from a remote queue.

Example 4-5 Creating, putting, and getting message from remote queue

```
//creating the message
MQeMsgObject  msg=new MQeMsgObject();


//Specify the field name and value
msg.putUnicode("myFieldName", name);
msg.putUnicode("myStatus", status);

//putting the message
myQueueManager.putMessage( "TargetQueueManagerName", "TargetQueueName", msg,
filter, confirmationIdforthemessage);

//getting the message
msg=   myQueueManager.getMessage( "TargetQueueManagerName", TargetQueueName",
filter, confirmationIdforthemessage);
```

4.10 Unconfiguring a Queue Manager

Example 4-6 shows how to unconfigure a queue manager if necessary.

Example 4-6 Removing a Queue Manager

```
// Remove the configured entries for the queues we defined.
configurator.deleteSystemQueueDefinition();
```

```
configurator.deleteDeadLetterQueueDefinition();
configurator.deleteAdminReplyQueueDefinition();
configurator.deleteAdminQueueDefinition();

// Remove the queue manager itself.
configurator.deleteQueueManagerDefinition();

// Close the queue manager configuration utility class.
configurator.close();
```

4.11 Enabling the Receiver to work with WebSphere MQ Everyplace

The Receiver.java specified in 3.2, “JTAPI-based sample code” on page 9 has to be enhanced to incorporate the functionalities, to be able to communicate with the Queue Manager for putting messages into the queue.

In the metaEvent() of the Receiver

- if(curEv instanceof CallCtlTermConnRingingEv) - means that the status is ringing. A message must be put into the common queue.
- if(curEv instanceof CallCtlTermConnTalkingEv) - means that the call has been ended by the originator. If the message is still in the queue, it means that the call has not been answered by the receiver (IBM Sales Center), but that the call has been ended by the originator. Whether the message still exists in the queue must be checked. If it exists, it is deleted from the queue.
- if(curEv instanceof CallCtlConnDisconnectedEv) - also means that the call has not been answered by the receiver (IBM Sales Center). Whether the message still exists in the queue must be checked. If it exists, it is deleted from the queue.

4.12 Processing messages from IBM Sales Center

A new view must be created for the graphical user interface (GUI) to reflect the call details and the customer details. This is implemented by creating a new Eclipse plug-in, in the IBM Sales Center environment, by extending the org.eclipse.ui.part.ViewPart.

A Message Listener is attached to the Queue Manager, which raises an event whenever a message is received by the queue. This can be done by extending the com.ibm.mqe.MQeMessageListenerInterface. The messageArrived(), which gets called whenever a message reaches the queue, must be implemented. The processing that has to be performed whenever a message arrives, can be included using this function.

Functionalities provided by the plug-in

The following are the functionalities provided by the plug-in:

1. It receives incoming calls using the JTAPI.
2. When a call is received, it puts a message in the queue using the connection and the listener.
3. The queue's message listener now raises an event. And if the user is not already attending any calls, it displays the details of this call to the user.

4. If the user is on another call, this message stays in the queue.
5. The message gets picked up after the user ends the previous call.
6. If the call ends or is ended by the caller by the time the message reaches the head of the queue, the message is taken back from the queue. (It is a missed call and therefore, its associated message is not available in the queue).

User interface design of the plug-in

The user interface (UI) design of the plug-in is as follows:

1. Whenever there is an incoming call (a message reaches the queue) and if there is no call being attended by the IBM Sales Center CSR, the caller's number is displayed in the telephony view for the CSR to take action.
2. *Online* status indicates that the phone is ringing. The message is received from the queue. The message is no longer available in the queue and the user is notified.
3. The CSR has the choice of viewing the customer details by right-clicking the call and selecting **Open Customer Editor**. This option is available only when the CSR is logged in.
4. The CSR can also choose to drop the call. A dropped call still appears in the call list of the view. Its status now becomes *Attended*.
5. When the call is dropped, the next message in the queue is received from the queue and its calling number is displayed.
6. If another call is waiting in the queue when the CSR is attending a call, a text appears on top of the view, indicating that a call is waiting.
7. A call that is missed is never displayed in the UI.

Using the plug-in

The plug-in that creates a new view, Telephony View, is provided in com.ibm.commerce.telesales.cti_1.0.0. (See Appendix A, “Additional material” on page 31 for this plug-in.)

Complete the following tasks:

1. Copy the com.ibm.commerce.telesales.cti_1.0.0 folder into the plug-in folder under the Eclipse home path in the IBM Sales Center installed environment. Copy the MqeGateway.jar to the \com.ibm.commerce.telesales.cti_1.0.0\lib folder. The MqeGateway.jar is available with the MQ Everyplace trial version, which is available in the following location:

   `http://www-128.ibm.com/developerworks/websphere/downloads/wmqe/`

   The QueueManagers will use the local disk place in the path C:\QueueManagers. This has been configured in the sample code.
2. Restart the IBM Sales Center environment.
3. Open the IBM Sales Center by selecting **Application** → **Open** → **IBM Sales Center** → **Order Management**.
4. Log in to IBM Sales Center and open the required store.
5. Open the Telephone View by clicking **View** → **Show View** → **Telephony View**.
6. Keep the view opened.

After setting up the CTI environment and obtaining two telephone numbers from the CTI environment (as described in Chapter 3, “JTAPI and CTI environment” on page 7), run the sample code:

1. Copy the folder called Sample into the file system, for example, <temp_path>. Set the class path to point to the jtapi.jar and the MqeGateway.jar. The QueueManagers will use the local disk place in the path C:\QueueManagers. This has been configured in the sample code.
2. Open the Cisco IP phone communicator.
3. Dial the receiver number.
4. Run the sample.bat file inside the Sample folder in the <temp_path> with the argument shown in Example 4-7.

Example 4-7 sample.bat file

```
<servername> <calleruser> <destinationuser> <callernumber> <destinationnumber>
from the command line.

For Example:
sample.bat servername user002 us3r002  2264 2265
Here,
 2264 is the caller
 2265 is the receiver
       user002 is the username associated with the caller.
       us3r002 is the username associated with the caller.
```

4.13 Scope of the model used

Based on the description and background provided in the previous section, the model can be extended to handle real-time functionalities.

Archived

5

IBM Sales Center and JTAPI production environment

The previous chapter discussed the use of JTAPI in a virtual CTI environment. This chapter shows how our test environment appears in a production environment.

Figure 5-1 shows a hardware and software stack that allows the network computer to communicate with the telephony server through a production telephone system.

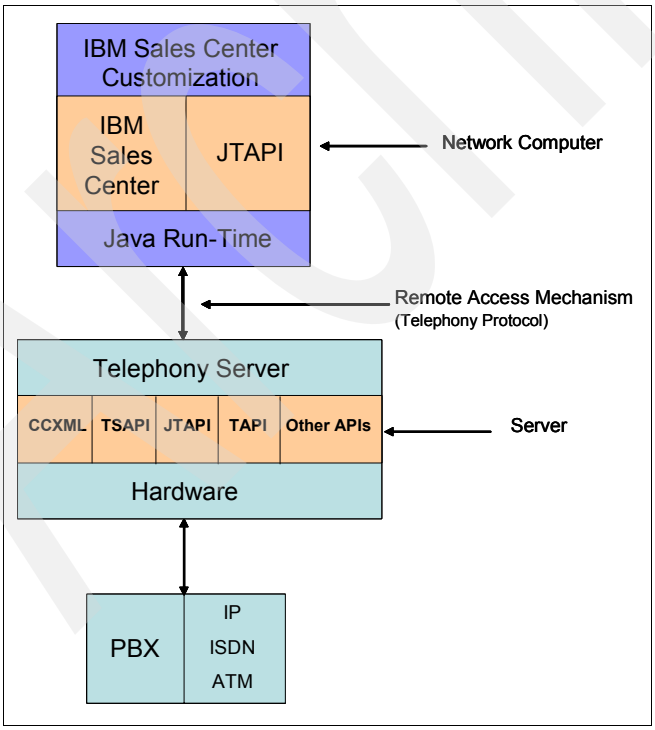

Figure 5-1 IBM Sales Center and JTAPI in a production environment

© Copyright IBM Corp. 2007. All rights reserved.

Figure 5-2 shows a full-scale CTI environment that includes an IVR system and WebSphere Commerce supporting IBM Sales Center for WebSphere Commerce.

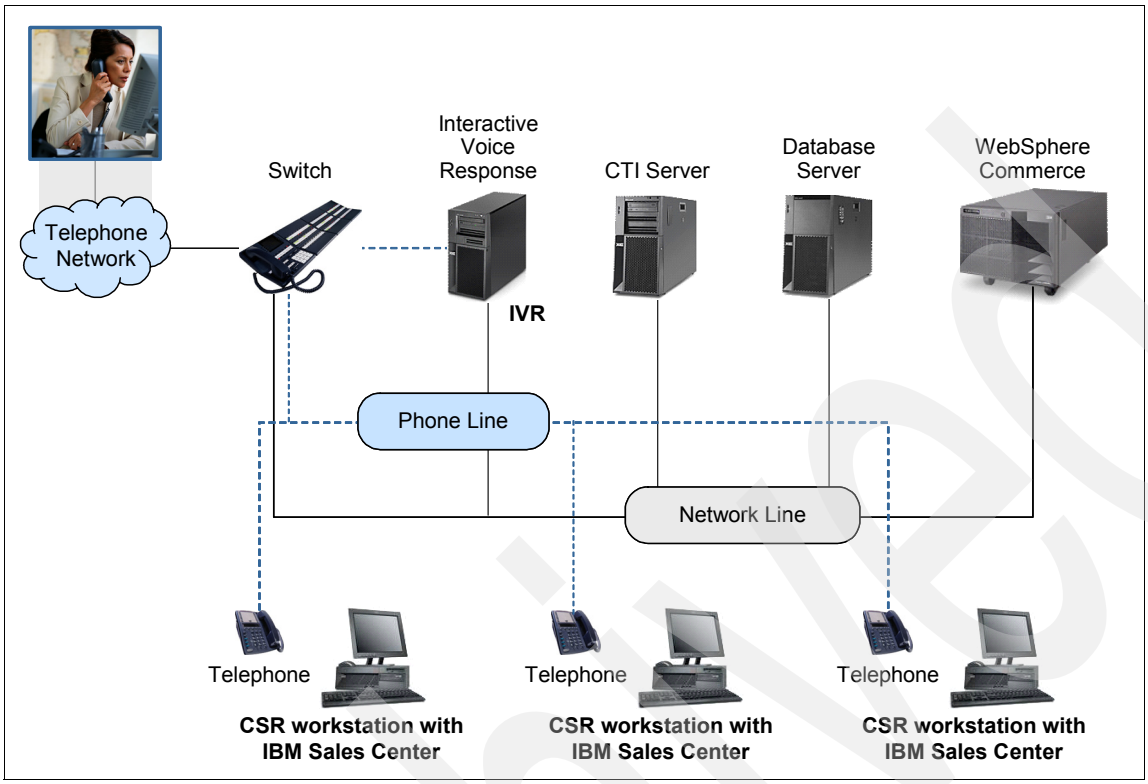

Figure 5-2 A full-scale CTI environment

5.1 Use cases

This section discusses use cases that were created and are highlighted to help capture the functional requirements for IBM Sales Center. Consider these use cases for your IBM Sales Center business solution.

5.1.1 The login use case

The login use case demonstrates the following:

1. A CSR logs in to IBM Sales Center.
2. IBM Sales Center attaches to the telephony system, using the server and the authentication data supplied by client preferences.
3. IBM Sales Center attaches to the telephony system, using the server and the authentication data that comes from the WebSphere Commerce server.

5.1.2 The non-monitored call use case

The non-monitored call use case demonstrates the following:

1. A CSR's supervisor calls the CSR to discuss vacation entitlement. This call is placed to the CSR's internal line, not to the external number that customers call.

2. The CSR answers the call, suspends the call, and resumes the call, but IBM Sales Center recognizes that these calls were not made to the call center external line and takes no action.

5.1.3 The direct monitored call use case

The direct monitored call use case demonstrates the following:

1. A call arrives to an external customer line. The line is typically accessible to more than one CSR phone.
2. CSR Joe answers the call.
3. Joe's IBM Sales Center automatically retrieves the caller's phone number and uses this information to look up the caller in WebSphere Commerce.
4. If exactly one matching customer record is found, IBM Sales Center opens a customer editor on that customer and makes that customer the active customer.
5. If no matching record is found or if more than one matching customer record is found, IBM Sales Center displays the Find Customer dialog box with the pre-populated number search fields and the results are displayed.

5.1.4 The forwarded call use case

The forwarded call use case demonstrates the following:

1. After the Direct monitored call sequence, the customer tells the CSR that Mary, another CSR, has been handling an issue, and could he therefore speak to her again, please?
2. Joe forwards the call to Mary's internal line.
3. Mary answers the call on her internal line, but because it was originally placed to the external customer line, Mary's IBM Sales Center does process it as though it is a direct monitored call scenario.
4. In another variation, Joe works with the customer for a while before forwarding to Mary. So Joe forwards his IBM Sales Center state to Mary along with the call.

5.1.5 The queued call use case

The queued call use case demonstrates the following:

1. Instead of going directly to the CSR shared line, the call is queued in the call handling system. There, Sally entertains the customer with elevator music and promises "Your call is important to us."
2. Joe eventually retrieves the call.
3. IBM Sales Center processes the call as it did in the forwarded call case because the call was originally made to the external customer support line.

5.1.6 Queue call with Interactive Voice Response interaction use case

This is similar to a queued call, but the call handling system includes a Voice Response Unit (VRU) that interacts with the customer to collect the customer number, which the customer enters using the touchpad of the phone. When IBM Sales Center processes this call, it retrieves that additional information from the call handling system and uses it to find the customer. If the lookup using the additional information is unsuccessful, IBM Sales Center reverts to using the calling number as in the cases described earlier.

Archived

This is a use case that requires customization because there is no standard cross-vendor interface to retrieve the IVR354904-collected data.

Archived

A

Additional material

This paper refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/REDP4315

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redpaper form number, REDP4315.

Using the Web material

The additional Web material that accompanies this paper includes the following files:

File name	*Description*
com.ibm.commerce.telesales.cti_1.0.0.zip	**Use for this Redpaper**

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

© Copyright IBM Corp. 2007. All rights reserved.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 33. Note that some of the documents referenced here may be available in softcopy only.

- *Deploying and Customizing IBM Sales Center for WebSphere Commerce V6*, SG24-7249

Online resources

These Web sites are also relevant as further information sources:

- An Introduction to the Java Telephony API (JTAPI) - PDF

  http://www.zurich.ibm.com/csc/distribsys/j323/jtapi-tutorial.pdf

- JTAPI-based sample codes

  http://www.cisco.com/cgi-bin/dev_support/access_level/product_support

- MQe information

  http://publib.boulder.ibm.com/infocenter/iwedhelp/v6r0/index.jsp?topic=/com.ibm.mqe.doc/ovr50010.html

- MQ Everyplace trial version

  http://www-128.ibm.com/developerworks/websphere/downloads/wmqe/

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

© Copyright IBM Corp. 2007. All rights reserved.

Archived

Archived

IBM Sales Center

With Computer Telephony Integration

Redpaper™

Integrate JTAPI technology into IBM Sales Center

JTAPI and CTI examples

IBM Sales Center and JTAPI use cases

This IBM Redpaper discusses how to integrate IBM Sales Center for WebSphere Commerce with a Computer Telephony Integration system that uses Java Telephony Application Programming Interface (JTAPI). This technology enables the Sales Center to identify a caller based on their phone number, and access customer-related information. This Redpaper also provides a demonstration using Cisco JTAPI sample code and provides necessary information to design, develop, deploy, and customize applications.

This Redpaper is a derivative of the IBM Redbooks publication *Deploying and Customizing IBM Sales Center for WebSphere Commerce V6*, SG24-7249.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

REDP-4315-00

Archive