Metreos Communications Environment

Scheduled Conference Application

8/16/2004
Version 1.0

OVERVIEW

The Scheduled Conference application bundled with the Metreos Communications Environment (MCE) is first and foremost intended for reference and educational purposes only. Additional features and error handling must be implemented before this application is suitable for use in a production environment.

However, basic functionality is present and the application should run without issues on a properly configured Application Server with a connected Metreos Media Server. See the installation chapter for a list of required components.

This application demonstrated the considerations, flow, and behavior necessary to create a conferencing application that keeps track of the starting time of a conference. The application uses the database interaction components of the Metreos Visual Designer to retrieve information about a conference that was previously scheduled by a user, and takes appropriate action based on user input. Scheduling a conference is accomplished by way of writing an entry into a database, and thus this task can be integrated into any environment that provides read and write access to a database, such as the PHP-enabled web page that is provided with this sample application that can be found in the Web sub-folder of this application.

To request a conference via the included web page, the user specifies the date, time, and duration of the desired conference. After clicking Submit, the script generates a random five-digit pin number, creates a database query, executes the query, and then returns the generated pin number to the user. The user can then dial into the system, enter his pin, and be connected to the conference, provided that it is not more than 15 minutes before the scheduled conference time.

The 15 minute limit is a soft one. The application could easily be modified to make the time restrictions configurable via the MCE Management Console or through a custom management system, by storing the information either in the application itself or in a database, respectively. Additional features such as user accounts, recording, and muting might also be implemented.

The obvious motivation for an application such as this one is convenience. One user sets up a conference, distributes the time and pin number for the conference to other individuals, which allows them all to conference together at a specific time. The less obvious motivation is media provisioning. The Metreos Media Server has a limited number of resources, which creates the possibility that all the resources may be in use when a user tries to create a non-scheduled conference. By scheduling the conferences, it is possible to keep track of reserved media resources that are expected to be used at any given point in time. This allows for the most optimal operation of a conferencing system, since users would be nearly guaranteed to have the required resources available to them when they need them.

The flow of interaction with the conferencing system is roughly as follows. The application is triggered by an incoming call. The call is picked up, and a welcome message requesting the conference pin is played. The user enters his conference pin, followed by the pound sign (#). The application checks with the database whether the pin number is a valid one. If it is not, it request that the user re-enter the pin, and performs the check again. If the pin number is valid, the application will check if the current date is not more than 15 minutes before the scheduled time of the conference, as listed in the database. If the user has called too early, he is informed of this fact via an announcement, he is asked to call back at the appropriate time, and then the user is hung up on.

If the pin number and the current time are an appropriate match for the conference, the application checks to see if the user is the first one into the conference. If the user is the first one in, the conference is created, the user is placed into it, and an announcement welcomes him to the conference. If the user is not the first one in, the application retrieves the proper conference data from the database. Next, a prompt asks him to speak his name and company name, and then to press pound. If the user talks for more than 10 seconds, he will be cut off. The application also will detect if the user doesn't say anything. Next, the application plays an announcement to the existing conference. Normally, it plays whatever the user said in the previous step, followed by: "has entered the conference." However, if silence was detected, a standard announcement consisting of "A user has entered the conference" is played. This is repeated for each additional user entering the conference.

When a user hangs up the phone, the application checks whether he was the last user to leave the conference. If he was not, it removes the hanging-up user from the conference, and then plays an announcement to the conference specifying that a user has left. If the user was the last one to leave, no announcement is played, the conference is destroyed, and its entry is removed from the database.

INSTALLATION

There are several components which must be configured in order for this application to function properly. The installation and setup described here should be very similar to the installation and setup involved with any call-control application.

Media Server

First, the Media Server should be setup according to the procedure described in the Media Server User's Guide. This covers the setup of Intel Host Media Processing (HMP) and the Metreos Media Server service.

Copy all of the files from the *media* sub-folder of this application to your Media Server's audio directory.

Start the Media Server.

Application Server

Like the Media Server, this guide assumes that the Application Server has been setup and configured according to the procedure outlined in the User's Guide. This application requires that the H.323 and Media Server protocol providers be installed.

Start the Application Server.

The Application Server connects to the Media Server by first properly configuring the IP address and port of the Media Server in the MCE Admin web page. Once logged in, go to the 'Media Servers' link found under the 'Component Management' section of the main page. From there, select *Add a media server,* then specify the IP address of the machine running the Metreos Media Server service. Do not change the default port unless you have likewise modified the Media Server configuration. Once the *Add Media Server* button is clicked, the Media Server provider will begin establishing a connection to the Media Server. Once the connection has been established, the Media Server provider will log an Info-level message indicating a successful connection, which should be corroborated on the website.

There are three settings that may need to be modified. Proceed to the *Applications* link on the MCE Admin main page. Select the *Configure* option for the ScheduledConference application. The authCode setting determines the pin number that the caller will need to input in order to be authorized to place the call. The callManagerIp setting should be set to the IP address of your CallManager. The *appServerDn* setting determines the routing pattern for which this application will be launched. This may be specified as a literal or as a .Net regular expression.

Cisco Call Manager

This application isn't terribly practical without the aid of a VoIP PBX such as the Cisco CallManager. CallManager should already be configured with at least one phone device. If you have not already done this, add a new H.323 phone device and enter the IP address of the Application Server as the device description. Now, begin adding DNs. For example, if you pick the Directory Number 65000 for the first DN and you want the conferencing system to support 4 callers at a time, create DNs with directory numbers of 65000 through 65003. Set DN 65000 to Forward Busy to 65001, DN 65001 to Forward Busy to 65002, etc.

Database Server (MySQL)

This application requires a database in which it will store information about the scheduled conferences. For each scheduled conference, we store the following information: *confNumber, confId, scheduledFor, numHours, and numParticipants. confNumber* holds the pin number generated for use with the conference, *confId* is the internal media server identifier for the conference, *scheduledFor* is the date for which the conference is scheduled, *numHours* is the number of hours that it is scheduled for, and *numParticipants* represents the number of participants currently in the conference. None of these fields are allowed to be null, and *confId* and *numParticipants* must be zero upon first insertion to the database, as they are manipulated by the application. *confNumber, confId, numHours*, and *numParticipants* are all integers. *scheduledFor* is of type DATETIME.

Included in the SQL sub-folder of this application is a database creation script designed to work with a MySQL database. It will create a database named *metreos*, and then will create a table named *scheduledConferences* in that database. It will then create a user with the name *metreos* that can connect from any host, and has delete, insert, select, and update privileges on the metreos database and all tables inside it. Next, it will set the password to *metreos*, using the MySQL ODBC 3.51 compliant password encryption. Should you be running a MySQL database, you should be able to run this script using the mysql client by typing: mysql –u *USERNAME* –p < [path to scheduledconf.sql file]

Should you not be running a MySQL database, you may attempt to recreate the above set-up inside your database of choice.

The application itself uses an ODBC Data Source called "metreosMySQL." You may create one by going to *Start->Settings->Control Panel->Administrative Tools->Data Sources,* and adding a source with the above name. This Data Source must be created on the same machine as the application server. While performing the add, be sure to select the right driver for the database you are using. MySQL ODBC drivers can be obtained from the MySQL homepage (www.mysql.com).

If you are using MySQL ODBC, specify the following information:

Data Source Name: metreosMySQL
Host/Server: 127.0.0.1
User: metreos
Password: metreos

Click on *Test Data Source* to make sure your set-up is correct. Refer to the MySQL ODBC documentation if any problems arise.

Should you not be using MySQL ODBC, duplicate the above settings with whatever database you are using.

Web Server (Internet Information Services)

The index.php file in the Web subfolder of this application is a PHP script that allows a user to pick a date and time for the conference using a series of list boxes, inserts the conference entry into a MySQL database, and returns the generated pin number to the user.

In order to use the included PHP webpage to schedule conferences you will need a web server capable of interfacing with the PHP scripting language (www.php.net) such as IIS or Apache, as well as a MySQL database server (discussed in above section). Please refer to www.php.net for required software packages and documentation. Once you have properly configured your web server and PHP, you need to edit the index.php file in the Web sub-folder of this application folder. If your database server is on a different machine than your web server, you will have to change one line in the index.php script:
Line 11 of index.php is as follows: *$dbHost = "localhost";*
If you need to, replace *localhost* with the IP address or hostname of your database server.

If you are using IIS, you will need to open *Start->Settings->Control Panel->Administrative Tools->Computer Management*. Expand *Services and Applications,* then expand *Internet Information Services*, then *Web Sites,* and finally, *Default Website*. Right click on *Default Website*, then pick *New->Virtual Directory*. Specify an alias that you want to use, click *Next*, specify the directory in which the index.php file resides, and click-through to the finish. Right-click the newly created entry and select *Properties->Documents.* Click on *Add,* and type in index.php. Click *Okay* to apply all changes.

The document should now be available under http://webserver/alias, where webserver is the hostname or IP address of the web server you created the virtual directory on, and alias is the alias that you specified while creating the virtual directory.

If you want to use this file on a web server other than IIS, please refer to the specific documentation for that product.

HOW IT WORKS

The application is triggered by a Metreos.CallControl.IncomingCall event.

Script: OnIncomingCall

This is the script that is triggered when the IncomingCall event is raised by the application server upon receiving a call. OnIncomingCall is the handler for an IncomingCall event. The first thing we do is set the g_ProcessDigits boolean global variable using CustomCode. g_ProcessDigit is later used by another event handler to determine whether to call ReceiveDigits or not. Next, we request a connection from the media server. If the request does not succeed, we reject the incoming call and exit the application. Otherwise, we report the details of the connection to the calling device, and answer the call. If the provisional response for attempting to answer the call denotes anything other than success, we delete the connection and exit. If the AnswerCall function is processed successfully, we will either receive an AnswerCall_Complete or an AnswerCall_Failed event, depending on what the final response is. OnAnswerCall_Complete and OnAnswerCall_Failed are the handlers for those events, respectively.

In OnAnswerCall_Failed, we delete the connection to the media server, and exit the script. In OnAnswerCall_Complete, we assign the IP and port of the calling device to global variables and then complete the half-connect to the media server by providing the media server with the details of the calling device. If this fails, we hang up the call. Otherwise, we play the welcome prompt. If the provisional response of this action is not success, we again hang up the call, since we are unable to play the announcement to the user. OnPlayAnnouncement_Complete and OnPlayAnnouncement_Failed are the handlers for the PlayAnnouncement action.

In OnPlayAnnouncement_Failed, we check to see if the global boolean variable g_ExitApp is set. This variable is used by the PlayAnnouncement handlers to determine whether the application should be exited or not. In some cases, it is not critical that an announcement is played, and failure can be ignored. Other times, we cannot proceed unless the announcement succeeds, and the application is exited. OnPlayAnnouncement_Complete checks to see if the above-mentioned g_ExitApp flag is set, and hangs up the call if it is. It is important to note that this handler is executed after the PlayAnnouncement operation receives a final response – thus the call will not be hung up until after the announcement is done playing. Next, we check if we're adding the calling party to an existing conference.

If we are not adding the user to a conference, which may not yet exist, we check if we want to process digits from the phone keypad. If we do not, we simply exit the function. Otherwise, we call ReceiveDigits. If the provisional response for receive digits fails, we set g_ExitApp to true, and play an announcement to the user explaining a problem with the system. When the announcement is done playing, the

OnPlayAnnouncement_Complete and OnPlayAnnouncement_Failed will see that g_ExitApp is true, and will hang up the user. If the provisional response for the PlayAnnouncement fails, we simply hang up the user. The above method is used throughout the entire application, and it is assumed that the reader will recognize the pattern. Should there be any variations, they will be discussed. If the provisional response from ReceiveDigits is successful, we turn off digit processing for further PlayAnnouncement actions, and exit the function.

If we are adding a user to an existing conference, we first use CustomCode to set g_AddingToConference to false, so this path is not taken again unless explicitly specified by setting g_AddingToConference to true. Usually when we go down this path, the announcement that raised the PlayAnnouncement_Complete event requested that the user specify his name and company name. Thus, the next thing we want to do is record what the user says. We set flags on the RecordAudio action to stop recording after 10 seconds of input, or detect 9.9 seconds of silence. The terminatingCondition of the RecordAudio action will be examined in OnRecordAudio_Complete.

After the user connects, he hears a prompt asking him to input his conference pin on the keypad. OnPlayAnnouncement_Complete will then call the ReceiveDigits action, which will process user input until the # key is pressed, as specified by the ReceiveDigits terminating conditions. If the user input is successful, we use CustomCode to remove all occurances of *s and #s from the received input. If after processing the input is empty, we return failure from the custom code, turn digit processing back on, play an announcement specifying that the input was invalid, and we essentially loop, since OnPlayAnnouncement_Complete will again call ReceiveDigits.

If the input is deemed valid, we check if the g_dbConnected boolean, which tells us whether we've already connected to the database or not, is true. If it is false, we proceed to the OpenDatabase action, found under the Database section of the Metreos Visual Designer toolkit. We specify the Data Source Name (DSN) here. The line used is:
DSN=metreosMySQL;UID=metreos;PWD=metreos
DSN=metreosMySQL specifies the name of the ODBC component that was discussed under the Installation section. UID and PWD specify the username and password, respectively, as set in the ODBC component. We specify *dbConnection* as the name that we will use throughout the application to refer to this particular database connection throughout all subsequent interactions with the database. We specify *odbc* for the type of connection. If the OpenDatabase action fails, we exit the application. Otherwise, we set g_dbConnected to true, and move on to the ExecuteQuery action. We also would have arrived at this point if g_dbConnected had been set to *true* when the condition was tested.

Next, we execute the ExecuteQuery action, which sends a query to the database connection referenced by the name 'dbconnection,' and place the results of the query into a System.Data.DataTable object. The query requests all rows from the database that have a confNumber value equal to the conference pin that the caller just specified. The scheduling webpage included with this application guarantees that at any given point in

time all the confNumber values will be unique. Thus, in the actions that follow, we can assume that there will either be one or zero rows returned by the query.

The CustomCode action that executes next, checks to see if the number of rows returned by the query is zero. If so, the action returns failure, and we proceed down the invalid user input path discussed several paragraphs ago. Otherwise, we pull out the row holding the desired results, and extract the confId, numParticipants, scheduledDate, and numHours columns of the database, and assign them to g_conferenceId, numberOfParticipants, scheduledDate, and numHours variables, respectively. We then return success.

The next CustomCode action checks whether the current system time is more than 15 minutes before the scheduled conference start time. If it is, we set g_ExitApp to be true. Next, we have an If action which branches on the value of g_ExitApp. If g_ExitApp is true, we take steps to exit the application. Otherwise, we switch on the retrieved value of g_conferenceId. If it is anything but 0, we set g_AddingToConference to true, the user is not the first person to connect to the conference. We proceed to the PlayAnnouncement action, which asks the user to speak his name and company name, then sets off a series of events based on the contents of PlayAnnouncement (discussed above).

If g_conferenceId is 0, the currently connected user is the first one into the conference. We create a conference using CreateConnectionConference by specifying 0 for the conferenceId and g_incomingConnectionId for connectionId. When CreateConnectionConference is called with conferenceId set to 0, it creates a new conference, and returns the actual conferenceId as a result of the action. CreateConnectionConference puts whatever connectionId is specified into the conference with the specified conferenceId. Next, we play an announcement to the user, welcoming him to the conference.

The next step for both of the above branches is to increment the number of users currently in the conference, as retrieved from the database. We increment the numberOfParticipants variable by one, set g_InConference boolean to true. This variable lets us determine whether the user has been added to a conference or not. Next, we send a query to the database that updates the number of users in the conference.

If the ReceiveDigits action failed, OnReceiveDigits_Failed is launched. We exit the application, since this is not a recoverable error as far as this application is concerned.

If we get to OnRecordAudio_Complete, it means that the user successfully recorded his name and company information. We add him to the conference. If this fails, we exit the application. Otherwise, we Switch on the terminating condition of the RecordAudio action. If the action terminated because of silence, we play a standard greeting. Otherwise we play whatever the user actually said. If the RecordAudio action fails, we enter OnRecordAudio_Failed. There, we attempt to add the user to the conference anyway, so

he may at least listen. If the add is successful, we announce the user to the conference, otherwise we exit.

OnHangup_Complete and OnHangup_Failed are handlers for the Hangup action. We use the Hangup action through the application in order to hang up the call. Regardless of whether the Hangup succeeds or fails, we call the custom function PerformHangup.

OnHangup is invoked when the user hangs up. Again, PerformHangup is called.

In PerformHangup, the first thing we do is check if the user is in a conference. If he is not, all we do is delete the connection and exit. If he is in a conference, we retrieve the current number of participants in the conference from the database, and decrease it by one. Next, we check to see if there is only one person remaining in the conference. If this is true, we delete the conference entry from the database, delete the connection, and exit. Otherwise, we send a query to the database, updating the number of participants, delete the connection, and send a PlayAnnouncement to the conference, notifying other conferees that a user has left. We then exit the application.

Closing Remarks

We have used a number of basic Metreos Visual Designer components to create a functional scheduled conference application. We have demonstrated how the Database native actions can be used to keep track of state. It should however be noted, that additional precautions need to be taken before this application is put into a production environment. The biggest issue that would need to be addressed is the possibility of database concurrency issues. It is possible, however unlikely, that two users could exit the conference at exactly the same time, both having retrieved the same number of current conference participants. Both would decrement that number by one, and both would write the same number to the database. At that point, the number of participants as recorded in the database would be higher than the number of participants in the conference. This is not a big issue from the standpoint of the Metreos Application and Media servers. The application will exit regardless, terminating on the Application Server, and the Media Server automatically removes empty conferences. This would mainly be an issue if any additional functionality was required from the scheduledconferences table.