Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

Level: Intermediate
Works with: Sametime
Updated: 03-Mar-2003

by John W Rooney
and Eben Stewart

In the *LDD Today* article, "**Building your own Sametime bots, Part 1**," we focused on how you can use an automated agent (known as a *bot*) to generate responses to user inquiries. In this month's article, we'll take this idea to the next level and show you how to connect your application to Sametime and use Instant Messaging (IM) as an alert system—and even work it into your workflows. The key point we will demonstrate is how an application can initiate the conversation with the user, rather than the other way around.

We'll start by discussing the important Sametime concepts of presence and awareness and how you can incorprate these into automatic alert and workflow programs. We'll then examine in detail AlertBot, a working Sametime bot that shows basic techniques needed to proactively send messages to users. (You can download the code for AlertBot from the **Sandbox**.)

This article assumes that you're an experienced Sametime and Java programmer, familiar with Sametime's developer toolkits.

A brief review of IM concepts

When we first rolled out Sametime instant messaging (IM) within IBM a few years ago, we initially encountered the normal resistance users feel towards using any new technology as a key tool for communication. Our email delivery within the company was highly optimized, frequently used to engage in "near real-time" dialogues with colleagues throughout the world. Given this, users often asked how IM differed from their email experience and what advantage could be gained by using it.

Today of course, we're all very familiar with how IM significantly changes user interactions. The most obvious differences between IM and email involve *presence* and *awareness.*We often merge these two concepts when speaking of IM in the enterprise, but for the sake of this article, let's consider them separately.

To begin, let's think about how presence and awareness relate to bots. When we speak with users about bots, we often highlight a number of important facts. For example, it's easy for users to see how bots provide a convenient way to work within the Sametime Connect client. But convenience is only the first part of the bots story. Using presence and awareness capabilities in Sametime, we can integrate IM deeply into our applications. In effect, not only can you know when the bot is online, but the bot can "know" when you are online and what your current online status is. Your applications can track changes in user status (such as moving from "away" to "active") and use this as a trigger to perform actions such as sending you a message. So let's dive a bit deeper into presence and awareness and see how you can incorporate these features into a Sametime bot that performs alert and notification services.

Presence

Presence is the ability for you to express your current state of activity to others on the network. In Sametime, these states can be:

- Online - Available
- Online - Away
- Online - Do Not Disturb
- Offline

This is an extremely powerful feature because it lets you know whether or not a particular user is available and able to respond to a message. With email and other types of messaging, you often send a note with no idea whether or not the recipient is presently there to read it. You may get a quick response, but you may not—a critical difference between email and IM.

Many users point out that the content of email is richer because unlike with IM you can attach files, embed images, and so on. This is true (at least for now, given the current state of IM technology), but that's not our point. We're not trying to promote IM as a complete replacement to email. Instead, IM augments email and other messaging systems—indeed, IM and email can be integrated into a single application such as **NotesBuddy**. But in a world where time is becoming an increasingly scarce commodity, the need to know whether or not a specific person is available at this moment can be an important component of a business-critical application.

Awareness
Awareness is the ability to sense the presence of others in an IM environment. As with presence, this boils down to letting the initiator of a chat know the likelihood of getting an immediate response. Awareness also allows the initiator to make decisions about the medium to use when conducting conversations. For example, let's say you are working on a presentation with a colleague, and you need to confirm the latest sales forecast for the current quarter. You expect this to be a simple conversation, so you open Sametime Connect to send her an instant message. However, when you look at her name, you see her status listed as Do Not Disturb—so you can't start a conversation with her. Additionally, she has left a message stating she is in an all-day customer briefing. With these two pieces of data in hand, you can now make a decision: If you don't need the forecast until tomorrow, you send her an email. If you require this information today, you can seek out another co-worker to obtain it. This is a simple example, but it demonstrates how being aware of another user's IM status affects your business process. This is very powerful, and we'll present you with examples of how applications can leverage awareness.

One last thing that we'd like to highlight here is the "psychological" impact that different message types have on the recipient and how we prioritize our responses to each. We all share in the sense of being overloaded with email. Every day our inboxes are flooded with numerous messages to sort through. Most email programs have tools to help handle this deluge (for instance, folders and rules in Notes). But it can still be difficult to keep up at times. Is it any wonder that we often hear users complain that they "live in their email"? IM, on the other hand, conveys the impression of "light conversation," immediate and easy. Plus, in our experience rolling out Sametime within IBM, we have seen numerous examples in which IM frees the user to multitask and to carry on several simultaneous conversations. Think about it—if you're in the middle of an important task, which are you more likely to respond to: an instant message or an email? Most users would probably answer the instant message first, even when the topic of conversation may seem less important. The message pops up on your screen, you have your dialogue, solve the problem, and move on. Additionally, it's easy to invite others into the conversation, expand the discussion, and improve your response time to inquiries. To paraphrase Marshall McLuhan, "The medium is often the message."

Integrating IM into your applications

Now let's talk about how you can take advantage of presence and awareness in your applications and integrate IM even more into your business processes, in particular user alerts and workflows.

Alerts
First, let's discuss using Sametime to deliver server-generated alerts to users. We all have applications that require a user to be notified of some event, for instance, a change in state or an exception to an anticipated process. An obvious example of this is the alert sent to administrators when a server crashes or when it reaches a performance threshold. (IBM's strategy for autonomic computing will, of course, make such alerts far less common and important in the future, but let's keep the administrator in the loop for the moment.) Such alerts can be automatically delivered to cell phones, pagers, or email.

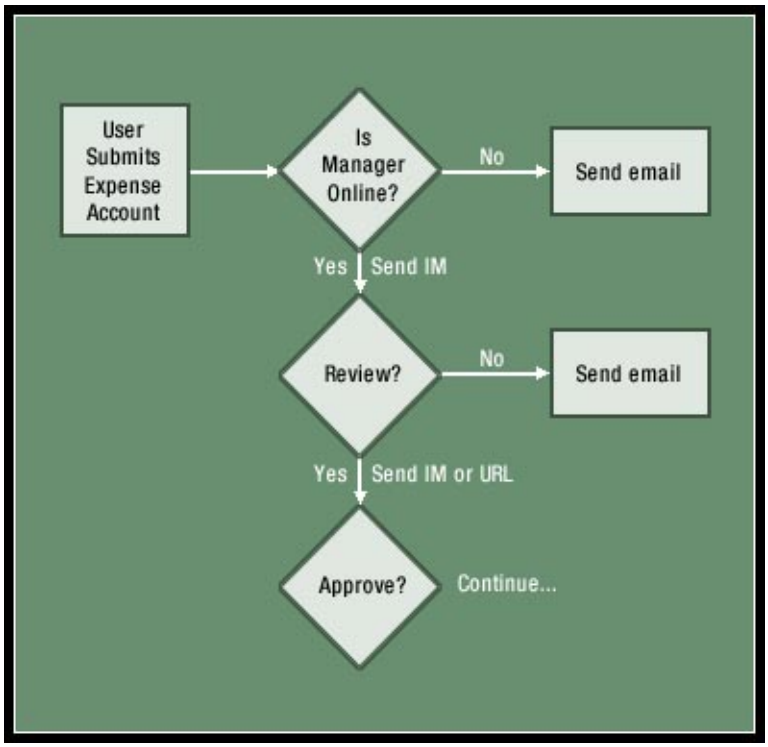
We can also deliver these via IM. We can add code to our applications that take a message, connect to Sametime to discover the intended recipient's state, and determine if a message can be routed to the user through Sametime. If the user is online and available, your application can send the user an instant message. If the user is in any other state, your application can select the next best path (such as email or text pager) to contact this user.

In early releases of Sametime, the message you sent opened a chat window on the user's desktop. With

Sametime 3, we now have another message type available to us—Announcement—which opens a dialog box on the desktop without the ability for the user to respond. This can be very useful if we don't want the user to carry on a conversation with the application generating the alert.

Workflows
We all have had at least some exposure to the concept of workflow within applications. Most of us participate in workflow management systems via forms routed through email. With the recent focus on moving applications to the Web, our workflow systems often route form summaries in email with a URL the user can click to perform the appropriate action. In general, these applications work well, but IM can provide a nice alternative when we need to put the information right in front of the user.

Let's look at a decision tree for how IM can be incorporated into a workflow:



When we discuss using IM to drive workflow processes, some have questioned the value of approving forms within Sametime when the old "tried and true" email-based systems seem to suffice. The key point here is to find the balance that suits your application. Keep in mind that you don't need to use IM to route all form information. If your application presents forms via a desktop application, you can simply use Sametime to send an alert as described in the previous section. If your application presents a Web interface to the form, you can send the approver a URL to click to open the form.

Of course, we encourage you to consider ways to allow the approver to indicate approval through the Sametime interface. The messages delivered by Sametime are currently limited to text, but that usually doesn't impact your ability to convey relevant information. Also keep in mind that Sametime conversations can be authenticated and encrypted. You can trust that the information is received by the correct user and that the content of the message is secure.

The AlertBot

Our sample AlertBot (which you can download from the **Sandbox**) demonstrates basic techniques needed for a bot to use both presence and awareness to proactively send messages to users. This can extend the capability of any bot to move beyond a simple query/response system into a full-featured application. In this example, the AlertBot watches the online status of a particular person and sends an alert when his/her status has changed.

AlertBot demonstrates the following:

- Resolving user identities (user IDs) to Sametime user objects
- Watching the online status of a Sametime user
- Initiating a conversation with a Sametime user

In discussing the AlertBot, we'll skip over basics such as logging in, using listeners, and sending and receiving text. These were covered in our **previous article**, which you can consult if you want to review these topics.

Let's look at some new objects that deal specifically with identifying the user and maintaining awareness. In the following two lines, the LookupService provides the Resolver object. The Resolver object reads a user ID as a string, and "resolves" it to an actual object that can be used to initiate a conversation:

```
protected LookupService lookupService;
protected Resolver resolver;
```

The AwarenessService provides the WatchList object. WatchList maintains the list of watched users and provides awareness messages (through the AwarenessListener interface):

```
protected AwarenessService awarenessService;
protected WatchList watchList;
```

To use these objects, you must first create and initialize them. This is usually done in the loggedIn( ) event (in the LoginListener Interface). First create the LookupService object. Then create a Resolver object from the LookupService and add a listener to it. (See the **Sametime Java API documentation** for a full explanation of the method parameters.)

```
//Get a handle to the Lookup Service and add a resolve listener
lookupService = (LookupService) stsession.getCompApi(LookupService.COMP_NAME);
resolver = lookupService.createResolver(true, false, true, false);
resolver.addResolveListener(this);
```

Next create the AwarenessService. Then create a WatchList object, and add a listener to that as well:

```
//Get a handle to the Awareness Service and create a WatchList
awarenessService = (AwarenessService) stsession.getCompApi(AwarenessService.COMP_NAME);
watchList = awarenessService.createWatchList();
watchList.addStatusListener(this);
```

In our example, the user is specified in the code and is immediately resolved. In more advanced applications, the user name list can be driven by the application itself. The resolve can be done at any time, not just at login time. For now, we just make a call to the Resolver to resolve our one user:

```
resolver.resolve(UTW); // UTW is the String user ID Constant
```

As always, make sure to remove these listeners, and clean up the Resolver and WatchList objects when the loggedOut( ) event is called:

```
resolver.removeResolveListener(this);
watchList.close();
watchList.removeStatusListener(this);
```

The ResolveListener interface consists of three methods. The only one we use in AlertBot is resolved( ). In this method, the string user ID has been resolved to a usable object, STUser. The STUser object is the primary object used by Sametime to represent a valid logged-in user. This STUser object is used by the WatchList to start monitoring the status of this STUser object. All that needs to be done here is to add the STUser object to the WatchList:

```
public void resolved(ResolveEvent re) {
        if (re.getResolved() instance of STUser) {
                STUser user = ((STUser) re.getResolved());
                String userName = user.getName();
                sysOut("Resolved " + userName);
                watchList.addItem(user);
                sysOut("Added " + userName + " to WatchList");
```



```
        }
}
```

The AwarenessListener has two event methods, only one of which we are concerned with: userStatusChanged( ). This method is called whenever the status of the watched user changes, including the first time the user is added to a WatchList. We also verify the current status of the user (only send an alert if the user is either Active or Away).

IM objects are created through the ImService using the createIm( ) method. Note that when the IM object is created, an ImListener must be added before the IM is opened to ensure that we receive the ImOpened( ) event:

```
public void userStatusChanged(StatusEvent se) {
        //We get an array of objects representing the users
        Object wu[] = se.getWatchedUsers();

        STWatchedUser stwu = (STWatchedUser) wu[0];
        sysOut(stwu.getName() + " status changed to " + stwu.getStatus().getStatusDescription());
                if (stwu.getStatus().getStatusType() == STUserStatus.ST_USER_STATUS_ACTIVE ||
                stwu.getStatus().getStatusType() == STUserStatus.ST_USER_STATUS_AWAY) {
                //Send an alert to the user
                sysOut("Alert called for " + stwu.getName());
                //Set the IM Type to IM_TYPE_CHAT
                int imt = ImTypes.IM_TYPE_CHAT;
                //Set the encryption level
                EncLevel enc = EncLevel.ENC_LEVEL_ALL;

                //Create the channel to the user
                Im im = imService.createIm(stwu, enc, imt);
                Im.addImListener(this);
                im.open();
        }
}
```

The ImOpened( ) event is very similar to the ImReceived( ) event described in our previous article, but ImOpened( ) is generated when the IM object is opened locally, while ImReceived( ) occurs when the IM is opened remotely. In the ImOpened( ) event, we have a known valid STUser object and an open IM object. Now we can send text, using the IM object. For this example, we tell the user what his/her current status and status message is:

```
public void ImOpened(ImEvent ie) {
        sysOut("IM Opened to " + ie.getIm().getPartner().getName());

        //To get the user status, we need to cast the IM object as an STWatchedUser
        STWatchedUser stwu = (STWatchedUser) ie.getIm().getPartner();
        if (ie.getIm().isOpen()) {
                String text = "I see that your status has changed to " + (stwu.getStatus().getStatusType() ==
                STUserStatus.ST_USER_STATUS_ACTIVE ? "Active" : "Away");

                ie.getIm().sendText(false, text + " with the message \"" + stwu.getStatus().getStatusDescription() +
                "\"");
                sysOut("Message sent to " + ie.getIm().getPartner().getName());
                try {
                        // This is to make sure the message is sent before closing the IM

                        Thread.sleep(2000);
                                } catch (InterruptedException e) {
                }
                ie.getIm().close(STError.ST_OK);
        }
}
```

Summary

The presence and awareness features of Lotus Sametime can turn simple query/response bots into powerful applications. By expressing presence and leveraging awareness, your applications can initiate conversations with

and respond to users. Using these features within your applications can drive additional value from your Sametime environment and improve your response time to critical business issues. Please feel free to examine and adapt our AlertBot code. We hope that you take advantage of these ideas within your organization.

ABOUT THE AUTHORS

John W. Rooney is a Manager of Internet Technology Development at IBM. Rooney (as his friends call him) and the Webahead/Internet Technology Group have developed numerous applications used widely within IBM, including several that rely heavily on Sametime bot technology.

Eben Stewart is a Software Engineer in the Webahead/Internet Technology Group at IBM. He is the primary developer of the BotServer, the toolkit used by IBM to provide bot applications for internal use, such as Bluepages (the corporate directory bot).

