Introducing the ToolTalk Service 1

The ToolTalk service enables independent applications to communicate with each other without having direct knowledge of each other. Applications create and send ToolTalk messages to communicate with each other. The ToolTalk service receives these messages, determines the recipients, and then delivers the messages to the appropriate applications, as shown in Figure 1-1.

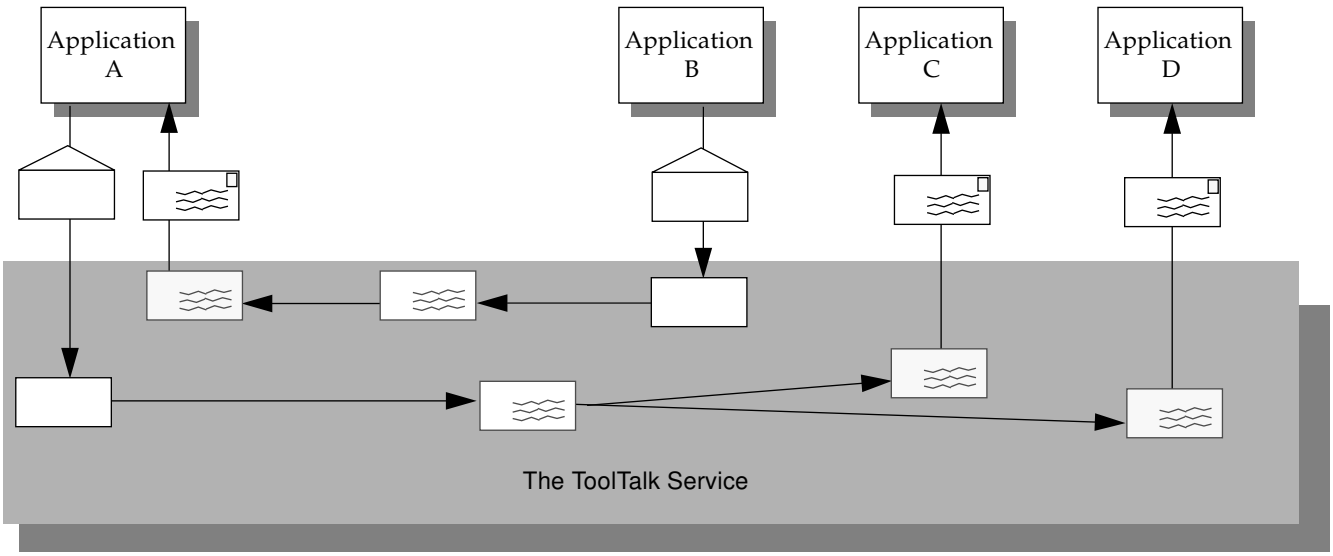

Figure 1-1 Applications Using the ToolTalk Service

1

1.1 ToolTalk Scenarios

The scenarios in this section illustrate how the ToolTalk service helps users solve their work problems. The message protocols used in these scenarios are hypothetical.

Using the ToolTalk Desktop Services Message Set

The ToolTalk Desktop Services Message Set allows an application to integrate and control other applications without user intervention. This section illustrates two scenarios that show how the Desktop Services Message Set might be implemented.

The Smart Desktop

A common user requirement for a graphic user interface (GUI) front-end is the ability to have data files be aware (or "know") of their applications. To do this, an application-level program is needed to interpret the user's requests. Examples of this application-level program (known as *smart desktops*) are the Apple Macintosh finder, Microsoft Windows File Manager, and the Solaris File Manager. The key common requirements for smart desktops are:

1. Takes a file
2. Determines its application
3. Invokes the application

The ToolTalk Service encompasses additional flexibility by allowing classes of tools to edit a specific data type. The following scenario illustrates how the Desktop Services Message Set might be implemented as a smart desktop transparent to the end-user.

1. **Diane double clicks on the File Manager icon.**
   - The File Manager opens and displays the files in Diane's current directory.
2. **Diane double clicks on an icon for a data file.**
   a. The File Manager requests that the file represented by the icon be displayed. The File Manager encodes the file type in the *display* message.

b. The ToolTalk session manager matches the pattern in the *display* message to a registered application (in this case, the Icon Editor), and finds an instance of the application running on Diane's desktop.

Note – If the ToolTalk session manager does not find a running instance of the application, it checks the statically-defined ptypes and starts an application that best matches the pattern in the message. If none of the ptypes match, it returns failure to the File Manager application.

---

c. The Icon Editor accepts the *display* message, de-iconifies itself, and raises itself to the top of the display.

3. **Diane manually edits the file.**

Integrated Toolsets

Another significant application for which the Desktop Services Message Set can be implemented is *integrated toolsets*. These environments can be applied in vertical applications (such as a CASE software developer toolset) or in horizontal environments (such as compound documents). Common to both of these applications is the premise that the overall solution is built out of specialized applications designed to perform one particular task well. Examples of integrated toolset applications are text editors, drawing packages, video or audio display tools, compiler front-ends, and debuggers. The integrated toolset environment requires applications to interact by calling on each other to handle user requests. For example, to display video, an editor calls a video display program; or to check a block of completed code, an editor calls a compiler. The following scenario illustrates how Desktop Services Message Set might be implemented as an integrated toolset:

1. **George is working on a compound document using his favorite editor.** He decides to change the some of the source code text.

2. **George double clicks on the source code text.**

   a. The Document Editor first determines the text represents source code and then determines what file contains the source code.

   b. The Document Editor sends an *edit* message request, using the file name as a parameter for the message.

c. The ToolTalk session manager matches the pattern in the *edit* message to a registered application (in this case, the Source Code Editor), and finds an instance of the application running on George's desktop.

Note – If the ToolTalk session manager does not find a running instance of the application, it checks the statically-defined ptypes and starts an application that best matches the pattern in the message. If none of the ptypes match, it returns failure to the Document Editor application.

---

d. The Source Code Editor accepts the *edit* message request.

e. The Source Code editor determines that the source code file is under configuration control, and sends a message to check out the file.

f. The Source Code Control application accepts the message and creates a read/write copy of the requested file. It then passes the name of the file back to the Source Code Editor.

g. The Source Code Editor opens a window that contains the source file.

3. George edits the source code text.

Using the ToolTalk Document and Media Exchange Message Set

The ToolTalk Document and Media Exchange Message Set is very flexible and robust. This section illustrates three applications of the ToolTalk Document and Media Exchange Message Set:

- Integrating multimedia into an authoring application
- Adding multimedia extensions to an existing application
- Extending the *cut and paste* facility of *X* with a media translation facility

Integrating Multimedia Functionality

Integrating multimedia functionality into an application allows end-users of the application to embed various media types in their documents.

Typically, an icon that represents the media object is embedded in the document. Upon selection of an embedded object, the ToolTalk service automatically invokes an appropriate external media application and the object is played as illustrated in the following scenario.

1. Daniel opens a document that contains multimedia objects.
2. The window shows the document with several icons representing various media types (such as sound, video, and graphics).
3. Daniel double-clicks on the sound icon.
   A sound application (called a *player*) is launched and the embedded recording is played.
4. To edit the recording, Daniel clicks once on the icon to select it and uses the third mouse button to bring up an Edit menu.
   An editing application is launched and Daniel edits the media object.

Adding Multimedia Extensions to Existing Applications

The ToolTalk Document and Media Exchange Message Set also allows an application to use other multimedia applications to extend its features or capabilities. For example, a calendar manager can be extended to use the audiotool to play a sound file as a reminder of an appointment, as illustrated in the following scenario:

1. Karin opens her calendar manager and sets an appointment.
2. Karin clicks on an audio response button, which causes the soundtool to pop up.
3. Karin records her message; for example, “Bring the report.”

When Karin’s appointment reminder is executed, the calendar manager will start the audiotool and play Karin’s recorded reminder.

Extending the X Cut and Paste Facility

The ToolTalk Document and Media Exchange Message Set can support an extensible, open-ended translation facility. The following scenario illustrates how an extensible multimedia *cut and paste* facility could work:

1. Maria opens two documents that are different media types.
2. Maria selects a portion of *Document A* and cuts the portion using the standard *X*-windowing *cut* facility.
3. Maria then pastes the cut portion into *Document B*.
   a. *Document B* negotiates the transfer of the cut data with *Document A*.
   b. If *Document B* does not understand any of the types offered by *Document B*, it requests a *tagged media type*. *Document B* uses the tagged media type to broadcast a ToolTalk message requesting a translation of the media type to a media type it understands.
   c. A registered translation utility accepts the request and returns the translated version of the media type to *Document B*.
   d. The paste of the translated data into *Document B* is performed.

Using the CASE Interoperability Message Sets

The CASE Interoperability Message Sets allow an application to integrate and control other applications without user intervention. This section illustrates several scenarios that show how the CASE Interoperability Message Sets might be used.

User Scenario: Fixing Bugs

This scenario steps through a complete cycle of how to fix a bug for a released application. It begins with receiving the bug report and describes the process required to fix the problem.

1. **Larry receives a bug report that indicates his application has a problem.**
2. **Larry invokes his CASE environment.**
   A CASE user interface is displayed. The functions Larry wants to perform are available in this CASE user interface.

3. **Larry writes a test case that duplicates the failure stated in the bug report.**
4. **Larry selects the debug function to run the application against the test case.**
   a. A Debug request is sent.
   b. The messaging environment selects the debugging application. It does not find an instance of the application running in Larry's CASE environment, so it automatically starts a debugger.
   c. The debugging application receives the request and loads the binary.
5. **Larry tests the code and reviews the debugging state in the debugging window.**
   He finds a function call that appears to be passing incorrect arguments.
6. **Larry selects the edit function to edit this code.**
   a. The debugging tool sends an edit request.
   b. The source editor receives a message to edit the specified source file.
7. **Larry wants to modify the source code, so he selects the checkout function.**
   a. The source code editor sends a checkout request.
   b. The source code editor receives the checkout notification, and changes the buffer state to modifiable.
8. **Larry edits the source code to fix the bug and selects the Build function to build the application.**
   a. The build request is sent.
   b. The build application receives the build request, and performs the build.
   c. When the build completes, the build application sends the BuildFinished notice.
   d. The debugger receives the BuildFinished notice and reloads the newly built application binary.

9. **Larry retests the application to confirm the bug fix works.**

10. **Larry exits the CASE environment.**

    a. Quit requests are sent to the source code editor, debugger, version manager, and builder applications.

1.2 How Applications Use ToolTalk Messages

Applications create, send, and receive ToolTalk messages to communicate with other applications. Senders create, fill in, and send a message; the ToolTalk service determines the recipients and delivers the message to the recipients. Recipients retrieve messages, examine the information in the message, and then either discard the message or perform an operation and reply with the results.

Sending ToolTalk Messages

ToolTalk messages are simple structures that contain fields for address, subject, and delivery information. To send a ToolTalk message, an application obtains an empty message, fills in the message attributes, and sends the message. The sending application needs to provide the following information:

- Is the message a notice or a request? (that is, should the recipient respond to the message?)
- What interest does the recipient share with the sender? (for example, is the recipient running in a specific user session or interested in a specific file?)

To narrow the focus of the message delivery, the sending application can provide more information in the message.

Message Patterns

An important ToolTalk feature is that senders need to know little about the recipients because applications that want to receive messages explicitly state what message they want to receive. This information is registered with the ToolTalk service in the form of *message patterns*.

Applications can provide message patterns to the ToolTalk service at installation time and while the application is running. Message patterns are created similarly to the way a message is created; both use the same type of

information. For each type of message an application wants to receive, it obtains an empty message pattern, fills in the attributes, and registers the pattern with the ToolTalk service. These message patterns usually match the message protocols that applications have agreed to use. Applications can add more patterns for individual use.

When the ToolTalk service receives a message from a sending application, it compares the information in the message to the register patterns. Once matches have been found, the ToolTalk service delivers copies of the message to all recipients.

For each pattern that describes a message an application wants to receive, the application declares whether it can *handle* or *observe* the message. Although many applications can observe a message, only one application can handle the message to ensure that a requested operation is performed only once. If the ToolTalk service cannot find a handler for a request, it returns the message to the sending application indicating that delivery failed.

Receiving ToolTalk Messages

When the ToolTalk service determines that a message needs to be delivered to a specific process, it creates a copy of the message and notifies the process that a message is waiting. If a receiving application is not running, the ToolTalk service looks for instructions (provided by the application at installation time) on how to start the application.

The process retrieves the message and examines its contents.

- If the message contains a notice that an operation has been performed, the process reads the information and then discards the message.
- If the message contains a request to perform an operation, the process performs the operation and returns the result of the operation in a reply to the original message. Once the reply has been sent, the process discards the original message.

1.3 ToolTalk Message Distribution

The ToolTalk service provides two methods of addressing messages: *process-oriented messages* and *object-oriented messages*.

Process-Oriented Messages

Process-oriented messages are addressed to processes. Applications that create a process-oriented message address the message to either a specific process or to a particular type of process. Process-oriented messages are a good way for existing applications to begin communication with other applications. Modifications to support process-oriented messages are straightforward and usually take a short time to implement.

Object-Oriented Messages

Object-oriented messages are addressed to objects managed by applications. Applications that create an object-oriented message address the message to either a specific object or to a particular type of object. Object-oriented messages are particularly useful for applications that currently use objects or that are to be designed around objects. If an existing application is not object-oriented, the ToolTalk service allows applications to identify portions of application data as objects so that applications can begin to communicate about these objects.

Note – Programs coded to the ToolTalk object-oriented messaging interface are not portable to CORBA-compliant systems without source changes.

Determining Message Delivery

To determine which groups receive messages, you *scope* your messages. Scoping limits the delivery of messages to a particular session or file.

Sessions

A *session* is a group of processes that have an instance of the ToolTalk message server in common. When a process opens communication with the ToolTalk service, a default session is located (or created if a session does not already

exist) and a *process identifier* (*procid*) is assigned to the process. Default sessions are located either through an environment variable (called process tree sessions) or through the X display (called X sessions).

The concept of a session is important in the delivery of messages. Senders can scope a message to a session and the ToolTalk service will deliver it to all processes that have message patterns that reference the current session. To update message patterns with the current *session identifier* (*sessid*), applications join the session.

Files

A container for data that is of interest to applications is called a *file* in this book.

The concept of a file is important in the delivery of messages. Senders can scope a message to a file and the ToolTalk service will deliver it to all processes that have message patterns that reference the file without regard to the process's default session. To update message patterns with the current file path name, applications join the file.

You can also scope a message to a file within a session. The ToolTalk service will deliver the message to all processes that reference both the file and session in their message patterns.

Note – The file scoping feature is restricted to NFS and UFS file systems; it does not work, for example, across tmpfs filesystems.

1.4 Modifying Applications to Use the ToolTalk Service

Before you modify your application to use the ToolTalk service you must define (or locate) a ToolTalk *message protocol*: a set of ToolTalk messages that describe operations applications agree to perform. The message protocol specification includes the set of messages and how applications should behave when they receive the messages.

To use the ToolTalk service, an application calls ToolTalk functions from the ToolTalk application programming interface (API). The ToolTalk API provides functions to register with the ToolTalk service, to create message patterns, to send messages, to receive messages, to examine message information, and so on. To modify your application to use the ToolTalk service, you must first include the ToolTalk API header file in your program. You also need to modify your application to:

- Initialize the ToolTalk service and join a session.
- Register message patterns with the ToolTalk service.
- Send and receive messages.
- Unregister message patterns and leave your ToolTalk session.