© 1991 by SunSoft—Printed in USA.
2550 Garcia Avenue, Mountain View, California 94043-1100

All rights reserved. No part of this work covered by copyright may be reproduced in any form or by any means—graphic, electronic or mechanical, including photocopying, recording, taping, or storage in an information retrieval system— without prior written permission of the copyright owner.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 (October 1988) and FAR 52.227-19 (June 1987).

The product described in this manual may be protected by one or more U.S. patents, foreign patents, and/or pending applications.

TRADEMARKS

The Sun logo and Sun Microsystems are registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

NFS, OpenWindows, and ToolTalk are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademarks of UNIX System Laboratories, Inc.

All other products or services mentioned in this document are identified by the trademarks, service marks, or product names as designated by the companies who market those products. Inquiries concerning such trademarks should be made directly to those companies.

Addressing

Although messages are ultimately delivered to processes, they can be addressed to processes, procedures, objects, or object types. Each ToolTalk process that receives messages has a procid (process identifier) that is unique within a session. A sender that knows a receiver's procid can send a message to that process.

However, it is unusual for one process to know another's procid; more often a sender doesn't care which process executes an operation (request message) or learns of an event (notice message). A procedure message is received by any process that has registered a matching pattern; the sender specifies an operation and arguments, leaving the ToolTalk service to find one handler and all matching observers.

An object message specifies an operation, arguments, and an object identifier (objid). Each object has a unique objid. Given an object message, the ToolTalk service looks up the object's type, determines which program implements the operation specified in the message, starts a process running that program if one is not already running, and delivers the message. As with procedure messages, an object message is delivered to at most one handler and to all interested observers.

Sometimes a sender does not know an object's objid. Such situations are accommodated by object type messages, in which the sender specifies the object's type, and the ToolTalk service delivers the message to a process that implements the operation for the type.

A request, as shown in Figure 4, takes a round-trip from sender to handler and back; copies of the message take one-way side trips to observers. The ToolTalk service delivers a request to, at most, one handler. The handler adds results to the message and sends it back. Other processes can observe a request before it is handled, after, or both; observers absorb a request without sending it back.

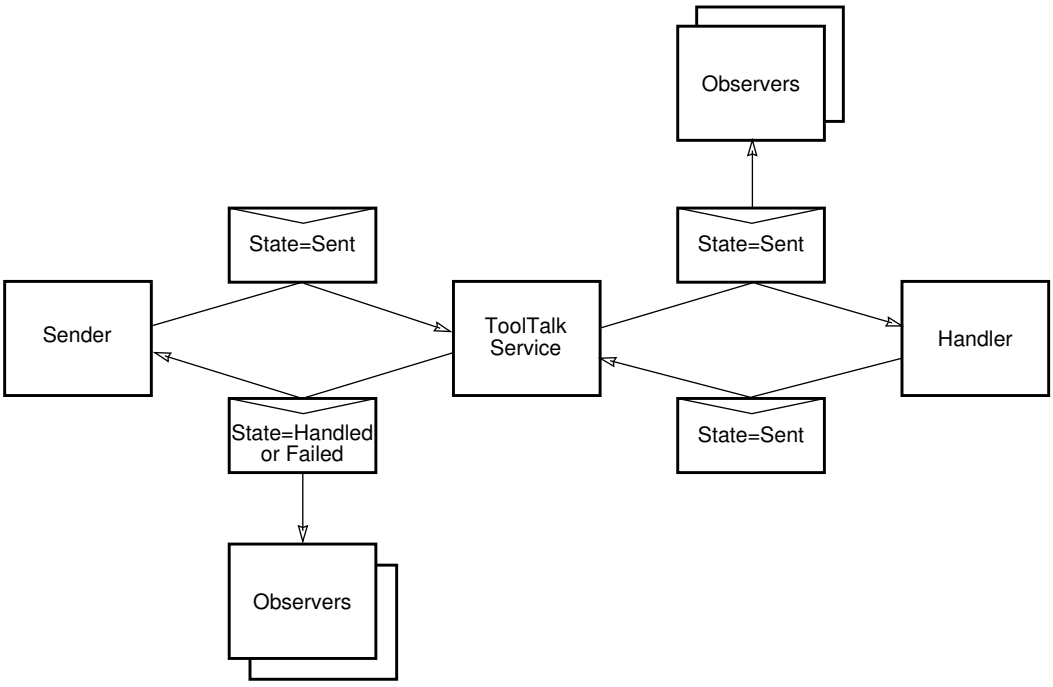

Figure 4 Request Routing

Handlers and Observers

For each pattern describing a message it wants to receive, a process also declares whether it can handle the message or merely wishes to observe it. Both requests and notices can be handled or observed. Although many processes can observe a message, only one process can handle it; thus, a requested operation is performed at most once. If the ToolTalk service cannot find a handler for a request message, it returns the message to its sender with a state of failed. Because notices do not return results, an unhandled notice is not an error.

Routing

A notice takes a one-way trip, as shown in Figure 3. The sender creates a message, fills in attribute values, and sends it. The ToolTalk service matches message and pattern attribute values, and gives a copy of the message to one handler and to all matching observers. File-scoped messages are automatically transferred across session boundaries to processes that have declared interest in the file.

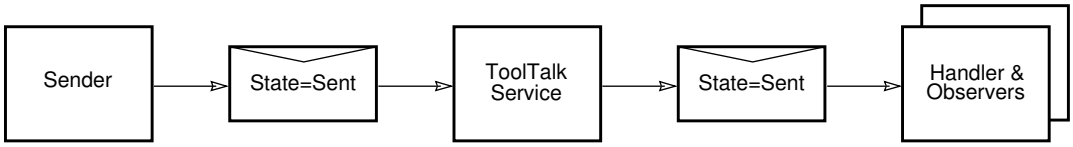

Figure 3 Notice Routing

Messages

Processes use the ToolTalk service to send each other messages. The set of messages that a group of processes exchange, together with the definitions of when messages are sent and how they are processed when received, constitute a message protocol.

Within a message's scope, the ToolTalk service determines the actual receivers by attribute-matching. Each message consists of attributes such as those shown in Table 1.

Table 1 Basic Message Attributes

Attribute	**Value**
Address	procedure, process, object, object type
Class	request or notice
Operation	action to be performed, or event that has occurred
Arguments	input values for operation, output results
Scope	session, file, both, file-in-session
State	created, sent, handled

Sending processes supply values for message attributes. Receiving processes specify, in the message patterns they register, the attribute values of messages they wish to receive. The ToolTalk service compares message attributes to receiver patterns, and delivers copies of a message to the processes that have registered matching patterns.

Notices and Requests

Processes can send two classes of ToolTalk messages, notices and requests. A notice is informational, a way for a process to announce an event. Processes that receive a notice absorb the message without returning results to the sender. A request is a call for an action, with the results of the action recorded in the message, and the message returned to the sender as a reply.

Each object is of some type; an application who creates and manages a certain type of object defines and implements the operations for that type.

An object-based application receives messages that are sent to its objects. If no application that manages a certain type of object is running in the sender's session, the ToolTalk service starts one automatically. To execute the operation specified in the message, the application retrieves the object's data, performs the operation, and updates the message with the result of the operation. The ToolTalk service returns the updated message, called a reply, to the process that requested the operation.

Object data is stored in two parts as shown in Figure 3. One part is called the object contents, and is managed by the application that creates or manages it. An object's contents is typically a piece, or pieces, of an ordinary file: a paragraph, a source code function, or a range of spreadsheet cells, for example. The second part of an object's data is called the object spec (for specification), and is managed by the ToolTalk service. An object's spec contains standard properties, such as the object's type, the name of the file in which the object's contents are located, and the object's owner.

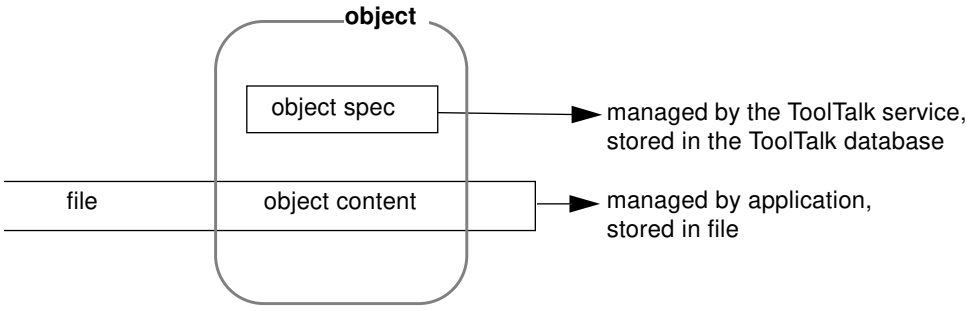

Figure 3 ToolTalk Object Data

By calling ToolTalk functions, applications can direct the ToolTalk service to create new specs, to retrieve property values from specs, to store application-defined properties in specs, to return the identifiers of all specs associated with a file, and so on. Because applications can store additional object data in specs, they can make objects of pieces of existing files without changing the formats of the files. They can as well make objects of pieces of read-only files.

How The ToolTalk Service Works

To deliver messages, the ToolTalk message service interacts with four principal entities, programs (sometimes called applications or tools), processes, objects, and messages.

Programs and Processes

One execution of an application, tool, or program that calls ToolTalk API functions is a *process*. A ToolTalk process is a SunOS process about which the ToolTalk service has some extra information.

Processes are grouped into *sessions*, a session usually representing the processes running in behalf of one user, regardless of where the processes are located. Users typically create a session by a command in an initialization file such as the `.xinitrc` file. Users can start additional sessions, for example, when logged in remotely from a terminal.

The ToolTalk service tracks the members of each session so it can deliver session-scoped messages. Each ToolTalk message has a scope that localizes its distribution; a message whose scope is a session can only be delivered to processes in the same session as the message sender. The other scope is called *file*; a file-scoped message can only be received by processes that have expressed their interest in the file named in the message. Messages can also be scoped to both a session and a file or a file within a session.

An application developer can instruct the ToolTalk service to consider the application as a potential message receiver when no process is running the application. This is done by describing the messages of interest in a process type, a file that is stored when an application is installed.

Objects

In addition to traditional interprocess messages, the ToolTalk service supports object-oriented message passing. A process that addresses a message to an object invokes an operation on that object, just as an object-oriented program invokes a method on an object. In contrast to most object-oriented programming languages, ToolTalk objects may be located anywhere in the network. Also unlike most object-oriented programming languages, a method is invoked by one process (the message sender) and executed by another process (the message receiver).

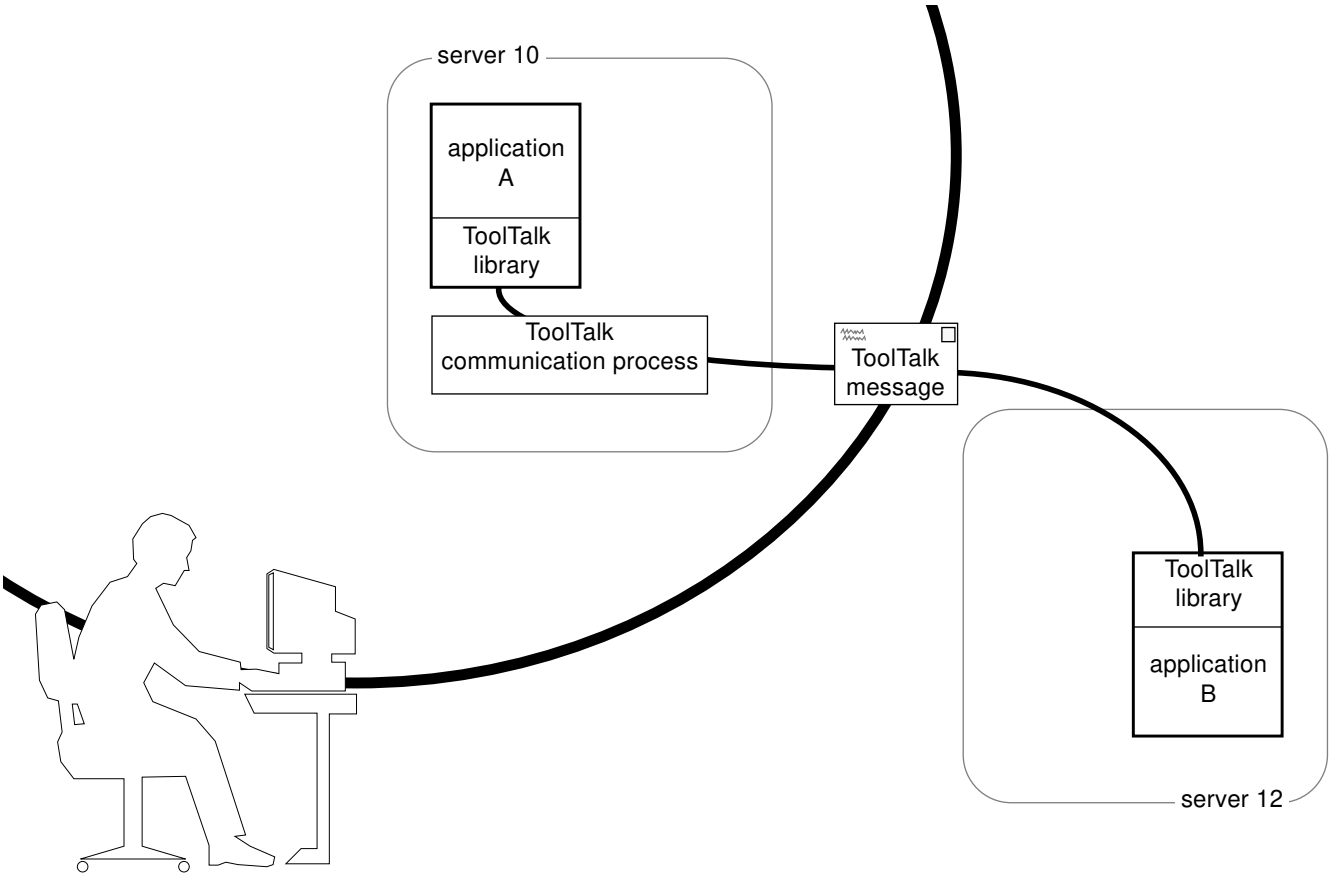

Figure 2 ToolTalk Messaging Between Applications In A Session

messages it wishes to receive beyond the list in the message protocol, for example, an application may want to observe all messages about a particular file.

Applications using the ToolTalk service to communicate usually have something in common – the applications are running in the same user session or they're interested in the same file or data. Applications register this interest in sessions or files (or both) with the ToolTalk service. This file and session information is used in conjunction with the message patterns by the ToolTalk service to determine what application should receive a message.

For example, suppose one application wants to inform all interested applications running in a user's session that an event has occurred. Because network window systems, such as the one within the OpenWindows™ system, allow users to run applications on multiple machines, it is difficult for the sending application to know which applications should be notified and where they are running. Using the ToolTalk library functions, the sending application simply sends a message scoped to the session; the ToolTalk communication process delivers copies of the message to all applications in the session that have registered a pattern that matches the message, no matter where the applications are running, as shown Figure 2.

When applications communicate, end users gain greater freedom in choosing the application that best suites their needs. Today, a single company can offer a set of applications that may or may not interact among themselves. Users choose sets of applications based on trade-offs of the entire set of applications, not on the basis of whether the individual application meets the unique needs for the portion of the problem the application solves. As users gain the freedom to choose applications that interoperate, they can tailor their environment for their unique work situation.

The distributed object paradigm results in even greater freedom for users to customize their environment. Users and developers customize applications by putting together reusable services that perform operations on objects. Focus is shifted from the application to the information or data being manipulated.

Using The ToolTalk Service

To communicate, applications create, send, and receive ToolTalk messages. Sending applications create, fill in, and send a message; the ToolTalk service determines the recipient and delivers the message to the receiving application.

ToolTalk messages are simple structures containing fields for address, subject, and delivery information. To send a ToolTalk message, an application need only provide this information:

- Should the recipient respond to the message; is this a notice or a request?
- What operation should be performed/has been performed? Are there arguments needed to perform the operation?
- What interest does the recipient share with the sender; is the recipient running in a specific user session or interested in a specific file?
- (For object-oriented messages) What object or type of object is involved in this operation?

An important ToolTalk feature is that sending applications need to know little about the receiving application. This is because applications that want to receive messages explicitly state what these messages should look like. This information is registered with ToolTalk in the form of message patterns. These message patterns usually look very similar to the message protocols that applications have agreed to use. An application may add more types of

Multicast messaging provides additional value to an application by taking a step from independent applications that perform a portion of the end-users work toward applications that interoperate to solve whole problems. Instead of developing an entire set of applications that solve a problem, developers can create or enhance existing applications to perform the functions that they perform best. Because little knowledge of other applications is needed to communicate via the ToolTalk service, developers can concentrate on applications that "plug and play" with other vendor's applications. This is particularly important because it means companies do not have to cultivate relationships with every vendor they need to communicate with, nor do they have to change their code every time a new application is added to the set of applications that interoperate.

To use multicast messaging, developers add ToolTalk calls to their applications to send and receive messages; the application does not need to be restructured. The ToolTalk service was designed to preserve investments in existing applications through a communications model that fits easily into existing applications.

Object-oriented messaging provides inter-application and inter-object communication for applications that create and manage objects. Even if an existing application is not object-oriented, the ToolTalk service provides a way for developers to identify portions of application data as objects so that applications can begin communicating about these objects. This step toward the object world provides additional functionality as well as experience in dealing with objects. As more applications move toward interchangeable objects and services, the ToolTalk service provides a transition path between existing applications with multicast message capabilities and new applications with object-oriented messaging.

End Users

As applications work together communicating about events that have occurred or operations that need to be performed, the user's workflow becomes automated. Instead of performing one task with one application and then inputting the results into another application to perform the next task, users focus on solving a whole problem, not performing a series of tasks that result in the problem's solution.

The ToolTalk Service Features

Some highlights of the ToolTalk service's capabilities are listed below:

- Message addressing is very flexible. A message can be addressed to a specific process, to a particular type of process, to an object, or to a process that can perform a particular operation on an object.
- Applications can send notice messages that announce events and request messages that ask for operations to be performed. Multiple processes may receive a copy of either kind of message for observation, but at most one is chosen by the ToolTalk service to execute a request.
- In addition to object-oriented messages, the ToolTalk service provides support for the objects themselves, including persistent description storage, application-defined properties, and automatic activation of applications to perform object operations.
- An application can register interest in receiving messages even though no process is running the application. It can further direct the ToolTalk service to start a process when a message is available, or to save the message until a process running the program is started by other means (typically by a user).

Who Benefits?

Application Developers

The ToolTalk service provides solutions meeting the requirements of two sets of application developers:

- Multicast[1] messaging for developers of procedural-based, self-contained applications
- Object-oriented messaging for developers of applications based on a distributed object paradigm

1. SunSoft's ToolTalk service provides multicast messaging, i.e. a single process sends a message that is delivered by the ToolTalk service to multiple receivers. Multicast messaging, with the concept of one-to-many communications, falls between broadcast messaging (one-to-all) and point-to-point messaging (one-to-one).

Both methods provide a unique opportunity for applications to bridge the communication gap and provide cross-application functionality to users.

The ToolTalk 1.0 service is built on top of SunSoft's ONC remote procedure call (RPC) product. While both provide communication capabilities, the ToolTalk service has a higher-level interface for application developers. While the initial ToolTalk technology is available on platforms running the ONC RPC, the service has been designed for portability to other platforms and RPC mechanisms.

Moving Toward Distributed Objects

As development trends move toward distributed objects, the ToolTalk service provides a transition path from today's procedure-oriented world to the object-oriented applications of the future. With the ToolTalk service, users can identify portions of their application data as objects, create object descriptions, and store this information with the ToolTalk service; Application data is not altered. Applications can then communicate with other applications about these objects.

As next-generation, object-oriented architectures such as those proposed by the Object Management Group emerge, SunSoft's commitment to providing distributed object facilities ensures a smooth migration path from the ToolTalk service's object facilities to new object environments. Developer's initial investment in ToolTalk objects will provide invaluable experience to help chart the move into the object-oriented world.

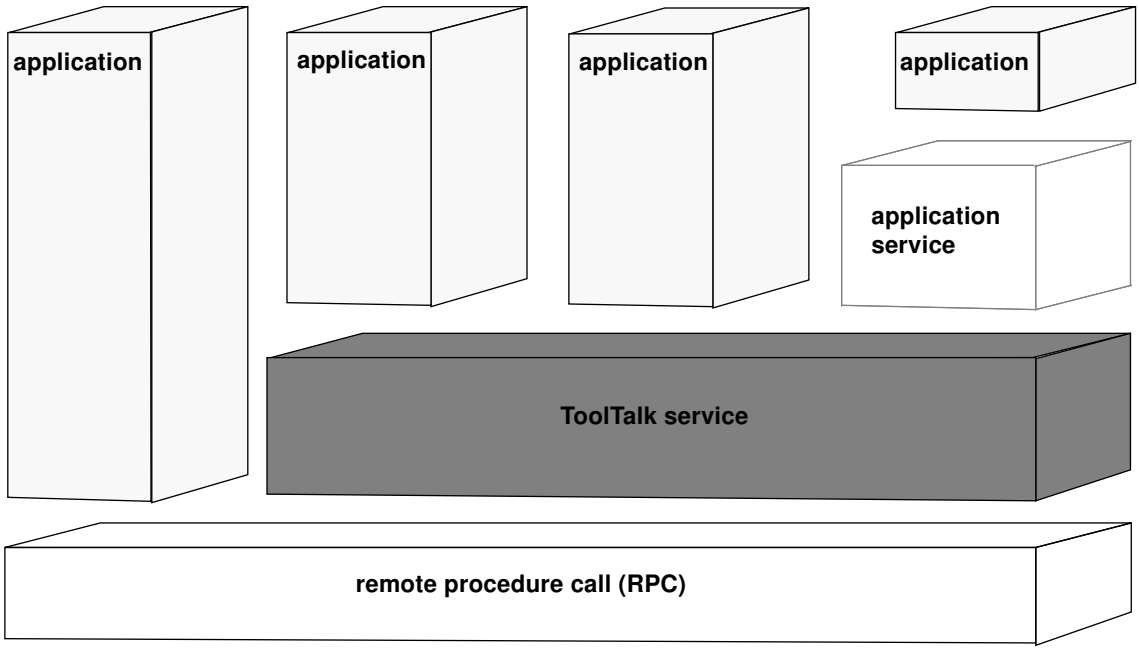

Figure 1 Application InterOperability Facilities

The ToolTalk service is a forerunner of application interoperability services that applications use to cooperate. It provides the first communication backbone that frees applications from needing to have direct knowledge of other applications before interacting with them.

The ToolTalk service is used in two ways:

- Applications use ToolTalk directly, calling functions from the ToolTalk application programming interface (API) library to create, send, and receive messages. Examples of these message may include messages to open, display or close application windows.
- Applications use a “service” built on top of the ToolTalk service. These application services use the ToolTalk service as a communication backbone and object manager. These types of services can provide linking, drag and drop, and other capabilities for application users.

Inter-Application Communication

The ToolTalk service provides a straightforward mechanism for applications to talk with each other. Applications that previously stood alone -- solving specific problems for their users -- can now interact with other applications, solving larger-scale problems. Some advantages of the inter application communication provided by the ToolTalk service are:

- Applications can do what they do best and ask other applications to perform the rest.
- inter-application communication is a small investment with the ToolTalk service; existing applications are easily modified to send, receive, and use shared information.
- No knowledge about the application receiving the information is needed – including its location on the network; the ToolTalk service determines which application receives the information by looking for applications who asked for the type of information being sent.
- Applications can communicate about objects; the ToolTalk service manages object descriptions and uses these descriptions to deliver information to the applications managing the object.

Applications that use the ToolTalk service cooperate by sending and receiving messages; they do not share stored data. The messages passed by cooperating applications form a small, well-defined message protocol that maximizes application autonomy. So long as message protocols are observed, cooperating applications can be modified, even rewritten, without affecting one another.

Where The ToolTalk Service Fits In

The ToolTalk service is one of the first commercial products in the new class of mechanisms designed to facilitate information exchange between applications from different developers. These mechanisms play a key role in application interoperability. Figure 1 shows the relationship the ToolTalk service has in SunSoft's initial framework for application interoperability.

For readers familiar with SunSoft's ONC ™ RPC (remote procedure call) mechanism, the ToolTalk service is a higher-level facility that is more flexible and easier to use. The first release of the ToolTalk service is, in fact, implemented atop RPC.

The ToolTalk Service

Executive Summary

As computer users increasingly demand that independently developed applications work together, “interoperability” and “integration” are becoming important themes for software developers. These terms convey the notion of inter-application cooperation: of applications achieving more by working together than by working separately. The ToolTalk™ service promotes cooperation among independently developed applications.

The ToolTalk service is a network-spanning interprocess message system. Cooperating application developers agree on a message protocol (the syntax and meaning of a set of messages), and use the ToolTalk service to deliver the messages. ToolTalk messages come in several styles, including, for example, procedural and object-oriented messages.

Applications create, send, and receive messages by calling ToolTalk functions from the ToolTalk application programming interface (API) library. The API gives the ToolTalk service the appearance of a simple library, hiding the details of its implementation; libraries included in applications interact with a collection of communicating processes running on machines throughout a local area network.

A primary feature of the ToolTalk service is that sending applications need to know little about the receiving applications or the network. Applications that want to receive messages explicitly state what these messages should look like in message patterns registered with the ToolTalk service. When the ToolTalk service receives a message to deliver, it compares the message attributes with the registered message patterns until a match is found. The ToolTalk service then locates the recipient that registered the matching pattern and delivers the message. Many applications may receive a copy of a message to observe the information but only one application “handles” the message.

The ToolTalk Service

A SunSoft White Paper

SunSoft, Inc.
A Sun Microsystems Subsidiary
2550 Garcia Avenue
Mountain View, CA 94043

Revision 01, June 1991