SunSoft, Inc.
2550 Garcia Avenue
Mountain View, CA 94043

For more information, call 1 800 227-9227.

Printed in USA 4/94 94079-001

The ToolTalkSession Scope Manager is a client of the ttsession — just like any other application in the session. The ttsession delivers the message to the ToolTalkSession Scope Manager, which acts as a representative for all MOS applications. The Scope Manager then delivers the message to interested application via the event channel.

Summary

The Messaging Object Service in the Project DOE environment provides many benefits to developers of inter-operating applications — and to the users of those applications. The inter-operability of existing ToolTalk applications with MOS-based applications on the Project DOE environment means that there are already an initial supply of applications available that are usable in the Project DOE environment.

The Messaging Object Service also leverages both the underlying ORB and the Object Services (such as the Event service, the Naming service, and Persistent Storage Management) that are part of the Project DOE environment. This leverage results in a number of capabilities in the Project DOE Application Environment, including:

- A more general message scoping mechanism. The distributed nature of the naming service will, for example, allow widely separated applications to communicate because there is a common mechanism (namespace) for scope lookup.
- A more versatile notification capability. The separation of event handling will allow message routing to better suit the needs of the application; for example, an application may want to use multiple notification channels.
- Persistent scopes. The persistence of objects and the manner in which objects are activated will eliminate the need for separate declarations of static patterns.
- Easier incorporation of existing (legacy) technology. Applications will be able to put “object wrappers” around existing software to take advantage of the functionality of other (non-object) applications.

When a MOS application sends a message into a ToolTalk scope, the ToolTalkSession Scope Manager examines the contents of the message. Figure 4 shows the relationship between the MOS application, the ToolTalkSession Scope Manager, the ttsession, and the ToolTalk application.

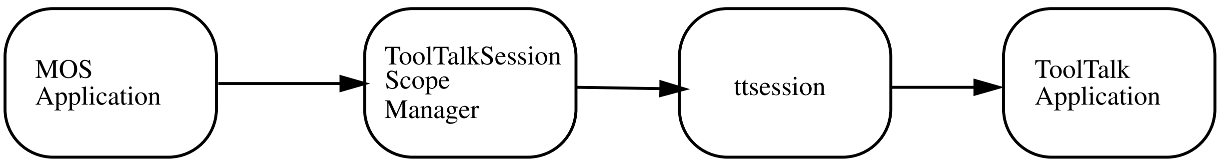

Figure 4 MOS-to-ToolTalk Inter-operability

If all parts of the message have analogs in ToolTalk, the ToolTalkSession Scope Manager determines that the destination of the message is a ToolTalk application. The ToolTalkSession Scope Manager then relays the message to the ttsession within the ToolTalk environment. The ttsession distributes the message just as it would distribute any ToolTalk message.

When a ToolTalk application sends a message to a MOS application, the application sends a message to the ttsession. Figure 4 shows the relationship between the ToolTalk application, the ttsession, the ToolTalkSession Scope Manager, the event channel, and the MOS application.

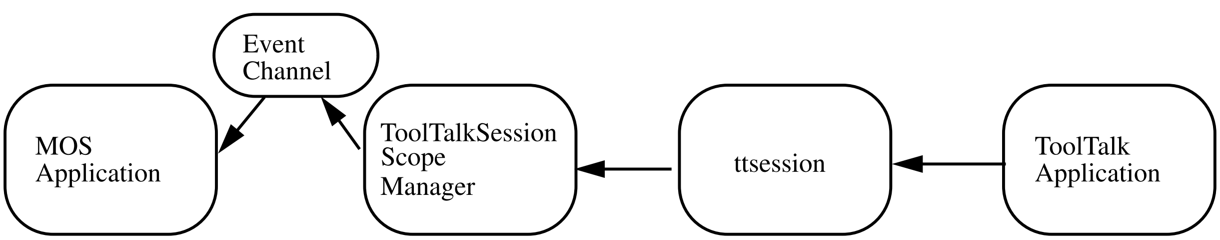

Figure 5 ToolTalk-to-MOS Inter-operability

What does this mean for the users of your ToolTalk-aware applications? It means that your customers will be able to continue to use your applications as they move into the new object paradigm. With MOS, you will be able to continue to use ToolTalk-compatible messaging in your existing applications as you enhance your applications to take advantage of the Project DOE distributed object facilities. This inter-operability means that you will be able to run a ToolTalk-aware application in the distributed object environment (and other CORBA-compliant environments that support the ToolTalk APIs and MOS interfaces). As additional components of the Project DOE become available, your existing ToolTalk-aware application interfaces (implemented as ToolTalk messages) will be available to other Project DOE components through the Messaging Object Service.

MOS applications can interact with ToolTalk applications, and vice-versa. From the point of view of a MOS application, it sends a message to the ToolTalk scope just as it would send a message to any other MOS scope. However, the MOS application uses ToolTalk language in its messages and patterns. From the point of view of the ToolTalk application, MOS is just another application within the ttsession. The interaction is transparent expect for the format of the messages.

In addition, MOS will inter-operate with ToolTalk on Solaris and ToolTalk on other platforms. ToolTalk and MOS will, therefore, provide distributed object capabilities to other platforms regardless of whether the platform has native distributed object capabilities.

Sending and Receiving Messages

A MOS application can send a message to or receive a message from a ToolTalk scope. The process for sending and receiving messages is the same as for a MOS scope except that, when a ToolTalk scope is specified, the Scope Provider finds a special ToolTalkSession Scope Manager that is designed to be bridge between MOS and ToolTalk.

All information communicated between MOS and ToolTalk applications is managed by the ToolTalkSession Scope Manager, which translates MOS messages and patterns to and from ToolTalk messages and patterns.

Inter-operating with ToolTalk

Open systems mean that developers are able to create more positive and productive computing environments. Project DOE technology represents SunSoft's support of open systems in the new object paradigm; the ToolTalk service represents SunSoft's support of open systems in today's (existing) computing environment. ToolTalk functionality is supported in the Project DOE environment through MOS.

If you develop an application that uses the ToolTalk message sets, you can be assured that you have made an excellent investment. Figure 3 shows how your investment in ToolTalk messages is protected in the Project DOE environment.

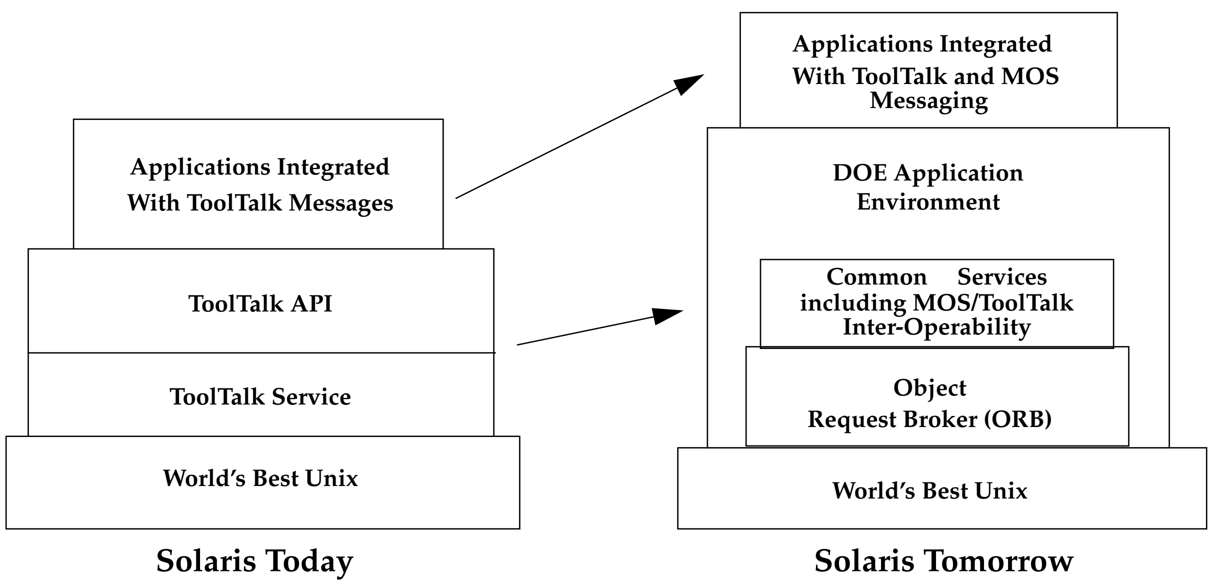

Figure 3 ToolTalk Investments are Protected in the Project DOE Environment

Existing ToolTalk applications will, of course, continue to interact with each other in the Project DOE environment. In addition, MOS enables existing ToolTalk applications to interact with *new object-based* applications. Inter-operability of applications from the current paradigm to the new object paradigm is provided. This provision means that the ToolTalk applications you are developing today can also work with Project DOE technology.

Trying a Message Before Sending It

The try message command allows you to determine the effect of a message without actually sending it. You can try the message, examine the results and, if the results are satisfactory, you can then send it.

Sending a Message

MOS enables an application to broadcast information in the form of a message, without needing to know which applications will receive the information. A message is sent to a scope, not to an individual application, so the Scope Manager orchestrates the delivery of the information in the message to interested applications. Table 13 describes the mossh commands used to send a message.

Table 13 mossh Commands to Send Messages

Command	Description
send message	Sends messages to a Scope Manager. When the Scope Manager receives the message, it matches the message against the patterns on its list. The Scope Manager pushes results to each event channel that corresponds to a matched pattern. The result remains in the event channel until it is retrieved.
dump channel	Displays the result. This command reads events off of the event channel and displays the associated results.

Ending a mossh Session

To end your mossh session, you need to first destroy all of the objects you have created, then quit the session. Table 14 describes these mossh commands.

Table 14 mossh Commands to End a Session

Command	Description
destroy	Destroys all the objects that mossh has created. You can also use the destroy command to destroy individual objects in a scope.
quit	Saves any object references that have not been destroyed and exits the mossh session.

Creating an Event Channel

The `mossh` commands described in Table 11 create and display event channels.

Table 11 mossh Commands to Create and Display Event Channels

Command	Description
make channel	Creates an event channel. You can also specify a maximum number of events that the channel receives, and a time-out value.
show channel	Analogous to the show message command. Enables you to examine the contents of an event channel.

Finding a Scope Manager and Registering Patterns

The Scope Manager acts as a central repository for all patterns from applications in the scope. Registering a pattern indicates which messages an application is interested in receiving by placing it on a list of registered patterns kept by the Scope Manager. Once you have registered a pattern, the Scope Manager does the work of matching incoming messages against the pattern and delivering results to the event channel you have specified. Table 12 describes the `mossh` commands to find a Scope Manager and register a pattern.

Table 12 mossh Commands to Find a Scope Manage and Register a Pattern

Command	Description
find scope	Locates or creates a Scope Manager. Option you can include a ToolTalk session id. Also, you can specify whether a new Scope Manager should be created if no scope exists.
register pattern	Registers a pattern, or pattern sequence, and the associated event channel with a given scope manager.
show scope	Analogous to the show message command. Enables you to examine the contents of a scope.

Registering a pattern requires three objects: an event channel, a Scope Manager, and a pattern.

Creating a Pattern

The `mossh` commands described in Table 9 create a pattern and specify the information it will contain.

Table 9 mossh Commands to Create Patterns

Command	Description
make pattern	Analogous to the make message command. This command creates a pattern with the name specified. You can supply a sentence to be contained in this pattern.
modify pattern	Analogous to the modify message command. This command modifies the various parameters of a pattern. The options described in Table 10 enable you to set the pattern parameters
show pattern	Analogous to the show message command. Enables you to examine the contents of a pattern. The maximum number of uses and the lifetime are set to zero.

Table 10 modify pattern Command Options

Option	Parameter	Definition
-w	sentence	Describes the information in the pattern.
-s	channelname	The mossh name of an event channel
-u	user_data	Sentence contents for use by the application, not by MOS. Defaults to NULL.

An Easy Way to Get Started Using MOS

Included with MOS is a demo utility, `mossh`, which enables you to perform a variety of essential MOS functions without having to master the underlying interfaces. mossh is a command interpreter that provides commands for most of the public portion of the MOS IDL.

Creating Messages

The `mossh` commands described in Table 7 create a message and specify the information it will contain.

Table 7 mossh Commands to Create Messages

Command	**Description**
make message	Creates a message with the name specified. You can supply a sentence, which is the data to be contained in this message. In general, this command is used to create objects. `mossh` names the object references for the created objects so that they can be easily used by other mossh commands.
modify message	Modifies any of the parameters of a message shown in Table 8.
show message	Enables you to examine the contents of a message.

Table 8 modify message Command Options

Option	**Parameter**	**Definition**
-w	sentence	Describes the information carried by the message.
-u	user_data	Sentence contents for use by the application, *not* by MOS. Defaults to NULL.
-m	max_uses	Limits the number of recipients of the message by restricting the number of patterns executed with this message to the number *max_uses*.
-l	lifetime	The number of seconds elapsed after 00:00 1/1/70 when the message should no longer be used.
-s	channelname	The `mossh` name of an event channel.

set of rules Brian has specified to MOS; therefore, the message does *not* match his pattern and the Stock Scope Manager does not send a message to its event channel.

▼ *Scene Five: The Buying Activity*

Julie receives the StockForSale message and, since both the index and indicators are up, she decides to purchase Steve's stock. She sends a query to the Stock Scope Manager to find out who would see her Buy request. The result returned to her indicates that the FTC and Brian are both interested in seeing the transaction. She sends a Buy message to her broker, indicating to the Stock Scope Manager to also deliver a copy of the message to the FTC, but *not* to deliver a message to Brian (because, in this example, she does not want Brian to know her personal business).

Julie's broker receives the Buy message. When the transaction has been executed, the Broker application automatically sends her a BuyAcknowledged message, which is also sent to the FTC.

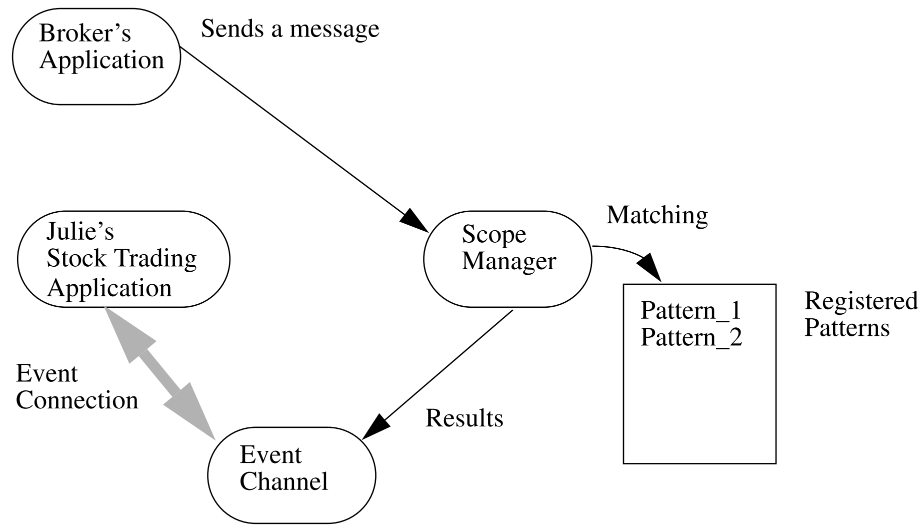

Figure 2 Sending a Message

▼ *Scene Three: The Observers*

In addition to the stockholders and their brokers, there are at least two other parties interested in the activities occurring in the Stock scope: Brian and the Federal Trade Commission (FTC). Brian has set up a complex set of rules to indicate to MOS that he only wants to be informed when something "really important" occurs; the FTC is monitoring all trading activity.

▼ *Scene Four: The Selling Activity*

Steve decides to sell some stock and his StockTrading application sends out a StockForSale message to the Stock Scope Manager. The Stock Scope Manager sends the message to the event channel of all the patterns that matched the message, including to the event channel in which the Observer FTC receives messages. However, this sale is not a "really important" sale according to the

Obtaining Results

StockTrading applications "listen" on their event channel for messages broadcast by Broker applications. For example, the following events occur might occur in this scenario:

1. A Broker application creates a message containing new index information.
2. The Broker application sends the message to the Stock Scope Manager.
3. The Stock Scope Manager matches the incoming message with the registered patterns and determines which applications are interested in receiving the message.
4. The Stock Scope Manager determines the event channel for each matched message.
5. The Stock Scope Manager sends a Result to the event channel.
   The result contains a reference to the matched pattern and the message that caused the match to occur.
6. Each StockTrading application obtains the Result from the event channel.
7. The StockTrading application examines the Result for specific information about the match.
8. The StockTrading applications use the information to buy, sell, or hold stock.

Figure 2 shows how the Broker application sends the message and the StockTrading applications receive it.

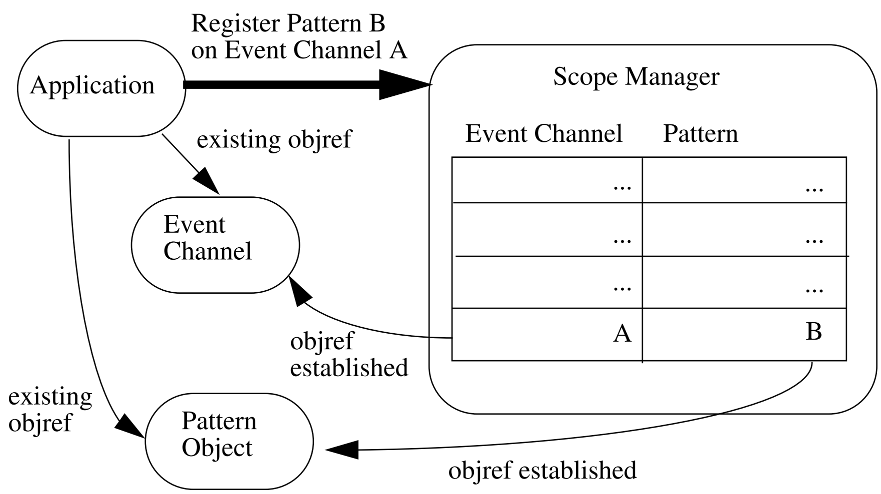

Figure 1 Registering Patterns

▼ *Scene Two: The Brokers*

The brokers also use MOS-based applications in the DOE environment. These applications (which we will call Broker applications) perform several tasks:

- When the indexes and indicators are announced, each application automatically send two messages to the interested stockholders in their scope: one message containing the Consumer Price Index and the other message containing the value of the indicators.
- They sends messages to an individual stockholder whenever the broker receives and executes a buy or sell order.
- They advertise the brokers' services and profile their capabilities.

Applications communicate by broadcasting messages to the scope and by "listening" for results on their event channel. In this example, the messages the Broker applications broadcast include messages containing index and indicator messages, messages about stocks for sale, and confirmation of orders being received and executed.

♦ **Each StockTrading application must do the following:**

1. Obtain a pattern factory.
   The pattern factory is a source of patterns.
2. Use the pattern factory to create a pattern.
   This empty template is used to hold the actual information to be matched.
3. Set the sentence to specify the messages it wants to receive.
   In this example, the sentence specifies messages containing index and indicator information.

Registering a Pattern

Once a set of patterns is created, an application must register these patterns with the scope. The scope retains all patterns registered by applications within the scope.

♦ **Each StockTrading application must do the following:**

1. Obtain a scope.
2. Obtain a pattern.
3. Obtain an event channel factory.
4. Use the event channel factory to obtain an event channel.
5. Identify the pattern and the event channel.
6. Register the pattern with the Scope Manager.

The event channel is used like a mailbox by the Scope Manager to keep all incoming messages that match the patterns registered by the application. Applications have unrestricted use of event channels; applications can share event channels with other applications, or "listen" to more than one event channel at once. Patterns can be registered on multiple event channels, and multiple patterns can be registered on any given event channel.

Figure 1 shows how the Scope Manager establishes and maintains a list of object references to event channels and their corresponding patterns.

Once the rules were set, the stockholders registered the patterns that will match the messages in which they are interested in receiving. Registering a pattern required the applications to each perform four tasks: finding the manager of the scope of interest (in this example, the Stock Scope Manager), creating patterns describing their stock interests, registering the pattern, and creating an event channel in which to receive the matched messages.

Finding the Scope

To inter-operate with other applications, an application must first find its scope. The StockTrading applications need to find the Stock scope in order to be able to receive the index and indicator messages.

♦ **Each StockTrading application must do the following:**

1. Obtain an object reference for a Scope Provider.
2. Ask the Scope Provider to find a particular scope (in this example, the Stock scope).
   With this invocation, the application indicates whether a new scope should be created if none exists.

♦ **The Scope Provider does the following:**

1. Receives the invocation and attempts to find the scope through the Naming service.
   The Naming service traverses the name space until the scope is found.
2. Relays a reference to the Scope Manager of each application's scope.
3. Once a scope is found, the Scope Provider is no longer needed by the applications.

Creating Patterns

To receive information, an application must first create *patterns* and then register these patterns with the Scope Manager. The patterns describe which messages the application is interested in receiving. The Scope Manager retains these patterns and matches them against incoming messages.

In this example, each StockTrading application must specify the range of messages it is interested in receiving in order to receive information from other applications in the Stock scope.

Table 5 Hypothetical MOS Message Protocol for StockTrading Applications

Message	**Description**
CPIndexAnnounce	Indicates whether the Consumer Price Index has gone up, gone down, or stayed the same, and the amount it has changed.
IndicatorsAnnounce	Indicates whether the composite Economic Indicators value has gone up, gone down, or stayed the same, and the among it has changed.
Buy	Places a buy order.
Sell	Places a sell order.
BuySellAcknowledge	Acknowledges that a buy or sell order has been received and successfully executed.

To set up their environments to listen for the combinations of information that interests them, each stockholder specified a set of "rules" when they first installed their StockTrading applications. These set of rules are implemented using MOS patterns. Julie's set of rules is shown in Table 6.

Table 6 Julie's Rules for Buying and Selling Stock

Rule	**Purpose**
Buy100	Specifies the conditions favorable for Judy to purchase 100 shares, namely, that *both* the Consumer Price Index *and* the Economic Indicators are *up*.
Sell100	Specifies the conditions appropriate for Judy to sell 100 shares, namely, that *both* the Consumer Price Index *and* the Economic Indicators are *down*.
Buy10	Specifies the conditions appropriate for Judy to buy 10 shares, namely that *either* the Consumer Price Index *or* the Economic Indicators are *up*.

An Example Scenario

MOS provides a high level of flexibility for specifying which messages an application is “interested” in receiving. This section provides an example of how MOS can be used. The scenario shows how applications can specify complex conditions and combinations of conditions for receiving messages.

▼ *Scene One: The Stockholders*

Julie, Thomas, and Steve all own stock in the Ritzee movie theaters. The value of each share of this stock is influenced by the Consumer Price Index and a group of Economic Indicators, which each stockholder receives from their broker once per month. The chart shown in Table 4 is used by each stockholder to make buy and sell decisions.

Table 4 Stock Decision Matrix

Stockholder	Index Up AND Indicators UP	Index Down AND Indicators Down	Index Up OR Indicators Up
Julie	Buy 100	Sell 100	Buy 10
Thomas	Buy 50	Sell 50	No activity
Steve	Buy 1,000	Sell 500	Buy 50

Note – Julie, Thomas, and Steve are referred to collectively as “stockholders” in this example.

The stockholders use MOS-based applications in the DOE environment. Each stockholder’s application is set up to “listen” for index and indicator messages in the combinations listed in Table 4. Depending on which combination is received, the individual stockholder’s application sends out a buy or sell order to the broker and notifies the stockholder that a buy or sell order has been sent. Table 5 shows the hypothetical MOS message protocol used by these StockTrading applications.

Note – The stockholders’ applications are referred to as the StockTrading applications in this example.

Table 3 Basic MOS Interfaces *(Continued)*

Interface	Description
Result	An application uses the Result interface to obtain the results of a message-pattern match. Each result contains object references for the pattern that was matched and the message that matched the pattern. The result also contains the set of variables in the pattern sentence paired with the values in the message sentence that they matched.
ResultDispatch	The ResultDispatch interface is implemented by the Scope Manager.
ScopeManager	An application uses the ScopeManager interface to register and unregister patterns, and to send and try messages. This interface includes a tooltalk_scope attribute that indicates whether the scope is a ToolTalk scope.
Message Factory	An application uses the MessageFactory interface to create messages.
Pattern Factory	An application uses the PatternFactory interface to create patterns.
ScopeProvider	An application uses the ScopeProvider interface to find, create, and delete scopes.

Basic Mechanisms

Two basic mechanisms manage the flow of information between applications: the Scope Provider and the Scope Manager (see Table 2).

Table 2 Basic MOS Mechanisms

Mechanism	**Definition**
Scope Provider	• Finds the scope into which messages will be sent. • If no scope exists, the Scope Provider creates one if the application has indicated that one should be created. • If the requested scope exists, the Scope Provider returns the object reference to the scope.
Scope Manager	• Coordinates the flow of information in a given scope. • Contains a list of patterns that applications in the scope have registered. • Maintains object references to the event channel(s) associated with the patterns. • Matches incoming messages against its list of patterns. • Sends results to the Event Channels.

Basic Interfaces

The basic MOS interfaces provide facilities for creating, cloning, and destroying messages and patterns, and for obtaining results from message-pattern matches.

Table 3 Basic MOS Interfaces

Interface	**Description**
Message	An application uses the Message interface to send messages to other applications. The interface defines a message and its attributes.
Pattern	An application uses the Pattern interface to specify the kinds of messages it wants to receive. A pattern contains a sentence that is used to match against sentences contained in incoming messages. This interface contains operations to set the sentence in the pattern and to destroy the pattern.

How Does MOS Work?

As in ToolTalk, MOS applications send and receive *messages*. Messages contain *sentences*, which are the formal statements of the information in the message. An application can specify in which messages it is interested receiving through *patterns*, which MOS uses to match against incoming messages.

MOS applications send messages to a *scope*. The scope limits message delivery to only the applications who have registered interest in the scope. For example, a collection of financial applications might registered interest in messages sent in a "stock trading" scope, or desktop applications might be interested in messages sent in a ToolTalk session scope.

MOS enables applications to "screen" its incoming messages by specifying that certain conditions must be true. An application can also specify that it wants to receive a message if ConditionA is true, *or* if ConditionB is true. MOS uses the Event service as the mechanism to pass information to the applications. Event channels provide a versatile means for applications to connect with MOS. This flexibility enables MOS applications to specify their interest very accurately.

Scopes

A scope is a name space representing a range of interest for a set of applications. A scope limits the range of communication for the group of applications. A scope is used to represent a group of applications with shared interests, such as numerical values, events, strings, or any other information that could change during a program's execution. A ToolTalk session is one possible kind of scope from the point of view of a MOS application. MOS uses the Naming service to organize the scopes served by MOS.

The Messaging Object Service

The Messaging Object Service (MOS) is a Project DOE Object Service that provides an inter-operability solution for applications in the DOE environment. MOS also provides inter-operability between legacy ToolTalk® applications and MOS-compiant applications running in the DOE environment.

What is MOS?

For DOE developers, MOS provides a simple and effective way to integrate different DOE applications. For developers now using ToolTalk to integrate applications in the Solaris environment, MOS assures that their ToolTalk applications will integrate smoothly into the DOE environment; and that users will be able to continue to run their ToolTalk applications in the DOE environment, inter-operating with DOE applications integrated through MOS.

MOS enables applications to broadcast messages that are picked up by interested applications. Applications can also specify which incoming messages they are interested in receiving. Only incoming messages that match the given set of criteria are passed on to the application.

Applications send, specify, and receive information through three basic objects: messages, patterns, and results (see Table 1).

Table 1 Basic MOS Objects

Objects	**Definition**
Message	A request or notification. For example, a request could be for an application to perform a computation and reply with a result. A notification is for broadcasting information (a state change, for example).
Pattern	A specification of exactly what kinds of messages an application is interested in receiving.
Result	A set of information that describes how a pattern matched a message.

Table of Contents

The Messaging Object Service

A White Paper

SunSoft
A Sun Microsystems Company

SunSoft, Inc.
2550 Garcia Avenue
Mountain View, CA 94043

For more information, call 1 800 227-9227.

Printed in USA 4/94 94079-001

The ToolTalkSession Scope Manager is a client of the ttsession — just like any other application in the session. The ttsession delivers the message to the ToolTalkSession Scope Manager, which acts as a representative for all MOS applications. The Scope Manager then delivers the message to interested application via the event channel.

Summary

The Messaging Object Service in the Project DOE environment provides many benefits to developers of inter-operating applications — and to the users of those applications. The inter-operability of existing ToolTalk applications with MOS-based applications on the Project DOE environment means that there are already an initial supply of applications available that are usable in the Project DOE environment.

The Messaging Object Service also leverages both the underlying ORB and the Object Services (such as the Event service, the Naming service, and Persistent Storage Management) that are part of the Project DOE environment. This leverage results in a number of capabilities in the Project DOE Application Environment, including:

- A more general message scoping mechanism. The distributed nature of the naming service will, for example, allow widely separated applications to communicate because there is a common mechanism (namespace) for scope lookup.
- A more versatile notification capability. The separation of event handling will allow message routing to better suit the needs of the application; for example, an application may want to use multiple notification channels.
- Persistent scopes. The persistence of objects and the manner in which objects are activated will eliminate the need for separate declarations of static patterns.
- Easier incorporation of existing (legacy) technology. Applications will be able to put "object wrappers" around existing software to take advantage of the functionality of other (non-object) applications.

When a MOS application sends a message into a ToolTalk scope, the ToolTalkSession Scope Manager examines the contents of the message. Figure 4 shows the relationship between the MOS application, the ToolTalkSession Scope Manager, the ttsession, and the ToolTalk application.

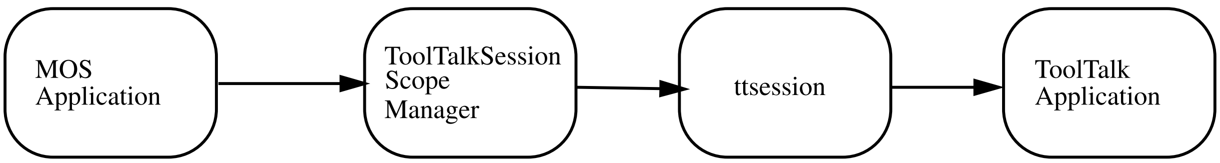

Figure 4 MOS-to-ToolTalk Inter-operability

If all parts of the message have analogs in ToolTalk, the ToolTalkSession Scope Manager determines that the destination of the message is a ToolTalk application. The ToolTalkSession Scope Manager then relays the message to the ttsession within the ToolTalk environment. The ttsession distributes the message just as it would distribute any ToolTalk message.

When a ToolTalk application sends a message to a MOS application, the application sends a message to the ttsession. Figure 4 shows the relationship between the ToolTalk application, the ttsession, the ToolTalkSession Scope Manager, the event channel, and the MOS application.

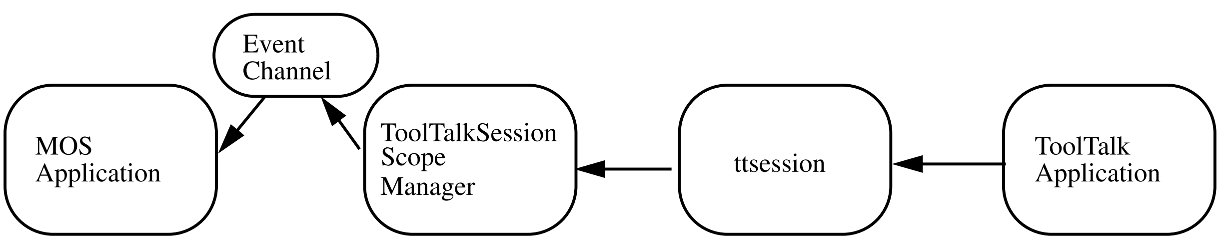

Figure 5 ToolTalk-to-MOS Inter-operability

What does this mean for the users of your ToolTalk-aware applications? It means that your customers will be able to continue to use your applications as they move into the new object paradigm. With MOS, you will be able to continue to use ToolTalk-compatible messaging in your existing applications as you enhance your applications to take advantage of the Project DOE distributed object facilities. This inter-operability means that you will be able to run a ToolTalk-aware application in the distributed object environment (and other CORBA-compliant environments that support the ToolTalk APIs and MOS interfaces). As additional components of the Project DOE become available, your existing ToolTalk-aware application interfaces (implemented as ToolTalk messages) will be available to other Project DOE components through the Messaging Object Service.

MOS applications can interact with ToolTalk applications, and vice-versa. From the point of view of a MOS application, it sends a message to the ToolTalk scope just as it would send a message to any other MOS scope. However, the MOS application uses ToolTalk language in its messages and patterns. From the point of view of the ToolTalk application, MOS is just another application within the ttsession. The interaction is transparent expect for the format of the messages.

In addition, MOS will inter-operate with ToolTalk on Solaris and ToolTalk on other platforms. ToolTalk and MOS will, therefore, provide distributed object capabilities to other platforms regardless of whether the platform has native distributed object capabilities.

Sending and Receiving Messages

A MOS application can send a message to or receive a message from a ToolTalk scope. The process for sending and receiving messages is the same as for a MOS scope except that, when a ToolTalk scope is specified, the Scope Provider finds a special ToolTalkSession Scope Manager that is designed to be bridge between MOS and ToolTalk.

All information communicated between MOS and ToolTalk applications is managed by the ToolTalkSession Scope Manager, which translates MOS messages and patterns to and from ToolTalk messages and patterns.

Inter-operating with ToolTalk

Open systems mean that developers are able to create more positive and productive computing environments. Project DOE technology represents SunSoft's support of open systems in the new object paradigm; the ToolTalk service represents SunSoft's support of open systems in today's (existing) computing environment. ToolTalk functionality is supported in the Project DOE environment through MOS.

If you develop an application that uses the ToolTalk message sets, you can be assured that you have made an excellent investment. Figure 3 shows how your investment in ToolTalk messages is protected in the Project DOE environment.

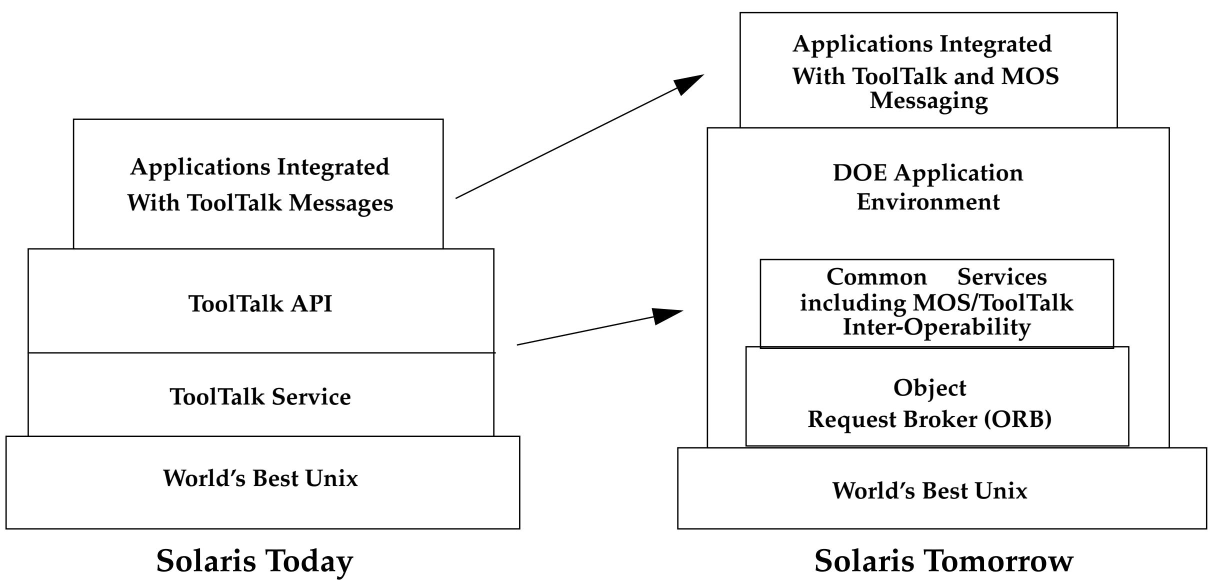

Figure 3 ToolTalk Investments are Protected in the Project DOE Environment

Existing ToolTalk applications will, of course, continue to interact with each other in the Project DOE environment. In addition, MOS enables existing ToolTalk applications to interact with *new object-based* applications. Inter-operability of applications from the current paradigm to the new object paradigm is provided. This provision means that the ToolTalk applications you are developing today can also work with Project DOE technology.

Trying a Message Before Sending It

The try message command allows you to determine the effect of a message without actually sending it. You can try the message, examine the results and, if the results are satisfactory, you can then send it.

Sending a Message

MOS enables an application to broadcast information in the form of a message, without needing to know which applications will receive the information. A message is sent to a scope, not to an individual application, so the Scope Manager orchestrates the delivery of the information in the message to interested applications. Table 13 describes the mossh commands used to send a message.

Table 13 mossh Commands to Send Messages

Command	Description
send message	Sends messages to a Scope Manager. When the Scope Manager receives the message, it matches the message against the patterns on its list. The Scope Manager pushes results to each event channel that corresponds to a matched pattern. The result remains in the event channel until it is retrieved.
dump channel	Displays the result. This command reads events off of the event channel and displays the associated results.

Ending a mossh Session

To end your mossh session, you need to first destroy all of the objects you have created, then quit the session. Table 14 describes these mossh commands.

Table 14 mossh Commands to End a Session

Command	Description
destroy	Destroys all the objects that mossh has created. You can also use the destroy command to destroy individual objects in a scope.
quit	Saves any object references that have not been destroyed and exits the mossh session.

Creating an Event Channel

The mossh commands described in Table 11 create and display event channels.

Table 11 mossh Commands to Create and Display Event Channels

Command	Description
make channel	Creates an event channel. You can also specify a maximum number of events that the channel receives, and a time-out value.
show channel	Analogous to the show message command. Enables you to examine the contents of an event channel.

Finding a Scope Manager and Registering Patterns

The Scope Manager acts as a central repository for all patterns from applications in the scope. Registering a pattern indicates which messages an application is interested in receiving by placing it on a list of registered patterns kept by the Scope Manager. Once you have registered a pattern, the Scope Manager does the work of matching incoming messages against the pattern and delivering results to the event channel you have specified. Table 12 describes the mossh commands to find a Scope Manager and register a pattern.

Table 12 mossh Commands to Find a Scope Manage and Register a Pattern

Command	Description
find scope	Locates or creates a Scope Manager. Option you can include a ToolTalk session id. Also, you can specify whether a new Scope Manager should be created if no scope exists.
register pattern	Registers a pattern, or pattern sequence, and the associated event channel with a given scope manager.
show scope	Analogous to the show message command. Enables you to examine the contents of a scope.

Registering a pattern requires three objects: an event channel, a Scope Manager, and a pattern.

Creating a Pattern

The `mossh` commands described in Table 9 create a pattern and specify the information it will contain.

Table 9 mossh Commands to Create Patterns

Command	Description
make pattern	Analogous to the make message command. This command creates a pattern with the name specified. You can supply a sentence to be contained in this pattern.
modify pattern	Analogous to the modify message command. This command modifies the various parameters of a pattern. The options described in Table 10 enable you to set the pattern parameters
show pattern	Analogous to the show message command. Enables you to examine the contents of a pattern. The maximum number of uses and the lifetime are set to zero.

Table 10 modify pattern Command Options

Option	Parameter	Definition
-w	sentence	Describes the information in the pattern.
-s	channelname	The mossh name of an event channel
-u	user_data	Sentence contents for use by the application, not by MOS. Defaults to NULL.

An Easy Way to Get Started Using MOS

Included with MOS is a demo utility, `mossh`, which enables you to perform a variety of essential MOS functions without having to master the underlying interfaces. mossh is a command interpreter that provides commands for most of the public portion of the MOS IDL.

Creating Messages

The `mossh` commands described in Table 7 create a message and specify the information it will contain.

Table 7 mossh Commands to Create Messages

Command	Description
make message	Creates a message with the name specified. You can supply a sentence, which is the data to be contained in this message. In general, this command is used to create objects. `mossh` names the object references for the created objects so that they can be easily used by other mossh commands.
modify message	Modifies any of the parameters of a message shown in Table 8.
show message	Enables you to examine the contents of a message.

Table 8 modify message Command Options

Option	Parameter	Definition
-w	sentence	Describes the information carried by the message.
-u	user_data	Sentence contents for use by the application, *not* by MOS. Defaults to NULL.
-m	max_uses	Limits the number of recipients of the message by restricting the number of patterns executed with this message to the number *max_uses*.
-l	lifetime	The number of seconds elapsed after 00:00 1/1/70 when the message should no longer be used.
-s	channelname	The `mossh` name of an event channel.

set of rules Brian has specified to MOS; therefore, the message does *not* match his pattern and the Stock Scope Manager does not send a message to its event channel.

▼ *Scene Five: The Buying Activity*

Julie receives the StockForSale message and, since both the index and indicators are up, she decides to purchase Steve's stock. She sends a query to the Stock Scope Manager to find out who would see her Buy request. The result returned to her indicates that the FTC and Brian are both interested in seeing the transaction. She sends a Buy message to her broker, indicating to the Stock Scope Manager to also deliver a copy of the message to the FTC, but *not* to deliver a message to Brian (because, in this example, she does not want Brian to know her personal business).

Julie's broker receives the Buy message. When the transaction has been executed, the Broker application automatically sends her a BuyAcknowledged message, which is also sent to the FTC.

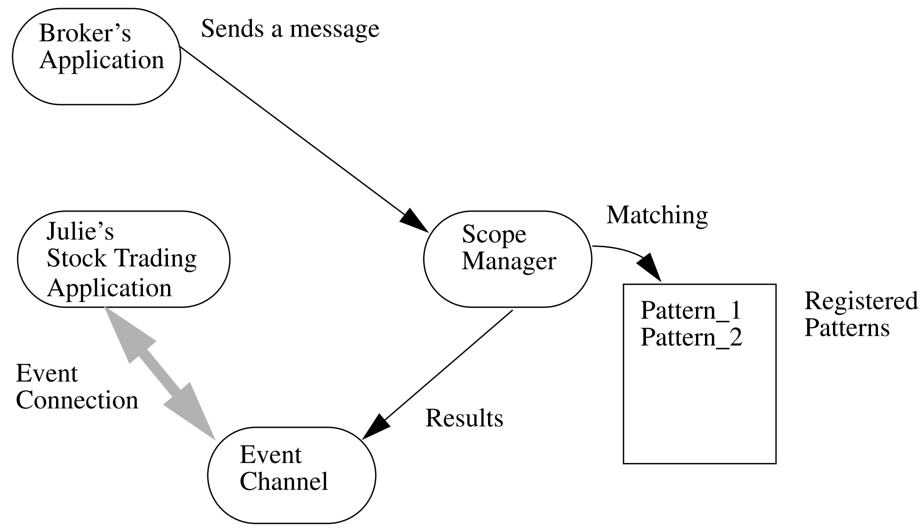

Figure 2 Sending a Message

▼ *Scene Three: The Observers*

In addition to the stockholders and their brokers, there are at least two other parties interested in the activities occurring in the Stock scope: Brian and the Federal Trade Commission (FTC). Brian has set up a complex set of rules to indicate to MOS that he only wants to be informed when something "really important" occurs; the FTC is monitoring all trading activity.

▼ *Scene Four: The Selling Activity*

Steve decides to sell some stock and his StockTrading application sends out a StockForSale message to the Stock Scope Manager. The Stock Scope Manager sends the message to the event channel of all the patterns that matched the message, including to the event channel in which the Observer FTC receives messages. However, this sale is not a "really important" sale according to the

Obtaining Results

StockTrading applications “listen” on their event channel for messages broadcast by Broker applications. For example, the following events occur might occur in this scenario:

1. A Broker application creates a message containing new index information.
2. The Broker application sends the message to the Stock Scope Manager.
3. The Stock Scope Manager matches the incoming message with the registered patterns and determines which applications are interested in receiving the message.
4. The Stock Scope Manager determines the event channel for each matched message.
5. The Stock Scope Manager sends a Result to the event channel.
   The result contains a reference to the matched pattern and the message that caused the match to occur.
6. Each StockTrading application obtains the Result from the event channel.
7. The StockTrading application examines the Result for specific information about the match.
8. The StockTrading applications use the information to buy, sell, or hold stock.

Figure 2 shows how the Broker application sends the message and the StockTrading applications receive it.

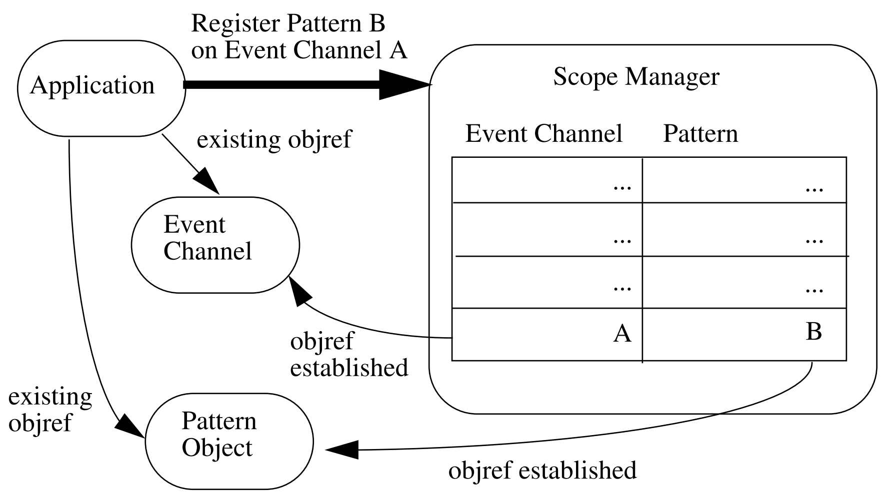

Figure 1 Registering Patterns

▼ *Scene Two: The Brokers*

The brokers also use MOS-based applications in the DOE environment. These applications (which we will call Broker applications) perform several tasks:

- When the indexes and indicators are announced, each application automatically send two messages to the interested stockholders in their scope: one message containing the Consumer Price Index and the other message containing the value of the indicators.
- They sends messages to an individual stockholder whenever the broker receives and executes a buy or sell order.
- They advertise the brokers' services and profile their capabilities.

Applications communicate by broadcasting messages to the scope and by "listening" for results on their event channel. In this example, the messages the Broker applications broadcast include messages containing index and indicator messages, messages about stocks for sale, and confirmation of orders being received and executed.

♦ **Each StockTrading application must do the following:**

1. Obtain a pattern factory.
   The pattern factory is a source of patterns.

2. Use the pattern factory to create a pattern.
   This empty template is used to hold the actual information to be matched.

3. Set the sentence to specify the messages it wants to receive.
   In this example, the sentence specifies messages containing index and indicator information.

Registering a Pattern

Once a set of patterns is created, an application must register these patterns with the scope. The scope retains all patterns registered by applications within the scope.

♦ **Each StockTrading application must do the following:**

1. Obtain a scope.

2. Obtain a pattern.

3. Obtain an event channel factory.

4. Use the event channel factory to obtain an event channel.

5. Identify the pattern and the event channel.

6. Register the pattern with the Scope Manager.

The event channel is used like a mailbox by the Scope Manager to keep all incoming messages that match the patterns registered by the application. Applications have unrestricted use of event channels; applications can share event channels with other applications, or "listen" to more than one event channel at once. Patterns can be registered on multiple event channels, and multiple patterns can be registered on any given event channel.

Figure 1 shows how the Scope Manager establishes and maintains a list of object references to event channels and their corresponding patterns.

Once the rules were set, the stockholders registered the patterns that will match the messages in which they are interested in receiving. Registering a pattern required the applications to each perform four tasks: finding the manager of the scope of interest (in this example, the Stock Scope Manager), creating patterns describing their stock interests, registering the pattern, and creating an event channel in which to receive the matched messages.

Finding the Scope

To inter-operate with other applications, an application must first find its scope. The StockTrading applications need to find the Stock scope in order to be able to receive the index and indicator messages.

♦ **Each StockTrading application must do the following:**

1. Obtain an object reference for a Scope Provider.
2. Ask the Scope Provider to find a particular scope (in this example, the Stock scope).
   With this invocation, the application indicates whether a new scope should be created if none exists.

♦ **The Scope Provider does the following:**

1. Receives the invocation and attempts to find the scope through the Naming service.
   The Naming service traverses the name space until the scope is found.
2. Relays a reference to the Scope Manager of each application's scope.
3. Once a scope is found, the Scope Provider is no longer needed by the applications.

Creating Patterns

To receive information, an application must first create *patterns* and then register these patterns with the Scope Manager. The patterns describe which messages the application is interested in receiving. The Scope Manager retains these patterns and matches them against incoming messages.

In this example, each StockTrading application must specify the range of messages it is interested in receiving in order to receive information from other applications in the Stock scope.

Table 5 Hypothetical MOS Message Protocol for StockTrading Applications

Message	**Description**
CPIndexAnnounce	Indicates whether the Consumer Price Index has gone up, gone down, or stayed the same, and the amount it has changed.
IndicatorsAnnounce	Indicates whether the composite Economic Indicators value has gone up, gone down, or stayed the same, and the among it has changed.
Buy	Places a buy order.
Sell	Places a sell order.
BuySellAcknowledge	Acknowledges that a buy or sell order has been received and successfully executed.

To set up their environments to listen for the combinations of information that interests them, each stockholder specified a set of "rules" when they first installed their StockTrading applications. These set of rules are implemented using MOS patterns. Julie's set of rules is shown in Table 6.

Table 6 Julie's Rules for Buying and Selling Stock

Rule	**Purpose**
Buy100	Specifies the conditions favorable for Judy to purchase 100 shares, namely, that *both* the Consumer Price Index *and* the Economic Indicators are *up*.
Sell100	Specifies the conditions appropriate for Judy to sell 100 shares, namely, that *both* the Consumer Price Index *and* the Economic Indicators are *down*.
Buy10	Specifies the conditions appropriate for Judy to buy 10 shares, namely that *either* the Consumer Price Index *or* the Economic Indicators are *up*.

An Example Scenario

MOS provides a high level of flexibility for specifying which messages an application is “interested” in receiving. This section provides an example of how MOS can be used. The scenario shows how applications can specify complex conditions and combinations of conditions for receiving messages.

▼ *Scene One: The Stockholders*

Julie, Thomas, and Steve all own stock in the Ritzee movie theaters. The value of each share of this stock is influenced by the Consumer Price Index and a group of Economic Indicators, which each stockholder receives from their broker once per month. The chart shown in Table 4 is used by each stockholder to make buy and sell decisions.

Table 4 Stock Decision Matrix

Stockholder	Index Up AND Indicators UP	Index Down AND Indicators Down	Index Up OR Indicators Up
Julie	Buy 100	Sell 100	Buy 10
Thomas	Buy 50	Sell 50	No activity
Steve	Buy 1,000	Sell 500	Buy 50

Note – Julie, Thomas, and Steve are referred to collectively as “stockholders” in this example.

The stockholders use MOS-based applications in the DOE environment. Each stockholder’s application is set up to “listen” for index and indicator messages in the combinations listed in Table 4. Depending on which combination is received, the individual stockholder’s application sends out a buy or sell order to the broker and notifies the stockholder that a buy or sell order has been sent. Table 5 shows the hypothetical MOS message protocol used by these StockTrading applications.

Note – The stockholders’ applications are referred to as the StockTrading applications in this example.

Table 3 Basic MOS Interfaces *(Continued)*

Interface	Description
Result	An application uses the Result interface to obtain the results of a message-pattern match. Each result contains object references for the pattern that was matched and the message that matched the pattern. The result also contains the set of variables in the pattern sentence paired with the values in the message sentence that they matched.
ResultDispatch	The ResultDispatch interface is implemented by the Scope Manager.
ScopeManager	An application uses the ScopeManager interface to register and unregister patterns, and to send and try messages. This interface includes a tooltalk_scope attribute that indicates whether the scope is a ToolTalk scope.
Message Factory	An application uses the MessageFactory interface to create messages.
Pattern Factory	An application uses the PatternFactory interface to create patterns.
ScopeProvider	An application uses the ScopeProvider interface to find, create, and delete scopes.

Basic Mechanisms

Two basic mechanisms manage the flow of information between applications: the Scope Provider and the Scope Manager (see Table 2).

Table 2 Basic MOS Mechanisms

Mechanism	Definition
Scope Provider	• Finds the scope into which messages will be sent. • If no scope exists, the Scope Provider creates one if the application has indicated that one should be created. • If the requested scope exists, the Scope Provider returns the object reference to the scope.
Scope Manager	• Coordinates the flow of information in a given scope. • Contains a list of patterns that applications in the scope have registered. • Maintains object references to the event channel(s) associated with the patterns. • Matches incoming messages against its list of patterns. • Sends results to the Event Channels.

Basic Interfaces

The basic MOS interfaces provide facilities for creating, cloning, and destroying messages and patterns, and for obtaining results from message-pattern matches.

Table 3 Basic MOS Interfaces

Interface	Description
Message	An application uses the Message interface to send messages to other applications. The interface defines a message and its attributes.
Pattern	An application uses the Pattern interface to specify the kinds of messages it wants to receive. A pattern contains a sentence that is used to match against sentences contained in incoming messages. This interface contains operations to set the sentence in the pattern and to destroy the pattern.

How Does MOS Work?

As in ToolTalk, MOS applications send and receive *messages*. Messages contain *sentences*, which are the formal statements of the information in the message. An application can specify in which messages it is interested receiving through *patterns*, which MOS uses to match against incoming messages.

MOS applications send messages to a *scope*. The scope limits message delivery to only the applications who have registered interest in the scope. For example, a collection of financial applications might registered interest in messages sent in a “stock trading” scope, or desktop applications might be interested in messages sent in a ToolTalk session scope.

MOS enables applications to “screen” its incoming messages by specifying that certain conditions must be true. An application can also specify that it wants to receive a message if ConditionA is true, *or* if ConditionB is true. MOS uses the Event service as the mechanism to pass information to the applications. Event channels provide a versatile means for applications to connect with MOS. This flexibility enables MOS applications to specify their interest very accurately.

Scopes

A scope is a name space representing a range of interest for a set of applications. A scope limits the range of communication for the group of applications. A scope is used to represent a group of applications with shared interests, such as numerical values, events, strings, or any other information that could change during a program’s execution. A ToolTalk session is one possible kind of scope from the point of view of a MOS application. MOS uses the Naming service to organize the scopes served by MOS.

The Messaging Object Service

The Messaging Object Service (MOS) is a Project DOE Object Service that provides an inter-operability solution for applications in the DOE environment. MOS also provides inter-operability between legacy ToolTalk® applications and MOS-compiant applications running in the DOE environment.

What is MOS?

For DOE developers, MOS provides a simple and effective way to integrate different DOE applications. For developers now using ToolTalk to integrate applications in the Solaris environment, MOS assures that their ToolTalk applications will integrate smoothly into the DOE environment; and that users will be able to continue to run their ToolTalk applications in the DOE environment, inter-operating with DOE applications integrated through MOS.

MOS enables applications to broadcast messages that are picked up by interested applications. Applications can also specify which incoming messages they are interested in receiving. Only incoming messages that match the given set of criteria are passed on to the application.

Applications send, specify, and receive information through three basic objects: messages, patterns, and results (see Table 1).

Table 1 Basic MOS Objects

Objects	Definition
Message	A request or notification. For example, a request could be for an application to perform a computation and reply with a result. A notification is for broadcasting information (a state change, for example).
Pattern	A specification of exactly what kinds of messages an application is interested in receiving.
Result	A set of information that describes how a pattern matched a message.

Table of Contents

The Messaging Object Service

A White Paper

SunSoft
A Sun Microsystems Company