A Direct Manipulation Toolkit for Awareness Support in Groupware

A Thesis Submitted to

the College of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of Master of Science

in the

Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Jason M. Hill

Permission to Use

Abstract

It is currently difficult to build synchronous groupware interfaces that support group awareness. This difficulty exists for two main reasons: 1) group awareness requires user interface components that provide complex visual representations of awareness data, and 2) group awareness requires a distributed groupware infrastructure. Current groupware toolkits do not provide awareness support in generic and reusable ways and interfaces therefore require considerable development effort. This thesis describes a new groupware toolkit has been developed that allows developers to efficiently add awareness support to groupware interfaces. The toolkit provides a suite of customizable and reusable components that provide group awareness information and that abstract away low-level groupware issues such as distributed communication. The toolkit has been shown to substantially reduce development effort, is scalable, extensible, flexible, and easy to use, and allows rapid prototyping and testing.

Acknowledgments

I would like to thank my thesis advisor, Carl Gutwin, for his help and extreme patience during the course of my thesis program. Managing a full-time career while in the Master's program often had me travelling and working long hours. Carl's patience allowed me to successfully manage both endeavors with significantly less difficulty.

Additionally, I would like to thank my employer, SED Systems, for their support, both financially and with respect to flexibility of work hours. Without that support this thesis might never have been completed.

Last but certainly not least, I would like to give my extreme thanks to my family and friends for all the love and support they have continually and unconditionally given to me throughout the years. This support is fundamental to all my successes.

Table of Contents

List of Tables

List of Figures

1 Introduction

The goal of real-time distributed groupware is to allow people, regardless of where they are physically located, to work together as simply and naturally as they can when they are face-to-face. One of the key requirements that groupware systems need to fulfill in order to achieve this goal is to provide support for *group awareness*. Group awareness is the up-to-the-moment understanding of another person's activities in a group environment [Gutwin & Greenberg 2001]. This includes information about who is using the system, where they are working, and what they are doing. Among the reasons for providing group awareness information are: coordinating actions, managing coupling, talking about the task, anticipating others' actions, and finding opportunities to assist one another. These are all highly beneficial when a group of people is trying to collaborate in a productive and natural manner.

It has traditionally been difficult to develop groupware interfaces that provide group awareness support. The main reason is a lack of reusable, widely available, generic groupware components that provide group awareness. In past groupware systems, awareness support has typically been custom-built from scratch, or code has been taken and modified from another existing application for the new groupware application. The custom-build process often involves extending or wrapping single-user graphical user interface (GUI) components with groupware-specific functionality, which often requires considerable effort. If a reusable and customizable set of off-the-shelf groupware components existed that could be used in new groupware applications, the difficulty in building applications that provide group awareness would be reduced.

Although reuse is not common in groupware, it is the basis of single-user GUI development. To facilitate the development of single-user interfaces, many toolkits have been developed over the years. The toolkits provide a standard set of components (both widgets and supporting classes) that can be used in the creation of user interfaces. These components are developed with many design-time (and sometimes run-time) customizable properties that help to make the components generic and, hence, reusable in a wide variety of applications. In addition, components are often designed for use with GUI builders, tools that provide developers a way of interactively constructing GUIs for applications. A builder allows a developer to drag and drop components into a GUI designer and modify the design-time properties of the components, providing direct-manipulation WYSIWYG (what-you-see-is-what-you-get) development and rapid prototyping and testing. GUI toolkits that integrate with builders facilitate reuse by simplifying the process of adding and customizing pre-built components to an application.

This thesis is concerned with simplifying the construction of groupware interfaces that supports group awareness. This has been accomplished by building a new groupware toolkit that provides generic, customizable, groupware-specific components to application developers. The components automatically provide customizable support for group awareness and provide a distribution architecture to allow rapid development and testing of the user interface. Additionally, the toolkit has been developed such that it integrates with a set of popular GUI builders, which will also help to simplify groupware GUI development.

1.1 The Problem

The problem addressed by this thesis is that *it is difficult to build groupware interfaces that support group awareness*.

Providing group awareness is difficult for two main reasons:

i. Group awareness requires user interface components that provide complex visual representations of awareness data.
ii. Group awareness requires a distributed groupware infrastructure.

There are many more considerations when visualizing multi-user data over single-user data in user interface widgets. First, multi-user widgets need to express which user is manipulating a widget. In a single-user interface there is only ever one user to consider. Secondly, the widget needs to determine if the manipulation is by the local user or a remote user. This is because when a local user performs an action, the widget must notify remote applications of the action. Also, the visualization for local and remote users is often different within a widget. Next, certain multi-user widgets may have two different modes of use that should be accounted for – coupled-use and individual-use. In coupled-use mode, there is a single data model for all users' copies of a specific user-interface widget. When any user manipulates their copy of the user-interface widget, the single model is changed and updated for all users. In individual-use mode, all users have their own model for each user interface widget. Manipulating their copy of the widget affects only their copy of the model. For multi-user widgets that support individual use mode, such as scrollbars and list boxes, the widgets need to display model information for all users of the system. This requires a more complex rendering strategy than a single-user widget or a coupled-use mode multi-user widget. Finally, for coupled-use mode multi-user widgets, the widgets need to handle simultaneous access to the widgets by multiple users. This in itself is often a complex issue.

Providing support for more complex types of awareness information requires even more complex rendering. One important type of group awareness information that is often missing from groupware applications is *process feedthrough* information. *Feedthrough* is the feedback produced when artifacts are manipulated that provides others with clues about that manipulation [Dix 1993]. In groupware, *process feedthrough* is feedthrough received by users as a result of another user manipulating an interface widget that provides the local user with visual reminders of what can be done [Gutwin & Greenberg 2001]. A real-world feedthrough example is the smoke that emanates from the barrel of a gun when it is fired. When someone sees this smoke they know that the gun has recently been fired, even though they may not have actually seen the event. An example of process feedthrough in the groupware world is the depression of a button on a remote GUI when the local user clicks the mouse on the corresponding button on the local GUI. When visualizing process feedthrough information in widgets, considerations must be made so that the visualization of the feedthrough does not interfere with the remote users' ability to do their work. Additionally, for individual-use widgets it must be considered how to handle feedthrough visualization for the actions of multiple users at potentially the same time. Process feedthrough issues generally complicate multi-user widget rendering.

In groupware, users distributed across a network are working as a team within the application. This means that the developer must put facilities within the application to share information between users across the network. This will involve using a distributed communication protocol, devising a remote messaging scheme, coordinating the transformation of user actions to remote messages, and transforming remote messages to updates on the GUIs of the remote users. All of these issues add complexity to the developer's task, and are required to be accessible by the multi-user widgets of a groupware toolkit.

1.2 Goals and Motivation

The main goal of this thesis is to reduce the development effort required to produce groupware systems that support group awareness. Reducing development effort is important because of the increasing popularity of groupware that follows the expanding availability of the Internet, and the increasingly distributed nature of organizations. To satisfy the growing need, developers will be required to build a larger amount of groupware with a greater number of features than systems currently have, in a shorter time period than is currently being done.

A second goal of the research is to improve groupware: that is, provide a more informative collaborative environment for groupware users through the provision of awareness information. The motivation for this goal is the idea that better support for awareness means better collaboration for users.

1.3 Solution

The problem introduced above was that it is difficult to build groupware systems that support group awareness. The solution presented in this thesis is a new groupware GUI toolkit that simplifies the development of groupware interfaces by providing developers with a suite of groupware-enabled GUI components. In the context of this toolkit, the term *component* is used to mean a user-manipulated widget in a GUI, support classes for the underlying model and visual rendering, and underlying classes that provide the distributed communication infrastructure. This toolkit simplifies the construction of groupware that supports group awareness in three ways: it provides groupware-enabled GUI widgets, it provides communication infrastructure components, and it integrates with a set of popular IDE's. Each of these will be discussed below.

Single-user interactive GUI components, such as buttons, scrollbars, and text fields have been extended to provide groupware functionality, including the display of process feedthrough and location awareness. In addition, new groupware-specific components have been developed that contain specialized groupware functionality. Using these components will simplify groupware application development by allowing developers to add groupware functionality to an application simply by adding components to a GUI (via direct manipulation) and setting values for properties of the components.

The toolkit also provides communication infrastructure components that allow user interfaces developed with the toolkit to be fully functional, self-contained groupware applications. The toolkit abstracts away remote communication issues by providing components that encapsulate a black-box communication framework. This infrastructure includes message-dispatching components, client message controllers, and a set of groupware message classes which, when instantiated, can be passed via the message-dispatching components. These components abstract communication issues away from the groupware application developers and allow them to concentrate on application-specific issues.

The toolkit has been built on top of an existing component suite that integrates with current IDEs and direct-manipulation GUI builders. This allows groupware application developers to leverage the power of these IDEs and build groupware applications in the same development environment as they build other types of applications. The component suite is the Java Swing [Sun Microsystems 2002 C] component suite, which follows the JavaBeans standard for component-based development and allows integration with tools that leverage JavaBeans technology. The toolkit components both wrap existing Swing components and create new JavaBeans components, following the JavaBeans convention. Core Swing classes and interfaces have been extended and new classes and interfaces have been written to provide a component infrastructure for the groupware toolkit. This allows consistent components to be added to the toolkit by both the toolkit developers and toolkit users. Java interfaces have been provided such that new components can be easily integrated into the toolkit and a Java IDE.

1.4 Steps in the Solution

The following steps have been carried out in completing the solution:

- Classification of GUI widgets in terms of groupware requirements.
- Development of the communication infrastructure.
- Development of the GUI component infrastructure.
- Development of a representative sample of GUI components.
- Creation of a set of applications using the toolkit.

1.4.1 Classification of Widgets in Terms of Groupware Requirements

Requirements for the toolkit have been determined in three areas: 1) the types of groupware information and features that need to be supported, 2) the types of widgets that exist and which specific widgets fall into each category, and 3) the groupware features that need to be supported by each widget. For the first area, the focus was on features that provide group awareness information (and in particular process feedthrough information). The awareness framework discussed in [Gutwin & Greenberg 2001] was used to guide the process. The results of the requirements analysis were used to determine what groupware features and feature support was to be added to the component infrastructure (section 1.4.3). For the second area, categorizations of different types of groupware widgets and their groupware requirements and features were determined. These categories include divisions among single-user widgets, as well as consideration of different types of groupware-specific components. From these categories, a representative set of widgets was chosen for implementation in the toolkit. For the third area, the awareness information to be supported was mapped to the list of widgets to produce a specification for the enhanced components.

This step in the solution produced a list of widget categories and widgets within each category, along with the groupware features supported by each widget (see Chapter 5).

1.4.2 Development of a Communication Infrastructure

The GUI components within the toolkit need some way of communicating state changes and awareness information to the user interfaces of other clients within a groupware application. The mechanism to transfer this information is the communication infrastructure of the toolkit.

The communication infrastructure needs to be distributed in nature and is based on top of a standard communication framework. For the thesis implementation of the toolkit, Java RMI was selected as the communication middleware. It should be noted that the toolkit design is such that any communication infrastructure should be usable in an implementation with little change in the design. The messages sent between clients need to encompass all the information required to reflect the state of a widget within the remote user interfaces. This information needs to be relative in nature because client GUIs need not all be configured identically. The server-side toolkit components have been developed such that the communication and distribution architectures are a black box to the client-side components. Because RMI is used, the distribution architecture of the thesis implementation is client-server in nature, with a centralized server that supports numerous distributed clients. The server is lightweight, acting as a message router between clients and as a registration system for a groupware application session. It should be noted that because the server implementation is centralized the performance is less efficient that it would be with a replicated distribution architecture.

This step of the solution produced the architecture and code base for the communication infrastructure.

1.4.3 Development of the GUI Component Infrastructure

The toolkit uses an object-oriented component-based architecture. All GUI components within this architecture have been derived from a set of groupware-component base classes and interfaces, some of which are extensions of the core Swing classes and interfaces. Specifically, the base classes and interfaces include integration with the communication infrastructure as well as multi-user features and feature support. Multi-user feature support includes support for process feedthrough and other group awareness mechanisms.

The result of this step of the solution is base code and support code for the toolkit components. The base code is the set of base classes and interfaces that are used as a starting point for the creation of GUI components. The support code includes any classes and interfaces that a developer can use to add non-required groupware features to a component. An example of a non-required groupware feature that has support code developed for it is a run-time customization dialog box.

1.4.4 Develop a Representative Sample of GUI Components

The development of a set of GUI components from which applications can be built was the next step in development of the toolkit. Because of the limited scope of the project, a complete set of GUI components for the toolkit was not developed. Instead, a representative sample of components that cover the different types of components identified in 1.4.1 has been created. The sample includes groupware versions of existing Swing components and groupware-specific components.

The deliverable from this step in the solution is the suite of components for the toolkit.

1.4.5 Creation of a Set of Applications Using the Toolkit

Using the toolkit and a Java IDE, a set of groupware applications has been built. The purpose was to evaluate the toolkit (see section 1.5). Several applications have been built that cover the main types of groupware currently being built.

The set of sample applications is the deliverable from this step in the solution.

1.5 Evaluation

The toolkit has been evaluated from two points of view. The first and most important is the development effort required of the groupware user interface developer. The second point of view is that of an end-user working with products developed using the toolkit.

In evaluating from the developer's point of view, the following question needs to be addressed: "How easy is it to build groupware applications that support awareness using the toolkit?" First, a simple messaging application that demonstrates the group awareness features of the toolkit (and in particular process feedthrough) was developed using three different development methods. These methods are: with the proposed toolkit, with another existing groupware toolkit (GroupKit), and from scratch using Java and Swing. The results from the three methods have been compared and contrasted to determine if the proposed toolkit makes it easier to develop groupware that supports group awareness as compared to other possible methods. As another measure of the ease of development, the actual number of lines of code that was written to develop each application was recorded, as well as the total amount of development time required for each development. From this, a conclusion about the savings provided by the toolkit was drawn. As further evaluation of the toolkit from the developer's point of view, the generality, flexibility, extensibility, and performance of the toolkit were also evaluated throughout the development and testing of the toolkit, as all these features are important for a groupware toolkit.

The second point of view of the evaluation is that of an end user. Two questions need to be answered in this part of the evaluation. The first question answered was: "How well do the toolkit components meet the groupware requirements identified in the requirements analysis step?" Second, this evaluation step took a brief look at the more specific question, "Is the group awareness information provided to end users by the toolkit components adequate?" To ensure that the awareness information provided by the toolkit is valuable to groupware users, the widgets in the toolkit have been evaluated with usability techniques during the development of the toolkit.

1.6 Contributions

The major contribution of the research is the groupware GUI toolkit that integrates with direct-manipulation JavaBeans GUI development tools, and contains a number of generic reusable groupware GUI components for supporting group awareness. This toolkit provides developers with a quicker and easier way of developing groupware applications that support group awareness than is currently available. The GUI components within the toolkit provide built-in support for group awareness, including support for process feedthrough and location awareness. Additionally, the toolkit infrastructure abstracts distributed communication away from the developers and allows them to quickly build and test working distributed applications.

A minor contribution of the research is the groupware requirements analysis results obtained from the step discussed in section 1.4.1.

1.7 Thesis Outline

The rest of this thesis is organized as follows:

- Chapter 2 reviews prior work in the area of groupware GUIs and toolkits. The areas discussed in the chapter include component-based development, IDEs, awareness and feedthrough, and groupware architectures. The chapter provides background knowledge on the key principles from which the toolkit has been developed.
- Chapter 3 discusses the requirements of an awareness toolkit and specific requirements of the awareness components that are provided in the thesis toolkit implementation.
- Chapter 4 discusses in detail the design and implementation of the communication infrastructure and GUI component infrastructure of the awareness toolkit.

- Chapter 5 introduces and discusses in detail the awareness widgets that have been implemented for the thesis version of the toolkit.
- Chapter 6 introduces some sample applications built with the toolkit as well as provides a walkthrough of how to create a simple groupware application with the toolkit.
- Chapter 7 discusses the evaluation of the toolkit, as well as some weaknesses and drawbacks of the toolkit architecture and thesis implementation.
- Chapter 8 provides a summary of the toolkit, discusses the important lessons learned in the research, and proposes directions for future work.

2 Background and Previous Work

This chapter provides background information for the research project. Areas of discussion include groupware-related concepts, distributed communication, group awareness, component-based software development, and groupware toolkits.

2.1 Basic Groupware Concepts

The work for this thesis deals specifically with the creation of real-time distributed groupware. This section provides a definition for the term groupware and looks at what real-time distributed groupware is, as defined by previous work in the CSCW field.

2.1.1 Definition of Groupware

Groupware is "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment" [Ellis et al. 1991]. This is in contrast to single-user applications that only allow one user to work on tasks. *Distributed groupware* allows users at different computers, in different geographical locations, to work together. Groupware systems are divided into two types based on how user interaction takes place: *synchronous* and *asynchronous* groupware [Rodden & Blair 1991].

2.1.2 Synchronous vs. Asynchronous Groupware

Synchronous groupware systems allow physically separated users to interact with one another and with shared computational objects in real time. This contrasts with *asynchronous groupware* systems where interaction is not in real-time and the state of one user's workspace will not be immediately updated by the actions of other users in their workspaces. This thesis focuses on the creation of synchronous groupware systems.

Synchronous collaboration is provided using one of two approaches: *collaboration transparency* or *collaboration awareness* [Lauwers & Lantz 1990]. Collaboration transparency is typically used when transforming a legacy single-user application into a groupware application. In this case, the collaboration is unknown or transparent to the developers of the application, because the collaboration support is usually added to the application after it has been developed and without modification to the actual application itself. In contrast, synchronous groupware applications developed using the collaboration awareness approach are developed from the beginning with the intent of providing collaboration support, and the collaborative aspects of the application are an integral part of the design. The work completed for this paper specifically deals with the creation of collaboration aware systems.

Whether a synchronous groupware application was developed using collaboration awareness or collaboration transparency has a significant impact on the degree of *coupling* provided by the application. Coupling is the degree to which collaborators work closely together [Begole et al. 1994]. Collaboration transparent systems tend to be more tightly coupled, not allowing much independent work to be performed by collaborators. Collaboration aware systems, on the other hand, tend to be much more flexible allowing individual users to perform independent work, as well as work in close coordination with other users of the system.

2.1.3 Concurrency Control

Developers must determine how to resolve conflicts that can occur as a result of users' simultaneous operations [Ellis et al. 1991]. This involves the managing of messages passed between groupware clients to eliminate interference and inconsistencies that may result when users attempt to simultaneously update data or interact with objects within a synchronous groupware system. This is known as *concurrency control*, which is the activity of coordinating the potentially interfering actions of processes that operate in parallel [Greenberg and Marwood 1994]. Greenberg and Marwood describe two common concurrency control schemes: serialization and locking. Serialization involves ensuring that atomic operations are executed serially across the system, or repairing the effects of operations that have been executed out of order to give the appearance of serialization. In contrast, locking provides a mechanism for a user to obtain sole access to a resource (a lock on the resource) until the lock has been released. It should be noted that although some concurrency control is required for a production quality toolkit, it is possible to get by without explicit synchronization mechanisms, such as the case of GroupKit [Greenberg and Roseman 1999]. The toolkit for this research does not focus on concurrency control.

2.1.4 Groupware Architectures

The architecture on which a toolkit is based is very important because it ultimately impacts the architecture of any applications built with the toolkit. Ideally, the architecture of a toolkit should be such that the architecture of the applications built with the toolkit are not overly restricted. In groupware toolkits, this is difficult to achieve because the toolkit needs to consider issues that are based on the architecture of applications, such as distributed communication. Toolkits for single-user applications do not need to consider such architecture-dependent issues. Thus, groupware toolkits have tended to be based on developing applications that have a particular *architectural style* and are deployed using a particular *distribution architecture*. This section briefly discusses architectural styles and distribution architectures as they relate to groupware.

a) Architectural Styles

The architectural style of a system is the underlying model followed during the system design process. Phillips has stated that the key question that developers must consider when selecting an architectural style for a groupware application centers around how to build systems which allow multiple users to concurrently interact with each other and with shared data [Phillips 1999]. Most of the architectural styles found in groupware systems follow a philosophy where application data (the model) is kept separate from interface code (the view). The Model-View-Controller (MVC) architectural style is based primarily on this philosophy and is the most commonly used architectural style in groupware. It extends the philosophy introduced above by adding the concept of a controller, which interprets user input and updates the model accordingly. The job of the view is to present the data found in the model. The MVC architecture or a variant of

MVC can be found in the COAST [Schuckmann et al. 1999], Clock [Graham et al. 1996], and GroupKit [Roseman & Greenberg 1995] groupware toolkits, to name a few. The toolkit developed for this thesis will also follow the MVC philosophy.

Alternative architectural styles to MVC that can be found in groupware systems include Abstraction-Link-View (ALV), found in Rendezvous [Hill et al. 1994], and variations of less abstract styles, such as an optional RPC style found in Groupkit [Roseman & Greenberg 1992]. ALV is similar to MVC in that it keeps the model (abstraction) separate from the view. In ALV, views and controllers are coupled together and are *linked* to the abstraction by a constraint mechanism. The optional Groupkit architectural style allows user interfaces to communicate directly via remote procedure calls.

b) Distribution Architectures

The distribution architecture (also known as the run-time architecture) of a groupware system describes the distributed nature of the system. The two most common distribution architectures that are found in current groupware systems are the *centralized* and *replicated* architectures. A centralized architecture is based around a central server computer. All client computers communicate through the central server, which performs processing and synchronization for the application. The clients are responsible only for sending data and requests to the server and visualizing data received from the server. A replicated architecture, on the other hand, is based on each site having a copy of the data and performing necessary processing and synchronization.

In terms of which approach is the better architecture to use for groupware, there is no absolute answer. Centralized architectures make it easier to provide synchronization since all data and objects are located in one location, but the centralized server often becomes the bottleneck of the system if the number of users or amount of traffic becomes large. Replicated architectures help to alleviate the bottleneck due to the fact that data is replicated locally on all client machines, but the scheme required to synchronize such a system is usually much more complex. Therefore, the decision of which architecture to use is often system-specific. In terms of a groupware toolkit, allowing a developer to choose between different architectures for different applications, or even easily changing the architecture of a specific application, is a desirable feature. For the thesis work, a distribution architecture was selected based on ease of integration with the toolkit design. However, the toolkit was constructed such that a variety of run-time architectures could be used.

Further benefits of centralized versus replicated architectures are discussed in [Greenberg & Roseman 1999] and [Phillips 1999]. Hybrid versions of these architectures are also introduced in [Phillips 1999] and [Dyck 2000]. A replicated framework is discussed in [Berlage & Genau 1993].

2.2 Distributed Communication

Groupware applications are distributed by nature and, therefore, rely on an underlying distributed communication layer to perform message passing between computers. These messages contain the data necessary to maintain and present a consistent application state. This may involve messages to update the application model in architectures such as MVC and ALV, and will always involve messages to update the user interfaces of the client computers to the current state of the application. In essence, the messages passed between computers in a groupware application have the same purpose as the messages (both verbal and non-verbal) transferred between participants in a real-life group situation – conveying knowledge from one individual to other individuals within the group.

Distributed messages are typically packaged in some form of generic wrapper that allows them to be transported using a standardized remote communications protocol or middleware. A number of standard transport mechanisms exist, some of which handle issues such as communication to different operating systems, message compression, efficiency, and level of service. Examples of existing transport mechanisms include plain TCP and UDP sockets, remote procedure calls, Java Remote Method Invocation (RMI) [Sun Microsystems 2002 B], CORBA [OMG 2000], and DCOM [Microsoft 2002]. Sockets are generally provided by operating systems and require more low-level knowledge about networking than higher-level middleware such as RMI, CORBA, and

DCOM. Application developers are tending to move away from using low-level sockets for distributed applications due to the development effort required to use them, although in terms of efficiency sockets have a low overhead and are almost always more efficient than a higher-level protocol. In fact, the higher-level protocols are generally built on top of sockets, such as is the case with CORBA, RMI, and DCOM.

CORBA achieves platform independence by providing a common interface definition language (CORBA IDL) that is used to define the remote system interfaces. Language- and platform-specific CORBA compilers can then be used to compile the IDL files, and application-specific implementations defined by developers, for the desired target environments. A CORBA ORB (Object Request Broker) runs on some computer on the system network. The ORB's job is to establish a communication link between client computers on the system. Once a communication link has been established, the client computers can use their implementation of the IDL-defined interfaces to communicate with each other. RMI and DCOM operate in a similar fashion, although implemented differently. RMI achieves its platform independence due to the fact that Java, itself, is platform independent. Thus, all RMI code is pure Java code, and the same Java compiler can be used to compile the distributed code as is used to compile the rest of the application code. While DCOM is platform-independent remote communication middleware, it is rarely used outside of a Microsoft Windows operating system and will not be discussed here.

The decision of distributed communication method used by a groupware system is often impacted by the choice of distribution architecture, as well as the expected network traffic of the system. Servers in centralized systems often become the bottleneck of high-traffic systems and it is desirable to have an efficient, lightweight communication scheme. Replicated systems or systems that are expected to have a lower amount of network traffic can often use a less efficient, easier-to-use communication scheme.

In terms of a groupware toolkit, groupware application developers ideally should not have to deal with the specifics of setting up a distributed communication link, but should have the flexibility to specify what information is communicated for a specific communication link and the priority of each. This provides the developers with the ability to fine-tune their applications in terms of efficiency and responsiveness [Greenberg & Roseman 1999]. For this thesis, the toolkit will encapsulate distributed communication issues away from developers as much as possible and allow development effort to be focussed on application-specific details. In doing this, a standard communication middleware was selected from the list of popular distributed communication middleware. Java RMI was selected because of its ease of use in development, platform independence, and free cost. Other candidates that were considered include CORBA, DCOM, or raw TCP or UDP sockets.

2.3 Awareness

When working in a group environment it is important to be aware of what others in the group are doing. In fact, in many circumstances it is crucial to the success of the task being performed because it provides an individual with the necessary context from which to base her actions on. Dourish and Bellotti have defined *group awareness* as an understanding of the activities of others, which provides a context for your own activity [Dourish & Bellotti 1992]. This context is used to ensure that individual contributions are relevant to the group activity as a whole, and to evaluate individual actions with respect to group goals and progress. The information allows groups to manage the collaborative work process.

Groupware applications that provide shared workspaces are concerned with providing one particular type of awareness information called *workspace awareness*, or "the up-to-the-moment understanding of another person's interaction with a shared workspace" [Gutwin & Greenberg 1996]. In synchronous applications, workspace awareness primarily provides information to answer three questions: who is in the workspace, what are they doing, and where are they working. It additionally may provide information regarding when an activity was performed, why it was performed, and how it was performed.

Traditionally, working in a group has meant working in close physical proximity with other individuals. In this type of group environment the majority of awareness information is generated and collected in a passive manner. Individuals can see the actions of others and perhaps even see and hear the decision process that others have gone through before performing the action, as it happens and where it happens. In a computerized groupware environment the individuals are typically distributed across a network in distant locations. In this type of environment, all awareness information must be explicitly collected and distributed by the groupware application. This includes information that is normally implicitly dealt with by the collaborators' senses and intuitions. Determining the requirements for providing awareness in a groupware application to the same degree that it is available in face-to-face collaborative environments is a difficult problem. This problem has often been a topic of research in the area of CSCW. An overview of this concept can be found in [Gutwin & Greenberg 2001], which discusses a framework for providing workspace awareness in real-time groupware applications and forms part of the basis for this thesis work and the discussion found in this section of the proposal. Additionally, examples of awareness support in CSCW applications are discussed in [Dourish & Bly 1992] and [Sohlenkamp & Chwelos 1994].

The awareness framework tries to answer the following three main questions regarding workspace awareness:

- What information makes up workspace awareness?
- How is workspace awareness information gathered?
- How is workspace awareness used in collaboration?

The answers provided by the framework are summarized in the next three subsections.

2.3.1 What Information Makes Up Workspace Awareness?

As mentioned earlier, workspace awareness provides information to help answer several questions pertaining to interactions between users and their workspace. This includes providing information for both the present and the past. Awareness of the present involves providing answers to who, what, and where, while awareness of the past includes information for who, what, where, when, and how. These categories of awareness for each of the past and present focus on particular elements of interaction. The categories are shown in Table 2.1 and Table 2.2, and are discussed in more detail below.

Table 2.1: Elements of workspace awareness relating to the present.

Category	Element	Specific questions
Who	Presence	Is anyone in the workspace?
	Identity	Who is participating? Who is that?
	Authorship	Who is doing that?
What	Action	What are they doing?
	Intention	What goal is that action part of?
	Artifact	What object are they working on?
Where	Location	Where are they working?
	Gaze	Where are they looking?
	View	Where can they see?
	Reach	Where can they reach?

Table 2.2: Elements of workspace awareness relating to the past.

Category	Element	Specific questions
How	Action History	How did that operation happen?
	Artifact History	How did this artifact come to be in this state?
When	Event History	When did that event happen?
Who (past)	Presence History	Who was here, and when?
Where (past)	Location History	Where has a person been?
What (past)	Action History	What has a person been doing?

The elements of workspace awareness of the present describe the current state of a workspace. The elements pertaining to the ‘who’ question include presence, identity, and authorship. Presence is whether anyone is in the workspace, identity identifies a particular individual within the workspace, and authorship identifies who is performing a particular action. The ‘what’ elements are action, intention, and artifact. Action relates to what participants are doing, intention identifies the goals of the actions, and artifact identifies what objects the actions are being performed on. The ‘where’ elements include location, gaze, view, and reach. Location pertains to where participants are working, gaze is where they are looking, view describes what they see, and reach is where in the workspace an individual can reach.

The elements of workspace awareness of the past describe the history of a workspace. The elements for who, what, and where are the historical information for presence, action, and location, respectively. The ‘when’ element is event history. Event history describes when a particular event occurred. The ‘how’ elements are action history and artifact history, which pertain to what a person has been doing in the workspace and how a particular operation happened. Artifact history describes how an artifact came to be in its current state.

2.3.2 How is Workspace Awareness Information Gathered?

Workspace awareness information in synchronous collaboration is generated by three primary sources: from people’s bodies, from workspace artifacts, and from conversations and gestures. There exist three corresponding mechanisms to gather this information: consequential communication, feedthrough, and intentional communication. These three mechanisms are summarized below.

Consequential communication is the mechanism of seeing and hearing other people active in the workspace, or as Segal (1994) defines it, information transfer that emerges as a consequence of a person’s activity within an environment. This type of communication is unintentional and not explicitly generated but is nonetheless key to successful workspace interaction. Studies have indicated that a significant percentage of a workspace participant’s time is spent observing and analyzing other participants’ actions within the workspace [Segal 1994]. In terms of a groupware environment, this means that it is important for the environment to provide each participant with workspace awareness information about the other participants so that consequential communication can occur.

The mechanism of *feedthrough* was introduced in the problem statement of this thesis but will be discussed here for completeness. As previously mentioned, when artifacts are manipulated the manipulation process provides information which would normally be feedback to the person performing the action but can also inform others that are watching [Dix et al 1993]. This mechanism is called feedthrough. There are several types of feedthrough; one particular type is *process feedthrough* [Gutwin & Greenberg 1998]. Process feedthrough provides information about an action as it unfolds; that is, the intermediate steps in the process of performing the action. While feedthrough is implicitly propagated to other users in a real-world workspace, in a groupware environment process feedthrough must be explicitly propagated to participants because of the distributed nature of groupware.

The third mechanism for gathering awareness information is through intentional communication. People perform intentional communication through conversation and gestures. In a groupware environment, intentional communication is the easiest type of workspace awareness to generate because the information is usually explicitly generated by a participant and directed to other participants.

2.3.3 How is Workspace Awareness Used in Collaboration?

The awareness framework identifies that collaborators use workspace awareness information in five different ways: management of coupling, simplification of verbal communication, coordination, anticipation, and assistance. The use of awareness information for these five activities will be discussed below.

In groupware, users often switch between working on tasks individually or with other participants. To determine when to work with others and when to work individually, a user needs to be aware of what others are doing and when they are doing them. Workspace awareness information allows groupware users to manage their coupling with other users. Without workspace awareness information, it would be difficult to coordinate user efforts.

Nonverbal actions allow individuals to simplify their verbal communication with other individuals. These actions are interpreted and understood by other people, therefore eliminating the need for other dialog that would be necessary without these nonverbal actions. The key to nonverbal communication is that the sender must be aware of what the receiver can see in order to create useful nonverbal actions.

According to Gutwin and Greenberg (1998), coordinating actions involves ensuring that actions occur in the correct order, at the right time, and that they meet the constraints of the task. Without workspace knowledge pertaining to what other participants have done, what they are going to do, and what is left to do in a task, a coordinated effort would rarely succeed in completing a task.

Anticipation occurs when people take action based on their expectations or predictions of what others will do in the future. In a groupware environment, it would not be possible to anticipate the actions of others unless workspace awareness information is provided.

The ability of one participant to assist another in a collaborative activity can be greatly assisted by workspace awareness information. It helps an individual to determine why assistance is necessary and how the assistance should be provided.

One of the main focuses of the thesis groupware toolkit is to provide workspace awareness information to the users of the resulting applications. In particular, the toolkit focuses on providing process feedthrough information. In doing so, an important tradeoff between individual power and workspace awareness must be considered. The crux of the tradeoff is the fact that existing groupware designs that provide group awareness capabilities often do so at the expense of the power of an individual user. These issues of the tradeoff are discussed extensively in [Gutwin & Greenberg 2000] and will not be discussed here, although they are addressed in the toolkit.

2.4 Component-Based Development

According to Rogerson (1997), component-based development is the process of breaking a monolithic application into separate pieces called components. These components are like mini-applications that come packaged as a bundle of binary code and can be linked to other components at run-time, via well-defined public interfaces, to form an application. Sparling (2000) provides a more concise definition for a component:

> *"A component is a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces."* (p. 47).

Others that have done work regarding component-based development include [Aoyama 1998], [Kirtland 1999], and [Rine et al. 1999].

There are five main benefits of component-based development ([Rogerson 1997], [Sparling 2000]).

i. Reuse, which is facilitated by the encapsulated nature of a component.
ii. Lower development costs and times for rapidly developed applications.
iii. Distributed components can be created that encapsulate repetitive distributed communication issues.
iv. Components can be plugged and unplugged from an application, because they are independent entities.
v. Version compatibility. Because components have well-defined public interfaces, component version upgrades typically do not require changes to applications that use the components when integrating the new components.

To be successful at component-based development a number of things need to be considered. Components need to be based on a complete specification, and assurances need to be made to ensure that the implemented component complies with that specification once completed. Components should encapsulate their inner workings from their users and provide a clearly defined, easy-to-use public interface to make their functionality accessible to developers. This interface should be unchanging – that it should not change (or change very little) from version to version of the component. This means that the interface should be carefully and completely designed before the component is developed.

Sparling (2000) refers to visual components as a good example of the power of component-based development. Each visual component is an example of an independent package of software services, which is, in essence, a component. The work of this thesis involves the creation of a toolkit that facilitates application creation by assembling visual components. Therefore, the thesis work is a prime candidate for a component-based development strategy. The JavaBeans specification has been selected as the component-based development strategy to implement the work for this thesis and is discussed in the next subsection.

2.4.1 JavaBeans

The toolkit resulting from the work for this thesis is built in Java [Topley 1998] and Swing [Drye & Wake 1999] and will make extensive use of the JavaBean component-based development standard. This section provides an introduction to Sun's JavaBeans specification and style of component-based development [Sun Microsystems 1997].

A JavaBean is a reusable software component that can be manipulated visually in a builder tool. JavaBeans include software elements such as GUI widgets, database viewers, and software entities that do not have a visual presence within a running application. These non-visual entities can still be visually manipulated within a builder tool. A builder tool does not necessarily allow complete visual manipulation of a JavaBean component; instead it may provide convenient ways to piece JavaBeans together, such as through a scripting language.

JavaBeans typically provide support for five component characteristics: introspection, customization, events, properties, and persistence. These characteristics are discussed in detail below.

a) Introspection

Introspection is the process used by a builder tool to determine the properties, methods, and events supported by a JavaBean. At a low level, Java provides an Introspector class that a builder tool can use to perform introspection on JavaBeans. The Instrospector performs the introspection of a JavaBean in one of two ways:

1. If the JavaBean provides what is known as a BeanInfo class with the JavaBean, the Introspector will use the BeanInfo class to learn the characteristics of the JavaBean.
2. If no BeanInfo class is provided, the Introspector will use the Java Reflection mechanism to learn about the JavaBean. This means that a JavaBean developer must follow the design patterns and naming conventions for creating a JavaBean, if the Introspector is to successfully use reflection to learn about the JavaBean.

b) Customization

Customization allows a JavaBean user to modify the look and behavior of a JavaBean to suit the application being developed. Builder tools typically use introspection to determine the set of customizable properties of a JavaBean and will list these properties in a property sheet that allows a user to modify the values of the properties. For more complex JavaBeans, the JavaBean developer may wish to provide a more sophisticated method of modifying the properties of a JavaBean. The JavaBean specification allows for this by providing specification for a JavaBean Customizer class. This customizer class is a Java AWT (Abstract Windowing Toolkit) component that can be opened within a builder tool. A Wizard is an example of a JavaBean customizer.

c) Events

Events are the mechanism used by JavaBeans to communicate with each other. Events are generated by source JavaBeans. Other JavaBeans will register with the source as event listeners and will receive the generated event and respond by performing some action based on the event. Events are typically used by a JavaBean to notify listeners of a change in its state. In GUI environments, events are used to notify components of mouse and keyboard input, as well as state changes in components. Builder tools usually provide the capability to easily have a component generate events and add and remove event listeners from the component in a graphical manner. The JavaBean specification provides for both unicast and multicast event delivery. This means that an event will allow only one listener for a particular event type in the unicast situation, but will allow many listeners for a particular event type in the multicast situation.

d) Properties

Properties are discrete, named attributes of a JavaBean that can affect its appearance or its behavior. A JavaBean property may be modified in different ways:

1. Programmatically through *getter* and *setter* methods.
2. Through the builder tools property sheet mechanism or through a JavaBean Customizer.

The properties of JavaBeans are usually persistent, such that design-time modifications will not be lost between customization sessions. The *getter* and *setter* methods of a JavaBean are commonly referred to as the property accessor methods. These accessor methods need to conform to a particular design pattern in order for builder tools to recognize them through introspection and allow property modification through a property sheet. Properties may be *bound* properties. Bound properties allow other components to be notified when a property of a JavaBean has been modified. Properties may also be *constrained* properties. If a property is constrained, another property is performing validation when it has changed, and potentially vetoing the change based on the validation.

e) Persistence

Because JavaBeans are components that allow design time customization, they need the ability to store their initial state and settings assigned to them at design time within the scope of a specific application. This includes storing the property settings for all exposed properties and may include storing unexposed property settings as well. The states of any JavaBean components that are part of the JavaBean are also saved. Pointers and references to external JavaBeans are not stored. These links are typically re-established at run-time by a higher-level software entity. The default persistent storage method for a JavaBean is through Java Object Serialization, but the Externalization option allows a custom persistence method to be used for a JavaBean.

2.5 *Groupware Toolkits*

A toolkit is designed to assist developers in creating applications of a specific nature by automating various common and repetitive development tasks for the developer. Existing toolkits provide a range of features (eg. [Greenberg & Roseman 1992, 1996, 1999]; [Schuckmann et al. 1996, 1999]; [Hill et al. 1994]; [Graham 1995], [Graham et al. 1999]; [Begole et al. 1998]; [Sun Microsystems 1999]; [Banavar et al. 1998]; [Bharat & Brown 1994]):

- Providing programmers with high-level constructs to deal with low-level issues without masking those low-level issues.
- Writing code shells, snippets, or scripts for common or well-known mechanisms such as for that of event handlers, callbacks, architecture framework shells, and database connectivity and usage.
- Linking generic, reusable binary components into an application.
- Providing a direct-manipulation environment that makes it easy to assemble components, commonly through drag-and-drop.
- Providing component customization through the ability to modify properties of a component to suit the needs of the application.
- Providing a development methodology with rules and conventions that a developer may follow and that the toolkit will make easy for the developer to follow.

The additional requirements that groupware has over single-user software impose additional requirements and desired features for groupware toolkits ([Greenberg & Roseman 1999]; [Shuckmann et al. 1996]). These additions include:

- Providing architectural support such that the toolkit makes the development of multi-user distributed systems easier. Ideally, the choice of architecture should be left to the developer. The toolkit should provide architectural support that is flexible enough to support numerous groupware architectures and should be adaptable to numerous types of groupware.
- Groupware-specific programming abstractions and components. These will assist by providing functionality pertaining to groupware issues such as

synchronization, coupling, concurrency, distributed communication, and the user interface.

- Automated support for process management and distributed communication and message passing. This involves abstracting out the low-level details of each.
- Reusable groupware widgets. These may include both multi-user versions of single-user widgets and groupware-specific widgets.
- Session management capabilities.

Because multi-user applications are a newer and more complex concept than single-user applications, the existing multi-user toolkits do not yet meet all their requirements to the same extent as other toolkit types. The next subsection will look at some of the existing groupware toolkits in terms of how they meet the groupware toolkit requirements listed above. The discussion does not include an exhaustive list of all existing groupware toolkits, but rather a representative sample of some of the more prevalent toolkits for the different types of groupware toolkits that can be found. The section will finish by discussing how the proposed toolkit will help to fill the void and meet some of the groupware toolkit requirements that have not previously been met by the existing groupware toolkits.

2.5.1 A Look at Some Existing Groupware Toolkits

Groupware toolkits are created to provide support for the creation of the elements of groupware applications. The toolkits provide tools that help create these elements by using common and proven designs and styles and by making use of abstraction. The elements and design styles that have been addressed by groupware toolkits include distribution architecture, architectural style, distributed communication and message passing, direct-manipulation graphical user interfaces, and group awareness issues.

Most of the groupware toolkits currently in existence provide a powerful environment for the creation of groupware applications and provide support for architectural issues associated with the distributed nature of groupware applications. In terms of distribution architectures, toolkits such as Groupkit [Roseman & Greenberg

1999], COAST [Schuckmann et al. 1999], and Flexible JAMM [Begole et al. 1998] support replicated architectures, while Rendevous [Hill et al. 1994] has a centralized architecture. Clock ([Graham et al. 1996]; [Greenberg & Roseman 1999]) is a toolkit which goes one step further and allows flexibility in the choice of architecture by hiding the details of the architectural implementation from the developers through the use of precisely defined semantics. A Notification Server [Patterson et al. 1996] is a centralized software entity to be used in an otherwise replicated architecture (resulting in a semi-replicated architecture). It allows data to be shared between clients without a complex concurrency control scheme, due to the centralized nature of the server. One of the biggest appeals of Notification Server is the fact that it contains no knowledge of the application it is placed within and is, therefore, generic and reusable.

The Model-View-Contoller (MVC) architectural style is the most prevalent among existing toolkits, including COAST, Clock, and GroupKit. The Abstract-Link-View (ALV) architectural style can be found in Rendevous [Hill et al. 1994]. Both styles involve the separation of a data model from one or more views of the data, and the toolkits provide sufficient facilities to create software that easily conforms to the selected style. The creation of the model is kept separate from the views, and the developers only need to define the application model and specify the associations between the model and the views. The updating of and consistency maintenance for the views and all concurrency issues are taken care of by the toolkit.

The distributed communication and message passing issues of existing toolkits are abstracted away from the developer to varying degrees depending on the toolkit. For example, in terms of distributed communication, Groupkit requires a developer to have knowledge about programming remote procedure calls (RPCs). The toolkit simplifies the process somewhat by allowing multicast RPCs to be specified in the same fashion as normal RPCs and abstracts away low-level details, such as knowledge of file descriptors and sockets. Nonetheless, the programmer still must create the RPCs specifying the required arguments when a distributed call is to be made. Groupkit also allows for the specification of events to communicate interesting occurrences to interested listeners of the events. Some events are generated automatically by the system, such as those for controlling session management, while the developer can define application-specific events and handlers. Rendezvous, Notification Server, Shared Data Objects (SDO) [Banavar et al. 1998], and Flexible JAMM also use event or notification mechanisms for distributed communication. For many cases, events for significant occurrences are generated by the toolkit infrastructure and the developer needs only to create handlers (callbacks) to deal with the events in an application specific way. At a low level, the Java Shared Data Toolkit (JSDT) [Sun Microsystems 1999], allows for the use of either TCP-IP sockets, the HTTP protocol, or LRMP (a light-weight Java multicast package) to provide distributed message passing. The choice of protocol is made by simply passing an argument to the JSDT registry at startup. At a higher level, the JSDT uses an event mechanism to provide message passing.

In terms of the user interface of groupware toolkits, several of the existing toolkits provide support for a direct-manipulation GUI environment. Toolkits providing the MVC and ALV architectures provide the ability to create multiple views of the models and abstractions. Updates to these views are handled dynamically by the toolkit infrastructure. For the most part, developers need only to design the views and attach the views to the model data. Many of the toolkits, including Groupkit, COAST, Flexible JAMM, Rendezvous, Clock, SDO, and Visual Obliq [Bharat & Brown 1994], allow relaxed-WYSIWIS (what-you-see-is-what-I-see) views such that some degree of both coupled and individual work is allowed. Toolkits that provide GUI support will usually provide a widget set for developing the GUIs that includes some multi-user versions of standard single-user widgets and, in the cases of Groupkit, Flexible JAMM, SDO, and COAST, additionally provide groupware-specific controls, such as multi-user scrollbars and radar views. Groupkit, SDO, COAST, Rendevous, and Visual Obliq provide the capability to build custom widgets by extending existing widgets or coding completely new widgets with custom functionality. While JSDT is an additional package for Sun's Java and thus allows GUI development using AWT and Swing, it does not directly provide any multi-user GUI support. Some of the toolkits provide the ability to customize widgets to behave in relation to certain groupware principles. For example, Suite [Dewan & Choudhary 1995] allows developers to set the level of coupling on the widgets in its widget set by setting their coupling attributes. Suite also allows access control to be set on its widgets through collaboration rights. This type of flexibility makes the widget set of a toolkit highly reusable over a broad range of groupware applications without the user having to specifically code the functionality into the application. Unfortunately, few toolkits provide this capability.

Group awareness is a concept that has not been addressed to any significant extent by existing groupware toolkits. Some toolkits provide awareness-rich widgets, such as radar views, for location and view awareness [Begole et al. 1998] and participant status for identity awareness ([Begole et al. 1998], [Roseman & Greenberg 1999], [Bharat & Brown 1994]), but there is little support for other types of awareness information, such as feedthrough and consequential communication. Further, the multi-user and awareness-rich widgets that are present in the toolkits are not customizable. They provide one level of functionality only, and cannot be tailored at run-time or design-time to scale the amount of awareness provided based on the operating conditions of the environment.

There are additional capabilities that existing groupware toolkits are lacking that would assist groupware application developers. While some toolkits provide GUI builders and the capability to build a groupware GUI through direct-manipulation ([Schuckmann et al. 1999], [Hill et al. 1994], [Bharat & Brown 1994]), few toolkits allow construction of complete groupware applications through a direct-manipulation builder tool.

2.5.2 Summary of Existing Toolkits

Table 2.3 below, summarizes the highlights and drawbacks of the representative sample of existing toolkits introduced in the previous section.

Table 2.3: Groupware toolkit summary.

Highlights	Drawbacks
COAST [Schuckmann et al. 1996, 1999]	
• *Architecture*: fully replicated, MVC; two shared models (application and domain); coupling on a per attribute basis, implemented in Smalltalk; uses TCP/IP connections. • *User Interface*: UI builder; extensible widgets; small set of simple awareness widgets.	• Applications must be built in Smalltalk, which is no longer commonly used. • Visual support for building applications is limited to the user interface. • Does not provide generic, reusable groupware-specific widgets.
GroupKit [Roseman & Greenberg 1992, 1995], [Greenberg & Roseman 1999]	
• *Architecture*: mostly replicated with a centralized conference application registrar; session managers provide access to conference registrar and applications; MVC with a shared key-value pair based data model; built using C++ and X-Windows, uses multi-cast RPCs or events with callbacks; includes a standard set of events. • *User Interface*: provides a number of groupware-specific widgets such as participant status display and telepointers.	• Programmers must be concerned with concurrency control and synchronization. • Programmers must deal with low-level communication issues. • Lacks a good GUI builder. • Lacks a generic and reusable set of awareness widgets.
Rendezvous [Hill et al. 1994]	
• *Architecture*: centralized; ALV; uses constraints to maintain consistency between model and views; built using LISP; CLOS; and X-Windows. • *User interface*: uses declarative graphics model with powerful rendering base class; event-driven GUI model with many standard events; GUI builder; has standard set of widgets; can create new widgets through assembly of existing widgets.	• Potentially slow because of centralized architecture. • Widgets are proprietary. They are created to mimic the Motif widget set but are not derived from it. • Little support for group awareness. • Tied to X-Windows.
Clock [Graham 1995], [Graham et al. 1999], [Greenberg & Roseman 1999]	
• *Architecture:* centralized, semi-replicated, or hybrid, implemented through an abstract architecture; allows rapid, high-level development by abstracting details of distribution, networking, and concurrency; MVC. • *User Interface:* uses a declarative programming language to build user interfaces.	• Only pessimistic concurrency is supported, which results in poor performance on slow or congested networks. • Uses a proprietary programming language – the Clock language. • GUI's are built using an unfamiliar declarative language process.
Flexible JAMM [Begole et al. 1998]	
• *Architecture*: replicated; provides collaboration transparency through dynamic object replacement to turn single-user applications into multi-user applications; Java- and Swing-based; provides explicit and implicit floor control. • *User interface*: provides telepointers, telecursors, a radar view, and a shared text editor; allows run-time customization of telepointers.	• Does not support new development, only the conversion of existing single-user applications into groupware applications. • Restricted to Swing and serialization. • It is limited to 4 types of multi-user widgets.

JSDT [Sun Microsystems 1999]	
Highlights:	Drawbacks:
• *Architecture:* Java-based; supports a number of protocols including sockets (TCP/IP), HTTP, and LRMP (lightweight remote multicast package); shared data channels with session objects; event model; provides shared data types including byte arrays and tokens; object managers control access to shared resources; supports authentication policies. • *User Interface:* no support other than Swing.	• Does not provide any mutli-user UI support. • Does not provide group awareness support. • Provides only a few low-level shared data types.
Shared Data Objects (SDO) [Banavar et al. 1998]	
Highlights:	Drawbacks:
• *Architecture:* built on top of a client-server synchronous collaboration infrastructure called *Live*; centralized server; event channels; conforms to JavaBeans specification; component-based architecture; encapsulates distributed communication and sychronous collaboration issues within the components; components are extensible through a high-level API; visually programmable components including ones for conference setup, typed data sharing, application synchronization, event and media streams. • *User Interface:* visually programmable components for user awareness; integrates with JavaBeans-based GUI builders.	• Built on top of a proprietary distributed infrastructure, *Live,* rather than a standard middleware. • Requires low-level programming knowledge, such as that concerning event channels and other low-level programming constructs. • Supports only a centralized server architecture. • It is not evident to what extent awareness is supported. It appears to be limited. • Focus is on visual programming rather than meeting awareness needs.
Visual Obliq [Bharat & Brown 1994]	
Highlights:	Drawbacks:
• *Architecture:* applications are developed by building forms via direct-manipulation with a GUI builder and adding callback code; provides rapid application development and low turnaround time; allows multi-threaded callbacks; callback synchronization through locks; runs within the GUI builder or as a standalone application. • *User Interface:* property sheets for each widget; extensible widget set – any FormsVBT widget can be added.	• Proprietary GUI builder. • Everything is forms-based. Many groupware applications require a more complex or different GUI environment. • Uses an uncommon development language – Obliq.

3 Toolkit Requirements

This chapter discusses the first step of the thesis solution, classification of GUI widgets in terms of groupware requirements. The requirements analysis phase for this development involves 3 activities:

(a) Identifying the types of information that groupware applications communicate, and the mechanisms within groupware applications that are used to communicate the information.

(b) Identifying the types of widgets currently used within applications – both single-user widgets and groupware widgets.

(c) Determining the information and mechanisms from (a) that are to be supported by the widgets within the new toolkit. The widgets of the new toolkit will be a subset of the widgets identified in (b). From this point forward the toolkit will be called the MAUI (Multi-User Awareness User Interface) toolkit.

3.1 Identification of Groupware Information

This step involves using the information gathered through existing literature to identify the types of group awareness and other groupware information that is used in existing groupware applications. This step was discussed thoroughly through the background work introduced in Chapter 2 and will not be discussed here.

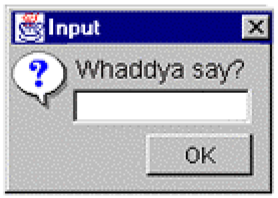
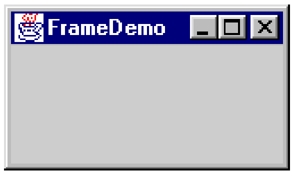
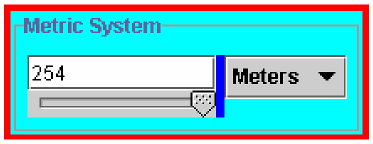
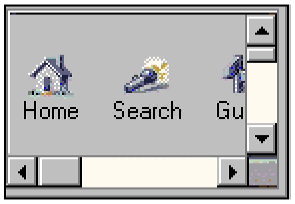
3.2 Types of Widgets

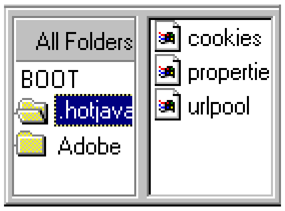
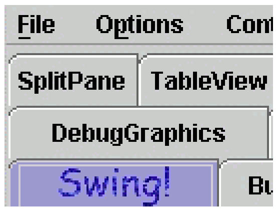
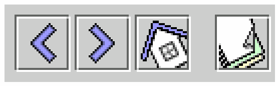
In groupware user interfaces, there are two main types of widgets that exist: those extended from single-user widgets, and those components that exist only in groupware.

3.2.1 Single-User Widgets

There are six widget classifications identified by Sun within the Java Swing Tutorial [Sun Microsystems 2002 A], which are representative of widget classification in general for existing single-user graphical user interfaces. These classifications and the widgets within each classification are shown in Table 3.1, below. The classifications include *top-level containers, general-purpose containers, special-purpose containers, basic controls, uneditable information displays, and editable displays of formatted information.*

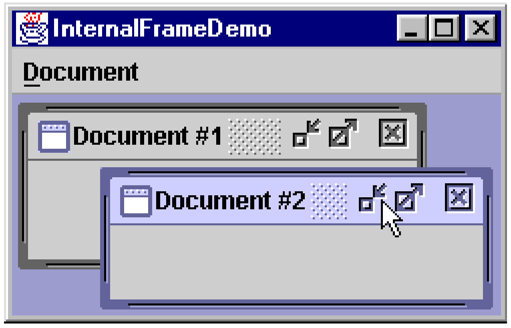
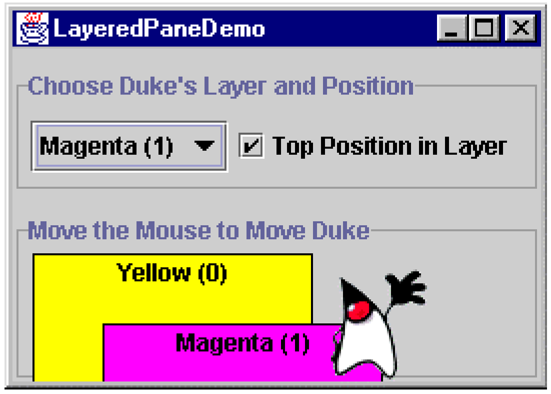
Table 3.1: Widgets of the Java Swing GUI Component Library

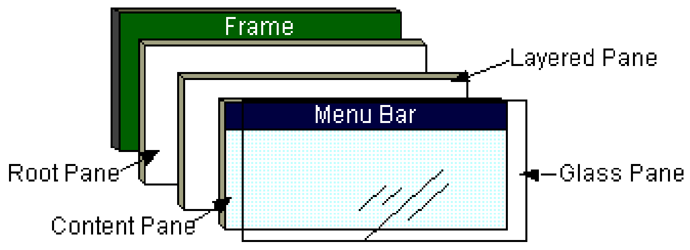
Top-Level Containers These are the components at the top of a containment hierarchy. Components can be assembled in a top-level container to collectively comprise a stand-alone user interface for an application.	
	Applet. An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.
	**Dialog.** A dialog is a top-level window with a title and a border that is typically used to take some form of input from the user.
	**Frame.** A frame is a top-level window with a title and a border.
General-Purpose Containers These are intermediate containers that can be used under many different circumstances to provide a specific type of containment relationship.	
	**Panel.** A panel is used to group related components together. The relationship between the components is often a visual relationship that is based solely on component layout.
	**Scroll Pane.** A scroll pane contains another container component. It provides the ability to navigate within the workspace of a component, which has a workspace larger than the viewable area. The viewport can be scrolled to various areas of the workspace.

 All Folders BOOT .hotjava Adobe cookies propertie urlpool	**Split Pane.** A split pane contains two other containment components. It allows the viewports of the two containers two be resized with respect to each other, such that the sum of the two viewable areas is equal to the viewable area of the split pane
 File Options Con SplitPane TableView DebugGraphics Swing! Bu	**Tabbed Pane.** A tabbed pane provides containment functionality similar to a tabbed binder. Container components can be added to the tabbed pane to represent the pages of the binder. Each tabbed container can be made visible by selecting its tab in the tab pane. The contents of only one tab at a time can be made visible within the tabbed pane.
	**Tool Bar.** A tool bar typically contains a row of basic controls used to that can be accessed to provide quick access to some application functionality.

Special-Purpose Containers

These are intermediate containers that play special roles within the graphical user interface.

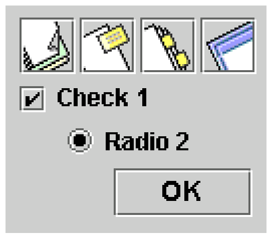
	**Internal Frame.** Internal frames are windows that can be contained within other windows. These are typically used to create an application workspace known as a multi-document interface or a desktop.
 LayeredPaneDemo Choose Duke's Layer and Position Magenta (1) Top Position in Layer Move the Mouse to Move Duke Yellow (0) Magenta (1)	**Layered Pane.** Typically, components within a GUI containment hierarchy have only a two-dimensional relationship to each other. Layered panes are used in conjunction with a frame to allow a three-dimensional component hierarchy.



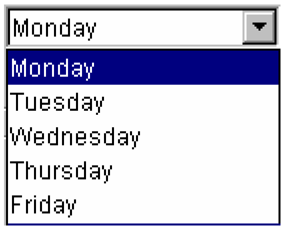
Glass Pane. Glass panes can be used as a cover over the root pane of a frame. The root pane is the component container of a frame. The glass pane can intercept mouse and keyboard events intended for components within the frame's root pane. These events can be modified, redirected, or thrown away, as is appropriate for an application. Additionally, a glass pane can be used as a canvas on which to paint over the frame as a whole.

Basic Controls

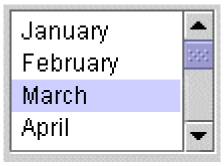
These are the atomic components that exist primarily to get input from the user. Basic controls generally also show simple state. They include controls such as buttons (push, check box, radio, and toolbar), combo boxes, lists, menus, sliders, and text fields.



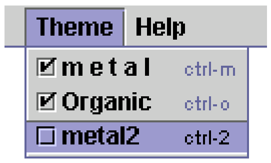
Buttons. Buttons are widgets used to trigger actions or change the state of data when manipulated.



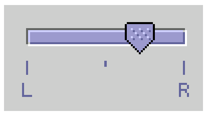
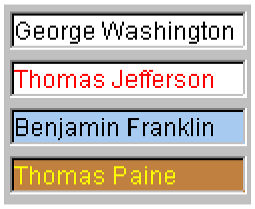
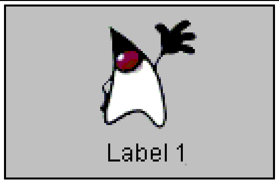
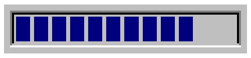
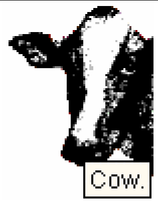
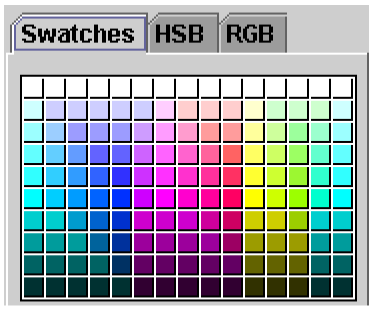
ComboBox. A component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request. If you make the combo box editable, then the combo box includes an editable field into which the user can type a value.

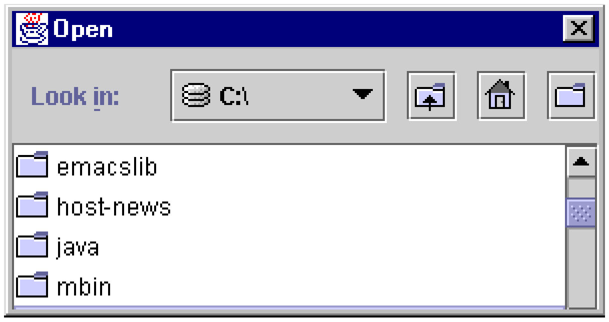
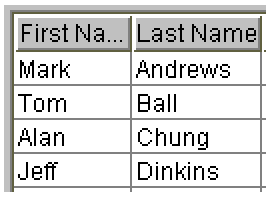
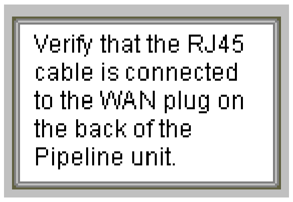
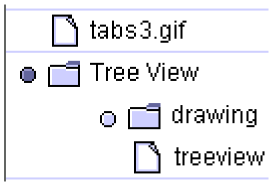


List. A list presents the user with a group of items, displayed in a column, to choose from. Lists can have many items, so they are often put in scroll panes.



Menu. A menu provides a space-saving way to let the user choose one of several options.

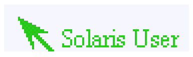
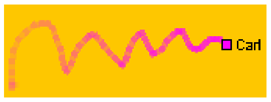
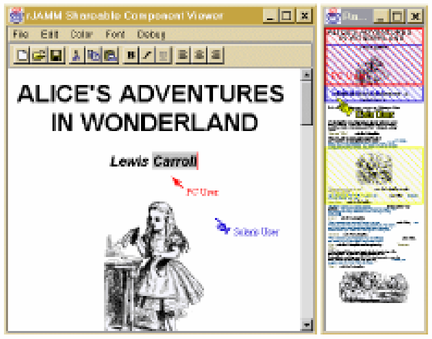
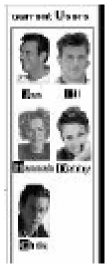
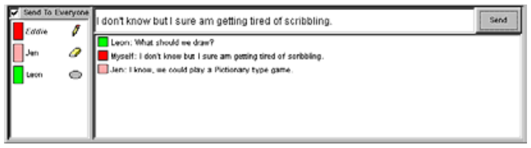
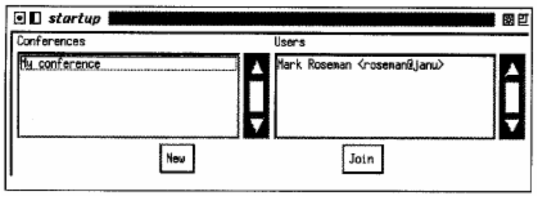
	**Slider.** A component that lets the user graphically select a value by sliding a knob within a bounded interval.
 George Washington Thomas Jefferson Benjamin Franklin Thomas Paine	**Text Field.** A component that allows the editing of a single line of text.
Uneditable Information Displays These are atomic components that exist solely to give the user information. A user does not interact with these components to execute actions or change the system state.	
	**Label.** A display area for a short text string or an image, or both. A label does not react to input events.
	**Progress Bar.** A component that, by default, displays an integer value within a bounded interval. A progress bar typically communicates the progress of some work by displaying its percentage of completion and possibly a textual display of this percentage.
	**Tool Tip.** Used to display a "Tip" for a Component.
Editable Displays of Formatted Information These are atomic components that display highly formatted information that can be edited by the user. Examples of editable displays of formatted information include tables, multi-line text displays that can display various formats of text (eg. HTML), hierarchical tree displays, file choosers and color choosers.	
	**Color Chooser.** A color chooser provides a pane of controls designed to allow a user to manipulate and select a color.

 Open Look in: C:\ emacslib host-news java mbin	**File Chooser.** File choosers provide a GUI for navigating the file system, and then either choosing a file or directory from a list or entering a file name or directory name.
 First Na... / Last Name Mark / Andrews Tom / Ball Alan / Chung Jeff / Dinkins	**Table.** A table is used to display and edit regular two-dimensional tables of cells.
 Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.	**Text.** Swing's text components display text and optionally allow the user to edit the text.
 tabs3.gif Tree View drawing treeview	**Tree.** A control that displays a set of hierarchical data as an outline.

3.2.2 Dedicated Groupware Widgets

For developing multi-user graphical user interfaces, the existing CSCW literature identifies the following additional widgets that do not have a single-user counterpart and, therefore, cannot be found in standard GUI component libraries [Ackerman & Starr 1995]. However, some of the existing groupware toolkits provide support for some of the widgets and techniques listed below. It should be noted that some of the items listed below are more of a technique than a widget. An example of these widgets and techniques is shown in Table 3.2, below.

Table 3.2: Multi-User Widgets

	**Telepointer**. A representation of a user's cursor on a remote user's GUI. A telepointer is used to show workspace location and may convey additional awareness information such as user state.
	**Telepointer Traces/Trails**. A trail of previous telepointer locations for a user on a remote user's GUI used to show location history. Telepointer traces typically show telepointer locations for a fixed time period.
 GroupKit / JAMM Shareable Component Viewer File Edit Color Font Debug ALICE'S ADVENTURES IN WONDERLAND Lewis Carroll PC User Solaris User	**Radar View**. A high-level miniature overview of the full workspace that typically denotes remote user positions and the locations of workspace artifacts. Radar views typically allow user interaction for workspace navigation.
	**Participant Display**. Displays information on all users of the group.
 I don't know but I sure am getting tired of scribbling. Leon: What should we draw? Myself: I don't know but I sure am getting tired of scribbling. Jen: I know, we could play a Pictionary type game.	**Chat Tool**. Facilitates textual communication between users.
 startup Conferences Users My conference Mark Roseman <roseman@janu> New Join	**Session Manager User Interface**. Provides facilities for connecting and disconnecting to groupware network.

3.3 Awareness Information for Each Widget Type

For each category of single-user widget and the list of multi-user widgets identified above, the following sections will list a table of the features for each widget the toolkit supports.

3.3.1 Multi-User Versions of Single-User Widgets

a) *Top-Level Containers*

The frame will not have any awareness features itself, but will be implemented to provide convenience when using the other awareness widgets that a frame may contain.

b) *General-Purpose Containers*

Table 3.3 shows the toolkit support for a scroll pane, scrollbar, and viewport.

Table 3.3: Awareness features provided by MAUI for scroll panes, scrollbars, and viewports.

Scroll Pane, Scrollbar, and Viewport	
WA Element	**Component's Support Features**
Present	
Who	
Presence	The scroll pane needs to indicate the location of other users within the workspace. (This will be done by implementing multi-user scroll bars that show the scroll locations of all users within the workspace).
Identity	Each user needs to be uniquely identified within the workspace (done by use of color).
What	
Action	Show feedthrough when control is used by reflecting manipulations of the scrollbars, for both the buttons and thumb.
Intention	Indicate when someone is going to use the scrollbars – either the thumb or the buttons (done by highlighting the buttons and thumb).
Artifact	Indicated through the Presence and Intention actions.
Where	
Location	Indicated through the Presence and Intention actions.
View	The viewports of other users should be indicated within the viewport of the local user (View rectangles will be implemented within the scroll pane. The view rectangle of a remote user will be visible through the viewport of the local user's scroll pane when there is overlap between the viewports of the local and remote user's.)

c) Special-Purpose Containers

The glass pane will be used to implement telepointers, traces, and various transparency effects. It is more of a mechanism for providing awareness effects than a widget.

d) Basic Controls

Table 3.4 shows the toolkit awareness support for the various types of button widgets, including push buttons, radio buttons, check boxes, and toggle buttons. Table 3.5 shows the support for combo boxes; Table 3.6 for lists; Table 3.7 for menus and menu items, Table 3.8 for sliders, and Table 3.9 for text fields.

Table 3.4: Awareness features provided by MAUI for buttons.

Buttons (push, radio, check, and toggle)	
WA Element	**Component's Support Features**
Present	
Who	
Presence	Indicate a user's presence over a button (done with highlighting of some kind – border, background, etc)
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Exhibit feedthrough with the button by animating it in some fashion (probably it's normal fashion) when it is manipulated. Also, if the button has state the data model for the button should be updated. The state will be shared between users (i.e. only shared state buttons will be done). {Perhaps mention that buttons with individual states for each user can exist, although not considered in this thesis}.
Intention	Done via presence.
Where	
Location	Indicated through the Presence and Intention actions.
Past	
How	
Action History	Indicate the last user to manipulate the button. This is most useful with buttons that exhibit state - radio buttons, check boxes, and toggle buttons.
Artifact History	Shown through action history.
When	
Event History	Show when the last manipulation occurred.
Who	
Presence	Shown through action history and event history.
What	
Action History	Indicated to some degree through action history.

Table 3.5: Awareness features provided by MAUI for combo boxes.

Combo Box	
WA Element	**Component's Support Features**
Present	
Who	
Presence	Indicate a user's presence over the combo box (done with highlighting of some kind – border, background, etc)
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Exhibit feedthrough with the combo box by animating it in some fashion (probably its normal fashion) when it is manipulated. This may involve animating the button, displaying the dropdown list and indicating the selected item in real time as it changes. The animation should keep from distracting the local user from his work, while providing a sufficient information about the manipulation.
Intention	When a user manipulates a combo box, the dropdown list will highlight the item that currently has the focus in the list to show which item the user is going to select. This indication should be displayed in some fashion to the remote user
Where	
Location	Indicated through the Presence and Intention actions.
Past	
How	
Action History	Indicate the last user to manipulate the combo box and the previously selected item (or items) in the combo box, to indicate the change in state and when it happened.
Artifact History	Shown through action history.
When	
Event History	Show when the last manipulation occurred.
Who	
Presence	Shown through action history and event history.
What	
Action History	Indicated to some degree through action history.

Table 3.6: Awareness features provided by MAUI for lists.

List	
WA Element	**Component's Support Features**
Present	

Who	
Presence	Indicate a user's presence over the list (done with highlighting of some kind – border, background, etc)
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Exhibit feedthrough with the list by animating it in some fashion (probably its normal fashion) when it is manipulated. This may involve changing the highlighted item in real time to reflect the change on the remote user's GUI.
Intention	Done via presence.
Where	
Location	Indicated through the Presence and Intention actions.
Past	
How	
Action History	Indicate the last user to manipulate the list and the previously selected item (or items) in the list, to indicate the change in state and when it happened.
Artifact History	Shown through action history.
When	
Event History	Show when the last manipulation occurred.
Who	
Presence	Shown through action history and event history.
What	
Action History	Indicated to some degree through action history.

Table 3.7: Awareness features provided by MAUI for menus and menu items.

Menu and Menu Items	
WA Element	**Component's Support Features**
Present	
Who	
Presence	Indicate a user's presence over the menu (done with highlighting of some kind – border, background, etc)
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Exhibit feedthrough with the menu by animating it in some fashion (probably its normal fashion) when it is manipulated. This may involve displaying the dropdown list and indicating the selected item in real time as it changes. The animation should keep from distracting the local user from his work, while providing a sufficient information about

	the manipulation. Also, if the menu item has state the data model for the button should be updated. The state will be shared between users (i.e. only shared state items will be done). {Perhaps mention that items with individual states for each user can exist, although not considered in this thesis}.
Intention	Done via presence.
Where	
Location	Indicated through the Presence and Intention actions.
Past	
How	
Action History	Indicate the last user to manipulate the menu and the previously selected item (or items) in the list, to indicate the change in state and when it happened.
Artifact History	Shown through action history.
When	
Event History	Show when the last manipulation occurred.
Who	
Presence	Shown through action history and event history.
What	
Action History	Indicated to some degree through action history.

Table 3.8: Awareness features provided by MAUI for sliders.

Slider	
WA Element	**Component's Support Features**
Present	
Who	
Presence	Indicate a user's presence over the slider (done with highlighting of some kind – border, background, etc)
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Exhibit feedthrough with the slider by animating it in some fashion (probably its normal fashion) when it is manipulated. This may involve animating the thumb as it moves. The animation should keep from distracting the local user from his work, while providing a sufficient information about the manipulation. Because the slider has state the data model for the slider should be updated. The slider may exhibit state in two ways: one shared state for all users or each user with individual state. Both these situations should be accounted for.
Intention	Done via presence.
Where	

Location	Indicated through the Presence and Intention actions.
Past	
How	
Action History	Indicate the last user to manipulate the slider and the previous state (or states), to indicate a change in state and when it happened. Note, that more history information will need to be maintained in a slider that shows individual states for each user.
Artifact History	Shown through action history.
When	
Event History	Show when the last manipulation occurred.
Who	
Presence	Shown through action history and event history.
What	
Action History	Indicated to some degree through action history.

Table 3.9: Awareness features provided by MAUI for text fields.

Text Field	
WA Element	**Component's Support Features**
Present	
Who	
Presence	Indicate a user's presence over the text box (done with highlighting of some kind – border, background, etc)
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Exhibit feedthrough with the text box by animating it in some fashion (probably its normal fashion) when it is manipulated. This may involve populating the remote client's text boxes with the contents of the text box on the local user's machine (i.e. keeping the contents in sync.) (Note that the text box may be designed such that it maintains content on a per user basis. If so, this must be indicated.)
Intention	Done via presence. Also, when a user highlights text in the text box this should be shown in remote user's text boxes.
Where	
Location	Indicated through the Presence and Intention actions. It should also be indicated by showing the cursor position for each user of the text box, as is done for the single user in a single-user text box.
Past	
How	
Action History	Indicate the last user to manipulate the text box and the previous state (or states), to indicate a change in state and when it happened. Note, that more history information will need to be maintained in a text box that shows individual content for each user.

Artifact History	Shown through action history.
When	
Event History	Show when the last manipulation occurred.
Who	
Presence	Shown through action history and event history.
What	
Action History	Indicated to some degree through action history.

e) *Editable Displays of Formatted Information*

Table 3.10 shows the toolkit awareness support for a Tree.

Table 3.10: Awareness features provided by MAUI for a tree.

Tree	
WA Element	**Component's Support Features**
Present	
Who	
Presence	Indicate a user's presence over a branch in the tree(done with highlighting of some kind – border, background, etc)
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Exhibit feedthrough with the tree by animating it in some fashion (probably its normal fashion) when it is manipulated. If this is a shared state control than the expansion and contraction of the tree should occur on other user's displays. In a control where individual state is maintained for each user, the tree needs to indicate the expansion, contraction, and traversal activities without changing the view of the local user.
Intention	Done via presence.
Artifact	The branch selection should be maintained whether or not the control is shared or maintains individual state for each user. If individual state is maintained the indication of another user's selected item needs to percolate up the branch hierarchy to the highest visible branch in the local user's view so that the local user can find the actual location of a remote user, if desired.
Where	
Location	Indicated through the Presence, Intention, and artifact actions.
View	A mode on the tree could be supported that allows the local user to view the visible tree hierarchy of the remote users, or even make his view the same as another's view.
Past	
How	

Action History	Indicate the last user to manipulate a branch in the tree and select each leaf node.
Artifact History	Shown through action history.
When	
Event History	Show when the last manipulation occurred.
Who	
Presence	Shown through action history and event history.
What	
Action History	Indicated to some degree through action history.

3.3.2 Dedicated Groupware Widgets

Table 3.11 shows the toolkit awareness support for telepointers and telepointer traces; Table 3.12 for a radar view; Table 3.13 for a participant display; and Table 3.14 for a chat component.

Table 3.11: Awareness features provided by MAUI for telepointers and telepointer traces.

Telepointer and Telepointer Traces	
WA Element	**Component's Support Features**
Present	
Who	
Presence	Existence of a telepointer by definition indicates that another user is present.
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	Movement of the telepointer in real-time in relation to other controls and the workspace as the user moves the cursor provides some indication as to what is happening.
Intention	Same reasoning as Action.
Artifact	Location of the telepointer indicates what object is being manipulated.
Where	
Location	Telepointers by definition indicate location.
Past	
When	
Event History	If a telepointer has not moved in a long period of time its image could be changed to reflect that the user is inactive.
Who	

Presence	Traces give a short-term indication of where a user was. A fading effect can give some indication of time.
Where	
Location History	Same as for Presence.
What	
Action History	Traces help to allow a user to decipher the movements and actions of other users.

Table 3.12: Awareness features provided by MAUI for radar views.

Radar View	
WA Element	**Component's Support Features**
Present	
Who	
Presence	The miniature view shows a small summary view of the main view and, therefore, will show presence the same as the main view.
Identity	Presence indication should be done in such a way that identity of the user can be easily determined (done via color).
What	
Action	The radar view should be updates in real-time, as the main view is updated.
Where	
Location	Individual user's locations should be shown on the radar view.
View	The views of individual users should be indicated in the radar view.
Reach	Indicated by View.

Table 3.13: Awareness features provided by MAUI for participant displays.

Participant Display	
WA Element	**Component's Support Features**
Present	
Who	
Presence	By definition the participant display shows all users that are in the system.
Identity	Users need to be uniquely identified (done via name and color).
What	
Action	The display can indicate user context and state information in some fashion (i.e., the display could indicate what mode or state the user is currently in, where applicable).

Table 3.14: Awareness features provided by MAUI for chat components.

Chat Component	
WA Element	**Component's Support Features**
Present	
Who	
Presence	The chat component needs to display the users available to chat with (an allow selection of the user or users that he wishes to chat with for each message to be sent).
Identity	Available users need to be uniquely identified (done via name and color).
What	
Action	Messages sent by one user need to be visualized by the recipients chat component so that the user can receive the message. At a minimum the visualization needs to contain the message contents and some identification of the sender.
Past	
When	
Event History	The messages should maintain some sort of timestamp to indicate when a message was sent/received
Who	
Presence	The history should maintain the sender for each message.
What	
Action History	The component needs to maintain some history information so that a user can refer back to previous messages to get context for new messages.

4 Toolkit Design and Implementation

This chapter discusses the second and third steps in the thesis solution introduced in Chapter 1:

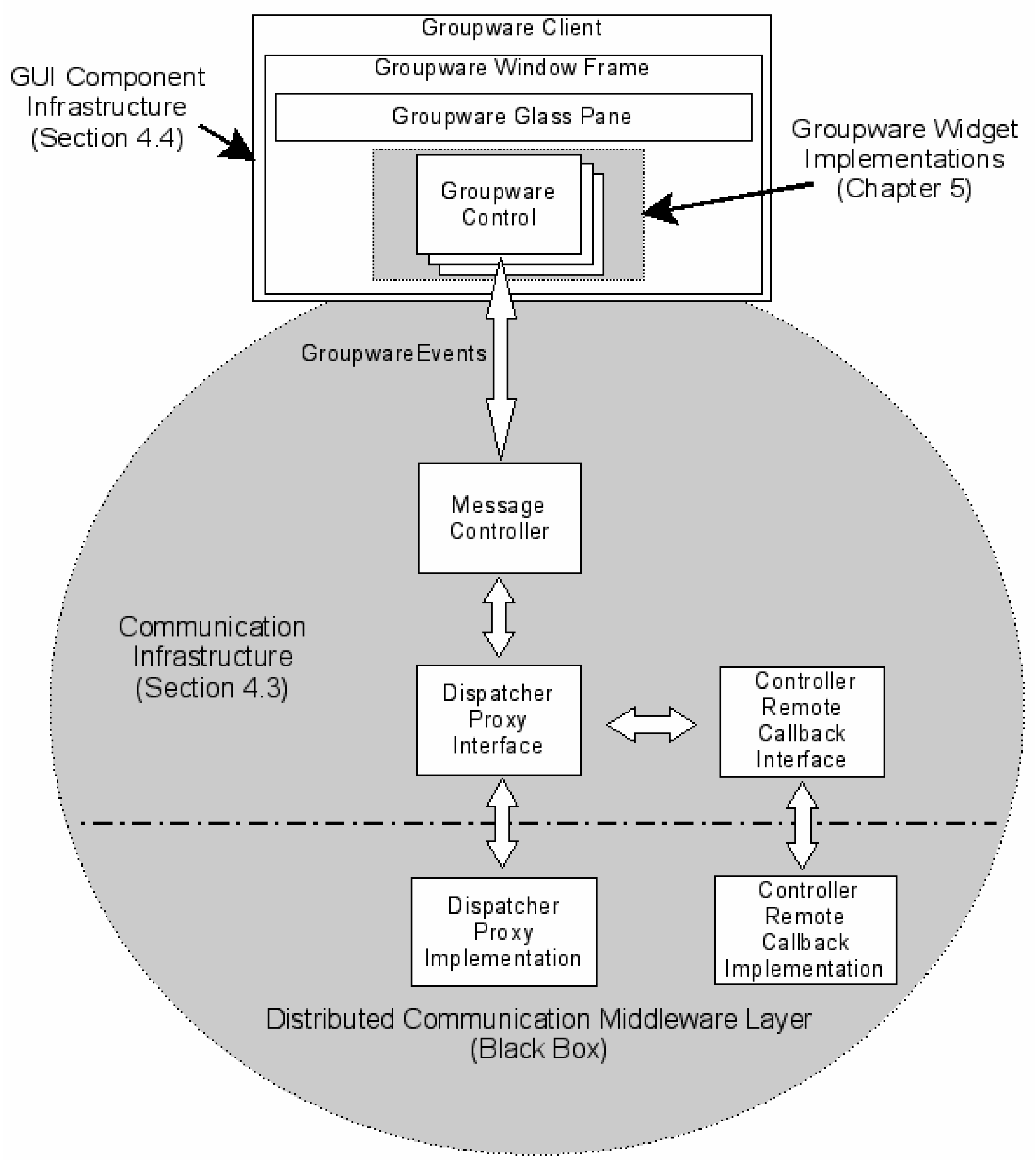
- Development of a Communication Infrastructure.
- Development of the GUI Component Infrastructure.

These steps comprise the design and implementation phases for the core of the toolkit development.

MAUI is a multi-layered architecture, as shown in Figure 4.1, below. At the base of the MAUI architecture is the communication infrastructure, which is built on top of a standard network communication middleware. The communication infrastructure provides the distributed communication layer and has been designed to be a black box, meaning that different implementations of the infrastructure are substitutable without affecting the other layers above and below it. The GUI component infrastructure is built on top of the communication infrastructure and provides the base functionality that is common to groupware GUI components. The GUI component infrastructure uses the communication infrastructure to transfer data from one computer to another. On top of the GUI component infrastructure are the GUI components themselves. This includes both extended single-user widgets and multi-user specific widgets. The Session Management layer runs parallel to the GUI component layer and provides functionality for the setup and tear-down of client sessions and for user management for an application built with the toolkit. Both the GUI component infrastructure and GUI component implementations make use of functionality within the session management layer.

This chapter gives a high-level overview of the highlights and features of the toolkit, followed by a discussion of the design and implementation of the

Communication Infrastructure, GUI Component Infrastructure, and the Session Management Layer.



Groupware Client
Groupware Window Frame
Groupware Glass Pane
GUI Component Infrastructure (Section 4.4)
Groupware Widget Implementations (Chapter 5)
Groupware Control
GroupwareEvents
Message Controller
Communication Infrastructure (Section 4.3)
Dispatcher Proxy Interface
Controller Remote Callback Interface
Dispatcher Proxy Implementation
Controller Remote Callback Implementation
Distributed Communication Middleware Layer (Black Box)

Figure 4.1: A High-Level View of the MAUI Architecture.

4.1 New Multi-User Toolkit Features Provided by MAUI

The MAUI toolkit attempts to fill a void, in terms of groupware features and development support, which existing toolkits have failed to address. Specifically, this void is caused by a lack of generic, reusable and extensible groupware widgets that provide awareness support to groupware application development. Because of the scope of this research, the toolkit will not attempt to provide the ultimate implementation for all the missing features discussed above. Rather, MAUI will provide a reasonable component-based architecture with "hooks" that will easily allow future adaptation and expansion of the toolkit to provide further functionality. The true purpose of the toolkit is to demonstrate that the proposed architecture and implementation facilitates the rapid development of groupware that supports group awareness. The unique or significant features that the proposed toolkit will attempt to provide are listed below (Note that this is an extension of what was discussed in Chapter 1):

- A fully extensible set of JavaBeans-based GUI widgets and supporting classes and interfaces built to provide awareness support to groupware applications, with the central focus being support for process feedthrough.
- A generic, reusable, and customizable set of multi-user versions of single-user widgets, as well as groupware-specific widgets.
- Rapid groupware application development through direct-manipulation and design-time property editing.
- Run-time application customization through application and widget customization dialogs. The run-time customization will focus on tailoring the level of awareness support provided by the client.

4.2 Toolkit Highlights

This section gives a high-level overview of the architecture and design of the toolkit, as well as the significant features that have been built into the toolkit. The toolkit has been developed using the following technologies:

- The Java language. This is because of Java's strong object-oriented nature (which assists in component-based development) and platform independence. Additionally, Java provides the AWT and Swing GUI development libraries upon which the GUI components for MAUI were built. Finally, Java is free and widely available.

- JavaBeans. The JavaBeans component-based methodology is a widely used standard that is compatible with a number of Java development tools. Swing follows the JavaBeans standard.

In terms of distribution architecture, the toolkit provides:

- A server framework that will allow any distribution architecture to be used (i.e centralized, replicated, hybrid, or adaptive) by creating the server components for the architecture such that they conform to the server interfaces provided by the framework.
- An implementation of a server used by the framework when testing applications built with the toolkit. Since the server component is not the main focus of the toolkit, the choice of distribution architecture was based on the simplicity of design and implementation. This lead to the selection of a centralized distribution architecture.

In terms of architectural style, the toolkit:

- Follows the Model-View-Controller (MVC) architectural style. The choice in architectural style is based on precedence, and the fact that the AWT and Swing libraries are built using the MVC architecture.

In terms of distributed communication and message passing, the toolkit uses:

- Java Remote Method Invocation (RMI). Java RMI is high-level and, therefore, easy to start using. It is written using pure Java code and compiled using a compiler supplied with the Java SDK.
- The event/listener mechanism. This strategy is Java convention and is heavily built into the language. Therefore, AWT and Swing also use this mechanism. The paradigm has been extended into the remote communication interfaces.

In terms of the user interface and group awareness, the toolkit:

- Extends the Swing widget set and provides multi-user versions of a selected set of widgets.
- Implements a generic, reusable set of groupware-specific widgets.

- Provides group awareness information processing and display for all controls provided by the toolkit.
- Provides a mechanism to attach both design-time and run-time customizers to the widget set. The customizers allow customization to the way widgets generate and display group awareness information.

4.3 Communication Infrastructure

At the base of a distributed system is a distributed communication infrastructure. This infrastructure is responsible for establishing communications between system nodes within the network. In the context of distributed groupware this includes the computer of each remotely distributed user and any server computers that the system may require. Additionally, the communication infrastructure is responsible for passing messages between each of the network nodes and may be responsible for the encoding and decoding of messages.

4.3.1 General Design Goals

For the scope of the thesis implementation of the MAUI toolkit, there were a number of design goals for the communication infrastructure. These goals are outlined below.

- *Lightweight messaging*. The messages passed between application clients should only contain information necessary to communicate the users' identities and states with respect to the actions being communicated. The information provided should be enough for the remote software to be able to create the necessary visual representation for the GUI control that is to communicate the action to the local user, but should not contain any extraneous information.
- *Generic and flexible design.* The messaging scheme should be flexible and generic enough to be used by a black box server implementation and a number of standard communications protocols.
- *Extensibility.* The design should be extensible enough so that new types of messages can be created and incorporated into the infrastructure for new types of toolkit widgets.

- *Robust inheritance hierarchy*. The design should be such that more specific and sophisticated types of messages can inherit content from the existing message types. The layering should be such that base message classes are available to be extended without adding extraneous information to the new message type (which helps achieve lightweight messaging).
- *Conforming to and leveraging a familiar pattern.* There are a number of existing patterns and paradigms for the development of a message-passing scheme for use with user interfaces. Existing development tools often provide support to leverage the messaging architectures of the more common patterns. The communication infrastructure for the toolkit should take advantage of this support by conforming to one of the common paradigms.
- *Building a black box, swappable architecture.* The choice of distribution architecture of a system is influenced by several factors: complexity of the system, power of the computers, available bandwidth, etc. Some of these factors are system-specific and, therefore, should not be restricted by the choice of toolkit used to implement the system. One of the design goals of this toolkit is to create the design such that the distribution architecture is a black box. From a developer's point of view, any distribution architecture should be able to be used simply by plugging it in (if the implementation exists). Also, use of the communication infrastructure and messages should not require that the developer have any knowledge about distributed programming and protocols. This is all hidden within the black box. Ideally, the architecture should be able to be changed at run-time in order to adjust to a change in the system's environment. The design should be such that changing the distribution architecture, whether at design-time or at run-time should have no apparent changes (other than perhaps performance and slight switchover lag) to the end-user's client application.

4.3.2 Java Implementation Goals

For the thesis implementation of the toolkit architecture, there are some extra goals based on the technology used for the implementation. More specifically, there are goals of the implementation based on development using the Java language. These goals are outlined below.

- *Simplification of development by leveraging existing Java technology*. The Java RMI protocol is 100% pure Java and is supported in most Java development tools. RMI is easy to use and allows rapid development of distributed communication architectures. Because of this, it has been used to wrap the messages and transport them between toolkit components.
- *Use of the event/listener* p*attern*. The adopted method used by Java and Swing to communicate between objects within their library hierarchy is an event/listener mechanism. This mechanism is well known and understood among Java developers and most of the development tools provide easy integration and support for using this mechanism in new Java code. Because of this, the event/listener pattern has been used in the development of the communication infrastructure.

4.3.3 Message Design

MAUI widgets communicate with their remote counterparts via distributed messages. The widgets of one user interface create the messages and send them to the communication infrastructure layer where they are packaged for remote transfer and sent to remote clients. The groupware message classes are central to the MAUI design and play a role in almost all aspects of the MAUI architecture. The groupware messages are passed from groupware controls in one client through the communication infrastructure layer to other clients. From here they are passed through the communication infrastructure of the remote clients to the corresponding control on each of the remote clients that the message was destined for. The flow of groupware messages within the MAUI architecture is illustrated in Figure 4.2.

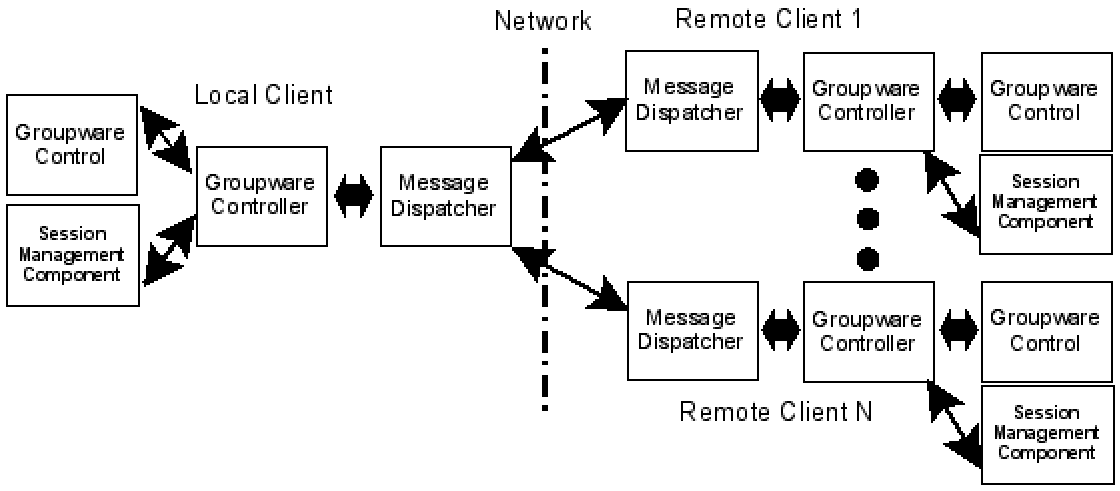


Figure 4.2: The Flow of Groupware Messages within the MAUI Hierarchy.

In general, MAUI messages have been implemented using the event/listener pattern commonly used in Java and Swing development. Because the implementation follows this pattern, messages from this point forward will be called events. The UML [Jacobson et. al. 1999] diagram in Figure 4.3 below shows the general event hierarchy and related classes of the MAUI toolkit.

At the top of the hierarchy is the Java EventObject class, found in the Java utility package. Two classes of MAUI events are extended from EventObject: GControlEvent and GUserEvent. The GControlEvent class is the base class from which all events that communicate widget manipulation information and changes to a widget's state are derived. The GControlEvent class has been extended into a number of subclasses. One key subset of GControlEvents has been created based on the elements of workspace awareness identified in the Awareness Framework. The GUserEvent class is used to communicate session management information, such as that associated with entry and exit of a user to and from a groupware session. Additionally, an extension of GUserEvent, GUserContextEvent, is used to communicate changes to a user's application-specific context and state. As is standard in the event/listener pattern, a listener interface exists for GControlEvents and GUserEvents. The listener interfaces are GControlListener and GUserListener, respectively. The next two sections will discuss the GControlEvent and GuserEvent classes and the related supporting classes used to communicate information within a MAUI application.

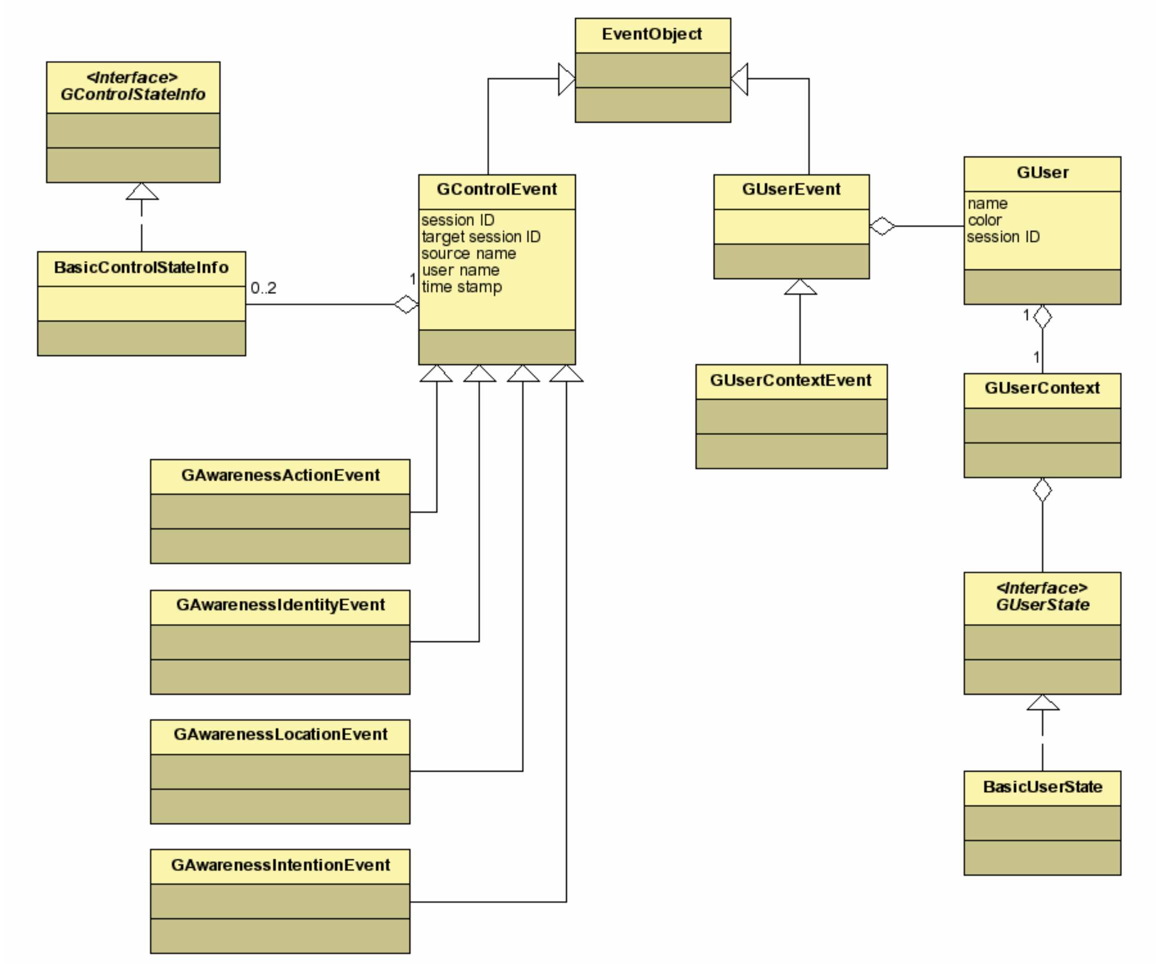


Figure 4.3: The Message Event Hierarchy of the MAUI Toolkit.

4.3.3.1 Control Events

The GControlEvent class is the widget communication base class that contains all the information necessary to transport a message between clients to the correct destination components, provide a context for the message, and provide state transition information for the source component. Two details should be noted about how state transition information is transported between clients. First, each event is independent from each previous or following event and, therefore, needs to contain all information required for correctly understanding the transition between states. Secondly, state information is not relevant to all messages and, hence, this information will not always be present. The source and type of the event determine whether or not state information will be sent. The GControlEvent attributes are summarized in Table 4.1 below.

Table 4.1: GControlEvent Attributes.

Session ID	A unique identifier for the client session that is the originator of the event (which involves using the IP address and a timestamp).
Target Session ID	Unique identifier of the target client, if the event is directed at a particular client. If not, the ID represents a broadcast message to all other clients.
Source ID	Identifies the source component that originated the message from within the source client. Note that this is a String and not an object reference, because the cost of maintaining a remote object reference is much higher than just an ID and almost always unnecessary.
User ID	Unique identifier of the user of the source client. This identifier will be useful in looking up user information used in visualizing the contents of a message. (The system design is such that all clients maintain common user information for all users, such as color).
Timestamp	The time at which the message was sent.
New State	Each event may contain information used in visualizing the state of the remote client's component that originated the message. The state object contained by the event is implementation specific. The GControlStateInfo interface that the objects must implement is an empty interface.
Previous State	Each event may contain information about the previous state that the component was in before the most recent action occurred. The previous state is included for historical purposes and because each event is independent of all other events

The GControlEvent class has been sub-classed into a set of events based on the elements of workspace awareness identified in the Awareness Framework. These events are intended to assist in meeting the requirements outlined in Chapter 3, as these requirements are divided into requirements for each element on a per-widget basis. The following events have been created for use in widget and groupware application development:

a) *GAwarenessActionEvent*. This event is used to convey information about what is happening or what has happened within the workspace.
b) *GAwarenessIdentityEvent*. This event can be used to provide information about the identity of a user performing an action within the workspace.

c) *GAwarenessIntentionEvent*. The awareness intention event is used to convey information about a user's intentions, or the potential actions that she may perform, within the workspace. To adequately convey a user's intentions, five fine-grained intention actions have been identified:

   i. *Entering Use Area*. This typically indicates when a user has moved the cursor or focus into the space occupied by a workspace artifact that can be manipulated in some fashion and, hence, the user is shown that she may intend to manipulate the artifact.

   ii. *Starting Manipulation*. Indicates that the user has begun to manipulate the artifact.

   iii. *Manipulation Performed*. Indicates that the user has manipulated the artifact in a significant way.

   iv. *Ending Manipulation*. Indicates that the user has completed the artifact manipulation but has not yet released focus on the artifact or moved the cursor away from the artifact.

   v. *Leaving Use Area*. Indicates that the user has moved the cursor from the artifact's space or released the focus on the artifact.

d) *GAwarenessLocationEvent.* This event is used to specify a user's work location within the workspace.

In addition to the framework-derived awareness events, several other events that extend GControlEvent or one of the awareness events exist. These events provide more specific information to a particular type of groupware control. For example, GTelepointerEvent extends GAwarenessActionEvent and additionally provides x and y coordinates and a color for use in rendering telepointers to the action identifier found in GAwarenessActionEvent. It should additionally be noted that the toolkit events are extensible and new events can be derived, as new types of controls require them.

4.3.3.2 User Events

MAUI provides a class of events related to session management. These messages help clients maintain a list of users and their associated information locally, for easy access. These events are typically fired when a client enters and exits a group session, and may be fired when information about the user changes. User information may include such information as an application-specific context and state within that context.

The base class for these events is the GUserEvent class, shown in Figure 4.3, which contains only a single piece of information – a reference to the GUser object that represents the user of the client GUI that originated the message. Unlike GControlEvent, GUserEvent is allowed to contain a reference to an object because GUserEvents need not be as lightweight as GControlEvents due to the presumption that these events will occur less frequently than GControlEvents. The GUserListener interface provides methods for receiving GUserEvents when a user is added to a workspace or removed from a workspace. These events are intended to be used when users enter and exit the system. A GUserContextEvent is a subclass of a GUserEvent. These events are fired as users change contexts and states within these contexts. The GUserContextListener interface provides method signatures for receiving events of this type.

GUser is the class representing users in the system. It contains the information in Table 4.2:

Table 4.2: GUser Attributes.

Name	The name given to the user.
Color	A representative color for the user, which is used in visualization of GControlEvent information.
Session ID	Identifies the client that the user belongs to.
Context	The application-specific context in which the user is working (a reference to a GUserContext object).

The GUserContext class allows information about the application-specific context in which the user is working within the workspace to be specified. It contains the information in Table 4.3:

Table 4.3: GUserContext Attributes.

Name	The name used to identify the context.
State	The current state of the user within the context (a reference to a GUserState object). Every context can have one or more states that a user may be in.

The GUserState interface is an empty interface. A BasicUserState implementation is provided by MAUI. It contains the information found in Table 4.4:

Table 4.4: GUserState Attributes.

Name	The name used to identify the state.

MAUI provides a several additional classes related to user management. These include:

- *GLocalUser.* This is a singleton extension of the GUser class intended to contain information regarding the local user of a client GUI. The Singleton design pattern provides a guideline for the creation of classes that have only one instance within an application, which can be globally accessed in a static context [Gamma et al. 1995].
- *GUserList.* This class is used to maintain the list of users and their related information within a client. GUserList extends the Swing AbstractListModel so that it can be used as a model object for Swing List components and components that extend List. Additionally, GUserList is a GUserListener and a GUserContextListener which allows it to be self-maintained (i.e., it will be up-to-date with respect to users entering and exiting the system and changing states and contexts).
- *GUserListSingleton.* This is a singleton GUserList that is meant to be the master user list in the system. It is a singleton to provide convenient access at the global scope.

4.3.4 Controller and Dispatcher Design

This section deals with the pieces of the communication infrastructure responsible for:

- collecting messages from the groupware widgets in the local client,
- sending the messages across the network to the remote clients,
- receiving the messages within the remote clients, and
- forwarding the messages to the destination widgets within the remote clients.

Figure 4.4 displays the controller and dispatcher design in the context of the MAUI hierarchy. Note that the portion of the design within the black box is specific to one particular implementation of the low-level communication infrastructure. Other implementations are possible without modification to the MAUI design.

GController is the client-side component that acts as a liaison between the widgets and the components ultimately responsible for distributed message passing. Typically, one controller is responsible for many widgets (and in most cases there will only be one controller per client). Because the distribution architecture is a black box, the controller does not talk directly do any components associated with the implementation of an actual communication protocol. That task is the responsibility of GDispatcherProxy. GDispatcherProxy implements a standard interface that GControllers know how to talk to. GDispatcherProxy presents a number of standard but generic distributed methods to the controller, and when one of these is called it translates it into a series of protocol-specific calls on the objects responsible for protocol-specific distributed communication. Because the proxy shields the controllers away from protocol-specific details, the classes and components that a developer using the toolkit works with are protocol independent.

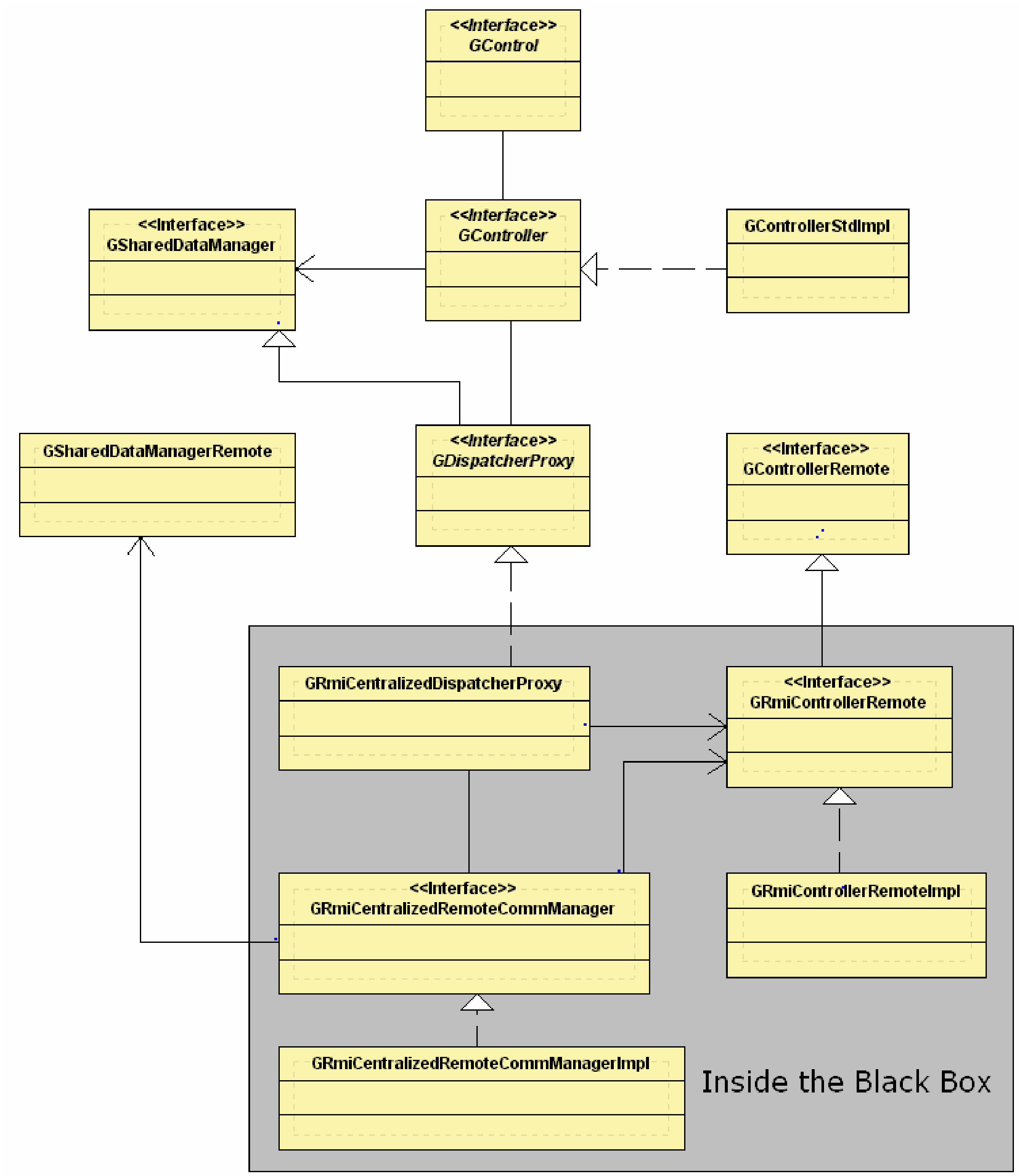


Figure 4.4: Controller and Dispatcher Architecture.

The responsibilities of GController are shown in Table 4.5 below.

Table 4.5: GController Responsibilities.

Remote Communication Setup	▪ Establishing a communication link through the dispatcher proxy. ▪ Creating an event listener to listen for GControlEvents for each control that registers with the controller. ▪ Adding listeners to GDispatcherProxy to listen for incoming GControlEvents from remote clients.
Event Forwarding	▪ Forwarding GControlEvents to the Dispatcher Proxy (in an asynchronous manner) that it receives through the GControlListener interface within the local client. ▪ Forwarding incoming remote GControlEvents to their associated groupware controls as the events are received from GDispatcherProxy. The controller also determines which of the GControls, if any, are the associated controls for incoming GControlEvents (Note that in the current MAUI implementation, the controller forwards events from a GControl to its counterpart within the remote clients).
Protocol-Independent Interfacing to the Dispatcher Proxy	▪ Retrieving the local client's session ID. ▪ Registering and deregistering users with the system. ▪ Setting and retrieving the local user's information, such as name and representation color. ▪ Retrieving a list of all current users of the system. ▪ Querying by user name for the existence of a user in the system. ▪ Changing context and state of the local user. ▪ Registering and deregistering groupware controls. ▪ Adding GUserListeners and GUserContextListeners.
Data-Sharing	▪ Providing a simple interface for sharing model-layer data. The interface is designed to be implemented as a shared key-value pair data registry. MAUI provides a simple centralized implementation of this interface.

The controller is implemented using the Bridge design pattern [Gamma et al. 1995]. This means that GController is actually divided into two pieces – the bridge interface and the bridge implementation. GController is actually the bridge interface, which is the public interface presented to the groupware controls. GController contains a reference to the bridge implementation, which can be swapped to a different implementation object at any point in time (either design time or run time). Because of this, multiple implementations of GController can exist for different communication protocols, if the need arises (although in most cases protocol-specific details are hidden behind the dispatcher proxy, so the standard implementation of GController is typically sufficient). Currently, there exists one standard GController implementation that inherits from Swing's JComponent so that it can be used as a drag-and-drop component within direct manipulation GUI builders.

The GDispatcherProxy interface can be implemented in different protocol-specific ways. The main responsibility of the dispatcher proxy is to hide implementation-specific details about distributed communications away from the controller. Thus, the dispatcher performs the underlying duties of the controller in a protocol-specific manner.

GDispatcherProxy makes use of the GControllerRemote interface, which has the responsibility of acting as a callback object for remote events. The reason for having the GControllerRemote class rather than having the DispatcherProxy act as its own remote callback object, is because an implementation of GDispatcherProxy can potentially become large in size, while a GControllerRemote implementation will tend to be much smaller. For distributed communication protocols that rely on a serialization scheme, sending a large serialized object across the network will likely lead to poor performance. A single implementation of the GDispatcherProxy interface has been created for the thesis version of the MAUI toolkit. This implementation is designed to communicate with an RMI-based centralized server.

4.4 GUI Component Infrastructure

The GUI component infrastructure provides the base on top of which the multi-user widgets created for the toolkit are built, and on top of which new multi-user widgets can be built. Figure 4.5, below, illustrates the role of the GUI Component Infrastructure within the MAUI hierarchy.

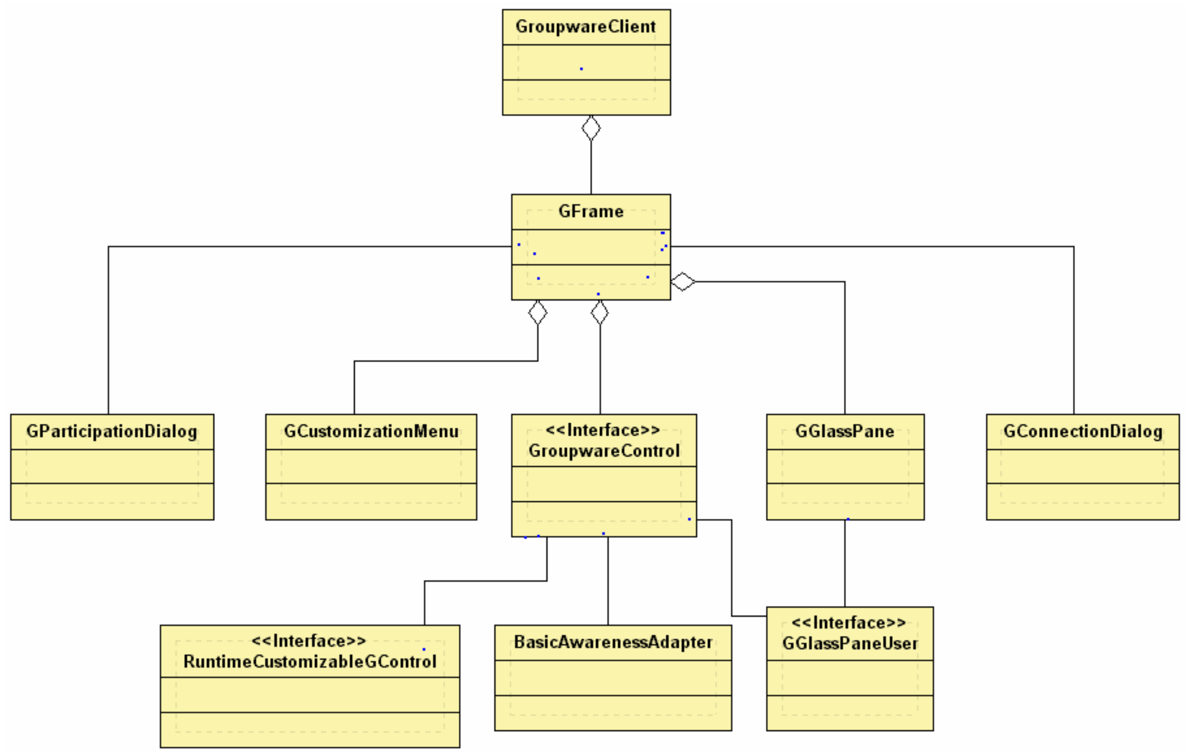


Figure 4.5: High-Level View of the MAUI GUI Component Architecture.

4.4.1 The GroupwareControl Interface

The GroupwareControl interface is implemented by all groupware widgets within the MAUI toolkit. Widgets within MAUI are not required to implement this interface, but by doing so the widgets can automatically integrate with other infrastructure components within the MAUI hierarchy that know how to communicate with GroupwareControl implementations. This includes the GController, discussed in the Communication Infrastructure section, as well as other GUI Infrastructure components yet to be introduced. These include the basic awareness adapter (BasicAwarenessAdapter), the groupware window frame (GFrame), and the design-time and run-time customization components. The interface provides for the groupware widget functionality outlined in Table 4.6.

Widgets that conform to this interface allow the GController to forward to it relevant GControlEvents received from remote clients, as well as allowing the widget to provide the GController with GControlEvents to forward to remote clients.

Table 4.6: GControl Interface Responsibilities.

Property Access	▪ Retrieval of the type identifier for the control. This is a unique per control type identifier, but not per control instance. The control instance identifier is known as the *groupware name*. ▪ Setting and retrieval of a reference to the GController that is responsible for remote communications. ▪ Retrieval of the groupware name, or the unique object identifier for the particular instance of the groupware control. It should be noted that the names for object instances must be the same for equivalent objects in all instances of the client user interface, otherwise the GController will not be able to locate the destination for an event received from a remote client. The toolkit provides this behavior by default. ▪ Setting and retrieval of the name of the last user to manipulate the control. ▪ Setting and retrieval of current and historical state information for the control. This includes getting the GControlStateInfo for both the current state of the control and the previous state of the control. ▪ Checking whether the control supports run-time customization and, if so, a method to get a reference to the customization properties for that control.
GControlEvent Management	▪ Addition and removal of GControlListeners so that widgets can capture the GControlEvents fired by the GroupwareControl. ▪ Handling of GControlEvents received by the GController from remote clients.

4.4.2 The BasicAwarenessAdapter

The BasicAwarenessAdapter has been created to provide support for the basic functions of a multi-user widget within the MAUI toolkit. The adapter handles the implementation of a number of the methods of GroupwareControl, as well as provides functionality for some of the common awareness features of the toolkit components. A groupware control can use the awareness adapter to handle the common operations that all groupware controls within the MAUI toolkit must perform, such as registration and deregistration with a GController, maintaining state information, visualization of certain awareness information, and generic customization functionality. Most of the widgets that have been built for the toolkit leverage the support provided by the adapter. Making use of the BasicAwarenessAdapter in the development of a groupware control will simplify the effort required to create the groupware control. A GroupwareControl makes use of the adapter by having it as a private aggregated member and supplying the adapter with a reference to the GroupwareControl itself. Figure 4.6 shows the relationship between the BasicAwarenessAdapter and a groupware control. Each oval labeled with an 'M' indicates a method call, while each oval labelled 'S' indicates that the call is on a method that has its signature defined by the GroupwareControl interface. The diagram illustrates that a groupware control implementation may implement the GroupwareControl interface by directly implementing the methods (B), passing the calls through to the BasicAwarenessAdapter implementation of the method (A), or adding functionality within a wrapper that calls the BasicAwarenessAdapter implementation (C). Additionally, a groupware control may make use of any other public method of the basic awareness adapter to perform awareness-based operations (D).

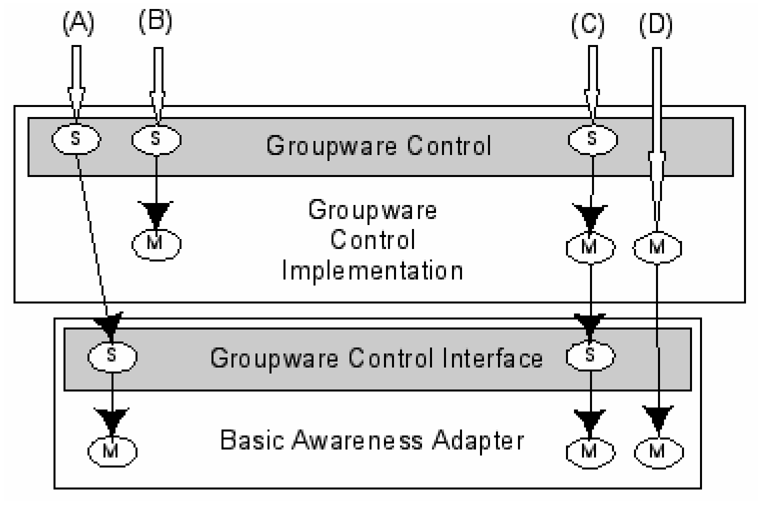


Figure 4.6: The Relationship Between the BasicAwarenessAdapter and a Groupware Control.

The BasicAwarenessAdapter provides the functionality to a GroupwareControl that is outlined in Table 4.7.

The BasicAwarenessAdapter is designed as an adapter and GroupwareControl is designed as an interface because some development languages, including Java, only allow single inheritance. If GroupwareControl was a class, it would mean that a multi-user control could not inherit functionality from another class in a toolkit implementation in one of these languages. The GroupwareControl interface is meant to be a generic, high-level interface and it is likely that specific controls would gain more benefit from inheriting from other classes, such as an existing single-user widget class, rather than inheriting from a generic base class. The BasicAwarenessAdapter allows the basic GroupwareControl functionality to be "plugged in" to a groupware control, therefore eliminating the need for a great deal of cut-and-paste code or error-prone re-implementation of something that's been done before.

Table 4.7: Services Provided by the BasicAwarnessAdapter.

Groupware Control Interface Implementation	▪ A specific implementation of a groupware widget will typically implement the GroupwareControl interface by wrapping the methods of the BasicAwarenessAdapter with simple fall-through methods (see Figure 4.6). The adapter can be used behind the scenes to handle all the common, generic functionality of a MAUI widget. If necessary, the groupware widget making use of the adapter can add widget-specific functionality into the wrapper methods.
GController Registration	▪ Performs registration with the GController on behalf of the groupware control and itself.
Usage History Maintenance	▪ Maintains the identity of the last user to manipulate the widget. ▪ Maintains the current and previous state information of the widget, but only if state information is relevant for the widget.
Awareness Visualization Support	▪ Provides one of a number of visual intention awareness effects to a widget. These are specifically related to entering and exiting the use area of the control. The available intention awareness effects include different border and background highlighting effects. ▪ Applies the desired intention awareness effect to the widget upon receiving an intention awareness event from the GController.
Customization Support	▪ Enables and disables awareness for a widget through the customization interface (discussed later), including deregistration and reregistration with the controller.

4.4.3 GFrame: The Groupware Window Frame

In section 3.2.1, window frames were introduced as one of three types of top-level containers found in graphical user interfaces. As a top-level container, a window will contain, directly or indirectly, all other GUI components within a containment hierarchy. Because of this, the GFrame class has been created as an extension of the basic window frame class to provide groupware functionality on a containment hierarchy, as a whole, as well as providing basic window functionality. At a high level, this extended functionality includes:

- Setting up and establishing distributed communications.
- Setting up facilities for application-level awareness effects, such as telepointers and transparency.
- Providing facilities for capturing identity information for the local user and maintaining remote user information.
- Providing facilities to support design-time customization at the application level and run-time customization, both at the application level and the widget level.

While GFrame could be considered a groupware control, the services provided by GFrame make it more of a GUI infrastructure component. The services that GFrame provides to a groupware application are discussed in, Table 4.8, below.

Table 4.8: Services Provided by GFrame.

Distributed Communication	▪ Automatic creation of a GController for use by the application. ▪ Automatic deregistration with the controller upon closure of the window. ▪ Automatic detection of all groupware controls contained within the frame's content pane (including those within nested containers, such as panels) and registration of them with the controller at window creation time. ▪ Optional creation and display of a server connection dialog at startup to capture server connection information.
Awareness Information Visualization	▪ Optional creation of the groupware glass pane (discussed below) for use by the widget to assist with telepointer and transparency effects. ▪ Binding of the glass pane to all contained controls that are an implementation of GGlassPaneUser. These are generally controls that make use of the glass pane for transparency effects. ▪ Optional enabling and initialization of the telepointer and telepointer traces functionality. ▪ Optional conversion of the main menu bar (JMenu) to a groupware menu bar (GMenuBar) that makes use of awareness visualization techniques.
User Management	▪ Optional creation and display of a dialog at startup to capture information about the local user, such as user name and color selection. The dialog ensures that the user name and color selections are unique among system users. ▪ Initialization of the main user list for the application and binding of the list to the controller. ▪ Optional creation and display of a Participation Dialog

	(GParticipationDialog), which displays and maintains a list of all participants of the group.
Customization	▪ Provides a design-time customization dialog for customizing application-level customizable properties. ▪ Optional creation of a menu item on the main menu bar to access the run-time customization dialog for each type of groupware control included within the frame, as well as an application-level customization dialog. ▪ Provision for the execution of a set of custom initialization steps. The doCustomInitialization method can be overridden by subclasses of GFrame. GFrame will call this method after it has completed other necessary initialization steps that might be required before custom initialization steps can occur. ▪ Management of customization properties for the window and the application (excluding groupware controls, which manage their own customization properties). This involves responding to customization events triggered by users through the design-time and run-time customization facilities.

4.4.4 GGlassPane: Groupware-Enabled Glass Pane

The groupware-enabled glass pane, GGlassPane, is an infrastructure component used to provide certain visualization services to groupware controls and a groupware application, as a whole. More specifically, the services provided by GGlassPane include rendering techniques that are performed on top of the highest-layer controls within a GFrame's containment hierarchy. This includes services such as transparency and telepointers. This section will discuss the details of how GGlassPane provides these rendering techniques, as well as how a typical groupware widget would make use of the GGlassPane.

In the Java world, a glass pane is a component that can be layered on top of a top-level container and used to intercept UI events that would generally go to the window or a component within the window's containment hierarchy. In addition to capturing events, the glass pane can be used to paint on top of a window and all of its contents. The GGlassPane utilizes both of these glass pane capabilities to provide rendering services to groupware widgets, as discussed in the paragraphs below.

Widgets that visualize awareness information through a translucent effect (or any other effect that is desired above the top level of the containment hierarchy) on a remote client do so by providing the GGlassPane with instructions on how to perform that paint operation. The GGlassPane makes use of the rendering information when it refreshes its image through a repaint operation. By default GGlassPane is an invisible component, like a sheet of glass, that is on top of a top-level container, therefore an end-user is unaware of its existence. Thus, the only visualization that GGlassPane performs is that required by other groupware components. As mentioned above, the GFrame will automatically bind the GGlassPane to widgets that are to make use of it. The binding is made possible by having the widget implement the GGlassPaneUser interface. This interface provides a single method signature to allow a GGlassPane reference to be set. As part of its initialization process, the GFrame will automatically locate all GGlassPaneUser components and set the glass pane reference for the components. Each component then can make use of the GGlassPane as is necessary. This typically involves adding a specialized rendering component contained by the widget to the GGlassPane. Each time the GGlassPane repaints itself, it has all components specified as rendering components paint to the Graphics object used by the GGlassPane. Thus, the widget is essentially performing its own painting to the glass pane.

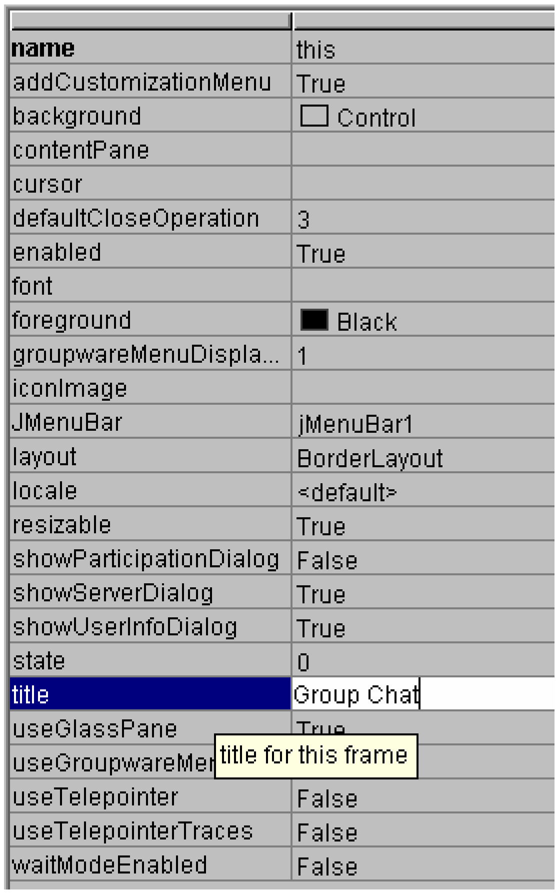
Additionally, the GGlassPane is responsible for providing the telepointer and telepointer traces multi-user functionality. These two multi-user features are not stand-alone widgets that can be added to an application via drag-and-drop. Rather, telepointer functionality is a feature of the GGlassPane. At a high-level, GGlassPane provides telepointer functionality by intercepting mouse motion events. When telepointers are enabled for an application the GGlassPane will use the mouse motion events to determine the current location of the user's cursor and send this information to remote clients. The glass pane of the remote clients will use the awareness event to render a telepointer at the location of the remote user's cursor. It should be noted that while GGlassPane is trapping user-input events, it is still propagating these events to the components in the containment hierarchy that would rightfully get the events if the GGlassPane were disabled. Chapter 5 provides a more detailed description of telepointers and telepointer traces.

4.4.5 The Groupware Client

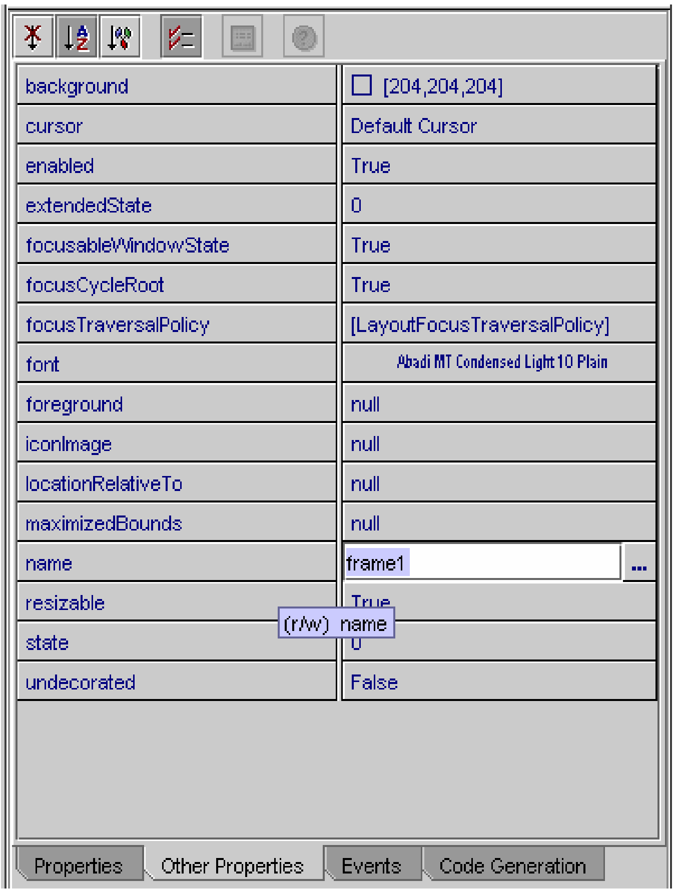
The groupware client is provided as a convenience class for constructing GFrame descendants and starting a groupware application built with the MAUI toolkit. The *main* method of GroupwareClient takes one input argument – the fully qualified Java class name (i.e. including package name) of the GFrame descendant used as the main frame of the application. The GroupwareClient constructs the GFrame descendant via reflection, initializes it, and sets it visible.

4.4.6 Design-Time Customization Support

The MAUI toolkit provides assistance for design-time customization within GUI builders for Java-based development, such as JBuilder, Forte for Java, or Visual Age for Java. Design-time customization involves the ability to change a number of application-level and widget-level properties while coding the application. This can be done by explicitly calling property setter methods or by making use of the GUI builder's built in functionality for direct manipulation code editing. Because the toolkit components comply with the JavaBeans development standard, developers using a Java GUI builder can take advantage of the direct manipulation functionality to modify component properties. Most Java GUI builders include a visual property editor that displays class properties as name-value pairs in a two column table. The first column generally contains the name of the property, while the second column contains the current value of the property. The name column is not editable but the value column is. When a user selects a cell to edit, an editor for the appropriate type of data appears in the cell. The GUI builder usually provides editors for the primitive Java data types and some of the common classes used as attribute types in GUI widgets, such as the AWT Color and Font classes. For new custom data types a developer can create property editors for that data type. If the JavaBeans convention is adhered to, the editor will either automatically integrate into the designer or will be very easy to integrate. Figure 4.7, provides examples of property editor panes from two popular Java IDE's.



name	this
addCustomizationMenu	True
background	☐ Control
contentPane	
cursor	
defaultCloseOperation	3
enabled	True
font	
foreground	■ Black
groupwareMenuDispla...	1
iconImage	
JMenuBar	jMenuBar1
layout	BorderLayout
locale	<default>
resizable	True
showParticipationDialog	False
showServerDialog	True
showUserInfoDialog	True
state	0
title	Group Chat
useGlassPane	True
useGroupwareMe	title for this frame
useTelepointer	False
useTelepointerTraces	False
waitModeEnabled	False



background	☐ [204,204,204]
cursor	Default Cursor
enabled	True
extendedState	0
focusableWindowState	True
focusCycleRoot	True
focusTraversalPolicy	[LayoutFocusTraversalPolicy]
font	Abadi MT Condensed Light 10 Plain
foreground	null
iconImage	null
locationRelativeTo	null
maximizedBounds	null
name	frame1
resizable	True
state	0
undecorated	False

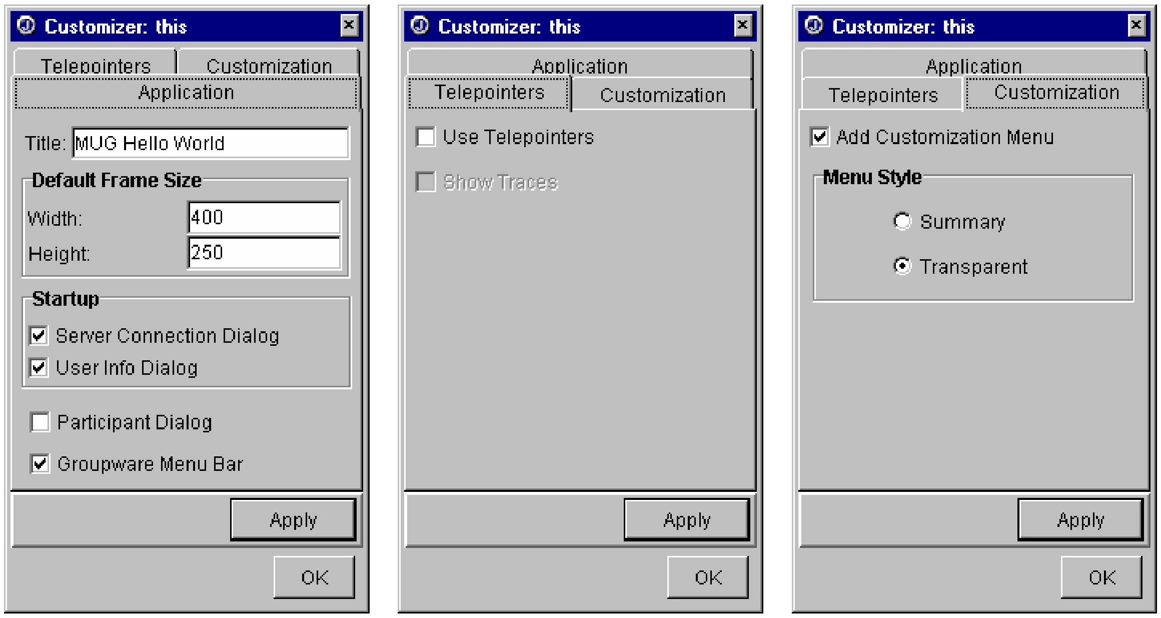
(r/w) name

Property Editor in JBuilder Property Editor in Forte for Java

Figure 4.7: Property Editors

For JavaBeans that require greater control over property editing, or for those JavaBeans that have tightly integrated or highly coupled properties, Java GUI builders will usually support JavaBean customization through the Java *Customizer* interface. Customizers provide a JavaBean-specific user interface from which to customize properties. So, rather than customizing and applying changes to individual properties one at a time, a customizer allows editing of multiple JavaBean properties before a commit is done, or allows greater support for maintaining dependencies between properties and constraining values of properties.

The MAUI toolkit provides a customizer for the GFrame JavaBean. The customizer allows customization to properties of GFrame as well as properties of the groupware application as a whole. Figure 4.8 illustrates the application-level properties of GFrame that are modifiable by the GFrame customizer.



Title: MUG Hello World

Default Frame Size
Width: 400
Height: 250

Startup
[✓] Server Connection Dialog
[✓] User Info Dialog

[ ] Participant Dialog
[✓] Groupware Menu Bar

[ ] Use Telepointers
[ ] Show Traces

[✓] Add Customization Menu

Menu Style
( ) Summary
(●) Transparent

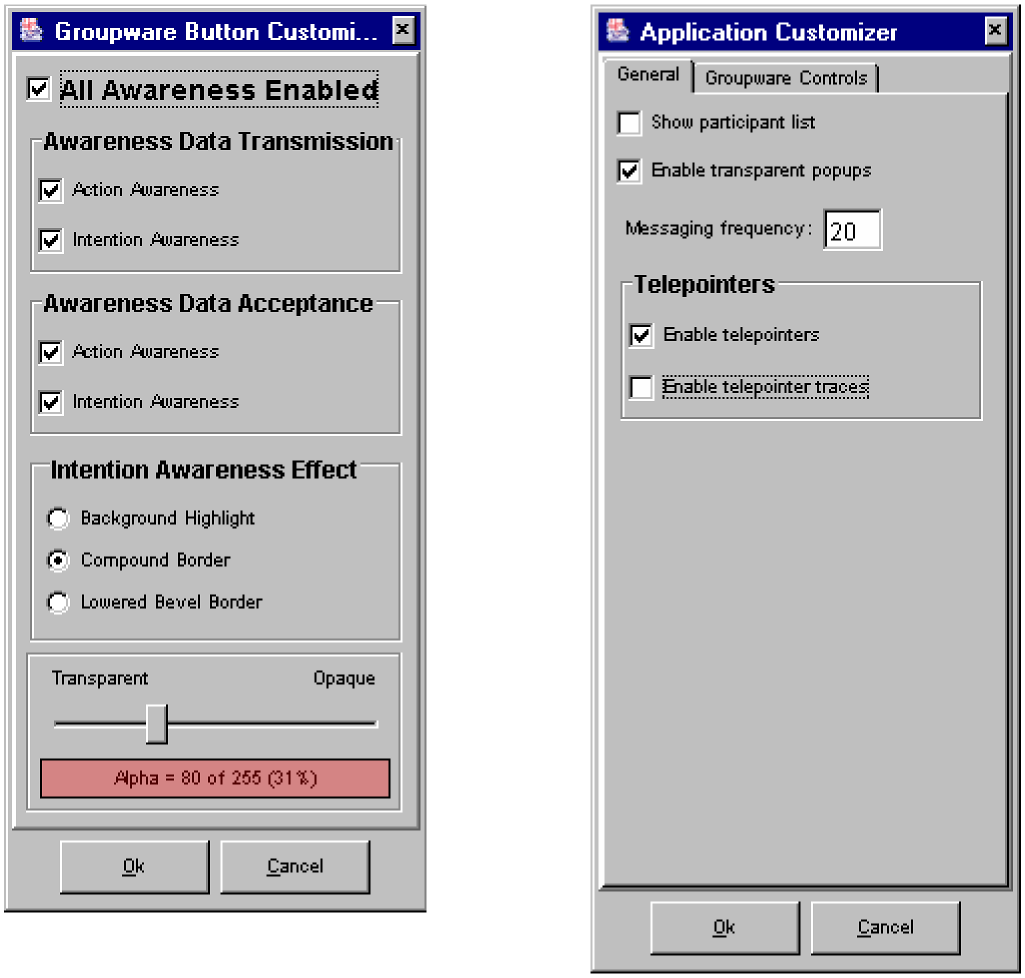
Figure 4.8: GFrame JavaBean Customizer.

4.4.7 Run-Time Customization Support

Run-time customization involves allowing end-users to tailor the look, feel, and behavior of their application through a graphical user interface at application run-time. The MAUI toolkit provides support for run-time customization of the properties of widgets and the application as a whole. Groupware widgets can implement the run-time customization interface, RuntimeCustomizableGControl, and take advantage of the basic facilities provided by MAUI for run-time customization.

MAUI provides the BasicAwarenessProperties class that can be used to manage the customizable properties common to all groupware controls. To use the BasicAwarnessProperties class, a multi-user widget will either instantiate or extend the class as an aggregate member, exposing the methods of BasicAwarenessProperties that implement the RuntimeCustomizableGControl interface. By doing this, the groupware widget can be customized using the BasicGControlRuntimeCustomizer (or an extension of it). The properties of a GroupwareControl that are customizable by this customizer are shown in Figure 4.9 (left). The customizer is accessible through the

CustomizationMenu that is optionally added to the main menu bar of the application window by the GFrame at application startup. In addition to run-time customization at a widget-type level, MAUI allows run-time customization at the application level. The right side of Figure 4.9 shows the application-level properties that are customizable in MAUI at run-time. The properties shown on the left side of Figure 4.9 that were customizable at the widget-type level are also customizable through the application customizer for groupware widgets, as a whole.



[✓] All Awareness Enabled

Awareness Data Transmission
- [✓] Action Awareness
- [✓] Intention Awareness

Awareness Data Acceptance
- [✓] Action Awareness
- [✓] Intention Awareness

Intention Awareness Effect
- ○ Background Highlight
- ● Compound Border
- ○ Lowered Bevel Border

Transparent — Opaque

Alpha = 80 of 255 (31%)

- [ ] Show participant list
- [✓] Enable transparent popups

Messaging frequency: 20

Telepointers
- [✓] Enable telepointers
- [ ] Enable telepointer traces

Figure 4.9: MAUI Basic Runtime Customization Support.

5 The Multi-User Widgets

This chapter introduces the multi-user widgets that have been created for the MAUI toolkit. The widgets have been built using the GUI component infrastructure discussed in Chapter 4, and have been designed to integrate with the communication infrastructure detailed in Chapter 3. More specifically, the majority of the multi-user widgets discussed in this chapter have been integrated with the functionality provided by the BasicAwarenessAdapter (Section 4.4.2).

The widgets are broken into two groups. Section 5.1 discusses multi-user versions of traditional single-user widgets, while Section 5.2 discusses multi-user specific widgets.

5.1 Multi-User Versions of Single-User Widgets

Most of the widgets used in creating single-user GUIs are applicable to multi-user applications as well. However, these single-user widgets in their normal form are not sufficient for multi-user applications because the widgets do not convey any awareness information. The MAUI toolkit provides extensions of many of the common single-user widgets that convey awareness information to remote GUIs of a groupware application and visualize awareness information received from remote GUIs. This section discusses the multi-user versions of the traditional single-user widgets provided by MAUI. The widgets are discussed in the same order that they were introduced in Section 3.2, with the general-purpose containers first followed by the basic controls.

Prior to discussing the individual widgets, the concept of coupled-use versus individual-use mode should be introduced. This concept is related to the shared nature of a widget. A widget in coupled-use mode provides one state model that is shared between all users of the widget and any manipulations performed on that widget by any user update that single model. This is in contrast to a widget in individual-use mode, where the widget has a model for each user of the widget and manipulations of the widget by the user affect only that user's model.

5.1.1 GScrollPane (General-Purpose Container)

A scroll pane, implemented as a JScrollPane in Swing, provides a scrolling view port to a workspace that is larger than the widget's screen area. The scrolling capability allows the viewport to be moved to different areas of the workspace. The groupware scroll pane, GScrollPane, extends a Swing JScrollPane by providing group awareness in one of two different modes – coupled-use mode or individual-use mode. In coupled use mode the scroll pane has the same appearance as the JScrollPane. All users share the same view port, and manipulation of a scroll bar by one user changes the view port on the scroll panes of all other users (i.e., there is one thumb for each scrollbar in the scroll pane, which is shared by all users). In individual-use mode, each user maintains their own view port, and controlling the thumb of their scroll bars only changes the location of the view port in the local view. In this way, all users can have a unique view port location within their view. A GScrollPane operating in individual use mode is illustrated in Figure 5.1, below.

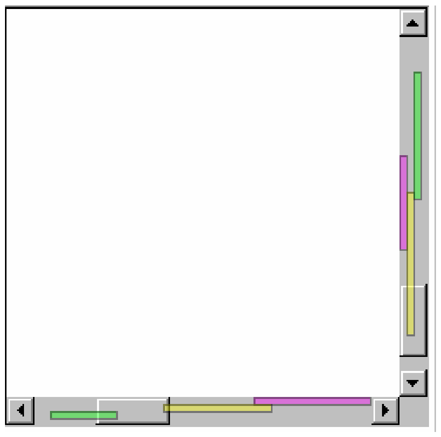
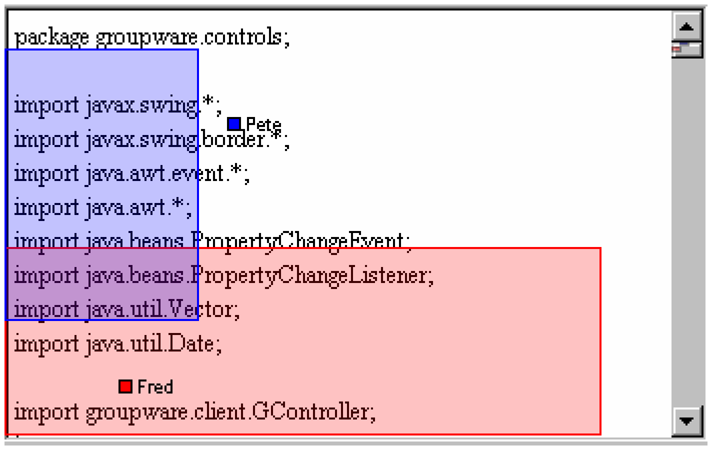


Figure 5.1: GScrollPane in Individual-Use Mode.

Location awareness information is shown for every user in the scroll pane of each user interface. While the scroll bar thumbs of the local user have the same look and feel as a JScrollPane, the thumbs of the remote users appear as colored bars within the scrollbar. The color of the bar matches the user's color represented by the bar. For horizontal scroll bars the width of the bar is representative of the proportion of the width of the view visible through the user's viewport, as it is for the thumb of a JScrollPane (for vertical scrollbars the height of the remote thumbs is determined in this same fashion). The space for the height of the horizontal scrollbar is divided among the remote thumbs. Initially, there is vertical space available for 4 remote user thumbs. If there are more than four remote users the height of the remote user thumbs decreases such that the vertical space is divided evenly among them. Collectively, the location of a remote user's thumb on each scrollbar and the size of the thumb, allow the local user to determine the size and position of the remote user's viewport within the workspace. Scrolling the local user's thumbs to the same place as a remote user's thumb, will place the local user's viewport at the same place in the workspace as the remote user.

The GScrollPane provides an additional representation of users' view ports. This is provided via *view rectangles*. A view rectangle is a translucent rectangle the same size and location as a remote user's view port but located within the local user's workspace. The rectangle is colored in the representative color of the user it represents. View rectangles can be useful in determining exactly what is visible within a remote user's viewport. Figure 5.2, below, shows a GScrollPane with view rectangles enabled.


```
package groupware.controls;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;
import java.awt.*;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.util.Vector;
import java.util.Date;

import groupware.client.GController;
```

Figure 5.2: A GScrollPane with View Rectangles Enabled.

In terms of message passing, a GScrollbar, which is the groupware scrollbar implementation, communicates with other GScrollbars via GScrollEvents. The GScrollEvent is a special extension of GControlEvent designed specifically for

GScrollBars. The event is comprised of three pieces of information in addition to the standard GControlEvent information: the orientation of the scrollbar (horizontal or vertical), the current position of the scrollbar, and the percentage of the view that is visible through the viewport. Assuming that all scrollbars have a fixed number of discrete positions, the current position of the scrollbar is the ratio of the current position over the maximum position. The relative visibility is used to determine the size of the scrollbar thumb and is the amount of the total view that is visible.

Discussions of previous implementations of mutli-user scrollbars can be found in [Hill & Hollan 1992] and [Brewster et al. 1994].

5.1.2 The Button Widgets (Basic Controls)

A button is a GUI widget that is used to trigger an action when it is pressed. MAUI extends five of the most common types of Swing buttons. Each groupware button widget extends a comparable single-user Swing button widget; therefore, the groupware buttons do not have a common groupware parent class even though their implementations are quite similar. However, the use of the BasicAwarenessAdapter has helped to reduce duplication within the implementations and provide some common implementation support. Note that at a low level the Swing button classes all inherit from a common base class – the AbstractButton class.

All groupware buttons have only a coupled-state mode. When any user manipulates the button and changes its state, the state is changed to that state for all copies of that button. Additionally, when a user manipulates a groupware button, the tooltip is set to indicate the time of the manipulation and the name of the user that performed it. The button only maintains information for the most recent manipulation.

All groupware button implementations display intention awareness information via one of the BasicAwarenessAdapter's standard intention awareness visualization techniques. A groupware button communicates intention awareness information as follows:

- When the mouse cursor of a user enters the space occupied by a groupware button, the comparable button on each remote GUI is highlighted to indicate the presence of the user and the potential intention of the user to manipulate the button. The highlighting effect uses the user's color so that the remote users can identify the user that may potentially manipulate the button.

- When a user manipulates the button, the feedthrough of the manipulation is shown on the remote GUIs. This feedthrough is the typical feedthrough exhibited by manipulating the button on the local GUI.
- When the mouse cursor of the user exits the space occupied by the button, the highlighting effect is removed from the remote GUIs.

As mentioned above, there are five groupware button types: GButton, GRadioButton, GCheckBox, GToggleButton, and GBasicArrowButton. These five button types are summarized in Tables 5.1 to 5.5 below.

Table 5.1: Summary of GButton.

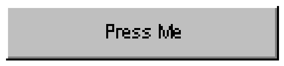
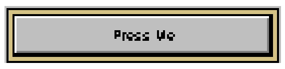
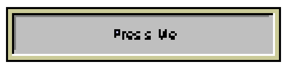
Type: GButton	▪ These are the basic buttons used to invoke an operation within a GUI	
States: *No Persistent States*	▪ Have a depressed state as a non-persistent state because the button is only depressed for a brief moment in time while the user is clicking it.	
Intention Awareness Effect: *Compound Border*	▪ The color of the outer border is the color for the user exhibiting the intention awareness, while the inner border is the original button border.	
Action Awareness Effect (Feedthrough): *Normal Action Animation*	▪ The feedthrough effect of a GButton is the animation of the pressing of the button.	
		
A GButton	A GButton Displaying Intention Awareness.	A GButton Providing Action Awareness

Table 5.2: Summary of GRadioButton.

Type: GRadioButton	▪ Used to select one option from a set of options and are usually gathered as a group of buttons. . Generally, only one radio button can be selected at any point in time.
States: *Selected, Unselected*	▪ As part of the action awareness event that provides the feedthrough effect, the new and previous states of the radio button are provided to the remote GUIs and the states of the remote radio buttons are updated to be continually in sync with the locally manipulated radio

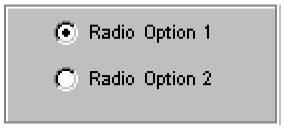
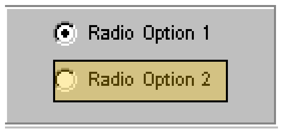
	button.
Intention Awareness Effect: *Transparent Background Highlighting*	▪ The color of the background is the color for the user exhibiting the intention awareness.
Action Awareness Effect (Feedthrough): *State Toggling*	▪ The feedthrough action awareness effect is performed by toggling the state of the radio button, which is the same effect that is provided to a single user of a Swing JRadioButton.
 Radio Option 1 Radio Option 2	 Radio Option 1 Radio Option 2
Group of GRadioButtons	A GRadioButton Providing Intention Awareness

Table 5.3: Summary of GCheckBox.

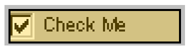
Type: GCheckBox	▪ Used to select or deselect a particular option within a GUI. Unlike radio buttons, check boxes are usually independent of each other.
States: *Selected, Unselected*	▪ As part of the action awareness event that provides the feedthrough effect, the new and previous states of the check box are provided to the remote GUIs and the states of the remote check boxes are updated to be continually in sync with the locally manipulated check box.
Intention Awareness Effect: *Transparent Background Highlighting*	▪ The color of the background is the color for the user exhibiting the intention awareness.
Action Awareness Effect (Feedthrough): *State Toggling*	▪ The feedthrough action awareness effect is provided by toggling the state of the check box on the remote GUIs, which is the same effect that is provided by the JCheckBox.
Check Me	 Check Me
A GCheckBox	A GCheckBox Providing Intention Awareness

Table 5.4: Summary of GToggleButton.

Type: GToggleButton	▪ Provide a similar function to that of a check box but provide a different visualization of the state information.
States: *Selected, Unselected*	▪ Same as for GCheckBox.
Intention Awareness	▪ Same as for GButton.

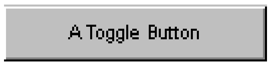
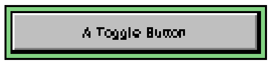
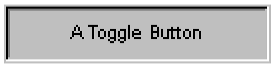
Effect: *Compound Border*		
Action Awareness Effect (Feedthrough): *State Toggling*	▪ Toggling the state of the button provides the feedthrough action awareness effect.	
		
A GToggleButton toggled to the unselected state	A GToggleButton providing intention awareness	A GToggleButton toggled to the selected state

Table 5.5: Summary of GBasicArrowButton.

Type: GBasicArrowButton	▪ Is a special button used by other widget implementations to perform specific actions. For example, a combo box makes use of an arrow button to drop down the list box portion of the control. Scroll bars use arrow buttons to move the view port left and right, in the case of a horizontal scroll bar, or up and down, in the case of a vertical scroll bar. An arrow button is basically a GButton with a scalable arrow icon on it to indicate a direction in which the context-specific action to be performed is to be applied.
States: *No Persitent States*	▪ Same as for GButton.
Intention Awareness Effect: *Transparent Background Highlighting*	▪ The color of the background is the color for the user exhibiting the intention awareness.
Action Awareness Effect (Feedthrough): *Normal Action Animation*	▪ Same as for GButton.
A GBasicArrowButton	A GBasicArrowButton Providing Intention Awareness

5.1.3 GComboBox (Basic Control)

In general, a combo box is a combination of a text box and a list control. It allows a single selection to be made from a list of options. By default, the list is hidden and only the text box portion of the combo box is displayed. The text box will show the selected option, if an option is selected. Beside the text box is an arrow button that is used to toggle the display of the list portion of the control. When the list is displayed the user can select one of the options in the potentially scrollable list of options. While the user is running through the list of options the list box will highlight the option that has the cursor or keyboard focus. Once the user has made a selection of an option, either by a mouse click or pressing the selection key on the keyboard (typically the enter key), the list is hidden and the text box displays the new current selection. In some cases, combo boxes also allow text to be typed into the combo box either as an alternative to the selection or as a shortcut to selecting an option from the list.

GComboBox widgets maintain state – the active selection in the combo box. When a user selects an option in the combo box, the current and previous states of the combo box are passed to the remote clients in the action awareness event that performs the feedthrough. The state information in the corresponding widgets in the remote clients updates their state information accordingly. The state of a combo box is the value of the selected option.

GComboBox widgets communicate intention awareness information. The process is as follows:

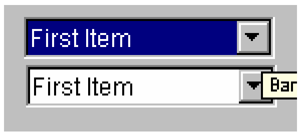
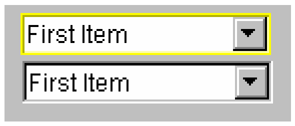
- When the mouse cursor of a user enters the space occupied by a combo box, the comparable combo box on each remote user's GUI is highlighted to indicate the presence of the user and the potential intention of the user to manipulate the control. The default highlighting effect is to change the color of the border of the combo box to that of the user for which the intention awareness is being exhibited.
- When a user manipulates the arrow button the feedthrough of the manipulation is shown on the remote GUIs. This feedthrough is the typical feedthrough exhibited by manipulating the button on the local GUI.
- The manipulation of the arrow button toggles the display of the list portion of the combo box from hidden to visible. On the remote GUIs, the combo box option list is displayed in one of two different ways depending on the configuration of the GComboBox. A GComboBox can be configured to display its popup list either as a translucent replica of the list that is shown on the local GUI or as an opaque summary list that displays only the selected

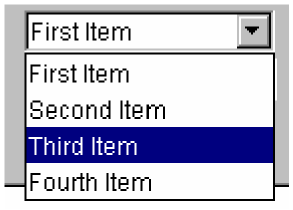
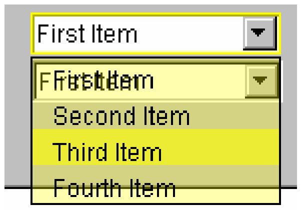
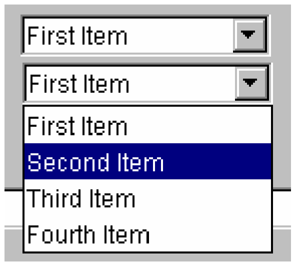
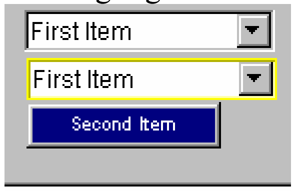
option. The translucent popup allows the contents of the window that the popup is displayed on top of to partially show through the list. This is so that the potentially unexpected feedthrough of a combo box displaying its popup list does not completely occlude the area of the workspace that the remote user may be working in when the feedthrough is exhibited. The summary popup list provides an alternative view by displaying a compact list that occupies only enough of the workspace to show the selected option. In this respect, the space occupied by the size of the popup is guaranteed and, hence, the designer can design the workspace such that displaying of the popup will not be likely to get in the way of other users that may be working.

- As the user moves through the selections in the combo box, the highlighted selection changes the selection on the remote GUIs in real time. For the transparent popup list, the highlighted option is reflected using a similar but translucent highlighting effect as in the local GUI. For the summary popup option, the single option displayed in the summarized list is updated with the highlighted option.
- When the user selects an option and causes the list portion of the combo box to disappear the effect is reflected in real time in the remote GUIs.
- When the mouse cursor of the user exits the space occupied by the combo box the highlighting effect is removed from the remote GUIs.

Table 5.6, below, illustrates the awareness effects of a GComboBox. The figure shows both the local and a remote view of two GComboBoxes stacked one on top of the other. The top combo box is configured to use a transparent popup list while the bottom combo box displays a summary popup list.

Table 5.6: Awareness Effects of GComboBox.

Local GUI	**Remote GUI**
 The user has the focus in the top combo	 Intention Awareness displayed for the top

box	combo box
 The user has placed the focus on the third item of the top combo box	 The transparent popup list of the top combo box is showing the third option in the list as highlighted.
 The user has given focus to the second item of the bottom combo box	 The summary popup list of the bottom combo box is displaying the second item as highlighted.

5.1.4 GList (Basic Control)

A list widget displays a group of options that a user may choose from. Depending on the selection model, a user may be allowed to choose only a single option or she may be allowed to choose multiple options. A list with many options is often placed within a scroll pane, such that the view port of the scroll pane only shows a subset of the list options at any point in time but allows the user to scroll through the entire list.

GList widgets maintain a state – the value of the selection. Unlike the widgets discussed above, GList objects provide two different models for displaying state information – individual-use mode and coupled-use mode, introduced in Section 5.1.

GLists visualize intention awareness in the following manner:

- When the mouse cursor of a user enters the space occupied by a Glist, the comparable GList on each remote GUI is highlighted to indicate the presence of the user and the potential intention of the user to manipulate the list. The default highlighting effect is to change the color of the border of the list to that of the user for which the intention awareness is being exhibited.

- When a user changes the selection within the list box, the selections are updated on the remote GUIs in one of two ways depending on the use mode set on the list. If the mode is coupled-use, the selected items are highlighted to mimic the selected state of the local GUI. The color of the highlighting of the remote GUIs is that of the user to make the selections. If the mode is individual-use mode, the selection updates the visualization of the selections for that user only. The selections for each user are highlighted in the color of the user to make the selection, except for the local user's selections. Those selections are highlighted, as they would be if the widget were a single-user widget. If the selections of multiple users overlap, the coloring of the selection is evenly split between the colors of the users with the selection (see diagram, below).
- When the mouse cursor of the user exits the space occupied by the list, the highlighting effect is removed from the remote GUIs.

Figure 5.3, below, illustrates a GList in individual-use mode, contained within a GScrollPane, and in multi-item selection mode. The figure shows four users of the GList, the local user and three remote users. The local user has the second item selected, which is highlighted in the same way as it would be for a single-user JList. The three remote users all have multiple items selected, with the selections highlighted transparently in the colors of the users. The green user has items three through seven selected, while the red user has items four through seven selected, and the yellow user has items six through eight selected.

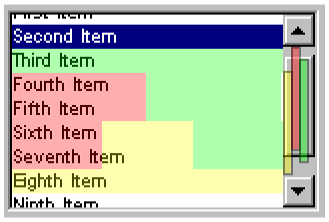


Figure 5.3: A GList in Individual-Use Mode.

5.1.5 GSlider (Basic Control)

A slider provides a way of selecting a single value from a discrete range of values. The user slides the slider thumb to the desired selection along the row (or column, if the slider is vertically oriented) of values. The values are generally numeric in nature and may have a textual label associated with each value that can be displayed on the slider.

The GSlider widget exhibits state, which is the value of the selection. In a fashion similar to the other widgets, the selection is updated in the action awareness event associated with the selection of a value with the slider thumb. Like the GList, GSlider can have one of two state models – coupled-use or individual-use mode.

GSliders visualize intention awareness in the following manner:

- When the mouse cursor of a user enters the space occupied by a GSlider, the comparable slider on each remote GUI is highlighted to indicate the presence of the user and the potential intention of the user to manipulate the slider. The default highlighting effect is to change the color of the background highlighting of the slider to the color of the user for which the intention awareness is being exhibited. The background is displayed in a translucent fashion.
- When a user changes the selection of the slider, the thumb positions are updated on the remote GUIs in one of two ways depending on the mode set on the slider. If the mode is coupled-use mode, the thumbs are highlighted to mimic the selected state of the local GUI. If the mode is individual-use mode, the slider updates the visualization of the thumbs for that user only. The thumbs for each user are drawn in a translucent version of the color of the user to make the selection, except for the local user's thumb. Those thumbs are drawn as they would be if the widget were a single-user widget. If the thumbs of multiple users are at the same selection, the coloring of the thumb bar is evenly split between the colors of the users with that selection (see diagram, below).
- When the mouse cursor of the user exits the space occupied by the slider, the highlighting effect is removed from the remote GUIs.

Figure 5.4, below, illustrates a GSlider in individual-use mode. The figure shows four users of the GSlider, the local user and three remote users. The local user has 40 selected, which is shown by the slider thumb. The three remote user's selections are highlighted transparently in the colors of the users. The green user and red users both have 60 selected, while the yellow user has 55 selected. Additionally, the yellow user is showing intention, which is indicated by the highlighting of the background color of the slider.

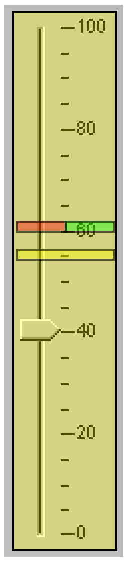


Figure 5.4: A GSlider in Individual-Use Mode.

5.1.6 GTextField (Basic Control)

Text fields are used to enter a single line of text. Typically, one or more characters can be highlighted and replaced at a single time and the cursor position can be repositioned at any place within the entered text string. The GTextField widget maintains state information, which is the text string contained by the widget. Whenever the text field is updated, the string is transferred in the action awareness event as GControlStateInfo and updated in the comparable text fields of the remote GUIs. GTextFields support only the coupled-use mode state model.

GTextFields visualize intention awareness in the following manner:

- When the mouse cursor of a user enters the space occupied by a GTextField, the comparable text field on each remote GUI is highlighted to indicate the presence of the user and the potential intention of the user to manipulate the text field. The default highlighting effect is to change the color of the border of the text field to that of the user for which the intention awareness is being exhibited.

- When the user gives keyboard focus to the text field, the location of the cursor is updated on the remote text field widgets. The cursor position is shown in a translucent version of the color of the user for which the intention awareness is being exhibited. Additionally, as the local user changes the cursor position, the remote GUIs are updated in real time to reflect the change. While the data contained by a text field is shared among all users, the visualization of intention awareness is done on an individual-by-individual basis. A GTextField shows the cursor position of each and every user that is using the text field.
- When a user highlights a portion of the text within the text field, the highlight is shown on the remote user's GUIs in the color of the user for which the intention awareness is being exhibited. Similar to the individual visualization of cursor position, the text highlighting effect is also done on an individual by individual basis.
- When the mouse cursor of the user exits the space occupied by the text field, the highlighting effect is removed from the remote GUIs.

Figure 5.5 illustrates a GTextField with four users – the local user and three remote users. The cursor position of the local user is indicated after the 'G' in the normal single-user widget fashion. The cursor position of the green user is indicated in a transparent fashion after the 's' character in 'is', while the red and blue users have text selected, which is indicated in a translucent fashion using the user's colors.

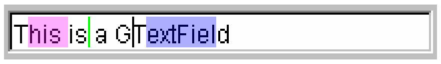


Figure 5.5: A GTextField.

5.1.7 GMenu and GMenuItem (Basic Control)

A menu is most commonly displayed in a menu bar across the top of a window, just below the title bar. The menu presents a selection of drop down lists that provide options for a user to select from. The options (referred to as menu items) may invoke some action (the same way a button would) or it may, itself, present a drop down list of sub options to users. Menus can also be used as popup lists within a window, commonly invoked by clicking the right mouse button.

Menus usually do not maintain any state information between uses, although controls such as combo boxes and check boxes can be used as menu items within a menu. The most common menu items, however, are the push button flavor that invoke actions rather than exhibit state. Currently, the push button flavor of menu item is the only menu item that has a multi-user counterpart in the MAUI toolkit.

GMenus and GMenuItems visualize intention awareness in the following manner:

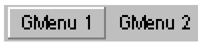
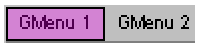
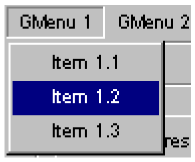
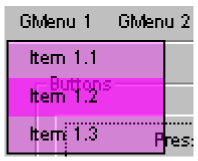
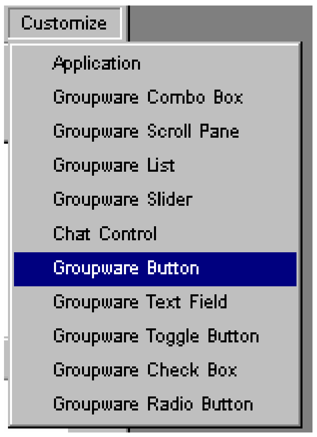
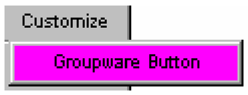
- When the mouse cursor of a user enters the space occupied by a GMenu. the comparable menu on each remote GUI is highlighted to indicate the presence of the user and the potential intention of the user to manipulate the menu. The default highlighting effect is to change the background color of the menu to that of the user for which the intention awareness is being exhibited.
- When the user manipulates the GMenu, the display of the drop down list is toggled to visible. Like a GComboBox, the visualization of the drop down list may be done in one of two ways, depending on the visualization option set on the GMenu. The two drop-down list visualization techniques are: a translucent replica of the local user's drop down list, and a summary drop down list. These two options are visualized in the exact same manner as the GComboBox.
- As a user moves the focus through the list of options in the drop down menu, the focus highlighting is propagated to the remote clients in real time. In the transparent drop down list, the menu items with focus are highlighted using a translucent coloring technique for the color of the local user. For the

summary list version, the list displays the highlighted item as the only item in the list.

- When the mouse cursor of the user exits the space occupied by the text field, the highlighting effect is removed from the remote GUIs.

Table 5.7 illustrates the awareness effects of a GMenu and GMenuItems. The figure shows both the local and a remote view of a GMenu exhibiting transparent awareness effects and a GMenu exhibiting summary awareness effects.

Table 5.7: Awarness Effects of GMenu and GMenuItem.

Local GUI	**Remote GUI**
 The user has the focus on GMenu1	 Intention Awareness displayed for GMenu1
 The user has placed the focus on the second item of GMenu1	 The transparent popup list of GMenu1 is showing the second option in the list as highlighted.
 Customize Application Groupware Combo Box Groupware Scroll Pane Groupware List Groupware Slider Chat Control Groupware Button Groupware Text Field Groupware Toggle Button Groupware Check Box Groupware Radio Button The user has given focus to the "Groupware Button" option of the "Customize" menu.	 The summary popup list of the "Customize" menu is displaying the "Groupware Button" option as highlighted.

5.2 Multi-User-Specific Widgets

This section discusses the multi-user-specific widgets developed for the MAUI toolkit. These widgets are considered multi-user-specific because they do not have relevance in a single-user environment and, therefore, do not have a counter-part in single-user widget sets.

5.2.1 Telepointers and Traces

A telepointer is a representation of a user's cursor on a remote user's GUI. A telepointer is used to convey location awareness information about a user. A cursor is used to mark the component on the GUI at which user input events will be directed. By conveying cursor location information to remote users as awareness information, they should have a reasonable idea of where a user is working and what it is they are doing. Telepointer information is usually transmitted in real time or near real time. By monitoring the progression of a remote user's telepointer, a user can often determine what workspace artifacts a user is manipulating at and what time, so that she may base her actions on the location awareness information of the other users.

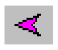
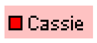
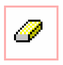
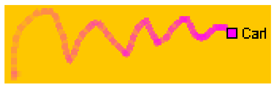
Telepointer traces (or trails) are "tails" on the ends of telepointers used to indicate the progression of a telepointer within the workspace over time [Gutwin 2001, 2002]. Traces provide historical location awareness information, as they show where a user has recently been. In the MAUI toolkit, traces are shown in the color of the user the telepointer is representing. Transparency is used to indicate the age of the trace point. For instance, the most recent telepointer trace point is opaque, while the least recent trace point is nearly transparent. The points in between the most and least recent points scale in transparency, such that the more recent trace points are more opaque than the less recent trace points. As time progresses and the trace points age, their transparency increases such that once a trace point reaches a certain age it is removed from the end of the trail altogether. Additionally, the number of points in a trace is limited to a configurable maximum. When a new point is added to the trace that contains the maximum number of trace points, the least recent trace point is removed, regardless of its age.

In MAUI, telepointers are implemented as a function of GGlassPane. Because a glass pane covers an entire window, drawing telepointers on top of the glass pane allows the implementation of telepointers to be isolated to one component. Otherwise, it would have to be the responsibility of each widget in the MAUI toolkit to implement telepointers for when the cursor is in their space. As mentioned previously, a glass pane intercepts user input for the workspace area over which it covers. The GGlassPane analyzes the events it intercepts and, if they are events related to changing the cursor position the GGlassPane, transmits a TelepointerEvent to the remote GUIs before forwarding the event to the underlying GUI component. Telepointer traces are simply the visualization of the historical TelepointerEvents received from remote GUIs for each user.

MAUI's telepointers have the ability to convey user context information. A user context describes the context in which a user is working within the application. User contexts additionally have states associated with them, to provide even more refined information about where and how a user is working within the workspace. User contexts and states are application-specific. There are no default contexts and states provided by MAUI. An example of user context and state might be as follows: a user within a groupware-brainstorming tool might be working in the diagramming context. Within that context, the user might be using an eraser tool, which would put her in the deletion state within the diagramming context. Telepointers can convey user context and state information through the visual representation of the telepointer icon. Developers can create a user state and context model within a MAUI application and assign icons to each user state. The telepointers can be configured to change their representation based on the state model such that when an event is received by a GUI indicating that a user has changed state, the telepointer icon for that user will change to the icon assigned to that state within the state model.

Table 5.8 shows three different telepointer visualizations in the three left-most columns – the default visualization, the label telepointer visualization, and an image telepointer visualization. The telepointer visualization can be set by specifying the telepointer renderer factory that glass pane should use to create its telepointer renderer. The LabelTelepointerRenderer and ImageTelepointerRenderer are currently the telepointer renderers included in MAUI. The LabelTelepointerRenderer draws the telepointer as the user's name with a small box colored in the user's color to the left of the name, while the ImageTelepointerRenderer draws the telepointer as an image inside a box colored the user's color. The image can be mapped to the user's current state, such that the telepointer reflects what state the user is in. By setting the glass pane to not use a telepointer renderer factory to create its renderer, it will use the default telepointer renderer. Additionally, programmers can create their own implementations of the TelepointerRenderer and TelepointerRendererFactory interfaces and set the glass pane to use them at application initialization. Table 5-8 also shows a telepointer with traces enabled, in the column on the far right of the table.

Table 5.8: MAUI Telepointers and Telepointer Traces.

			
Default Telepointer	Label Telepointer	Image Telepointer Exhibiting State Information	Telepointer with Traces Enabled

5.2.2 Participant Display

A participant display is a list of the members that are actively part of the groupware session. MAUI provides a default participant display component. The MAUI GParticipantPanel shows the name of each user and the representative color for that user within the workspace. The name of the local user is shown in Italics to distinguish it from the remote users. Additionally, the MAUI participant display can show user state information in the form of an icon to the right of the user name for which the state is associated with. A participant display is configured to show state information from a user context model in the same fashion that a telepointer is (see the previous section). In MAUI, the GParticipantDialog is a dialog that contains a GParticipantPanel. It can be activated via a property setting on the GFrame, as discussed previously.

Figure 5.6 shows the default MAUI participant display.

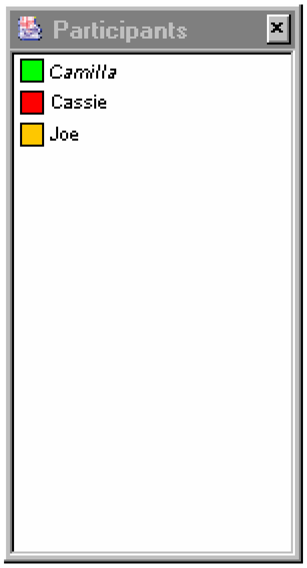
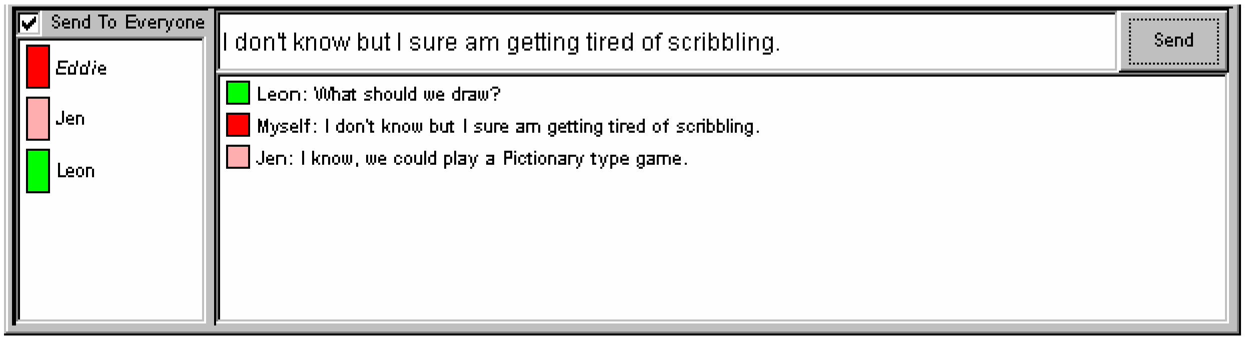


Figure 5.6: The MAUI Participant Display.

5.2.3 Chat Tool

The GChatTool is a widget that allows text messages to be sent between participants in real-time so that they can engage in conversation. The chat tool is a simple componentization of what is often considered a complete groupware application. The chat tool allows messages to be directed to specific participants or broadcast to all participants. Message recipients are selected from a GParticipantPanel that is attached to the chat tool. For broadcasting, a check box can be selected to enable broadcast mode so that all participants are sent outgoing messages regardless of the selections made in the participant panel. To send a message, a user types a message into a text box and presses the send button, which is a GButton that conveys intention and action awareness information when manipulated. Incoming messages are displayed in a scrolling text area in the bottom right-hand corner of the chat tool. Each new message is added to a new line in the text area and is preceded by color and name of the user that sent the message. The order messages are received is maintained in the chat tool so that a user can scroll back through the messages and review the progression of a conversation. Figure 5.7 shows the MAUI chat tool.



I don't know but I sure am getting tired of scribbling.

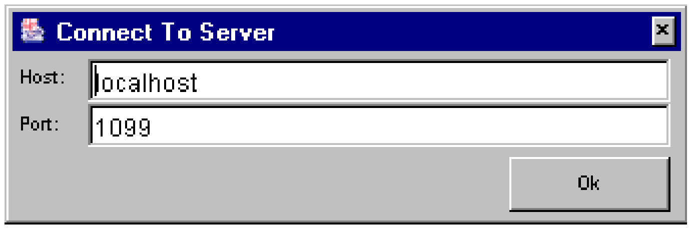
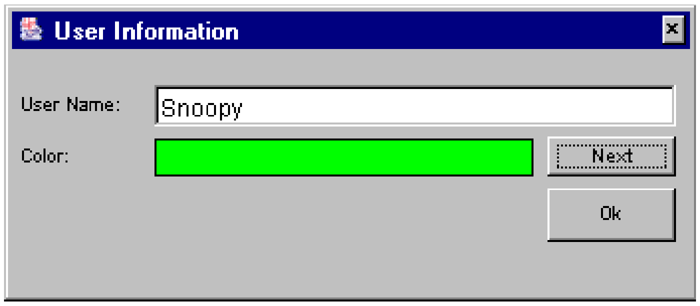
🟩 Leon: What should we draw?
🟥 Myself: I don't know but I sure am getting tired of scribbling.
🟥 Jen: I know, we could play a Pictionary type game.

Figure 5.7: The GChatTool.

5.2.4 Session Management Facilities

MAUI provides two optional dialogs for gathering session management information – GServerConnectionDialog and GLocalUserDialog. GServerConnectionDialog can be displayed at client startup time to gather information regarding the groupware server and port number to connect to. This dialog can be used for applications that allow connections to different servers or for applications where the server may be roaming. However, most applications are likely to provide a more elegant solution to gathering connection management information and will likely choose not to use this dialog in favor of a different solution. For example, in systems where there is a single, fixed location server, the application will likely provide automatic connection to the server at startup. In systems that connect to multiple servers the application will likely present a more elegant connection dialog that does not require entry of a server name or knowledge of port numbers. The GLocalUserDialog can be displayed at client startup to obtain information about the local user. The information obtained by the dialog is the user's name and a representative color for the user within the application. The dialog ensures that the user name and color choices are unique across all participants within the session. Once again, the GLocalUserDialog is a default dialog and many groupware applications may choose a more sophisticated method of obtaining local user information. GServerConnectionDialog and GLocalUserDialog are not widgets that can be added to an application via drag-and-drop. They can be enabled via a property setting on the GFrame, like the GParticipantDialog. GsSrverConnectionDialog and GLocalUserDialog are shown in Table 5.9.

Table 5.9: The Default Session Management UI Facilities in MAUI.

GServerConnectionDialog	 Connect To Server Host: localhost Port: 1099
GLocalUserDialog	 User Information User Name: Snoopy Color: 🟩

6 Developing Applications With MAUI

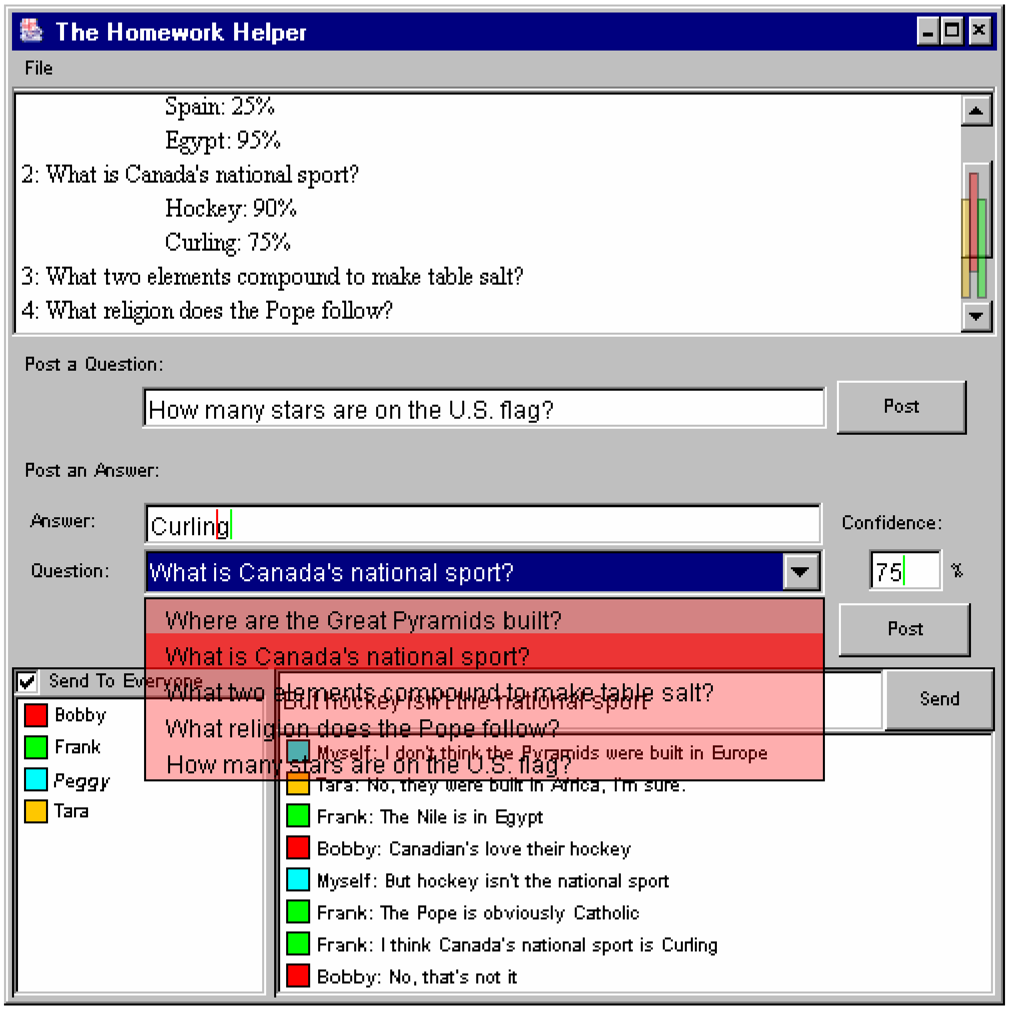
This chapter provides a detailed walkthrough of how to build an application with the MAUI toolkit. The walkthrough assumes that the reader is familiar with GUI development via a GUI builder tool, so it will be given using Borland's JBuilder IDE. Additionally, the walkthrough assumes that the reader has some familiarity with Java development and the Java development concepts outlined in chapter 2, such as JavaBeans and the use of the Event-Listener pattern. Later in the chapter, some of the sample applications that were built for testing MAUI will be briefly introduced.

6.1 A MAUI Development Walkthrough: The Homework Helper Application

This walkthrough will provide detailed instructions on how to build a simple multi-user application using MAUI and the JBuilder IDE. Note that development with another IDE may be done in a slightly different fashion, although the concepts and steps given here are generally applicable to development with any Java IDE.

6.1.1 Introduction to The Homework Helper

The application produced by the walkthrough has been called "The Homework Helper" and allows students to collaborate in real-time about their homework assignments by posting questions, discussing the problems among the group members, and posting answers for the questions with a confidence level for each answer. Note that the application is only a simple example application and is not necessarily meant to be an example of a real-world multi-user application. Figure 6.1, below, illustrates The Homework Helper in action. The scrolling text pane in the upper portion of the application window displays all the posted questions, with each of the answers posted for each question indented below the question. The confidence level for each answer is written to the left of the answer. The middle portion of the window allows the posting of questions and answers. A question can be entered into the top text field and posted to the message pane via the "Post" button the right of the text field. An answer is posted for a question by first typing the answer into the text field to the right of the "Answer" label and selecting the question for which the answer applies from the "Question" combo box. Next, the user enters a confidence level for the answer in the "Confidence" text field and presses the "Post" button below the confidence text field to post the answer to the message pane. The chat control on the bottom portion of the window is used to discuss the questions with the members of the group.



Spain: 25%
Egypt: 95%
2: What is Canada's national sport?
Hockey: 90%
Curling: 75%
3: What two elements compound to make table salt?
4: What religion does the Pope follow?

How many stars are on the U.S. flag?

Curling

What is Canada's national sport?

75

Where are the Great Pyramids built?
What is Canada's national sport?
What two elements compound to make table salt?
What religion does the Pope follow?
How many stars are on the U.S. flag?

But hockey isn't the national sport

🟥 Bobby
🟩 Frank
🟦 Peggy
🟨 Tara

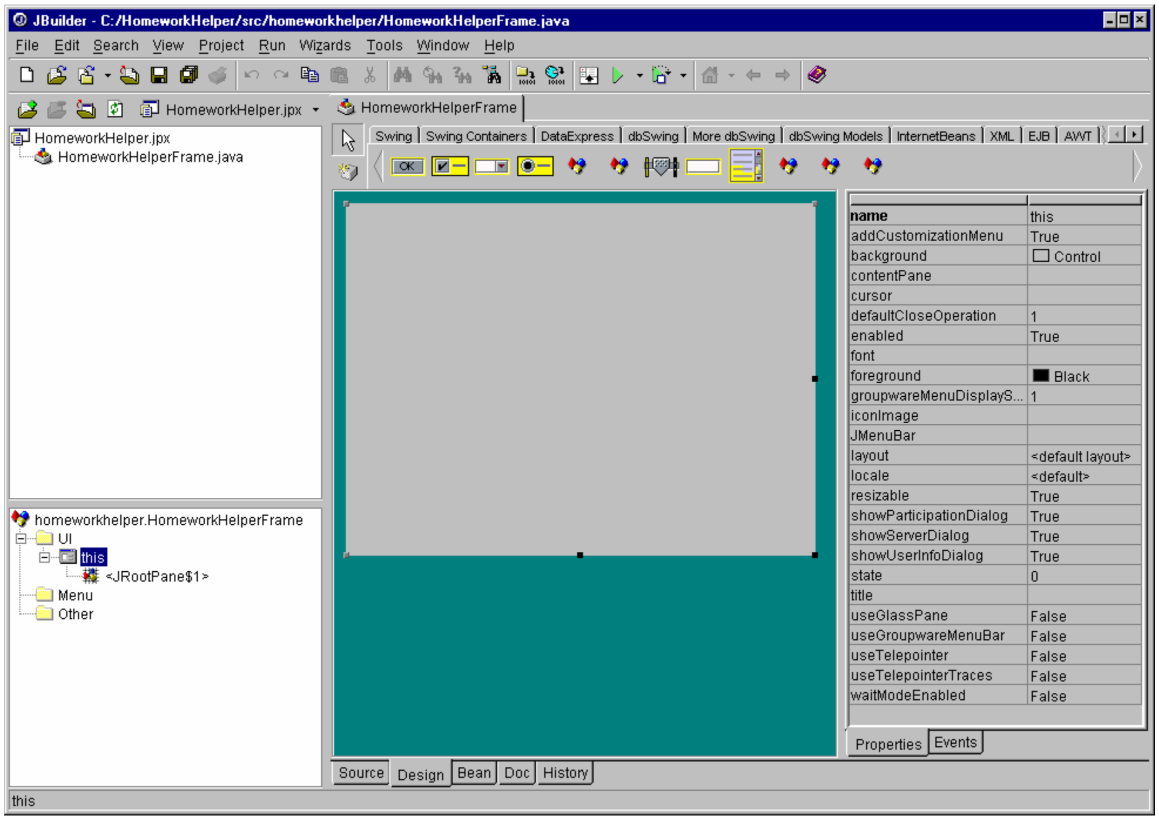
Myself: I don't think the Pyramids were built in Europe
Tara: No, they were built in Africa, I'm sure.
Frank: The Nile is in Egypt
Bobby: Canadian's love their hockey
Myself: But hockey isn't the national sport
Frank: The Pope is obviously Catholic
Frank: I think Canada's national sport is Curling
Bobby: No, that's not it

Figure 6.1: The Homework Helper Application.

6.1.2 Building the Application

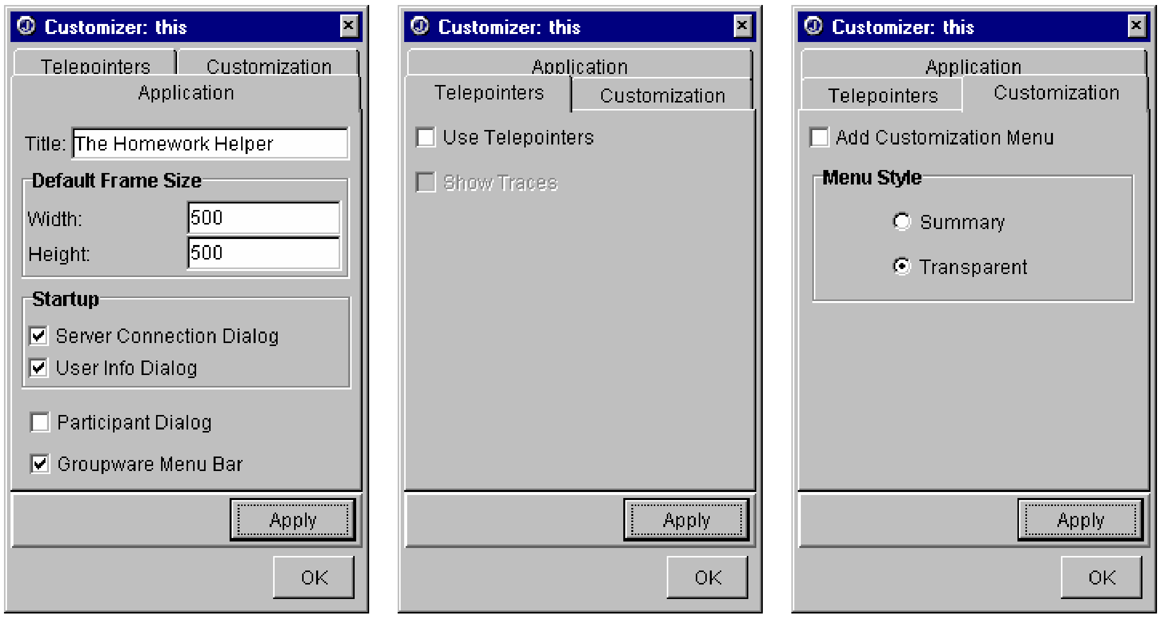
Most of the walkthrough steps detailed below are the same steps that would be followed if developing the application as a single-user application. Each step that is different from single-user application development is marked with '***'. In terms of coding, the only steps that are specific to groupware development with the MAUI toolkit are steps 6, 14 and 15. Step 6 customizes the groupware frame, while steps 14 and 15 add awareness event handling functionality.

1) The first step in creating The Homework Helper application is to create a new project named HomeworkHelper. Choose the default project settings.
2) *** Add the MAUI library as a library dependency for the project.
3) *** Add the MAUI component set to the list of available components on the JavaBean palette for the designer.
4) Create a new class in the homeworkhelper package. Name it HomeworkHelperFrame and make its base class the GFrame class, located in the groupware.controls package.
5) In the GUI Builder, activate the visual designer for the HomeworkHelperFrame. Figure 6.2 shows what the JBuilder IDE should look like after steps 1 through 4 have been completed.
6) Activate the GFrame customizer and set the properties for each tab as shown in Figure 6.3, below. Apply the property settings to the GFrame by pressing the apply button. Press the Ok button to close the customizer.


JBuilder - C:/HomeworkHelper/src/homeworkhelper/HomeworkHelperFrame.java
File Edit Search View Project Run Wizards Tools Window Help
HomeworkHelper.jpx
HomeworkHelperFrame
Swing | Swing Containers | DataExpress | dbSwing | More dbSwing | dbSwing Models | InternetBeans | XML | EJB | AWT
HomeworkHelper.jpx
HomeworkHelperFrame.java
homeworkhelper.HomeworkHelperFrame
UI
this
<JRootPane$1>
Menu
Other
Properties
Events
Source
Design
Bean
Doc
History
this

name	this
addCustomizationMenu	True
background	☐ Control
contentPane	
cursor	
defaultCloseOperation	1
enabled	True
font	
foreground	■ Black
groupwareMenuDisplayS...	1
iconImage	
JMenuBar	
layout	<default layout>
locale	<default>
resizable	True
showParticipationDialog	True
showServerDialog	True
showUserInfoDialog	True
state	0
title	
useGlassPane	False
useGroupwareMenuBar	False
useTelepointer	False
useTelepointerTraces	False
waitModeEnabled	False

Figure 6.2: The Homeworker Helper Frame in the JBuilder GUI Designer.



Title: The Homework Helper

Default Frame Size

Width: 500

Height: 500

Startup

[✓] Server Connection Dialog

[✓] User Info Dialog

[ ] Participant Dialog

[✓] Groupware Menu Bar

[ ] Use Telepointers

[ ] Show Traces

[ ] Add Customization Menu

Menu Style

○ Summary

● Transparent

Figure 6.3: Application-Level Property Settings for the Homework Helper Application.

7) Create a menu bar for the application and add a File menu. Under the File menu add an "Exit" menu item that exits the application when used.

8) Set the layout of the frame to a BorderLayout, add two JPanel objects to the frame, and set the layout constraints for one panel to "South" and the other to "Center". Set the layout of each of the added panels to the BorderLayout.
9) To the center panel add a GScrollPane object. Configure the scroll pane by setting its layout constraint to "Center". The default settings for the rest of the properties of GScrollPane are sufficient. Note that adding the GScrollPane is done in exactly the same way as adding a JScrollPane would be.
10) Add a JTextPane to the GScrollPane. Doing so should set the GScrollPane as the viewport to the JTextPane. Configure the text pane by setting its text property to an empty string and setting its editable property to false.
11) To the south panel add two components. First, add a GChatControl, binding it to the center of the panel. Add a new JPanel to the south panel and bind it to the north area of the south panel. We will call this new panel the message panel. Set the layout manager for the message panel to a null layout manager. Set the height of the preferred size property to 165 pixels. The width of the preferred size does not matter.
12) Next we will build the message and answer composition panel. To do this, add six JLabel objects, three GTextField objects, a GComboBox object and two GButton objects to the message panel. Organize the widgets on the message panel and set the text property for applicable widgets as shown in Figure 6.4, below. The name to give to each groupware widget is suggested by the text-filled ovals in the figure.
13) We now need to add some logic to the application to handle posting questions and answers and updating the text pane with the questions and answers as they are posted. First, we will add a LinkedHashMap as a private class member of HomeworkHelperFrame. Name the variable for the LinkedHashMap "questionsAndAnswers", as it will locally hold a mapping of each posted question to the posted answers for that question. The reason the map is a LinkedHashMap is to preserve the order in which questions were posted.

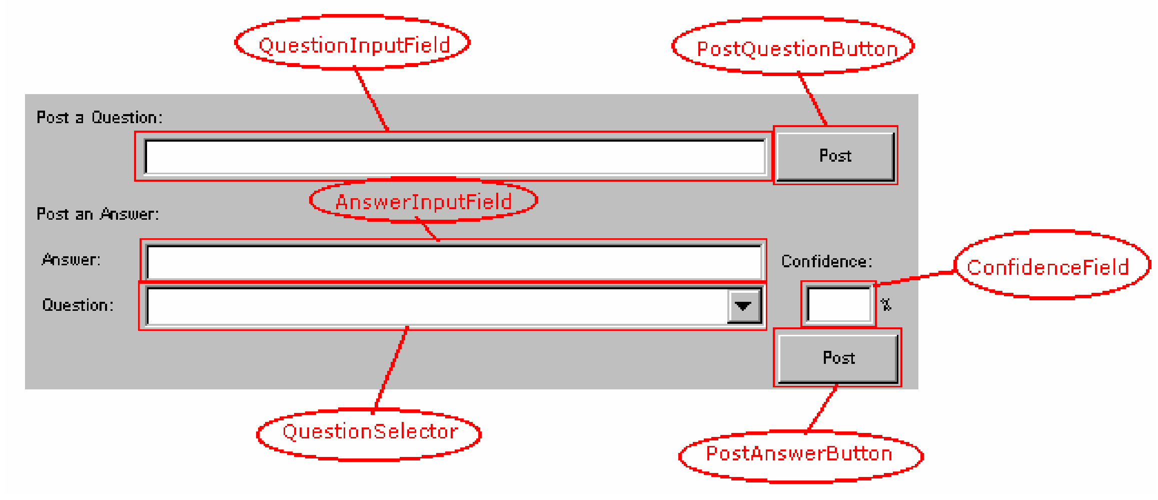


Figure 6.4: Names for the Groupware Widgets of the Message Panel.

14) *** Next, add two event handlers for the post button. The first handler will be for the actionPerformed event and will handle the addition of a posted question by the local user. The second handler will be for the gAwarenessActionReceived event and will handle receiving the event raised when a remote user posts a question. The thing to note about this step is that in addition to handling the action performed event for the post question button, there is an event handler for the event associated with receiving a remote awareness action event for the post question button. However, the action taken is identical for the two handlers. The handlers call a method that adds the question to the questionsAndAnswers map and the question combo box, and update the view. The way in which the events are handled would likely be done in a different manner in a production quality application. Rather than maintain the shared data within the client frame class, the application would have a model layer object that maintains the question and answer data and that notifies the client of changes to the data.

15) *** Next, add event handlers for the posting of answers in the same fashion as was done for the addition of questions in step 14. Again, the thing to note about this step is that in addition to handling the action performed event for the post answer button, there is an event handler for the event associated with receiving a remote awareness action event for the post answer button. However, the action taken is identical for the two handlers. The handlers call a method that takes the answer and confidence

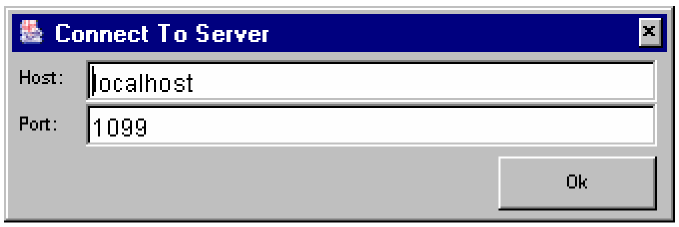
level received and adds it to the list of answers for the appropriate question and the view accordingly.

16) Finally, add the method that updates the message text pane that is called in the addQuestion() and addAnswer() methods defined above. This method will iterate through the question and answer map and add each question and its related answers to the message text pane. There is no special groupware-specific code in this method.

6.1.3 Running the Application

The following steps define the procedure that is used to run the HomeworkHelper application.

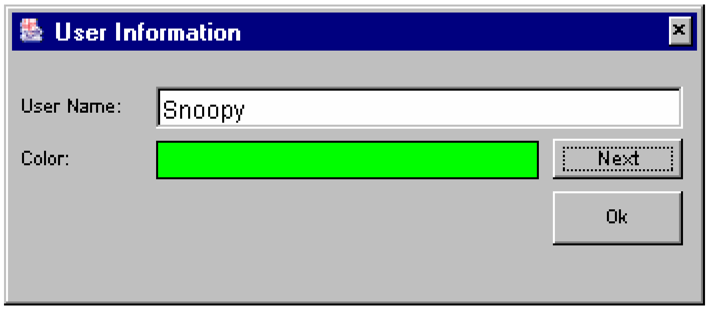
1) Start the Java RMI Registry with the default port setting.
2) Start the test server using the startup script appropriate for the platform where the server will be run. The startup scripts are included as part of the MAUI toolkit.
3) Start the application. Be sure to use the groupware.client.GroupwareClient class as the main class for the application. As an application command line parameter pass the fully-qualified class name of the main frame for the application, which will be homeworkerhelper.HomeworkHelperFrame.
4) Once the application has been started, the Server Connection Dialog will be presented, as shown in Figure 6.5 below. If the client has been started on the same computer as the server, accept the default server setting, otherwise enter the name of the server in the dialog. The RMI Registry was started on the default port so the port setting in the dialog does not need to be changed. Press the "Ok" button to continue.



Host: localhost
Port: 1099

Figure 6.5: Connecting to the Homework Helper Server.

5) Next, the Local User Information Dialog will be presented, as shown in Figure 6.6, below. Enter your user name and select a representative color. Press the "Ok" button to continue.



User Name: Snoopy

Color:

Figure 6.6: Entering User Information for the Homework Helper Application.

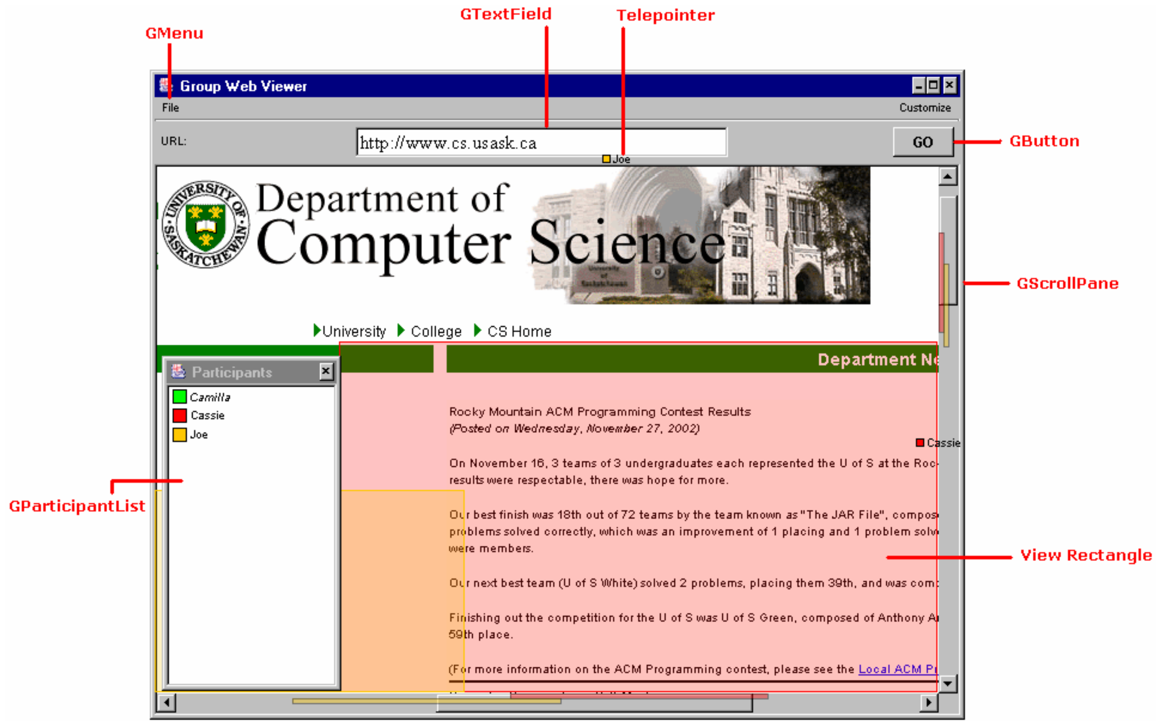
6) Now you can start using the application. Any other clients that are started and that connect to the same server will join the group session.

6.2 Test Applications

This section briefly introduces three of the sample applications that were built using MAUI, for the purpose of testing the toolkit. In addition to the HomeworkHelper application, the Shared Document Viewer, the Shared Drawing Application, and the Auto Shopper application were built with MAUI. These applications were not the only test applications built but were three of the more real-world type applications that were developed. The user interface of the three applications proved to be about the same development effort as a single-user version of each application. The multi-user applications additionally required the development of a model layer to support the shared model between users. The next three subsections will discuss the three applications.

6.2.1 The Shared Document Viewer Application

The Shared Document Viewer Application allows a group to simultaneously navigate a text (including HTML) document. One user will enter the URL of a document, press the "GO" button and all users will be connected to that document. The GScrollPane provides users with the location information of all other users via its multi-user scroll bars and view rectangles. The participant dialog provides the name and representative color of each user. The widgets and multi-user features that were used from MAUI (in addition to the core infrastructure) for the user interface include the scroll pane, push button, menu and menu item, text field, telepointers, view rectangles, and participant list. Figures 6.7, below, shows an annotated screen shot of the Shared Document Viewer.



Rocky Mountain ACM Programming Contest Results
(Posted on Wednesday, November 27, 2002)

On November 16, 3 teams of 3 undergraduates each represented the U of S at the Roc
results were respectable, there was hope for more.

Our best finish was 18th out of 72 teams by the team known as "The JAR File", compos
problems solved correctly, which was an improvement of 1 placing and 1 problem solv
were members.

Our next best team (U of S White) solved 2 problems, placing them 39th, and was com

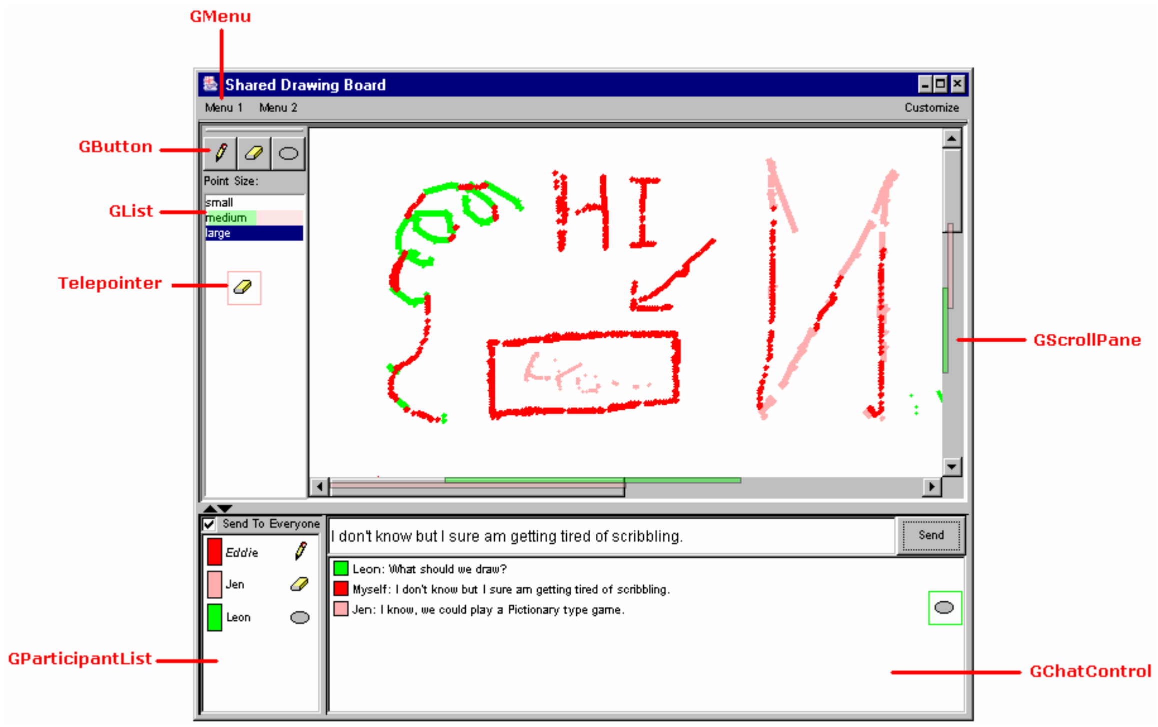
Finishing out the competition for the U of S was U of S Green, composed of Anthony A
59th place.

(For more information on the ACM Programming contest, please see the Local ACM Pr

Figure 6.7: Shared Document Viewer.

6.2.2 The Shared Drawing Application

The Shared Drawing Application allows groups to simultaneously sketch within the workspace. Each user draws in their representative color. Three drawing tools are provided: the pencil, the eraser, and the circle drawing tool. The currently selected tool for a user is indicated in the participant display found in the bottom right-hand corner of the application window. Also, the telepointer of a remote user is changed based on the selected tool. There are three drawing point sizes available: small, medium, and large. The point size for each user is indicated in the GList located in the middle region of the right-hand side of the application window. A chat area is provided at the bottom of the window for discussion about what is being drawn. The widgets and multi-user features that were used from MAUI (in addition to the core infrastructure) for the user interface include the scroll pane, push button, menu and menu item, list, telepointers, and chat control (with embedded participant list with user state indicators). Figure 6.8, below, shows the Shared Drawing Application.



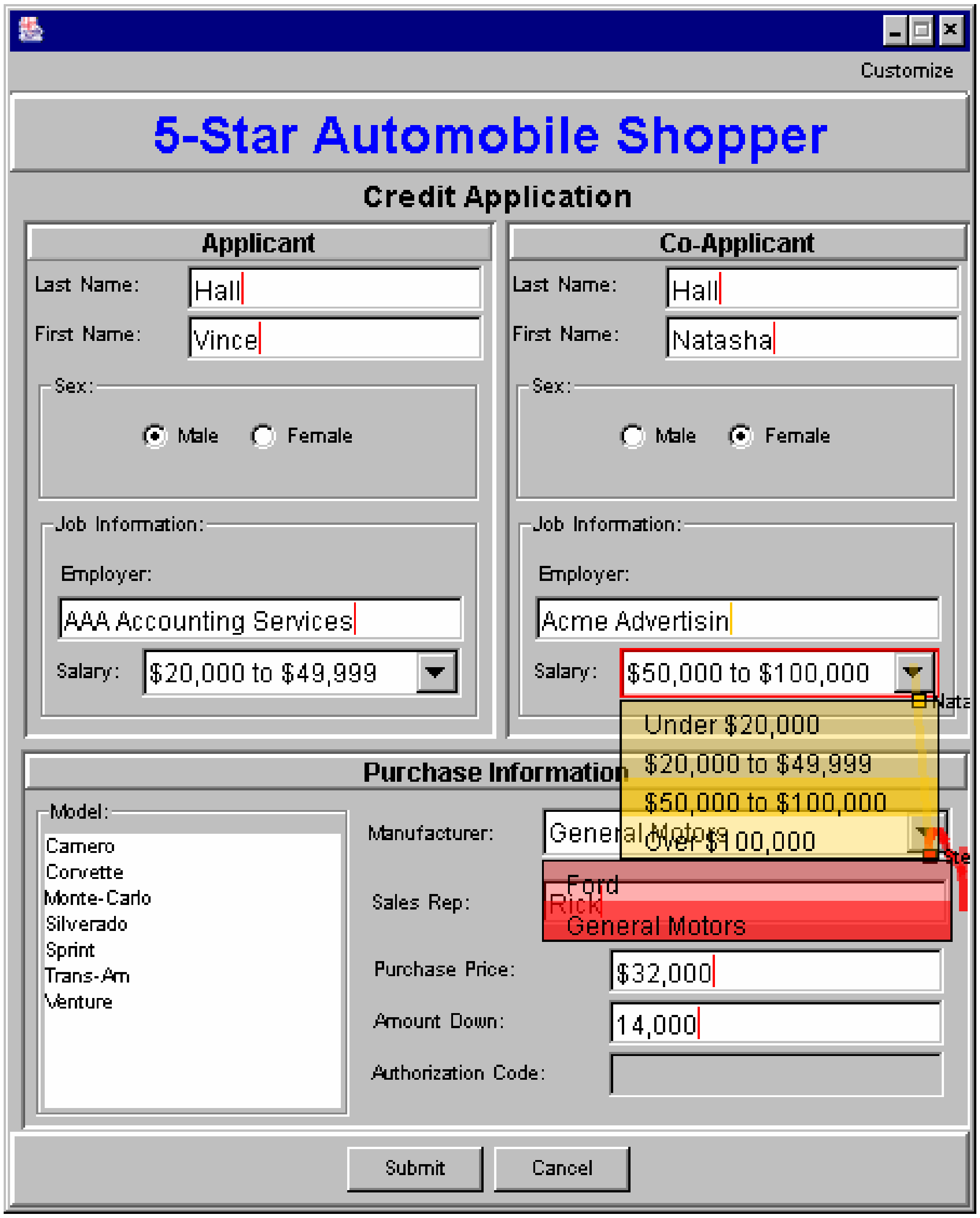
I don't know but I sure am getting tired of scribbling.

Leon: What should we draw?
Myself: I don't know but I sure am getting tired of scribbling.
Jen: I know, we could play a Pictionary type game.

Figure 6.8: The Shared Drawing Application.

6.2.3 The Auto Shopper Application

The Auto Shopper Application is a fictitious but representative example of a real-world on-line purchasing application. This application allows an applicant, co-applicant, and salesman to work together on-line to fill out an automobile purchasing application. Each user can fill in their portion of the required information and also can correct the mistakes of others within the other areas of the form. The widgets and multi-user features that were used from MAUI (in addition to the core infrastructure) for the user interface include the scroll pane, push button, radio button, combo box, menu and menu item, list, and telepointers with traces. Figure 6.9, below, illustrates the Auto Shopper Application.



Customize

5-Star Automobile Shopper

Credit Application

Applicant		Co-Applicant	
Last Name:	Hall	Last Name:	Hall
First Name:	Vince	First Name:	Natasha
Sex:	[● Male] ○ Female	Sex:	○ Male [● Female]
Employer:	AAA Accounting Services	Employer:	Acme Advertisin
Salary:	$20,000 to $49,999	Salary:	$50,000 to $100,000

Under $20,000
$20,000 to $49,999
$50,000 to $100,000
Over $100,000

Purchase Information

Model:
Camero
Corvette
Monte-Carlo
Silverado
Sprint
Trans-Am
Venture

Manufacturer:	General Motors
Sales Rep:	Rick
Purchase Price:	$32,000
Amount Down:	14,000
Authorization Code:	

Ford
General Motors

Figure 6.9: The Auto Shopper Application.

7 Evaluation

MAUI has been evaluated based on a number of criteria that are important to software developers and end users. With respect to software developers, the criteria include simplicity (effort reduction), generality, flexibility, extensibility, and performance. In addition, developers should be aware of the limitations of using MAUI, as compared to development without a toolkit. From the end user's perspective, some analysis should be done to determine the adequacy of the awareness information provided by the widgets and the visualization of that awareness information. The remainder of this chapter will discuss the evaluation process in general before delving into each of the evaluation criteria in detail.

7.1 The Evaluation Process

To evaluate MAUI, three different evaluations were performed. The first evaluation involved creating a number of sample applications with MAUI to determine how easy it is to create multi-user applications from scratch using the MAUI components and an IDE with a GUI builder. More specifically, the sample applications were built to determine the simplicity of building applications with MAUI. In building the applications, the limitations of developing with MAUI could also be determined. During the process of testing the applications, the performance of MAUI was analyzed. Selecting a range of applications with different requirements allowed the design-time flexibility of the toolkit to be analyzed. A subset of the test applications was introduced in Chapter 6.

The second evaluation involved building a multi-user application three different ways; without the use of a toolkit, using the GroupKit toolkit, and using MAUI. The JBuilder IDE was used for development in both the non-toolkit and MAUI approaches. This evaluation provided information about effort savings when using MAUI as compared to other approaches, as well as a comparison in performance between different approaches.

While the first two tests were mainly performed to test MAUI from a developer's perspective, the third evaluation was performed to gather information about MAUI from an application end-user's point of view. The evaluation was in the form of a user survey. A group of graduate students familiar with using groupware were given the sample applications to evaluate. As part of the evaluation, the students were given a questionnaire regarding each of the applications. The questionnaire focussed on the contribution of the awareness information provided by the widgets, the visualization of this awareness information, and the performance of the applications. The results of the evaluation were used to draw conclusions about the effectiveness of the MAUI widgets from the point of view of end users.

7.2 Simplicity (Effort Reduction)

The most important contribution of any toolkit is the reduction in effort that it provides to application developers. A toolkit should provide application development methods that are easy to learn, understand, and apply to a development process. For the MAUI toolkit, the evaluation for effort reduction involved the creation of an application in three different ways: without the use of a groupware toolkit, with the Groupkit toolkit, and with the MAUI toolkit. For each approach, the length of time to build and test each application was recorded and the total number of lines of code written for the application was calculated. It should be noted that a number of lines of code are added to the applications by the development tools and can be added by these tools easily and quickly. Therefore, using lines of code as a measure of the amount of effort required to write an application does not necessarily provide an accurate result. As a consequence, this evaluation places its highest confidence in the development time measure. The application that was built was a simple messaging program that maintains a list of participants and some history of the messages that have been sent by all users. The button used to send the messages to remote clients provides feedback to the remote users when the local user has given the button mouse focus or when the button is pressed.

Table 7.1, below, shows the statistics recorded for developing the application with each method.

	No Toolkit	GroupKit	MAUI
Number of classes & interfaces	14	N/A (9 procedures)	1
Total lines of code	670 (62 for feedthrough)	69 (18 for feedthrough)	44 (all added via GUI builder)
Approximate total development time	4.5 hours	2 hours (1.25 for feedthrough)	15 minutes

Table 7.1: Statistics for the Development of a Simple Messaging Application.

By analyzing the results, it is evident that MAUI provides a significant reduction in the effort required to develop a distributed real-time groupware application, both for an application developed from scratch and one developed with a notable Groupware toolkit. It should also be mentioned that the MAUI version of the application provided additional awareness features over the other two versions, such as a multi-user scrollbar on the message history list and feedback awareness on the exit menu. As far as performance was concerned, there was no visible difference in performance from the end-user's point of view between the applications. This test, however, was only performed for the development of one representative sample application.

7.3 Generality

The term generality is used to refer to the scope of applications where MAUI is applicable. Looking at the MAUI architecture outside of the scope of the current implementation, the design can be applied to any multi-user widget set. The architecture can be extended to any platform and language that supports an event/notification or similar mechanism and the ability to provide a visualization of awareness information.

In the current implementation of the MAUI architecture, the toolkit can be used to create synchronous distributed multi-user applications with the Java language and the Swing and AWT component suites. Within this scope there are no further limitations.

In fact, because of the ability to turn on and off awareness information transfer and reception, MAUI can be used to create single-user applications as well as multi-user applications. With the awareness information propagation completely turned off for a MAUI application, it will behave the same as it would if it were built as a single-user application. In this way, application developers no longer need to worry about whether they are building single-user or multi-user applications. Developers can just create MAUI applications and allow end-users to use run-time customization to enable or disable the awareness information propagation as is necessary. Thus, MAUI as an extension of Swing allows a wider range of applications to be developed more easily than with Swing and AWT themselves.

To apply the MAUI architecture to another development language, the approach would vary depending on the language. For example, to create a C# version of MAUI the approach taken would be quite similar to the approach that was taken for the Java version, due to the similarities between C# and Java. Both languages are completely object-oriented and provide an event notification mechanism. While Java provides the JavaBeans set of conventions for component-based development, C# provides for component-based development as part of the language syntax, making the task of creating components potentially less error prone than development in Java. To develop a TCL/TK version of MAUI, however, would require a slightly different approach. TCL/TK is not object-oriented and, therefore, does not provide an extension mechanism for its widget set. However, TCL/TK does provide a notification mechanism that would allow widget extensions to be developed through event/notification handlers.

7.4 Flexibility

The flexibility of the MAUI toolkit refers to how adaptable the MAUI architecture and its components are with respect to the requirements and design of particular applications. MAUI exhibits flexibility in several ways. First, the black-box nature of the communication infrastructure allows use of different network communication protocols for different applications and even multiple protocols within a single architecture. Additionally, the communication infrastructure performs message clustering that, to some extent, is configurable and can go a long way towards tuning the network performance of an application.

MAUI uses a combination of design-time and run-time customization to increase the flexibility of its component set. By adhering to the JavaBeans standard for Java component-based development, the highly flexible design-time customization facilities of Java development tools can be leveraged. Every MAUI widget provides a number of properties that are settable through the property editor of a Java GUI builder. As discussed in Chapter 4, MAUI provides an application-level customizer GUI that can be used to easily modify a number of application-level properties. In the future, customizers can be added for each widget without modification to the widget itself (provided the widget allows modification to the properties that are desired to be customizable). Similarly, if a MAUI widget conforms to the run-time customization interface any of its properties can be exposed and editable via a run-time customizer. Currently, all MAUI widgets use the same run-time customizer to allow editing of a few of the basic common widget properties.

The flexibility of MAUI was evaluated through the building of sample applications and via end-user surveys. Many of the comments found in the end-user questionnaires were regarding the way in which awareness information was visualized within an application. For instance, some users felt telepointers and/or traces should not be used in applications where they were used and vice-versa. Another example is regarding the intention awareness highlighting effects used for particular controls. Some users felt the effects were too distracting. These types of issues can be dealt with through the run-time customization interface for the application of a particular widget. For instance, telepointers and traces can be enabled and disabled through the application run-time customizer and awareness effects can be selected through a widget run-time customizer. Also, the intensity of an awareness effect can also be scaled up or toned down via the customizers. However, the run-time customizability of a widget is limited to those properties that are exposed for run-time customization for the widget, and in the current implementation the list of customizable options is not extensive. It is clear from the evaluations that the range of customizable options for each widget will need to be more extensive to better satisfy the differing needs and tastes of end users.

7.5 Extensibility

Extensibility refers to the ability of the MAUI architecture and component set to be extended to add new functionality that is not provided in its standard form. This includes adding widgets, extending widgets, extending message types, changing the communication infrastructure, and extending customization. New widgets can be plugged into the architecture as long as the widgets implement the GroupwareControl interface. A new widget can extend a Swing or AWT widget and add multi-user functionality, or it can be a completely new widget implementation that does not extend any pre-existing implementation. Implementing the GroupwareControl interface provides the widget with the capability to communicate with the communication infrastructure, however, the widget must still implement out-going-message construction, incoming-message processing, and awareness visualization. Depending on the nature of the widget, this may require differing levels of complexity. As mentioned previously, there are several pre-existing message types that are used by the existing widgets. These message types can be used in the creation of new widgets, either as is or they can be extended into new message types. To ease implementation, the BasicAwarenessAdapter can be used to provide visualization techniques and standard implementations for communication infrastructure connectivity. Finally, the widgets can take advantage of both design-time and run-time customization facilities. By following the JavaBeans guidelines, the widgets will be design-time customizable and can make use of the customization facilities of Java development tools. By implementing the RuntimeCustomizableGControl interface, the run-time customization facilities can be leveraged.

The extensibility of MAUI was put to an informal test by having a developer create a new widget for the toolkit. The widget was a tabbed pane that provides feedback when a tab is selected. The developer made use of the BasicAwarenessAdapter and a subset of the standard widget set to create the new component in a few hours' time.

7.6 Performance

With respect to the thesis version of MAUI, the performance goal was to be capable of providing awareness information in real-time, while maintaining a level of performance that is not significantly different than if the awareness provision had been added without the MAUI toolkit. Performance was continually analyzed in an informal fashion throughout the development and testing of MAUI. Because of the black-box nature of the lower level of the MAUI communication infrastructure, low-level communication protocols were not heavily weighted in the evaluation. The low-level communication protocols are the same mechanisms that would be used in non-toolkit-based groupware application development. What the evaluation does focus on is performance with respect to message generation and processing by the message controller, dispatcher proxy and widgets, as well as the awareness information visualization performed by the widgets.

In general, MAUI performs nearly as well as a custom groupware application could be expected to perform. The controller-dispatcher mechanism adds very little overhead to the message processing (there is only a little overhead with respect to the use of the dispatcher proxy to make the infrastructure a black box). The dispatcher-controller mechanism serves mainly as a message router. If a customized architecture was built, linking components directly to each other via direct socket connections might enhance performance. However, the small performance penalty to make the mechanism generic seems to be worth the tradeoff in the test cases. As far as the messages are concerned, message size may be slightly reduced if the messages were completely specialized to their task. MAUI generalizes the messages slightly by encoding some generic knowledge about the communication link into the message. Also, a custom message processing mechanism may be able to intelligently dispose of unnecessary messages and negotiate transfer priorities based on specific application knowledge. Message prioritizing and quality of service are issues that have not yet been addressed within the MAUI toolkit.

As far as the performance of the widgets and view components are concerned, visualization and response times are adequate enough to provide a smooth and responsive system, with a few exceptions. For the basic controls (buttons, combo box, list box, etc.), scroll bar, participation list and a few other widgets, the number of messages being generated is low and the visualization of the awareness information is simplistic enough that the amount of time to update a client is noticeable, subject only to the amount of lag in the network. Even for a moderate group size (four or five users) this appears to remain true. For telepointers and telepointer traces the findings are slightly less positive. Awareness events for telepointers are generated based on the mouse motion events generated within a user interface. This may be dozens to hundreds of events in a one-second period. This may exceed the rate at which the network communication middleware can process, send, and receive network communication messages. Because of this, MAUI events may be queued up causing a large delay. Currently, MAUI events are not prioritized, so if a large a large number of telepointer events block the network message queue, other awareness events will also be blocked. This problem was partially alleviated by adding the concept of message bundling to the MAUI dispatcher proxy. Message bundling allows MAUI events to be packaged together and sent to remote clients via one network communication call, thus lessening the amount of network traffic. Another performance bottleneck in MAUI is with respect to the rendering of telepointer traces. Recall that telepointer traces are visualized as a trail of translucent points following the telepointer. Over time, these translucent points fade away. The process of translucently fading away telepointer trace points has proven to be a rather CPU intensive process. For traces with a large number of points (over 100) the result is often a jittery user interface. The effect is compounded when there is a moderate number of users in the system. A point to note is that typically the number of trace points is relatively small because a large number of points are often too distracting for users. It should be noted that the transparency problem is due to the inefficiency of the Java transparency implementation and not MAUI. One other performance bottleneck that will be mentioned is the event-handling mechanism of the groupware glass pane. When activated, the glass pane captures all user-input events and forwards them to the control they were intended for, providing additional handling for certain input events.

The process of trapping and forwarding events necessitates an entire extra layer of event processing.

7.7 *Effectiveness from the User's Perspective*

It is important that the widgets provided by a toolkit provide users with adequate functionality that allows them to perform the tasks they desire. As part of the evaluation process, a user evaluation with small group (four) of experienced groupware users was performed. The group members were given the three sample applications introduced in Chapter 6 to provide feedback on with respect to the adequacy of the awareness information and visualization provided by the toolkit widgets. In general, all group members agreed that the awareness information provided was beneficial to the application it was provided within, and that it made the application more useful than if the awareness information was not present. Performance did not present a problem, and the awareness information was provided in a timely fashion. While all group members agreed that the implementation of the visualization of the information was adequate, all users provided comments about how the visualizations for particular widgets could be presented differently under different circumstances. However, the users often disagreed in how they thought the awareness information could better be visualized. It was concluded that the presentation of awareness information requires further study and is itself a potentially large research project. In general, though, the MAUI design-time and run-time customization facilities could be used to provide a number of different representations of awareness information for a single widget, as well as allow a developer to create a custom representation that could be plugged in to the widget. Both of these customization enhancements will be the topic of future work. One other significant comment that was provided was with respect to widget consistency. One of the users felt that it was important to ensure consistency in the operation of the widgets and the visualization of the awareness information. This is an issue that will be focussed on more heavily in future work.

7.8 Limitations

This section will discuss the limitations of development with the MAUI toolkit. Many of the limitations discussed are due to the scope of the current project, and future development will attempt to address limitations of this nature.

a) *Use of the Glass Pane.* Because MAUI uses a glass pane to implement transparency and other visualization techniques, an application developer cannot use their own glass pane in a MAUI application. GGlassPane can be extended, but caution must be taken when doing so to prevent damaging the awareness capabilities of the glass pane. It should be noted that this limitation is a result of the choice of implementation of a set of visualization techniques and not the architecture.

b) *Groupware Frame Contains Application-Level Functionality.* Considerable application functionality has been placed in the GFrame class rather than within a special application class. This limits applications to using a frame class as their main GUI window rather than using other window classes such as dialogs or applets as their main application windows.

c) *Incomplete Widget Set.* Only a subset of available single user widgets have multi-user versions implemented, and only a subset of the multi-user widgets found in existing groupware systems and literature have been implement to date.

d) *Missing Use Mode Implementations.* All widgets do not have both use modes (coupled and individual) implemented. In fact, it has not yet been determined if both use modes are even applicable to all possible widgets.

e) *Bounds of Transparency Effects.* Transparency effects cannot be shown outside the bounds of the glass pane. Therefore, when widgets such as menu items need to be drawn outside the window bounds they cannot make use of transparency effects.

f) *No Method of Updating Late Joiners.* Currently, there is no built in method of bringing users up to date with the rest of the group when they join an application session. This means that if no change in the state of awareness occurs for an aspect of the application, a client will never be aware of that aspect of the application.

g) *Widgets May Not Be Refreshed Upon User Exit.* When a user exits an application, the widgets of a remote client that are exhibiting stateful awareness information, such as intention awareness, may never receive a message indicating the intention has been cleared. This will result in the widget exhibiting the awareness state until the termination of the client.

h) *No Message Priorities.* Messages do not have priorities associated with them. Messages are always processed on a first-come-first-served basis.

i) *Runtime Customization is at the Widget Type Level.* The runtime customization of widgets occurs for a widget type rather than an individual widget.

8 Conclusion

This chapter concludes the thesis by providing a brief summary of the research problem, goals, and approach to the solution. The major and minor contributions of the thesis are restated. The final section of the chapter provides an introduction to the planned future work and directions for the research started in this thesis.

8.1 Summary of Research and Contributions

This thesis confronted the problem that *it is difficult to build groupware interfaces that support group awareness*. Group awareness is the up-to-the-moment understanding of another person's activities in a group environment. The difficulty exists because of the need for custom development in groupware applications, due to the lack of reusable groupware components in existing groupware toolkits. Groupware GUIs must take into consideration many issues beyond those typically found in single-user GUIs. These issues include distributed communication, devising a remote messaging scheme, coordinating the transformation of user actions to remote messages, and transforming remote messages to graphical updates on the GUIs of the remote users.

The main goal of the research was to reduce the development effort required to produce groupware systems that support group awareness. A second goal of the research was to provide a more informative collaborative environment for groupware users. The motivation is the notion that better support for awareness means better collaboration for users. The problem was addressed by developing a new groupware GUI toolkit that allows developers to create groupware applications quickly using a suite of reusable components. The main focus of the functionality of the components is group awareness support, abstracting away details of remote communication, and message passing. Process feedthrough information is the key type of awareness information provided by the components.

The toolkit simplifies the construction of groupware that supports group awareness in three ways:

- It provides reusable, groupware-enabled GUI widgets.
- It provides communication infrastructure components.
- It integrates with a set of popular IDEs.

Five steps were completed for the research:

- Classify GUI widgets in terms of groupware requirements.
- Develop a communication infrastructure.
- Develop the GUI component infrastructure
- Develop a representative sample of GUI components.
- Build a set of applications using the toolkit.

The major contributions of the research are:

- A groupware toolkit that simplifies the construction of groupware interfaces. The toolkit integrates with direct-manipulation JavaBean development tools and contains a number of generic reusable groupware components.
- A quicker and easier way of developing groupware applications that provide group awareness than is currently available.
- Groupware GUI components that provide built-in support for group awareness, including support for feedthrough.
- A groupware toolkit infrastructure that abstracts distributed communication issues away from application developers.

A minor contribution of the research is:

- The requirements analysis results obtained from the first solution step, which is the classification of GUI widgets in terms of groupware requirements.

8.2 Future Work

Following the completion of this thesis, enhancements and extensions have been planned for a future release of the MAUI toolkit. First of all, more testing and bug fixes will be done to the existing toolkit. Currently, MAUI contains a number of performance and usability bugs. These will be resolved before any other work is pursued. Secondly, the performance of MAUI will be analyzed and performance enhancements will be put in place. Through development and testing, a number of potential performance enhancements have been noted that will be evaluated and adopted, if proven beneficial. Next, more discussions and studies will occur regarding visualization techniques for awareness information, as this was one of the biggest areas of concern based on the end user evaluation performed for this thesis. The likely result of this discussion and evaluation will be enhancements and extensions to the widget customization interface and the customizable properties for each widget. Following this, new widgets will be developed. First, multi-user versions of the remaining Swing widgets will be developed followed by the creation of the missing multi-user widgets found in current groupware literature. Finally, any new widget types will be created that are discovered through the discussion and evaluation process. Naturally, following all of this there will be testing, testing, and more testing.

Two other areas of enhancement not directly related to widgets and awareness that will be pursued will be support for collaboration transparency and testing the black-box nature of the communication infrastructure. Support for collaboration transparency will be added through the post-processing of single-user applications by scripts that will swap in MAUI components for their single-user counterparts automatically. Additional scripts driven through a user-interface can be used to easily add multi-user specific widgets to the applications, as well allow finer-grained control over which widgets should be swapped in. Also, the capability to provide a run-time engine that will swap in multi-user components for single-user components will be investigated. The black-box nature of MAUI will be tested by swapping in different communication infrastructures for the one currently provided by MAUI.

9 References

[Ackerman & Starr 1995] Ackerman, M., Starr, B. (1995). “Social Activity Indicators: Interface Components for CSCW Systems.” *UIST ’95*, Pittsburgh, PA, USA, November 14-17, 1995, p159-168.

[Aoyama 1998] Aoyama, M. (1998). “New Age of Software Development: How Component-Based Software Engineering Changes the Way of Software Development.” Department of Information and Electronics Engineering , Niigata Institute of Technology, 1719 Fujihashi, Kashiwazaki 945-11, Japan.

[Banavar et al. 1998] Banavar, G., Doddapeneni, S., Millar, K., Mukherjee, B. (1998). “Rapidly Building Sychronous Collaborative Applications By Direct Manipulation.” CSCW ’98, Seattle, Washington, USA.

[Beaudouin-Lafon & Karsenty 1992] Beaudouin-Lafon, M., Karsenty, A. (1992). “Transparency and Awareness in a Real-Time Groupware System.” Laboratoire de Recherche en Informatique, Universite de Paris-Sud – Batiment 490 91405 ORSAY Cedex, France.

[Begole et al. 1998] Begole, J., Rosson, M., Shaffer, C. (1998). “Supporting Worker Independence in Collaboration Transparency.” UIST ’98, San Francisco, CA, USA.

[Berlage & Genau 1993] Berlage, T., and Genau, A. (1993). “A Framework for Shared Applications with a Replicated Architecture.” UIST ’93, November 3-5, 1993.

[Bhurat & Brown 1994] Bhurat, K., Brown, M. (1994). “Building Distributed, Multi-User Applications by Direct Manipulation.” UIST ’94, Marina del Rey, California, USA, November 2-4, 1994.

[Brewster et al. 1994] Brewster, S., Wright, C., Edwards, A. (1994). “Design and Evaluation of an Auditory-Enhanced Scrollbar.” *CHI ’94*, Boston, Massachusetts, April 24-28, 1994.

[Dewan & Choudhary 1995] Dewan, P, Choudhary, R. (1995). “Coupling the User Interfaces of a Multiuser Program.” ACM Transactions on Human Computer Interaction, v2, n1, 1995, p1-39.

[Dix et al. 1993] Dix, A., Finlay, J., Abowd, G., Bealle, R. (1993). “Human-Computer Interaction.” Prentice Hall.

[Dourish & Bellotti 1992] Dourish, P., Bellotti, V. (1992). “Awareness and Coordination in Shared Workspaces.” CSCW ’92.

[Dourish & Bly 1992] Dourish, P., Bly, S. (1992). “Portholes: Supporting Awareness in a Distributed Work Group.” *CHI ’92*, May 3-7, 1992, p541-547.

[Drye & Wake 1999] Drye, S., Wake, W. (1999). "Java Foundation Classes Swing Reference." Manning Publications Co., Greenwich, CT.

[Ellis et al. 1991] Ellis, C., Gibbs, S., Rein, G. (1991). “Groupware: Some Issues and Experiences.” Morgan Kaufmann Publishers, 1993, p9-28.

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). "Design Patterns. Elements of Reusable Object-Oriented Software." Addison-Wesley.

[Graham 1995] Graham, T. (1995). “Declarative Development of Interactive Systems.” Department of Computer Science, York University, Toronto, Ontario, Canada.

[Graham et al. 1996] Graham, T., Urnes, T., Nejabi, R. (1996). “Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware.” UIST ’96, Seattle, Washington, USA.

[Greenberg & Marwood 1994] Greenberg, S., Marwood, D. (1994). “Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface.” CSCW ’94, Chapel Hill, NC, USA, October 1994.

[Greenberg & Roseman 1999] Greenberg, S., Roseman, M. (1999). “Groupware Toolkits for Synchronous Work.” In M. Beaudouin-Lafon, edito, *Computer-Supported Cooperative Work (Trends in Software 7),* Chapter 6, p135-168.

[Gutwin 2001] Gutwin, C. (2001). “Traces: Visualization of Interaction.” Department of Computer Science, University of Saskatchewan, Saskatoon, SK, Canada.

[Gutwin 2002] Gutwin, C. (2002) “Traces: Visualizing the Immediate Past to Support Group Interaction.” Proceedings of the Conference on Human-Computer Interaction and Computer Graphics (GI'02), Calgary, Morgan Kaufman, 43-50.

[Gutwin & Greenberg 1998] Gutwin, C., Greenberg, S. (1998). “Effects of Awareness Support on Groupware Usability.” *CHI ’98*, Los Angeles, CA, USA, April 18-23, 1998, p511-518.

[Gutwin & Greenberg 2001] Gutwin, C., Greenberg, S. (2001). “A Descriptive Framework of Workspace Awareness for Real-Time Groupware.” *Computer Supported Cooperative Work*, Kluwer Academic Press.

[Gutwin & Greenberg 2000] Gutwin, C., Greenberg, G. (2000). “Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness.”.

[Gutwin et al. 1996] Gutwin, C., Roseman, M., Greenberg, S. (1996). “A Usability Study of Awareness Widgets in a Shared Workspace Groupware System.” CSCW ’96, Cambridge, MA, USA.

[Hill et al. 1994] Hill, R., Brinck, T., Rohall, S., Patterson, J., Wilner, W. (1994). “Language for Constructing Multiuser Applications.” *ACM Transactions on Computer-Human Interaction*, Vol. 1, No. 2, June 1994, p81-125.

[Hill & Hollan 1992] Hill, W., and Hollan, J. (1992). “Edit Wear and Read Wear.” CHI ’92, May 3 – 7, 1992.

[Jacobson et al. 1999] Jacobson, I., Booch, G., Rumbaugh, J. (1999). "The Unified Software Development Process." Addison Wesley Longman, Inc., Reading, Massachusetts.

[Kirtland 1999] Kirtland, M., (1999). "Designing Component-Based Applications." Microsoft Press, Redmond, Washington.

[Lauwers & Lantz 1990] Lauwers, C., Lantz, K. (1990). "Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window System." Proceedings of ACM CHI'90, 1990, p303-311.

[Microsoft 2002] Microsoft (2002). "DCOM" http://www.microsoft.com/com/tech/dcom.asp.

[OMG 2000] Object Management Group (2000). "CORBA" http://www.corba.org/.

[Phillips 1999] Phillips, G. (1999). "Architectures for Synchronous Groupware." Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada.

[Rine et al. 1999] Rine, D., Nader, N., Jaber, K. (1999). "Using Adapters to Reduce Interaction Complexity in Reusable Component-Based Software Development." *SSR '99*, Los Angeles, CA, USA, p37-43.

[Rodden & Blair 1991] Rodden, T., Blair, G. (1991). "CSCW and Distributed Systems: The Problem of Control." Morgan Kaufmann Publishers, 1993, p389-396.

[Roseman & Greenberg 1992] Roseman, M., Greenberg, S. (1992). "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications." CSCW '92 Proceedings, November 1992.

[Roseman & Greenberg 1995] Roseman, M., Greenberg, G. (1995). "Building Real Time Groupware with GroupKit, A Groupware Toolkit." ACM Transactions on Computer Human Interaction 1996.

[Schuckmann et al. 1996] Schuckmann, C., Kirchner, L., Schummer, J., Haake, J. (1996). "Designing Object-Oriented Synchronous Groupware with COAST." CSCW '96, Cambridge MA, USA.

[Schuckmann et al. 1999] Schuckmann, C., Schummer, J., Seitz, P. (1999). "Modelling Collaboration Using Shared Objects." Proceedings of ACM GROUP '99, Phoenix, Arizona, USA.

[Segal 1995] Segal, L. (1995). "Designing Team Workstations: The Choreography of Teamwork." *Local Applications of the Ecological Approach to Human-Machine Systems*, P. Hancock, J. Flach, J. Caird and K. Vicente ed., 392-415, Lawrence Erlbaum, Hillsdale, NJ.

[Sohlenkamp & Chwelos 1994] Sohlenkamp, M., Chwelos, G. (1994). "Integrating Communication, Cooperation, and Awareness: The DIVA Virtual Office Environment." *CSCW '94*, Chapel Hill, NC, USA, October 1994, p331-343.

[Sparling 2000] Sparling M. (2000). “Lessons Learned Through Six Years of Component-Based Development.” *Communications of the ACM*, Vol. 43, No. 10, p47-53, October 2000.

[Sun Microsystems 1997] Sun Microsystems, (1997). ”The JavaBeans Specification.”.

[Sun Microsystems 1999] Sun Microsystems, (1999). “The Java Shared Data Toolkit.”.

[Sun Microsystems 2002 A] Sun Microsystems, (2002 A). “The Java Tutorial: A practical guide for programmers.” http://www.javasoft.com/docs/books/tutorial/?frontpage-spotlight.

[Sun Microsystems 2002 B] Sun Microsystems, (2002 B). “Java Remote Method Invocation (RMI)” http://java.sun.com/products/jdk/rmi/index.html.

[Sun Microsystems 2002 C] Sun Microsystems, (2002 C). “The Swing Connection” http://java.sun.com/products/jfc/tsc/index.html.

[Topley 1998] Topley, K., (1998). "CORE: Java Foundation Classes." Prentice-Hall, Inc., Upper Saddle River, NJ.