

The MAUI Toolkit: Groupware Widgets for Group Awareness

JASON HILL & CARL GUTWIN

*Computer Science Department, University of Saskatchewan, Saskatoon, Canada S7N 5A9
(E-mails: {carl.gutwin, jason.hill} @usask.ca)*

Abstract. Group awareness is an important part of synchronous collaboration, and support for group awareness can greatly improve groupware usability. However, it is still difficult to build groupware that supports group awareness. To address this problem, we have developed the Multi-User Awareness UI toolkit (MAUI) toolkit, a Java toolkit with a broad suite of awareness-enhanced UI components. The toolkit contains both extensions of standard Swing widgets, and groupware-specific components such as telepointers. All components have added functionality for collecting, distributing, and visualizing group awareness information. The toolkit packages components as JavaBeans, allowing wide code reuse, easy integration with IDEs, and drag-and-drop creation of working group-aware interfaces. The toolkit provides the first ever set of UI widgets that are truly collaboration-aware, and provides them in a way that greatly simplifies the construction and testing of rich groupware interfaces.

Key words: awareness, feedthrough, groupware interfaces, group widgets

1. Introduction

Group awareness – the up-to-the-moment understanding of others’ activities in a shared space – is a crucial part of successful collaboration. However, it is still difficult and time-consuming for developers to support group awareness in synchronous groupware. Much of the code for gathering awareness information, distributing it, and displaying it on remote screens must be built from scratch, and too often there is little chance of later reuse in other applications. This situation exists despite the fact that groupware toolkits have greatly simplified the construction of many other aspects of synchronous groupware (e.g., Bharat and Brown, 1994; Hill et al., 1994; Roseman and Greenberg, 1996; Banavar et al., 1998).

To address this problem, we have developed a toolkit to simplify the construction of interfaces that support group awareness. The Multi-user Awareness UI toolkit (MAUI) is a Java-based toolkit that provides a comprehensive suite of awareness-enhanced interface components. The main difference between this toolkit and others is that MAUI provides shared and multi-user versions of standard UI widgets like buttons, sliders, menus, lists, and tables. These types of components have for the most part been

overlooked by designers of groupware toolkits, perhaps because of a common focus on custom-built domain artifacts that appear in dedicated shared workspaces. However, 'ordinary' GUI widgets are also important settings for collaboration and awareness: when there is a shared workspace, considerable activity still goes on in the controls around that space; when there is no custom-built workspace (such as in a form), then the UI components are the primary work artifacts.

GUI components can provide valuable awareness information, and are good candidates for implementation in a groupware toolkit. MAUI components provide several types of awareness information, but they are primarily based on the idea of showing people's activities as they manipulate the application interface. This mechanism is called feedthrough – feedback to the single user that also helps others understand the activity (Dix et al., 1993). For example, watching another person navigate through the items in a menu gives valuable clues about what they intend to do next. Feedthrough information is particularly valuable when people are working closely together, when one person is demonstrating something in the interface, or in expert-novice situations.

The MAUI toolkit includes groupware versions of standard UI widgets, and also provides specialized groupware components such as telepointers and participant lists. In addition to the widget set itself, MAUI introduces several concepts that have not been seen before in groupware toolkits:

1. A focus on standard widgets as a legitimate part of the 'shared workspace' where awareness information is important for helping collaborators understand each others' activities.
2. The idea of providing both single-state and multi-state versions of standard UI widgets, so that users can either have a shared data model or replicated models with appropriate visualizations of the multiple states.
3. The idea of run-time customization for a component's visual awareness effects (e.g., changing the visual strength of a highlight effect), which can be controlled either by the user or by the application program in response to changing group conditions.
4. The inclusion of 'black-box' network and session-management modules that enable rapid prototyping and testing, but that do not constrain the eventual redesign of these modules.
5. The packaging of multi-user components in the portable JavaBeans format (Sun Microsystems, 1997), which enables wide reuse and allows the widgets to be used in a number of standard integrated development environments (IDEs).

With the MAUI toolkit, developers can produce flexible awareness-enhanced groupware interfaces in far less time than was previously possible. MAUI is

the first toolkit that provides for simple and fast construction of true group interfaces. In this paper, we describe the parts of the MAUI toolkit, demonstrate the awareness displays of several components, provide an example of how a groupware interface is built, and discuss our experiences using and evaluating the toolkit over the past year. We begin by describing the types of awareness information that are used in the MAUI components, and by reviewing the interface support that is provided by existing groupware toolkits.

2. Awareness and feedthrough

Our goal is to help synchronous distributed collaborators maintain awareness of the other people in the group. One main way that the toolkit supports awareness is by distributing feedthrough information – the visual feedback that guides a local user through the operation of a component. In this section we review these two concepts.

2.1. GROUP AWARENESS

Group awareness is “an understanding of the activities of others...that provides a context for your own activity” (Dourish and Bellotti, 1992, p. 107). Maintaining awareness of others is an important part of collaborative activity, and support for awareness in groupware interfaces has been shown to improve groupware usability (Gutwin et al., 1996; Gutwin and Greenberg, 1998). Group awareness can be subdivided into basic questions about *who* is collaborating, *what* they are doing, and *where* they are working (Gutwin and Greenberg, 2002). When collaborators can easily gather information to answer these questions, they are better able to simplify their verbal communication, coordinate and anticipate one another’s actions, and assist one another.

In face-to-face situations, people gather awareness information in three main ways: through explicit communication, through observing the orientation and movement of another person’s body in the workspace, and through observing the effects of a person’s actions on the objects in the space. Although the MAUI toolkit provides some awareness information through both of the first two mechanisms, its primary means of supporting awareness is the third – through feedthrough.

2.2. FEEDTHROUGH

The visual objects in a shared space, either in a face-to-face or a groupware setting, can be a rich source of awareness information. When objects and

tools are used by a person, they ‘give off information’ that indicates what is being done to them. For example, switches show the movement of the lever as the switch is thrown, and a pencil makes a scratching sound as a line is drawn (Gaver et al., 1991). This information given off by objects is valuable feedback to the person performing the action – but to others nearby, the information is also *feedthrough* that provides awareness information about the person’s activities (Dix et al., 1993). In the physical world, objects produce this information naturally, but in groupware, feedthrough information must be explicitly gathered, transmitted, and redisplayed.

In a computer application, feedback (and therefore feedthrough) can be produced either by domain artifacts or by the UI components that make up the interface. Feedback produced by domain artifacts, however, is by nature domain-specific, and so is difficult to account for in a toolkit. UI widgets, in contrast, all have standard and well-known feedback mechanisms that are used to assist the local user in manipulating the widget. For example, pushbuttons change from a flat to a sunken state to confirm the press, menus highlight the command currently underneath the cursor to assist in selection, and many widgets show a highlight when the user moves over them in order to aid targeting.

Although this feedback is automatically displayed for the local user, remote users in a groupware system do not see it. However, as mentioned above, there are two reasons why UI components should be considered as sources of awareness information. First, groupware interaction and collaboration do happen outside the domain workspace, just as in the real world when collaboration occurs around a whiteboard as well as overtop of it. In particular, being able to see people’s actions and choices on the controls and tools that surround a domain workspace (e.g., palettes, toolbars, and menus) can greatly improve understanding of their intentions and plans for activity inside the workspace. Second, in many groupware applications the UI components may themselves make up the shared workspace. For example, in a form-based system, the entry fields, drop-down lists, and check boxes on the form are the domain artifacts; in a text editor, the text widget is the primary shared workspace. In these systems, standard UI components are the setting and the stage for the interaction (see Figure 1).

To support awareness in these situations, groupware components can collect, distribute, and display feedthrough information (Gutwin and Greenberg, 1998). As a simple example of how the feedthrough might look, consider a basic pushbutton in a groupware interface. When a person’s cursor moves over the button, it becomes highlighted on all user’s screens; when a person presses the button, it is shown being pressed on all screens. As feedthrough, the highlight no longer provides targeting feedback, but shows where another person is and that they may be about to press the pushbutton; similarly, the animation of the button press for a remote user is no longer

confirmation but instead shows that the action has been taken by another person.

Since a feedthrough display in groupware is completely synthetic, it is also possible to change the remote visualizations of the information. In some cases effects can be augmented to draw attention to particularly important actions; in other cases, the effects must be reduced or changed to minimize the chance of distracting the local user. For example, it would be problematic to duplicate the appearance of a dropdown menu on every person's screen, since it would occlude local work areas. Remote menus can still show feedthrough information, but they require more subtle and smaller remote representations.

3. UI issues in groupware toolkits

A number of groupware toolkits have been built in the past decade to simplify the process of developing multi-user distributed systems. Different toolkits focus on different aspects of the groupware development problem: some focus on what can be done through different programming approaches (e.g., Graham et al., 1996; Schuckmann et al., 1996), some on distribution architectures (e.g., Hill et al., 1994; Patterson et al., 1996), and others on simplicity (e.g., Roseman and Greenberg, 1996). These toolkits provide a wide range of services that are needed in groupware applications, such as network communication, session management, data replication and sharing, concurrency control, and event notification. However, although most toolkits provide some facility for building a GUI, there is almost no support for building collaboration-aware interfaces. Below, we first review the issue of group interfaces in terms of a general coupling model, and then summarize the interface components that have been seen in previous toolkits.

3.1. INTERFACE COUPLING

The degree to which groupware interfaces are coupled is intimately related to the architecture of the distributed system. All groupware systems must maintain a shared state in order to allow collaboration over a common set of data. However, the level at which state is shared can vary depending on the architecture of the system. Researchers have used the idea of application layers to specify the level of state sharing (Hill et al., 1994; Dewan and Choudhary, 1995). In Figure 2, five layers are shown, from the domain model through to the user's workstation screen. It is possible for layers to be either centralized (shown as a single box), replicated but synchronized through communication (two boxes connected by arrows), or replicated and

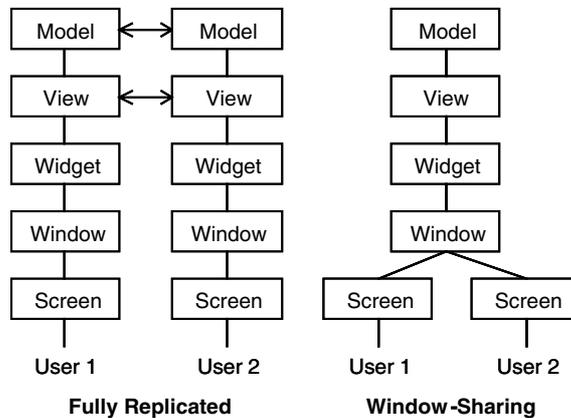


Figure 2. Sharing and synchronization at different layers of the groupware application – fully replicated at left, or window-sharing at right.

not synchronized (shown as unconnected boxes). Figure 2 shows two canonical groupware examples: a fully replicated architecture where models and views are synchronized, and a window-sharing architecture where a centralized application window is served to multiple user screens.

In this characterization of groupware architectures, we are interested in coupling the application's widget layer, since this would allow feedthrough information to be distributed. However, the only existing toolkits that currently allow this layer to be coupled are those that centralize it entirely – that is, application transparency systems (e.g., SharedX or VNC). No toolkit for replicated groupware that we have found allows generalized synchronization at the widget layer, although Dewan's generic model certainly makes this a possibility (Dewan and Choudhary, 1995). It is much more common – e.g., in Rendezvous (Hill et al., 1994), GroupKit (Roseman and Greenberg, 1996), COAST (Schuckmann et al., 1996), Clock (Graham et al., 1996), and Visual Obliq (Bharat and Brown, 1994) – to have a shared or synchronized model, with some synchronization of views, but no connection at the lower layers. Many toolkits do allow fine-grained sharing of the model elements underlying visual components (such as the String object underlying a TextEntry widget) (Schuckmann et al., 1996), but they do not synchronize the widget's appearance. One toolkit by Smith and Rodden (1993) did provide different versions of a widget for different participants in the groupware session, but the differences were based in access rights rather than awareness, and no communication was done at the widget level.

Full synchronization of the widget layer, however, is also not a perfect solution. State-sharing at this level implies that widgets replicate their exact behaviour on all screens, which may cause distraction and occlusion

(e.g., dropdown menus). Instead, we need to distribute the synchronization events, but have visual representations of those events that are appropriate for either the local or remote user.

3.2. GROUPWARE-SPECIFIC DISPLAYS AND WIDGETS

Although toolkits do not provide a generalized mechanism for communication at the widget layer, several do provide specific groupware widgets (e.g., Roseman and Greenberg, 1996; Begole et al., 1998). These are devices such as telepointers (see Figure 4), multi-user scrollbars (Baecker et al., 1993) (see Table II), participants lists (Figure 4), shared text panes (Begole et al., 1998), and 'radar' overviews (Smith et al., 1989). Of these, only the multi-user scrollbar and the shared text pane are related to existing single-user interface components; the others are either specific to a groupware context (e.g., telepointers, participant lists) or do not yet exist in standard widget sets (e.g., radar views).

In addition to these widgets, there is a large number of other custom-built awareness displays that have been seen in groupware systems. These displays provide a wide variety of information in forms that can be either situated in the artifacts or separate from it, and either literal or symbolic representations of another person's actions (Gutwin and Greenberg, 2002). Displays may include video images (Dourish and Bly, 1992), miniatures of others' screens (Gutwin, Greenberg, and Roseman, 1996), activity indicators (Sohlenkamp and Chwelos, 1994; Ackerman, 1995), or traces of past activity (Gutwin, 2002). Also, there have also been a number of architectures and protocols proposed for handling the transmission of awareness information at the infrastructure level. Although these do not provide visual representations of awareness information, they can form the basis for widget-layer devices that do have visual forms. Most of these systems are based on the idea of a notification server (Patterson et al., 1996), where awareness events are collected at the source and distributed to different participants. Three systems in common use are NSTP (Day et al., 1997), Elvin (Fitzpatrick et al., 2002), and NESSIE (Prinz, 1999). Of these, NESSIE also provides the basis for various visible or tangible representations of awareness notifications.

Among existing toolkits, GroupKit (Roseman and Greenberg, 1996) and Flexible JAMM (Begole et al., 1998) provide the widest range of groupware widgets. GroupKit supplies telepointers, a participant list, a multi-user scrollbar, and a basic radar view. Flexible JAMM provides a shared text pane, telepointers, and a radar view. These displays and widgets have been shown to be valuable in helping groups maintain awareness, and can substantially improve the usability of groupware systems (Gutwin and

Greenberg, 1998). However, there are only a few of these components in a few groupware toolkits, providing scant coverage of the range of interface components that are likely to be used in groupware systems. In addition, none of these components consider the possibilities of multi-state vs. shared-state use, run-time customization, rapid prototyping and testing, or integration with standard development tools.

The MAUI toolkit addresses these issues: it explicitly supports both single-state and multi-state versions of most widgets; it provides run-time customization that can be controlled either by the user or by the application program; it includes black-box network and session components that enable rapid prototyping and testing; and it packages components as Java Beans, which allows integration with standard IDEs.

4. The MAUI toolkit

The MAUI toolkit is made up of five main parts: a component set, an event model, a run-time customization facility, and detachable communications infrastructure and participant manager. The toolkit is based on Java, on the AWT event system, and on Swing (Figure 3).

4.1. MAUI INTERFACE COMPONENTS

The core of the MAUI toolkit is a set of awareness-enhanced UI components (see Table II). These are visual widgets that can be added to a groupware interface through an application builder, and that collect, distribute, and visualize group awareness information. The components can be divided into two groups: groupware versions of existing Swing widgets, and groupware-specific components that do not appear in single-user toolkits.

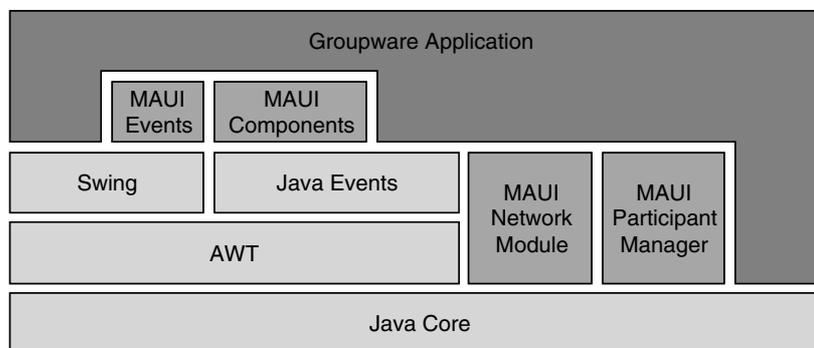


Figure 3. Integration of the MAUI toolkit with Java and with application programs.

4.2. COMPONENTS DERIVED FROM SWING WIDGETS

Java provides a large single-user component set commonly known as Swing (Walrath and Campione, 1999). There are four main types of visual widgets that support user interaction, as shown in Table I. Of these widgets, many are good candidates for gathering and providing some type of awareness information. In the MAUI toolkit, we have built groupware versions of widgets in each of the Swing categories in Table I.

For each Swing widget to be extended, we go through the following steps:

- determine whether the component can have both shared and multi-user forms;
- determine what elements of awareness and feedthrough that the component should collect and display;
- determine which AWT events to trap in order to gather the awareness information;
- extend the Swing class (from the *JComponents* in Swing, we create *GComponents* in the MAUI toolkit);
- add listener objects and adapters to trap AWT events and generate MAUI awareness events;
- add custom painting code to visualize the awareness information when an awareness event is received.

The first of these steps – determining whether components can be both shared and multi-user – means that many of the widgets actually have two distinct forms. A *shared* component has a single state for all members of the group, and manipulation by any person changes the state of the component for everyone. Simple widgets like buttons, combo boxes, menus, and text fields have only this form; it also makes the most sense for some complex components like text windows. For other widgets, however, it is reasonable to allow multi-user operation, where each person puts the component into a different state. Scrollbars are an example of a widget that can be used in

Table I. Swing categories, swing widgets, and widgets implemented in the MAUI toolkit

Category	Swing components	MAUI components
Top-level containers	Applet, dialog, frame	Dialog, frame
General-purpose containers	Panel, scroll pane, split pane, tab pane	Scroll pane, tab pane
Basic controls	Buttons, combo box, list, menu, slider, text field	All
Editable displays	File chooser, table, text, tree view	Table

either shared or multi-user forms; others include lists, sliders, tab panes, and (perhaps) tree views. When each user can be in a different state, visualizing awareness information becomes much more complex, since the states of all users must be shown in the component. Multi-user widgets show different types of awareness information as well – e.g., location awareness in a multi-user scrollbar.

A visual index to the MAUI components is given in Table II, showing their visual effects and the types of awareness they support (a video demonstration is available at hci.usask.ca/projects/maui/). Note that we do not include the GFrame in the index, since it is used primarily for administration

Table II. A visual index to the MAUI components

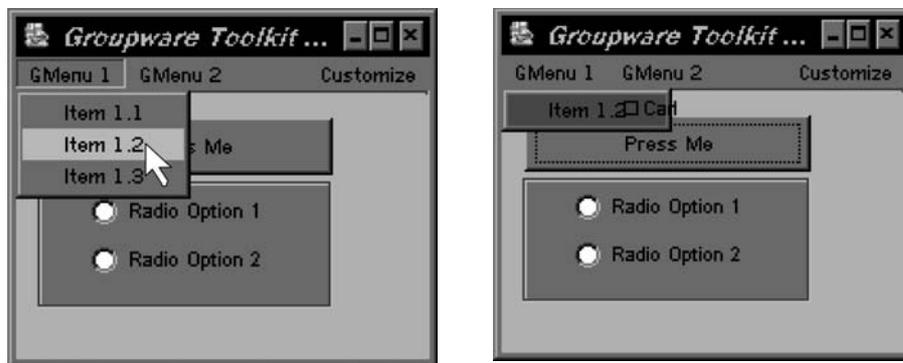
Local view	Remote view

GButtons have only a shared-state form. They visualize intention awareness information (based on enter and leave events) using either a border or background highlight, and action awareness information by animating the button press on the remote screen. Other button types (GCheckBox, GRadioButton, GToggleButton) are not shown but behave similarly.

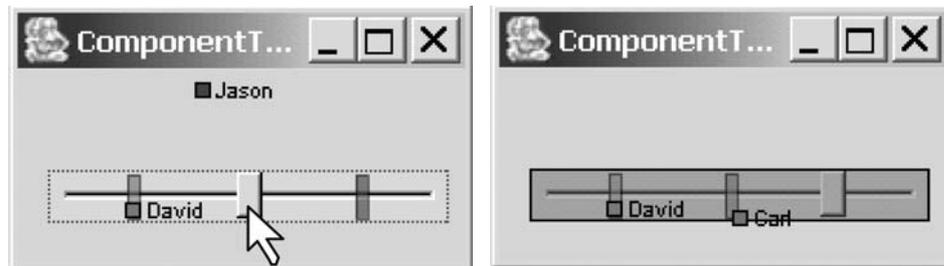
Table II. Continued

Local view	Remote view
------------	-------------

GMenus have only shared versions, but have two different remote representations. The first is the transparent menu. A custom popup is rendered onto the GlassPane of the GFrame (since normal popups cannot be made transparent). The transparent menu does not trap events, so components under the popup can be manipulated normally with the mouse. The highlight colour shows which user is manipulating the menu.



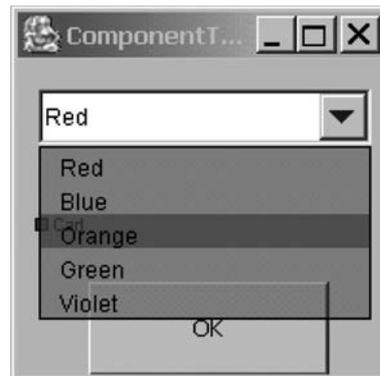
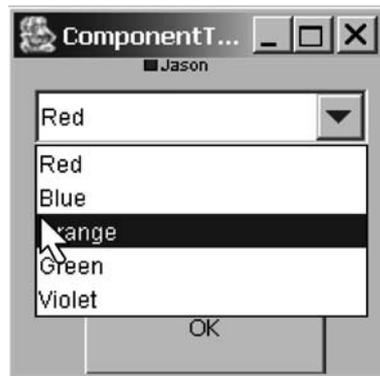
GMenus also have a summary representation, where only a small popup is rendered to show the current item under the other user's cursor. This representation shows less process information, but takes less space than the transparent menu.



GSliders have both shared and multi-user forms. In the multi-user form (shown here), the position of each person's slider knob is shown as a coloured overlay. In the example above, three users have moved their sliders to three different positions. Entry and exit information can also be shown either with a border effect or with a transparent highlight.

Table II. Continued

Local view	Remote view
------------	-------------



GComboBoxes have only a shared version, but can show their dropdown list in either a transparent (shown here) or a summary representation. These representations are displayed in the same way as for GMenus – by custom rendering to the GlassPane of the GFrame.



GTextFields use a combination of shared and multi-user properties. The text is shared, but selections in that text are multi-user, and are shown as coloured transparent overlays.

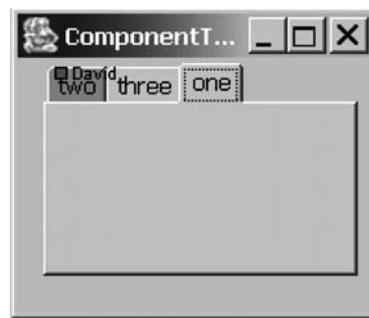
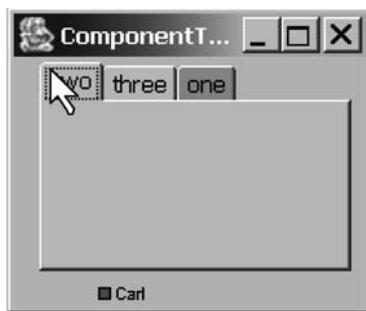


GscrollPanes have both shared and multi-user forms. In shared form, any user's manipulations change the scroll position for all participants; the scroll bar shows feedthrough of the arrow-button presses and scroll thumb movement. In multi-user form (shown here) the widget shows location awareness through a highlight bar corresponding to

Table II. Continued

Local view	Remote view
------------	-------------

each user's local scroll thumb, and can also show each user's viewport as a transparent rectangle inside the pane. When there are more than four other users, the scrollbar expands to fit all of the highlight bars. In its multi-user form, the component is equivalent to the multi-user scroll bar seen in other toolkits (e.g., Roseman and Greenberg, 1996). The `GScrollPane` is also used as a subcomponent in `GLists` and `GTables`.

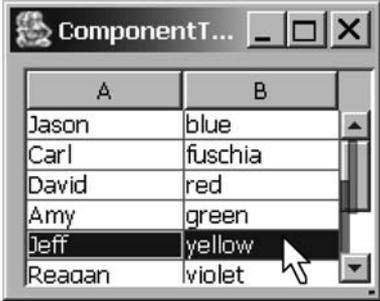
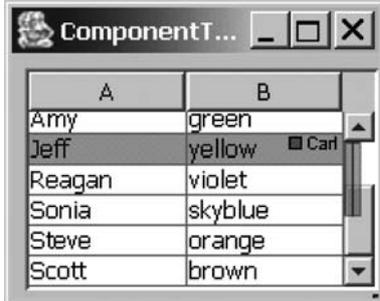


`GTabbedPanels` have both shared and multi-user forms. In shared form, any tab click changes the tab for all participants. In multi-user form (shown here), each user's current tab is shown highlighted in their colour.



`GLists` have shared or multi-user forms; the sharing relates to the selection. In shared form, any user's selection is duplicated as the selection on all clients. In multi-user form (shown), selections are shown as transparent highlights. When more than one person selects the same list item, the highlight space is divided between them. Above, three people have chosen the item "fourteen."

Table II. Continued

Local view	Remote view																												
 <table border="1"> <thead> <tr> <th>A</th> <th>B</th> </tr> </thead> <tbody> <tr> <td>Jason</td> <td>blue</td> </tr> <tr> <td>Carl</td> <td>fuchsia</td> </tr> <tr> <td>David</td> <td>red</td> </tr> <tr> <td>Amy</td> <td>green</td> </tr> <tr> <td>Jeff</td> <td>yellow</td> </tr> <tr> <td>Reagan</td> <td>violet</td> </tr> </tbody> </table>	A	B	Jason	blue	Carl	fuchsia	David	red	Amy	green	Jeff	yellow	Reagan	violet	 <table border="1"> <thead> <tr> <th>A</th> <th>B</th> </tr> </thead> <tbody> <tr> <td>Amy</td> <td>green</td> </tr> <tr> <td>Jeff</td> <td>yellow</td> </tr> <tr> <td>Reagan</td> <td>violet</td> </tr> <tr> <td>Sonia</td> <td>skyblue</td> </tr> <tr> <td>Steve</td> <td>orange</td> </tr> <tr> <td>Scott</td> <td>brown</td> </tr> </tbody> </table>	A	B	Amy	green	Jeff	yellow	Reagan	violet	Sonia	skyblue	Steve	orange	Scott	brown
A	B																												
Jason	blue																												
Carl	fuchsia																												
David	red																												
Amy	green																												
Jeff	yellow																												
Reagan	violet																												
A	B																												
Amy	green																												
Jeff	yellow																												
Reagan	violet																												
Sonia	skyblue																												
Steve	orange																												
Scott	brown																												

GTables also provide shared-state data with multi-user selections. Future versions of the GTable will allow more multi-user flexibility (such as independent column sorting).

and for rendering other components (e.g., telepointers and transparent popup menus). In addition, we do not include the shared versions of some widgets, since the remote view looks the same as the local view.

4.3. GROUPWARE-SPECIFIC COMPONENTS

In addition to the extended Swing widgets, the MAUI toolkit provides several groupware-specific devices that have been seen in other groupware systems and toolkits. In particular, the toolkit includes components for telepointers, participant lists, and a chat tool. These components show a variety of awareness information including who is in the session, where they are working, and how active they are.

Telepointers. Telepointers (Figure 4) can be added to any application by turning them on in the design-time customizer available in the IDE (see Figure 8). Telepointers have several design-time options. There are three styles of telepointer available (arrow, block, or configurable image), and participant names can be added to the pointer representation. Telepointers can also leave fading trails (as in Figure 4), which can assist people in interpreting gestural communication, particularly in jittery networks. Telepointers are implemented on the top-level component, the *GFrame*. To support telepointers, *GFrames* contain listener objects to trap and distribute mouse movement events. Display is handled by rendering the telepointers on the *GFrame*'s *GlassPane* (a transparent overlay pane). Using the top-level component to show telepointers is valuable since it is possible to show pointers in the entire window, not just in the panel of a custom shared

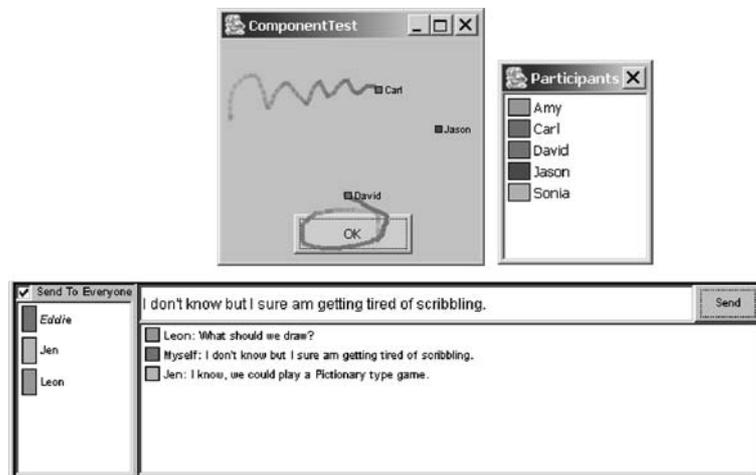


Figure 4. MAUI telepointers, participant list, and chat tool.

workspace. This means that users will get two sources of information about a person's activities and intentions in the interface – the feedthrough information provided by the widgets, and the embodiment information provided by the telepointer. However, the top-level approach does require extra calculation when the telepointer is over a scroll pane, since the pointer must be correctly registered to the position of the scrolled panel.

Participant list. The MAUI participant list is a simple component that shows the names and colours of all connected users (see Figure 4). This component does not need to listen to any standard AWT events, and only deals with connection and disconnection events that are handled by the participant manager (described below). The participant list can be used either as a top-level dialog or as a panel inside a GFrame.

Chat tool. This tool is a compound component with several subpanes (Figure 4). It is an example of how complex (but common) groupware functionality can be encapsulated as a widget. The chat tool allows messages to be directed to specific participants or broadcast to all. To send a message, a user types into the entry box and presses the send button; when messages are received, they are displayed in the scrolling text area at the bottom of the tool. Each new message is shown with the color and name of the user that sent the message. The scrollbar of the text area (when visible) is a GScrollbar, which helps people determine where others are looking and thus whether they will be able to see a new message.

4.4. EVENT MODEL

MAUI components gather and distribute awareness information through events. The MAUI event model has facilities for capturing incoming AWT

events from the local user, for generating MAUI-specific awareness events for distribution, and for handling awareness events that have arrived from another machine.

4.4.1. Capturing AWT and Swing User Events

Components gather awareness and feedthrough information by collecting two types of user events at the local machine. First, there are events that imply *intention* with a widget – for example, moving the mouse over a pushbutton suggests the intention of pressing the button. Second, there are *action* events that generally lead to callbacks – for example, actually pressing the button to execute its functionality. Both types are feedthrough, but action events are widget-specific, whereas intention events are more generic and can be reused for several widget types.

To handle these two types of awareness events, MAUI components include two types of listeners: a widget-specific listener built into the GComponent to handle action events, and a MAUI adapter (a BasicAwarenessAdapter) to handle intention events. For example, a GButton gets MouseEntered and MouseExited (intention) events through a BasicAwarenessAdapter, and gets ActionPerformed (action) events through an ActionListener (see Figure 5).

Collecting these events, however, does not mean that they cannot be used by the application programmer: the Java Listener model notifies any object that registers interest, so the events are also sent to any listeners that the application programmer has set up. Therefore, MAUI components also behave ‘normally’ from the programmer’s perspective. Note, however, that the MAUI toolkit does not handle any application semantics – so what happens in the application when the button is pressed is up to the developer.

4.4.2. Creating and Distributing Awareness Events

Whenever a MAUI component receives an AWT event that implies that awareness information should be distributed, it creates a MAUI event called a GControlEvent. We do not send the original AWT events, since they are generally large and contain information that is not needed at the remote machines. Instead, we extract the necessary data elements and repackage them in a smaller object more suitable for sending across the network. There are several types of MAUI events (see Figure 6): components generate wid-

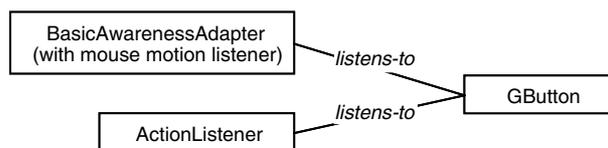


Figure 5. Collecting user events for a GButton.

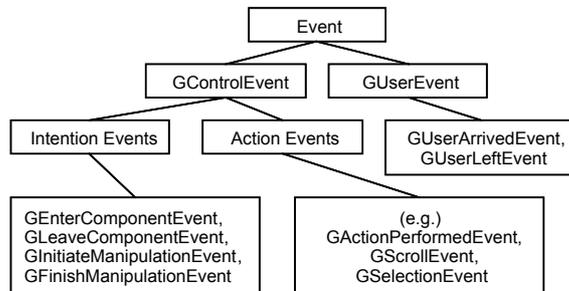


Figure 6. MAUI event hierarchy.

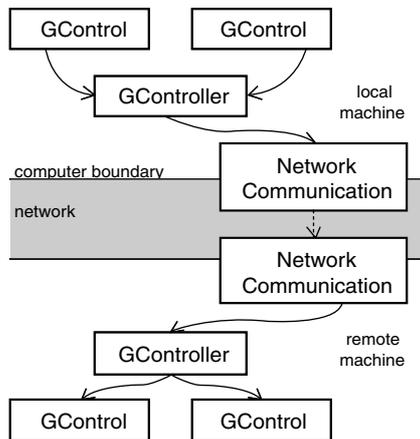


Figure 7. Information flow through MAUI components in a groupware application.

get-specific events and generic awareness events. There are also user events created by the participant manager.

Distribution of events happens as follows. Each MAUI component has previously registered with a `GController`, and this controller listens for the creation of `GControlEvents`. The controller forwards them to a `Dispatcher` object, which acts as a façade on the network communication layer. Once through the network, the event is received by another `Dispatcher` who directs the event to the appropriate `GController` for the UI component matching the originator of the event. The distribution process is illustrated in Figure 7.

4.4.3. Handling Awareness Events at the Remote Component

Components receive remote events from their `GController` by implementing the `GControllerListener` interface. This interface specifies the `gActionPerformed` method, into which is passed the `GControlEvent` object. At this point the event is inspected and rerouted according to its actual type.

The eventual result of receiving an awareness event is that some type of awareness visualization must be created or updated. This rendering may be simple or complex. For example, showing that another user has moved their mouse over a button involves changing the button's border colour or drawing a transparent highlight on the button (depending on which effect the designer has chosen). In some cases, the original Swing classes contain code to animate the widget to show manipulation (such as the `doClick()` method of `JButton`).

However, other widgets require more specialized rendering that must be custom-built. Menus and combo boxes, for example, require special treatment to show either the transparent or the summary representations of the component (see menu examples in Table II). In these cases, custom renderers are incorporated into the `GComponent`. We define two for menus, `TransparentPopupRenderer` and `SummaryPopupRenderer`.

4.5. DESIGN-TIME AND RUN-TIME CUSTOMIZATION

MAUI components allow both design-time and run-time customization. Design-time customizers are part of the JavaBean standard, and allow the developer to choose whether different types of awareness information will be shown, and if so, using which type of effect. Figure 8 shows the customizer for a `GFrame`, which allows the developer to easily add and configure telepointers and a collaboration menus.

Run-time customization, however, is separate from the JavaBean standard. MAUI includes these capabilities so that the user or the application can easily control the awareness visualizations. One of the dangers of supporting awareness is that the additional visual information can distract users when they are concentrating on individual rather than group tasks. The MAUI run-time customizers allow visualization effects to be changed, 'turned down,' or switched off entirely.

MAUI supports both widget-level customization and application-level defaults. Customizable properties for each component type are stored in static class dictionaries, and the `GFrame` stores another for the application properties. Whenever the component renders itself on the screen, it consults these properties to determine what effects to show and how 'heavy' to make them (e.g., transparency level). If customization is allowed in the application (this can be set as a design-time switch), a 'Customization' menu is added to the application menu bar. At application startup, the `GFrame` is inspected for customizable components, and these types are added to the menu. Selecting a menu item brings up a customization dialog that allows adjustment of the properties in the dictionary. An example customization dialog is shown in Figure 8.

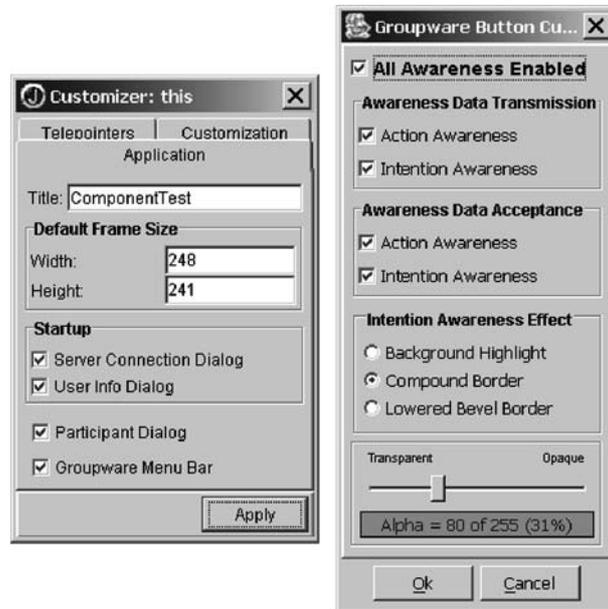


Figure 8. Customization dialogs for a GFrame (left, design-time) and for a GButton (right, run-time).

Certain properties are specific to particular widgets, but many others are common to multiple widgets. These can be set and manipulated as application-level properties that affect all widgets in the main frame. Using the application-level customization dialog, a user can globally turn up or turn down the visualizations in all components.

In addition to user changes, the customization system provides a simple interface for programmatic control of the visualizations. For example, an application programmer could easily tie the visual weight of awareness information to dynamic factors such as another person's proximity in the application (i.e., the closer someone is, the more obvious their activities become).

4.6. NETWORK COMMUNICATION INFRASTRUCTURE

MAUI components require some form of network communication to distribute awareness events. However, any groupware application that uses the MAUI toolkit for its interface will also require its own network layer. We did not wish to duplicate the network services in a single application, and we did not want to force application developers into a particular communications architecture. Therefore, we designed the MAUI toolkit to have a clear separation between network communication and toolkit behaviour, and to be able to easily switch between different network modules. Only one class in the

toolkit – the GController – communicates with the network infrastructure, and through only one method (`sendToOthers(GEvent g)`). This means that switching to a new communication system can be easily done when the developer builds an application.

The toolkit does contain a basic network package, a server-based implementation that uses Java Remote Method Invocation (RMI). An available default package is valuable because it allows developers to build and test real distributed interfaces without needing to design the communications architecture for the full system. Although the default system may be sufficient for small applications, we recognize that RMI is too heavyweight for sending the type of awareness messages that make up most of MAUI's communication. In particular, RMI sends all messages as using TCP transport; performance is sufficient on a local area network, but to enable testing across wider area networks, we plan to add a lighter default that uses UDP.

4.7. PARTICIPANT AND SESSION MANAGEMENT

Keeping track of information about the participants in a groupware session – such as their names and highlight colours – is another facility required by MAUI components that will also be needed by the overall groupware system. Again, we wanted to avoid restricting a developer's choice of participant management, and so the toolkit is designed to work with any session management module, but provides a default implementation for design and testing. The default module is automatically started by the GFrame application class, and is very simple: it provides dialog interfaces for making a network connection to a server, and for recording people's names and highlight colours. These dialogs are presented whenever an application starts (see Figure 9).

Switching to a new session management module requires two steps. First, we have created a Java interface (GParticipant) that specifies 'get' methods for the properties needed by MAUI components. Whatever class is eventually created to represent participants in the groupware system can be used with the MAUI toolkit as long as it implements this interface. Second, if the participant list widget is to be used, the participant management system must send

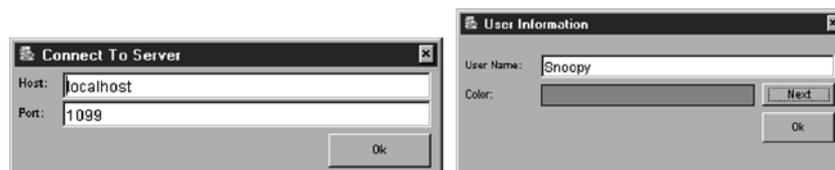


Figure 9. Default session manager's connection and user information dialogs.

GUserEvents (defined in the toolkit) to the GController, so that the widget can determine when people join and leave the groupware session.

5. A developer's view of MAUI

To illustrate the use of the MAUI toolkit, we present an example of how a developer would construct a simple group interface (Figure 10) in JBuilder, a popular Java integrated development environment (although any Beans-compliant IDE will work).

To use the toolkit, the developer first adds it to the project's library path, and then creates a new tab in the UI designer to show the groupware widgets (Figure 10). To start building the UI, the developer then creates a new class based on a GFrame; this places a new frame into the designer window. The group interface can then be built by dragging MAUI components from the palette onto the GFrame. A property list editor (at right in Figure 10) allows changes to both the awareness properties and the normal inherited properties of the widget. If other types of components are needed in the interface besides MAUI components (such as ordinary Swing widgets), they can be added from other tabs of the designer. Finally, test values for widgets such as lists and combo boxes are set by adding a few lines of code in the editor pane of the IDE.

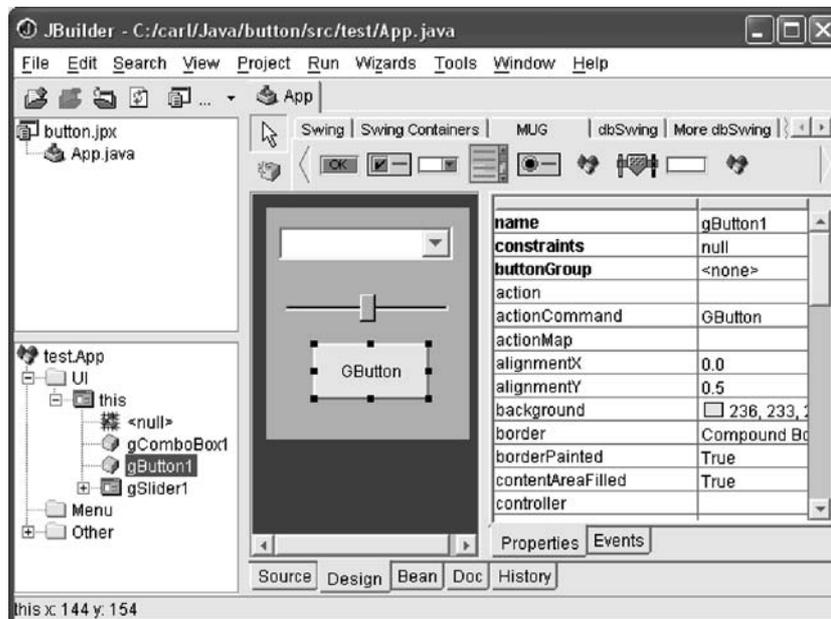


Figure 10. JBuilder IDE, showing UI designer, MAUI widget tab, and property list for selected GButton.

If the developer does not wish to make other modifications, the toolkit takes care of several remaining issues:

- any unset properties will be set to default values;
- unique IDs will be given to all groupware components;
- all `GComponents` will be assigned to a default `GController`;
- a default `GroupwareClient` class containing a `main()` method will be created so clients can be run;
- participant and server information will be requested automatically when clients are started.

To test the interface, the developer starts the MAUI black-box server, and then opens any number of clients from the IDE. As each client starts, a dialog will ask for the server address, the participant's name, and their choice of highlight colour. Once two clients are running, the groupware capabilities will be fully functional and ready for testing. A video of MAUI development can be seen at hci.usask.ca/projects/maui/.

6. Evaluation

MAUI has been used and tested in a variety of situations for approximately one year: within our own lab to develop groupware tools, with a limited external user community, and as part of a graduate course in groupware. Here we discuss our evaluation of the toolkit and the lessons that we have learned from our experiences. Three types of evaluation have been carried out: tests to determine the effort saved by the toolkit, its performance, its flexibility, and its extensibility; a usability analysis to look for usage problems from the developer's perspective; and end-user evaluations of the feed-through widgets themselves.

6.1. EFFORT

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 a sample application that was built in three different ways: in Java without the use of a groupware toolkit, with the Groupkit toolkit (Roseman and Greenberg, 1996), and with the MAUI toolkit. For each approach, we recorded the length of time to build and test each application, and the total number of lines of code written.

The test application was a simple forms-based program with buttons and text fields. The visual components were chosen such that implementation would be possible in all three paradigms (particularly Groupkit, which does

Table III. Development effort statistics for test application

	Java (no toolkit)	GroupKit	MAUI
Classes or functions	14 Classes	9 Procs	1 Class
Total lines written	670	69	0 (44 added by GUI builder)
Lines written for feedthrough	62	18	0
Development time	4.5 hours	2 hours	15 min

not provide arbitrary control over widget appearance), and as such the application was a 'lowest-common-denominator' for the three approaches. (see Table III).

This test shows that MAUI does reduce development effort for groupware interfaces, even in comparison to a toolkit which is designed for simplicity (i.e., Groupkit). The success of MAUI in this test is not surprising, given that this was the intent of the toolkit; however, it is also worth noting that for some of the widgets in MAUI's widget set, a toolkit like Groupkit could not compete at all without writing C code (since the painting of TCL/Tk widgets can normally only be controlled through the existing, and limited, widget API).

6.2. PERFORMANCE

This evaluation analyzed MAUI's performance in terms of 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 applications perform nearly as well as a custom groupware application. The MAUI controller-dispatcher mechanism adds very little overhead to the message processing, since this mechanism serves mainly as a message router. If a customized architecture was built, performance might be enhanced by linking components directly to each other via direct socket connections. However, the small performance penalty to make the mechanism generic seems to be worth the tradeoff in most cases.

Message processing in MAUI performs well, although there are again some aspects that are slightly slower due to the general approach. For example, message sizes could be slightly reduced if the messages were completely specialized to their task. MAUI generalizes the messages slightly by encoded 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 toolkit.

6.3. FLEXIBILITY

To test the ability of the toolkit to support a wide range of groupware application paradigms, we built four different applications that are all commonly-seen groupware examples. The applications were: a shared whiteboard, a shared web browser (shown in Figure 11), a form tool for jointly applying for a loan (Figure 12), and a discussion board.

The experience of building these four applications showed that MAUI is flexible enough to support a variety of groupware application paradigms. The flexibility of MAUI is rooted in the basic capabilities of the underlying Swing toolkit; since our goal was to add transparent groupware functionality to this existing layer, flexibility of the underlying tool is not reduced substantially.

However, the process of building the test programs did identify a set of limitations to applications built with MAUI; although these do not present serious problems, they should be understood by developers:

- MAUI uses a glass pane to implement menu transparency and telepointers, so an application developer cannot use their own glass pane in a MAUI application – they must add their code to the MAUI component.

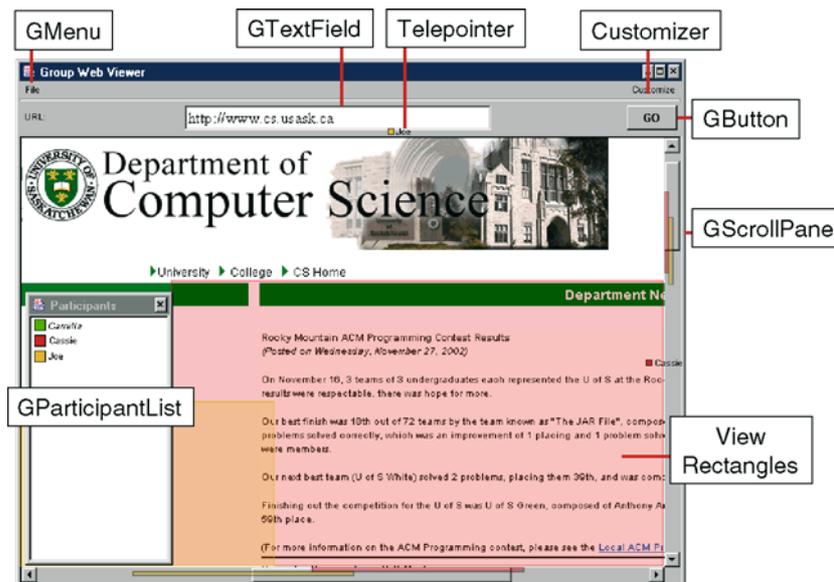


Figure 11. Example shared browser built with MAUI.

Figure 12. Example forms-based application built with MAUI.

- 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.
- Transparency effects cannot be shown outside the bounds of the glass pane. Therefore, if widgets such as (long) menus need to be drawn outside the window bounds they cannot make use of transparency effects.
- Awareness events do not currently have priorities associated with them, and messages are always processed on a first-come-first-served basis.
- The runtime customization of widgets occurs for all widgets of a particular type rather than an individual widget.

6.4. EXTENSIBILITY

We analysed MAUI to consider how easy it is for developers to extend the toolkit – to add new widgets, extend existing widgets, and extend message types. All of these extensions are possible, and should present little problem to a Java developer. In most cases, once the basic architecture of MAUI

communication is understood, the effort needed to integrate new work into the toolkit will be relatively small.

Any new widget can be added to the toolkit as long as it implements 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. Implementing the `GroupwareControl` interface provides the widget with the capability to communicate with the networking infrastructure; however, the widget must still implement message construction, incoming-message processing, and awareness visualization. There are several pre-existing message types that are used by the existing widgets, and these can be used in the creation of new widgets. To simplify new implementations, the `BasicAwarenessAdapter` can be used to provide visualization techniques and standard implementations for communication infrastructure connectivity. Finally, new widgets can take advantage of existing design-time and run-time customization facilities. If the developer follows the JavaBeans guidelines, a new widget will automatically support design-time customization (through the IDE) and can also make use of the existing MAUI run-time customization facilities.

6.5. USABILITY FOR DEVELOPERS

Informal usability testing was done with six students who used MAUI for an assignment in a graduate CSCW class. These students were not experienced groupware designers or developers, although they all had extensive single-user programming experience. The students built a basic groupware application using JBuilder and MAUI; one student also built a more complex system as a second test.

In general, the toolkit was seen as being simple to understand and easy to use. All of the students were able to complete the basic application with minimal assistance, and it was clear that people did not have to spend extra effort on the groupware parts of their application code. People liked the integration of MAUI widgets with the IDE, and reported that it was no more difficult to add groupware widgets to an application than it was to add ordinary Swing components. They also liked being able to set application properties in the design-time customizers. However, there were also a number of minor problems with the setup of the toolkit and the way that MAUI applications can be built.

- *Configuration is not as simple as it should be.* Importing the MAUI library into a JBuilder project, making sure that RMI was installed correctly, and setting up the IDE to show the MAUI widgets on a palette caused some students initial problems. Although these are partly problems of the IDE, and are problems that usually only have to be solved

once, they take away from the goal of further simplifying groupware development.

- *Telepointers and scroll panes cause problems.* Some of the developers realized a problem that exists when telepointers are used in conjunction with scroll panes: when a user scrolls to a different area, their telepointer location on others' screens must be adjusted by the scroll amount. However, this means that the telepointer can disappear when a user is in a scroll region not visible to another user, and that telepointers will make odd jumps as a user moves from the scroll pane to other areas of the interface. Unfortunately, we do not see any solution to this problem, other than perhaps animating the cursor jump to prevent it becoming lost from view.
- *Resizing application windows causes problems.* A similar telepointer problem exists if users are allowed to resize their windows: at the moment, only the literal cursor position is distributed, and so if widgets are different sizes, the telepointer will not appear to be in the correct place (note, however, that widget feedthrough will still work correctly). Again, this is a basic problem of all groupware systems; it could possibly be solved by recalculating telepointer positions based on the geometry of the source and local windows.
- *Multi-user widgets and changing contents.* Although the student developers liked the ability to use either shared-state or multi-user widgets, one person ran into problems when building a system where the contents of listboxes could be changed with filters. If multi-user widgets are used, changing the contents of a list results in a fundamental discontinuity between the interfaces – no longer does the multi-user scrollbar correctly indicate where the other person is or what they can see. Although it is possible to only use shared-state components, this removes the ability to support independent exploration.

6.6. USABILITY FOR END USERS

The awareness information provided by the MAUI widgets is the reason why the toolkit was built. Although MAUI is as much an infrastructure for providing awareness visualizations as it is the visualizations themselves, we are interested in whether the overall idea of feedthrough and the specific ways that we show feedthrough information are useful to end users. Therefore, we have collected user feedback about the awareness visualizations over the past year. Our sources include: an evaluation with a focus group of four experienced groupware users who were given the sample applications described above and asked to assess the awareness information and visualizations; student experiences where MAUI applications were used in classes, and comments sent to us by people who have downloaded and used the toolkit.

First, all of the feedback about the general idea of widget feedthrough has been positive. Users are almost always surprised and interested in the new widgets when they see them, and people are able to use the widgets without any training. Although part of the enthusiasm may be due to the novelty of the components, people regularly report that they like the idea of extending the shared workspace to include the entire application interface. Toolkit performance has not presented major problems, and the awareness information was seen to be provided in a timely fashion. Several people have suggested situations where the widgets would have assisted them: for example, one person recounted a story where their work was mistakenly deleted by another user, something that could have been prevented if they had been able to see the other person's actions in a menu before the selection was made; another person stated that multi-user widgets would have been useful in a situation where they tried to explain how to carry out a particular task with the interface.

The design and effectiveness of individual widgets and awareness visualizations has received both positive and negative feedback, although in all cases the reviews have suggested fairly minor revisions rather than fundamental changes. A number of users have provided comments about how the visualizations for particular widgets could be presented differently in certain circumstances and for certain tasks. For example, several people stated that the effects were 'too obvious' for their liking. MAUI does provide some facilities for changing the appearance of the visualizations through the runtime and design-time customizers (such as turning various effects on and off, or changing the visual weight of the effects), and most of these people agreed that the ability of 'turn down' the effects in the customizers provided a sufficient solution. It is clear, however, that reducing distraction must be considered further in the toolkit: one person suggested that a user setting to reduce or remove feedthrough around the local mouse cursor could be one way to reduce distraction.

Users suggested that although the awareness displays would be sufficient in many cases, there is still a need for custom-designed visualizations for specific applications. At present, changing the visualizations requires that a developer extend one of the MAUI widgets (which is not difficult), but it may also be possible to pre-package different effects, and allow developers and users to select different 'themes' of awareness representations.

In summary, our evaluations show that overall, MAUI is successful in meeting its goals: it provides a set of effective awareness widgets that are easy to understand and easy to use; it greatly simplifies the development and testing of groupware interfaces; and it is flexible enough to support a variety of groupware application types. Although there are issues and directions to be considered in future research, MAUI shows that true groupware interfaces are possible, feasible, and flexible.

7. Future work

There are several areas where we are continuing work on MAUI. These include enhancements to the component set, provision of hooks for easier extension, experiments with porting single-user applications, and addition of new generic awareness support tools.

One ongoing area of work involves enhancements in the component set. We are refining the existing awareness visualizations and are adding to the types of effects that can be produced by different widgets. For example, we are adding audio support to experiment with the effectiveness of sound feedthrough (e.g., Brewster et al., 1994). We are also expanding both the standard and the groupware-specific parts of the component set. Complex Swing widgets such as the text box and the tree view will first be built as shared-model components with multi-user highlighting, with full multi-user editing to come later. In addition, we are building several new groupware-specific components including a radar view and an extension to telepointer traces that will allow more permanent gestural annotations.

A second goal in our future work is to develop a set of simpler hooks for developers to add new components, change existing awareness effects, or add new effects. Currently, adding a new component involves implementing a series of MAUI-specific interfaces and adding code for listeners and adapters (most of which can be cut and pasted). This process is not difficult, but we wish to simplify it further to reduce developer effort.

Third, we are experimenting with the toolkit as a way to assist the conversion of existing single-user Java applications to groupware. Single-user widget classes can be swapped at run-time for groupware equivalents, as is done in Flexible JAMM (Begole et al., 1998). Although this will not completely convert the application (since MAUI handles only interface semantics, not application semantics), it would be an easy way to translate one part of the system.

Finally, we are interested in designing a set of more generic awareness tools, to simplify the process of supporting group awareness in objects that are not UI components. As discussed earlier, the collection and display of awareness information for domain artifacts in custom-built workspaces is difficult to generalize since different artifacts vary widely in their structures and behaviours. However, there are a number of common elements in the awareness support code of many different domain-specific workspaces, such as awareness events, participant information, visual highlights, traces, and fading. We plan to add a set of low-level awareness services to the MAUI toolkit that correspond to these common elements. This will provide developers with a set of building blocks for adding awareness support to any object, and will extend the coverage of the toolkit to all parts of a groupware interface.

8. Conclusion

Group awareness is an important part of collaborative activity, but awareness support has traditionally been difficult to add to groupware interfaces. To address this problem, we built the MAUI toolkit, a JavaBean based toolkit that provides a wide variety of both standard and groupware-specific interface components. MAUI contains several new concepts not seen previously in groupware toolkits: a focus on standard widgets as a legitimate shared workspace; the idea of providing both single-state and multi-state versions of UI widgets; the idea of run-time customization for a awareness visualization; and the packaging of multi-user widgets in a form that enables wide reuse.

The MAUI toolkit provides the first set of true groupware widgets; it substantially reduces the effort required to build collaboration-aware group interfaces, and integrates smoothly with popular development environments. Using MAUI, groupware developers are able to quickly build and test true awareness-enhanced group interfaces.

References

- Ackerman, M. and B. Starr (1995): Social Activity Indicators: Interface Components for CSCW Systems, In *UIST'95. Proceedings of the Symposium on User Interface Software and Technology*. New York: ACM Press, pp. 159–168.
- Baecker, R., D. Nastos, I. Posner and K. Mawby (1993): The User-Centred Iterative Design of Collaborative Writing. In *INTERCHI'93. Proceedings of the Conference on Human Factors in Computing Systems*. New York: ACM Press, pp. 399–405.
- Banavar, G., S. Doddapeneni, K. Millar and B. Mukherjee (1998): Rapidly Building Synchronous Collaborative Applications By Direct Manipulation. In *CSCW'98. Proceedings of the Conference on Computer-Supported Cooperative Work*. 1998. New York: ACM Press, pp. 139–148.
- Begole, J., M. Rosson and C. Shaffer (1998): Supporting Worker Independence in Collaboration Transparency. In *UIST'98. Proceedings of the Symposium on User Interface Software and Technology*. New York: ACM Press, pp. 133–142.
- Bharat, K. and M. Brown (1994): Building Distributed, Multi-User Applications by Direct Manipulation. In *UIST'94. Proceedings of the Symposium on User Interface Software and Technology*. New York: ACM Press, pp. 71–81.
- Brewster, S., C. Wright and A. Edwards (1994): Design and Evaluation of an Auditory-Enhanced Scrollbar. In *CHI'94. Proceedings of the Conference on Human Factors in Computing Systems*. New York: ACM Press, pp. 173–179.
- Day, M., J. Patterson and D. Mitchell (1997): The Notification Service Transfer Protocol (NSTP): Infrastructure for Synchronous Groupware. *Computer Networks*, vol. 29, nos. 8–13, pp. 905–915.
- Dewan, P. and R. Choudhary (1995): Coupling the User Interfaces of a Multiuser Program. *ACM Transactions on Computer-Human Interaction*, vol. 2, no. 1, pp. 1–39.
- Dix, A., J. Finlay, G. Abowd and R. Bealle (1993): *Human-Computer Interaction*. Hemel Hempstead: Prentice Hall.

- Dourish, P. and V. Bellotti (1992): Awareness and Coordination in Shared Workspaces. In *CSCW'92. Proceedings of the Conference on Computer-Supported Cooperative Work*. New York: ACM Press, pp. 107–114.
- Dourish, P. and S. Bly (1992): Portholes: Supporting Awareness in a Distributed Work Group. In *CHI'92. Proceedings of the Conference on Human Factors in Computing Systems*. New York: ACM Press, pp. 541–547.
- Fitzpatrick, G., S. Kaplan, T. Mansfield, D. Arnold and B. Segall (2002): Supporting Public Availability and Accessibility with Elvin: Experiences and Reflections. *Computer Supported Cooperative Work*, vol. 11, no. 3, pp. 447–474.
- Gaver, W., R. Smith and T. O'Shea (1991): Effective Sounds in Complex Systems: The ARKola Simulation. In *CHI'91. Proceedings of the Conference on Human Factors in Computing Systems*. New York: ACM Press, pp. 85–90.
- Graham, T., T. Urnes and R. Nejabi (1996): Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware. In *UIST'96. Proceedings of the Symposium on User Interface Software and Technology*. New York: ACM Press, pp. 1–10.
- Gutwin, C. and S. Greenberg (1998): Effects of Awareness Support on Groupware Usability. In *CHI'98. Proceedings of the Conference on Human Factors in Computing Systems*. New York: ACM Press, pp. 511–518.
- Gutwin, C., S. Greenberg and M. Roseman (1996): Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets and Evaluation. In *HCI'96. Proceedings of the Conference on People and Computers*. New York: Springer-Verlag, pp. 281–298.
- Gutwin, C. (2002): Traces: Visualizing the Immediate Past to Improve Group Interaction. In *GI'02. Proceedings of the Conference on Graphics Interface*. Los Angeles, CA: Morgan Kaufman, pp. 43–50.
- Gutwin, C. and S. Greenberg (2002): A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer-Supported Cooperative Work*, nos. 3–4, pp. 411–446.
- Gutwin, C. and G. Greenberg (1998): Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness. In *CSCW'98. Proceedings of the Conference on Computer-Supported Cooperative Work*. New York: ACM Press, pp. 207–216.
- Gutwin, C., M. Roseman and S. Greenberg (1996): A Usability Study of Awareness Widgets in a Shared Workspace Groupware System. In *CSCW'96. Proceedings of the Conference on Computer-Supported Cooperative Work*. New York: ACM Press, pp. 258–267.
- Hill, R., T. Brinck, S. Rohall, J. Patterson and W. Wilner (1994): The Rendezvous Architecture and Language for Constructing Multiuser Applications. *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 2, pp. 81–125.
- Patterson, J., M. Day and J. Kucan (1996): Notification Servers for Synchronous Groupware Protocols for Groupware. In *CSCW'96. Proceedings of the Conference on Computer-Supported Cooperative Work*. New York: ACM Press, pp. 122–129.
- Prinz, W. (1999): NESSIE: An Awareness Environment for Cooperative Settings. In *ECSCW'99. Proceedings of the European Conference on Computer Supported Cooperative Work*. Dordrecht: Kluwer, pp. 391–410.
- Roseman, M. and G. Greenberg (1996): Building Real Time Groupware with GroupKit, A Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 1, pp. 66–106.
- Schuckmann, C., L. Kirchner, J. Schummer and J. Haake (1996): Designing Object-Oriented Synchronous Groupware with COAST. In *CSCW'96. Proceedings of the Conference on Computer-Supported Cooperative Work*. New York: ACM Press, pp. 30–38.

- Smith, G. and T. Rodden (1993): Access as a Means of Configuring Cooperative Interfaces. In *COOCS'93. Proceedings of the Conference on Organizational Computing Systems*. New York: ACM Press, pp. 289–298.
- Smith, R., T. O'Shea, C. O'Malley, E. Scanlon and J. Taylor (1989): Preliminary Experiments With A Distributed, Multi-Media, Problem Solving Environment. In *ECSCW'89. Proceedings of the European Conference on Computer-Supported Cooperative Work*. Dordrecht: Kluwer, pp. 19–34.
- Sohlenkamp, M. and G. Chwelos (1994): Integrating Communication, Cooperation and Awareness: The DIVA Virtual Office Environment. In *CSCW'94. Proceedings of the Conference on Computer-Supported Cooperative Work*. New York: ACM Press, pp. 331–343.
- Sun Microsystems (1997): *The JavaBeans Specification*, available from java.sun.com/products/javabeans/docs/spec.html. Accessed August 2004.
- Walrath, K. and M. Campione (1999): *The JFC Swing Tutorial: A Guide to Constructing GUIs*. Boston: Addison Wesley.