

Awareness Support in a Groupware Widget Toolkit

Jason Hill and Carl Gutwin

Computer Science Department, University of Saskatchewan
Saskatoon, Canada, S7N 5A9
+1 306 966-8646

jason.hill, carl.gutwin @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 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.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Tools and Techniques—User interfaces; H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces—Computer-supported cooperative work.

General Terms

Performance, Design, Human Factors.

Keywords

Groupware interfaces, group widgets, awareness, feedthrough.

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 to 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. [1,3,14,15]).

To address this problem, we have developed a toolkit to simplify

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '00, Month 1-2, 2000, City, State.

Copyright 2000 ACM 1-58113-000-0/00/0000...\$5.00.

the construction of interfaces that support group awareness. The Multi-user Awareness UI (MAUI) toolkit 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 [6]. 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 ‘shared workspace’ where awareness information is important.
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 a portable format (i.e. Java Beans), 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, and provide an example of how a groupware interface is built using the toolkit. We begin, however, 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 and 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” ([7], 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 [10,13]. Group awareness can be subdivided into basic questions about *who* is collaborating, *what* they are doing, and *where* they are working [11]. When collaborators can easily gather information to answer these questions, they are able to simplify their verbal communication, able to better organize their actions and anticipate one another’s actions, and better able to assist one another (e.g. [11]).

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 (e.g. [6,8]). 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 [8].

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 [6]. 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 rarely 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.

To support awareness in these situations, groupware components can collect, distribute, and display feedthrough information [12]. 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 (e.g. [1,3,9,14,15,16]). Different toolkits focus on different aspects of the groupware development problem: some focus on programming languages (e.g. [3,9]), some on architectures (e.g. [9,14]), and others on simplicity (e.g. [1,3,15]). 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 [5,14]. In Figure 1, 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 not synchronized (shown as unconnected boxes). Figure 1 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, 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 [5]. It is much more common – e.g. in Rendezvous [14], GroupKit [15], COAST [16], Clock [9], and Visual Obliq [3] – 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) [16], but they do not synchronize the widget's appearance. One toolkit by Smith and Rodden [17] 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.

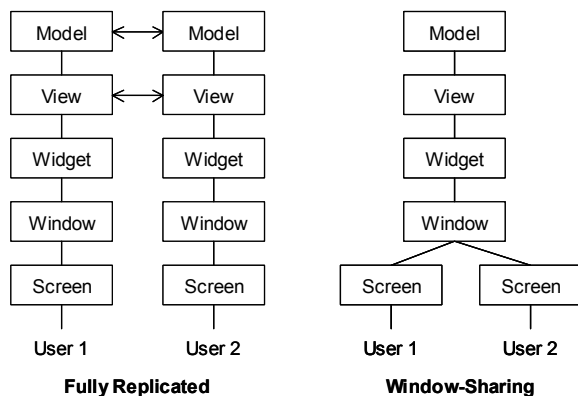


Figure 1. Sharing and synchronization at different layers of the groupware application – fully replicated at left, or window-sharing at right.

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 [2,15]. These are devices such as telepointers (Figure 3), multi-user scrollbars (see Table 2), participants lists (Figure 3), shared text panes [2], and 'radar' overviews [10]. Of these, only the multi-user scrollbar [15] and the shared text pane [2] 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).

Among existing toolkits, GroupKit [15] and Flexible JAMM [2] 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 [13]. 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 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 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 (see Figure 2).

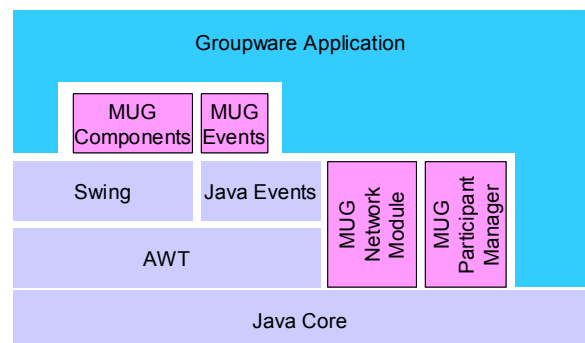


Figure 2. Integration of the MAUI toolkit with Java and with application programs.

4.1 MAUI interface components

The core of the MAUI toolkit is a set of awareness-enhanced UI components (see Table 2). 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.

4.2 Components derived from Swing widgets

Java provides a large single-user component set commonly known as Swing [19]. There are four main types of visual widgets that support user interaction, as shown in Table 1. 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 1.

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.

Table 1. Swing categories, Swing widgets, and widgets implemented in the MAUI toolkit.

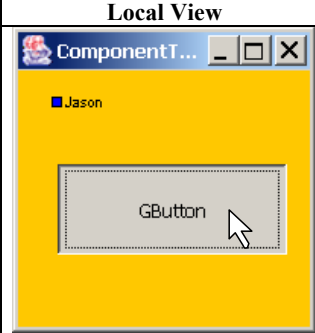
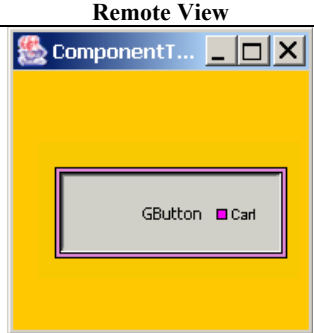
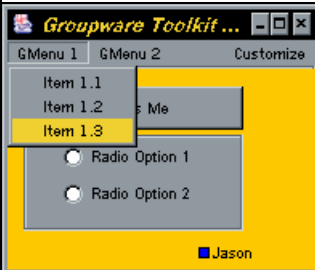
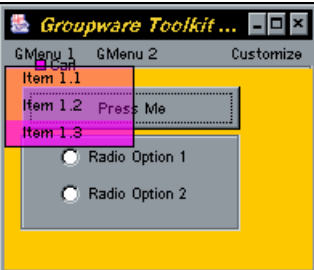

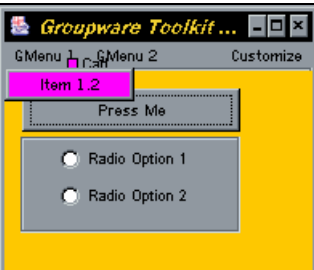
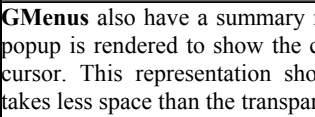
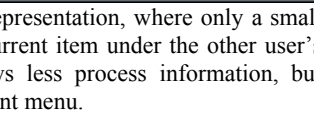
Category	Swing components	MAUI components
Top-Level Containers	Applet, Dialog, Frame	Dialog, Frame
General-Purpose Containers	Panel, Scroll pane, Split pane, Tab pane	Scroll pane, Tab pane
Basic Controls	Buttons, Combo box, List, Menu, Slider, Text Field	All
Editable Displays	File chooser, Table, Text, Tree view	Table

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 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 2, showing their visual effects and the types of awareness they support (see also the accompanying video figures). Note that we do not include the *GFrame* in the index, since it is used primarily for administration 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.

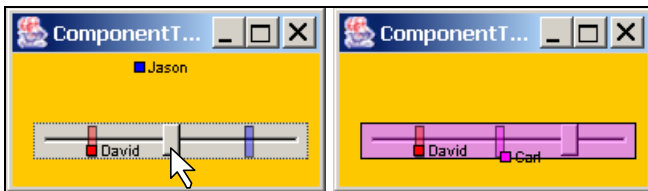
Table 2. A visual index to the MAUI components.

Local View	Remote View
	
	
	
	

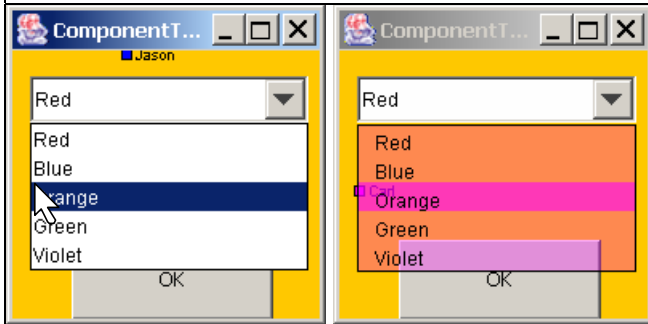
GButtons have only a shared-state form. They visualize intention awareness information (based on enter and leave events) using either a border or background highlight, and action awareness information by animating the button press on the remote screen. Other button types (*GCheckBox*, *GRadioButton*,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

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.

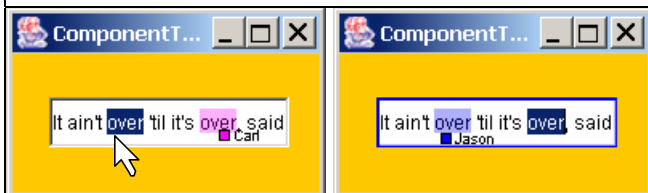
GMenus also have a summary representation, where only a small popup is rendered to show the current item under the other user's cursor. This representation shows less process information, but takes less space than the transparent menu.



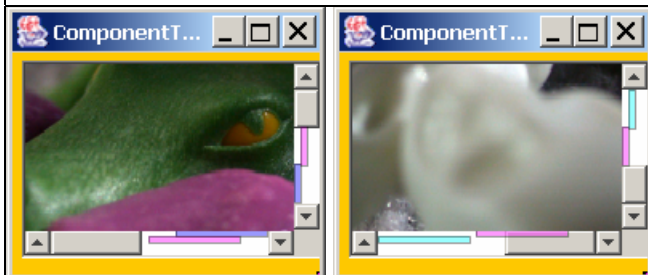
GSliders have both shared and multi-user forms. In the multi-user form (shown here), the position of each person's slider knob is shown as a coloured overlay. In the example above, three users have moved their sliders to three different positions. Entry and exit information can also be shown either with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



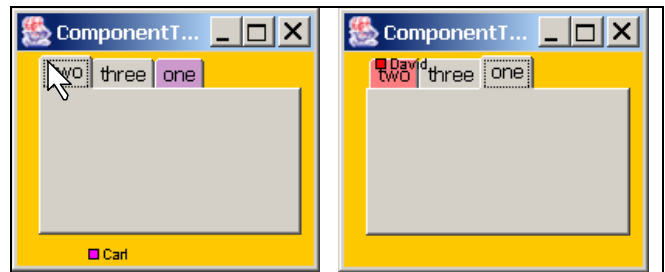
GComboBoxes have only a shared version, but can show their dropdown list in either a transparent (shown here) or a summary representation. These representations are displayed in the same way as for GMenus – by custom rendering to the GlassPane of the GFrame.



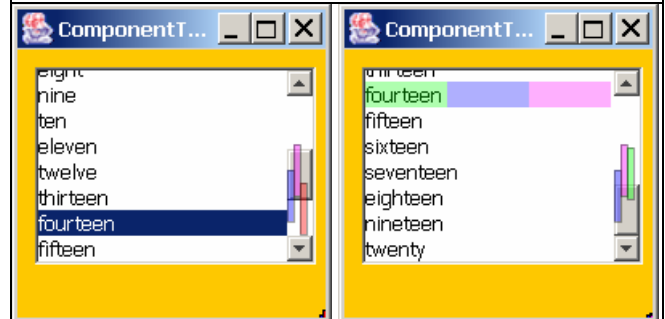
GTextFields use a combination of shared and multi-user properties. The text is shared, but selections in that text are multi-user, and are shown as coloured transparent overlays.



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 each user's local scroll thumb, and can also show each user's viewport as a rectangle inside the scrolled 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. [15]). The GScrollPane is also used with GLists and GTables.



GTabbedPanes 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."



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).

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 (soon) radar views. These components show a variety of awareness information including who is in the session, where they are working, and how active they are. Below, we outline how telepointers and participant lists have been incorporated into the toolkit.

Telepointers. Telepointers (Figure 3) are implemented on the top-level component, the GFrame, and can be added at design-time through a property list in the IDE. 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).

Telepointers have several design-time options. There are three styles of telepointer available (arrow, block, or icon), and participant names can be added to the pointer representation. Telepointers can also leave fading trails (as in Figure 3), which can assist people in producing and interpreting gestural communication, particularly in jittery networks.

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 ‘official’ shared workspace panel. 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 3). 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.

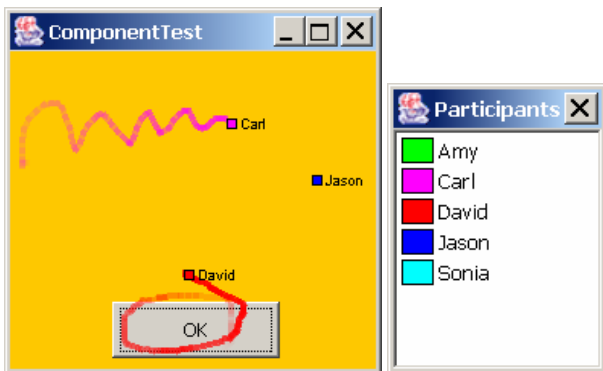


Figure 3. MAUI telepointers and participant list.

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 4).

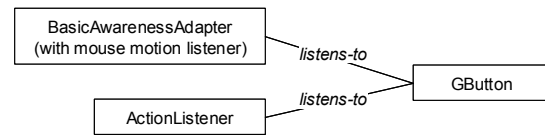


Figure 4. Collecting user events for a GButton.

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 5): components generate widget-specific events and generic awareness events. There are also user events created by the participant manager.

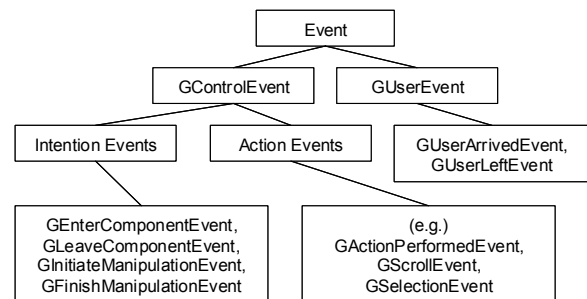


Figure 5. MAUI event hierarchy

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 6.

4.4.3 Handling awareness events at the remote component

Components receive remote events from their GController by implementing the GControlListener 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.

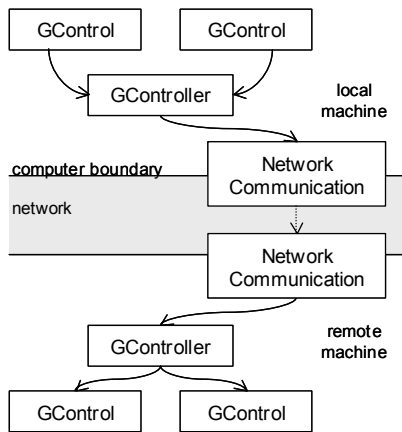


Figure 6. Information flow through MAUI components in a groupware application.

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 both the transparent or the summary representations of the component (see menu examples in Table 2). In these cases, custom renderers are incorporated into the `GComponent`. We defines two for menus, `TransparentPopupMenuRenderer` and `SummaryPopupMenuRenderer`.

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 7 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 7.

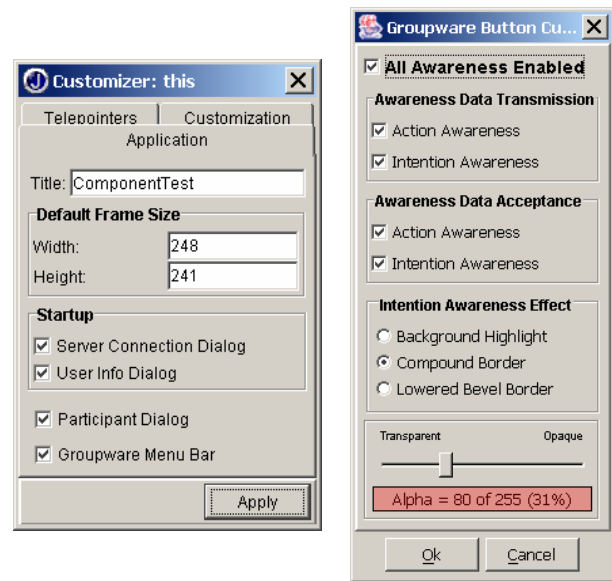


Figure 7. Customization dialogs for a `GFrame` (left, design-time) and for a `GButton` (right, run-time).

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 (i.e. `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 (currently based on Java Remote Method Invocation). 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. In addition, the default system may be sufficient for small applications.

4.7 Participant 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 management module.

Switching to a new 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 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 8) in JBuilder, a popular Java integrated development environment (although any Beans-compliant IDE will work).

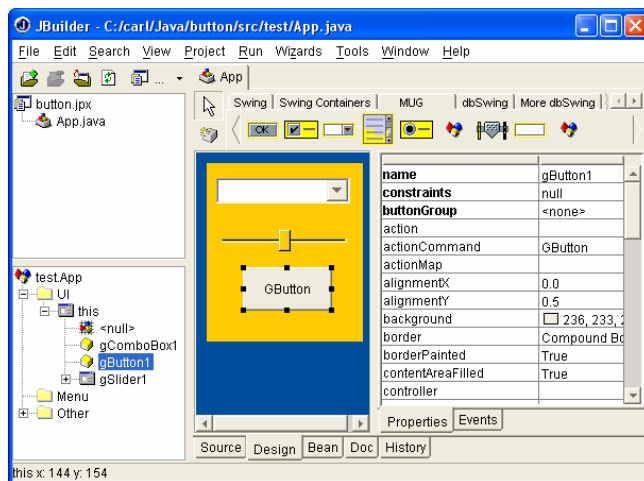


Figure 8. JBuilder IDE, showing UI designer, MAUI widget tab, and property list for selected GButton.

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 8). To start building the UI, the developer then creates a new class based on a JFrame; 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 JFrame. A property list editor (at right in Figure 8) 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.

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.xml.

6. EVALUATION

Four evaluations of MAUI have been carried out, three from the perspective of the developer, and one from the perspective of the end users of MAUI applications. The first considered effort – whether the toolkit does save work when compared to other development methods. The second considered the toolkit’s effect on application performance. The third tested flexibility by constructing several representative groupware interfaces. The fourth tested the end-user usability of the visual components themselves and the run-time customization facilities.

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 an application in three different ways: in Java without the use of a groupware toolkit, with the Groupkit toolkit [15], 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.

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 not provide arbitrary control over widget appearance), and as such the application was a ‘lowest-common-denominator’ for the three approaches.

Table 3. 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 minutes

This test shows that MAUI does reduce development effort for groupware interfaces, even in comparison to a toolkit which are also 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 performs 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 9), a form tool for jointly applying for a loan, and a discussion board.

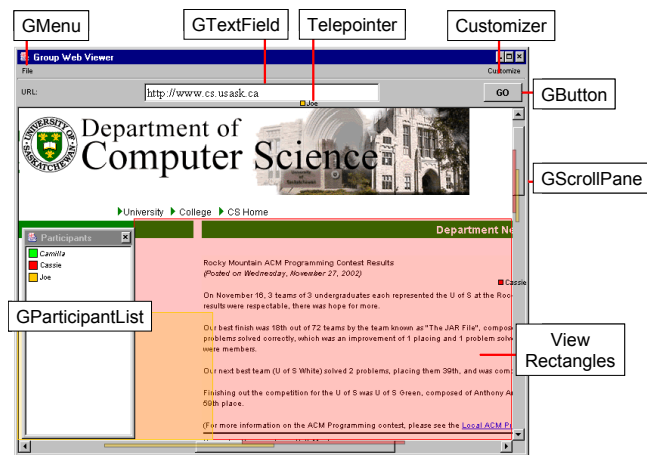


Figure 9. Example shared browser built with MAUI.

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 flexibility 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, we did determine a set of limitations to applications built with MAUI:

- 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.
- 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 menu items need to be drawn outside the window bounds they cannot make use of transparency effects.
- The 'black box' network and session modules are limited – the network module provides only TCP-based communication, for example, and the session management module does not provide methods for updating late entrants.
- 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 Widget and Awareness Usability

A basic evaluation of the widgets themselves was performed with a focus group of four experienced groupware users. The group was given the four sample applications described above, and asked to assess the awareness information and visualizations provided by the toolkit widgets. In general, all group members agreed that the awareness information provided would be beneficial to users of the applications. End-user performance did not present a problem, and the awareness information was provided in a timely fashion.

While all group members agreed that the visualizations of awareness information was adequate, all users provided comments about how the visualizations for particular widgets could be presented differently in special circumstances. The need for custom-designed awareness visualizations for specific applications is clear. MAUI provides some facilities for specialization by the application programmer through the run-time and design-time customizers, and we believe that this offers enough range to support most groupware applications. However, it would also be possible to extend these customizers to provide completely different representations of awareness information for a widget.

Despite these discussions, our evaluators agreed that the MAUI widgets worked well in a range of groupware applications.

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. [4,8]). 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: a radar view, a text-chat widget, 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 [2]. Although this will not completely convert the application (since MAUI handles 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.

9. REFERENCES

[1] Banavar, G., Doddapeneni, S., Millar, K., and Mukherjee, B. (1998). Rapidly Building Synchronous Collaborative

- Applications By Direct Manipulation. *Proc. ACM CSCW '98*, 139-148.
- [2] Begole, J., Rosson, M., and Shaffer, C. (1998). Supporting Worker Independence in Collaboration Transparency. *Proc. ACM UIST '98*, 133-142.
- [3] Bharat, K., and Brown, M. (1994). Building Distributed, Multi-User Applications by Direct Manipulation. *Proc. ACM UIST '94*, 71-81.
- [4] Brewster, S., Wright, C., and Edwards, A. (1994). Design and Evaluation of an Auditory-Enhanced Scrollbar. *Proc. ACM CHI '94*, 173-179.
- [5] Dewan, P., Choudhary, R. (1995) Coupling the User Interfaces of a Multiuser Program, *ACM ToCHI*, 2, 1, 1995, 1-39.
- [6] Dix, A., Finlay, J., Abowd, G., Bealle, R. (1993). *Human-Computer Interaction*, Prentice Hall, 1993.
- [7] Dourish, P., Bellotti, V. (1992). "Awareness and Coordination in Shared Workspaces." *Proc. ACM CSCW '92*, 107-114.
- [8] Gaver, W., Smith, R., and O'Shea, T. (1991) Effective Sounds in Complex Systems: The ARKola Simulation, *Proc. ACM CHI'91*, 85-90.
- [9] Graham, T., Urnes, T., Nejabi, R. (1996). Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware. *Proc. ACM UIST '96*, 1-10.
- [10] Gutwin, C., Greenberg, S. (1998). Effects of Awareness Support on Groupware Usability. *Proc. ACM CHI '98*, 511-518.
- [11] Gutwin, C., Greenberg, S. (in press). A Descriptive Framework of Workspace Awareness for Real-Time Groupware." *JCSCW*, in press.
- [12] Gutwin, C., Greenberg, G. (2001). Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness, *Proc. ACM CSCW'98*, 207-216.
- [13] Gutwin, C., Roseman, M., and Greenberg, S. (1996) A Usability Study of Awareness Widgets in a Shared Workspace Groupware System. *Proc. ACM CSCW '96*, 258-267.
- [14] Hill, R., Brinck, T., Rohall, S., Patterson, J., Wilner, W. (1994). Language for Constructing Multiuser Applications. *ACM ToCHI* 1,2, June 1994, 81-125.
- [15] Roseman, M., Greenberg, G. (1996) Building Real Time Groupware with GroupKit, A Groupware Toolkit. *ACM ToCHI*, 3,1, 66-106.
- [16] Schuckmann, C., Kirchner, L., Schummer, J., and Haake, J. (1996) Designing Object-Oriented Synchronous Groupware with COAST. *Proc. ACM CSCW '96*, 30-38.
- [17] Smith, G., and Rodden, T. (1993). Access as a Means of Configuring Cooperative Interfaces, *Proc. COOCS '93*, 1993.
- [18] Sun Microsystems (1997) *The JavaBeans Specification*, available from java.sun.com/products/javabeans/docs/spec.html.
- [19] Walrath, K., and Campione, M., (1999) *The JFC Swing Tutorial: A Guide to Constructing GUIs*, Addison Wesley, 1999.