Novel Widgets and Toolkits For Creating Them

- Toolkits for novel interaction techniques.
  - SwingStates
  - subArctic
- Toolkit for groupware applications.
  - MAUI
- All of these toolkits are Java libraries.

SwingStates

- SwingStates is a library that adds state machines to the Java Swing user interface toolkit.
- The state machines are used as a control structure for programming interaction.
  - Goal is to allow for programming advanced interaction and creating novel interaction techniques.

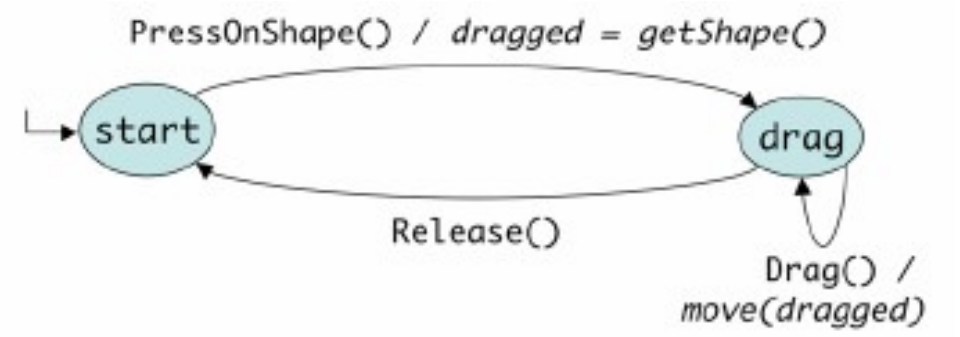
- http://www.lri.fr/~appert/SwingStates/index.html

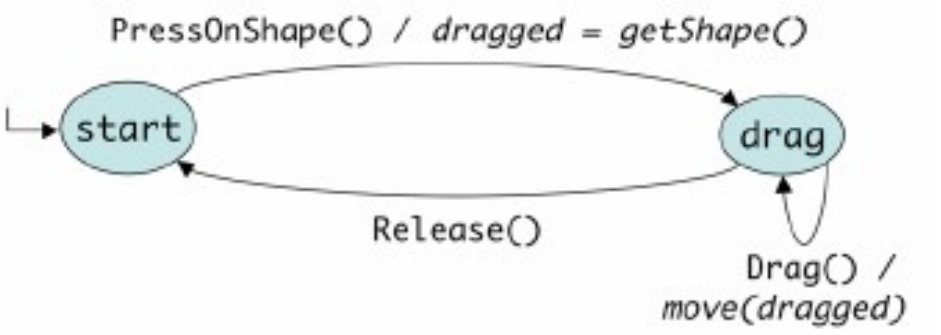
SwingStates Canvas

- SwingStates includes a Canvas widget.
  - Similar to the Tk canvas widget – GmlCanvas.
  - Holds a display list of shapes, including simple or arbitrary paths, text strings, images and some Swing widgets.
  - Each shape can have a geometric transform, a parent shape and a clipping shape.
  - Shapes can be tagged (similar to Tk).

State Machines

- A state machine consists of the following:
  - A set of *states* and a set of *transitions* labeled with *events*.
  - Each transition may have an associated *guard* and *action*.
  - Each state has a set of *output transitions* and a set of *input transitions* (may have an *enter* and *leave* action).






```
1  StateMachine sm = new StateMachine() {
2    SMShape dragged = null;
3    public State start = new State() {
4      Transition dragOn =
5         new PressOnShape(BUTTON1, "drag") {
6           public void action() {
7             dragged = getShape();
8           }
9      };
10   };
11   public State drag = new State() {
12    Transition drag = new Drag(BUTTON1, "drag") {
13       public void action() {
14         move(dragged);
15       }
16     };
17     Transition dragOff =
18       new Release(BUTTON1, "start") { } ;
19   };
20 };
```

Attaching State Machines to UI Objects

- Attach directly to regular Swing widgets to extend or redefine their behavior.
- Attach directly to the canvas and to individual shapes on the canvas.
- Via Tags
  - *Extensional tags* – added to or removed from objects explicitly.
  - *Intentional tags* – specified using a predicate.
    - Example: all objects with a blue background

Extending Swing Button Behavior

- A crossing interface for Swing Buttons

```
1 JStateMachine cross = new JStateMachine() {
2   public State out = new State() {
3       Transition enter = new EnterOnTag
                    ("javax.swing.JButton", "in") { };
4   };
5   public State in = new State() {
6       Transition leave = new Leave("out") {
7           public void action() {
8               ((JButton)getComponent()).doClick();
9           }
10      };
11 };
// attach this state machine to the quit button of Fig. 2
12 cross.attachJComponent(quitButton);
```

Applet

An Example Using Multiple Features

- Pie Menu example using the following elements
  - Canvas
  - Glasspane
  - Two state machines (communicating with each other)
  - Multiple widgets with tags

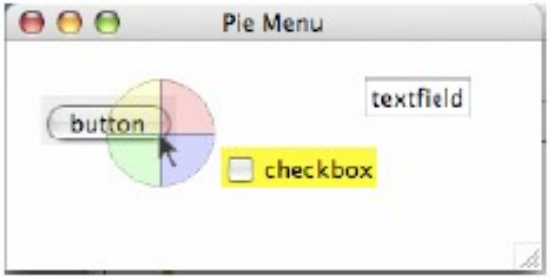
Applet

```
// ColorTag are designed to be added to each menu item.
class ColorTag extends CExtensionalTag {
    Color color;
    ... // constructor
}

// Color events are sent by smMenu to smWidget.
class ColorEvent extends VirtualEvent {
    Color color;
    ... // constructor
}

// The state machine that manages the pie menu
 smMenu =  new CStateMachine(canvas) {
   public State menuOff = new State() {
     Transition show = new Press(BUTTON1,"menuOn"){
       public void action() {
         showMenu(getPoint());
       }
     };
   };
   public State menuOn = new State() {
     Transition command = new ReleaseOnTag
                (ColorTag.class, BUTTON1, "menuOff") {
       public void action() {
         Color c = ((ColorTag)getTag()).color;
         smWidgets.processEvent(new ColorEvent(c));
         hideMenu();
       }
     };
     Transition hide = new Release
                                (BUTTON1, "menuOff") {
       public void action() {
         hideMenu();
       }
      };
   };
};
```

```
// "colorable" widgets must be attached to this state machine
 smWidgets = new JStateMachine() {
   JComponent picked;
   public State noSelection = new State() {
    Transition select = new PressOnComponent() {
      public void action() {picked = getComponent();}
    };
   } ;
   public State selection = new State() {
    Transition deselect
           = new Event("cancel", "noSelection") {} ;
    Transition color
           = new Event("color", "noSelection") {
     public void action() {
       ColorEvent e = (ColorEvent)getVirtualEvent();
       if (selected) picked.setBackground(e.color);
     }
    };
   };
};
```



Why use SwingStates?

- SwingStates lets you design and implement the interaction technique separate from the UI object.
- Once you have a state machine it can be attached to UI components in flexible ways.
- The canvas and tags borrowed from Tk make some of the drawing in Swing easier.

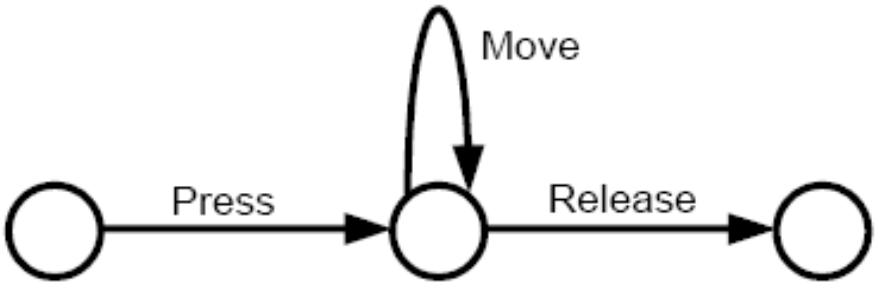
subArctic Toolkit

- The subArctic toolkit was designed to "*supply a powerful library of reusable interactive objects, and to make it easy to create new, unusual, and highly customized interactions*".
- Provides standard reusable components.
- Allows input to be handled in flexible ways.

- http://www.cc.gatech.edu/gvu/ui/sub_arctic/

Dispatch Agents

- Each agent handles a certain type of event.
  - Text editing input
  - Various types of dragging
  - Pressing, clicking, or double-clicking
- A simple drag agent would implement the following state controller.



Dispatch Agents (cont.)

- Dispatch agents use an input protocol to communicate with components.
- The component would then have to implement the Java interface that defines the protocol.
- Example of a simple drag agent protocol.

```
public interface simple_draggable
    extends focusable {
        public boolean drag_start(…);
        public boolean drag_feedback(…);
        public boolean drag_end(…);
};
```

Picking

- SubArctic gives the programmer control over what gets picked under the cursor position.
- Provides an explicit list (*pick collector*) and lets the components decide how to fill in the list
- Picking is performed by a top-down recursive traversal of the component tree.

Picking (cont.)

- A shadow drag container example for modifying picking order.
  - Any widget placed in the container will be dragged if any one of the widgets in the container is picked.
  - The container's pick() method creates an empty pick collector and passes it to the child objects.
  - When the container gets the list back it adds itself to the original pick collector followed by all the children in the list.

Applet

Dragging Dispatch Agents Provided by subArctic

- Conventional Dragging
- Constrained Motion
  - Uses a drag filter to limit the motion to an area.
- In/out dragging
  - Similar to the crossing interface seen in SwingStates example
- Semantic snap-dragging
  - Selected targets act as gravity wells for object being dragged. The target must pass a semantic test before the snap to target occurs.

Other subArctic Features

- Recording input
  - Ability to record input events to see the stream of input produced by the user.
    - Low level recording of user input.
  - Ability to record the input protocol and the method within the protocol used along with the object it is sent to.
    - Recording the semantics of user input.
- Animation support allowing for various effects.
  - Slow-in/slow-out motion
  - Squash and stretch

The MAUI Toolkit

- Multi-user Awareness UI toolkit
- Goal is to help synchronous and distributed collaborators maintain awareness of other people in the group.
- Extends many Java Swing widgets and adds some groupware specific widgets to provide *feedthrough*.

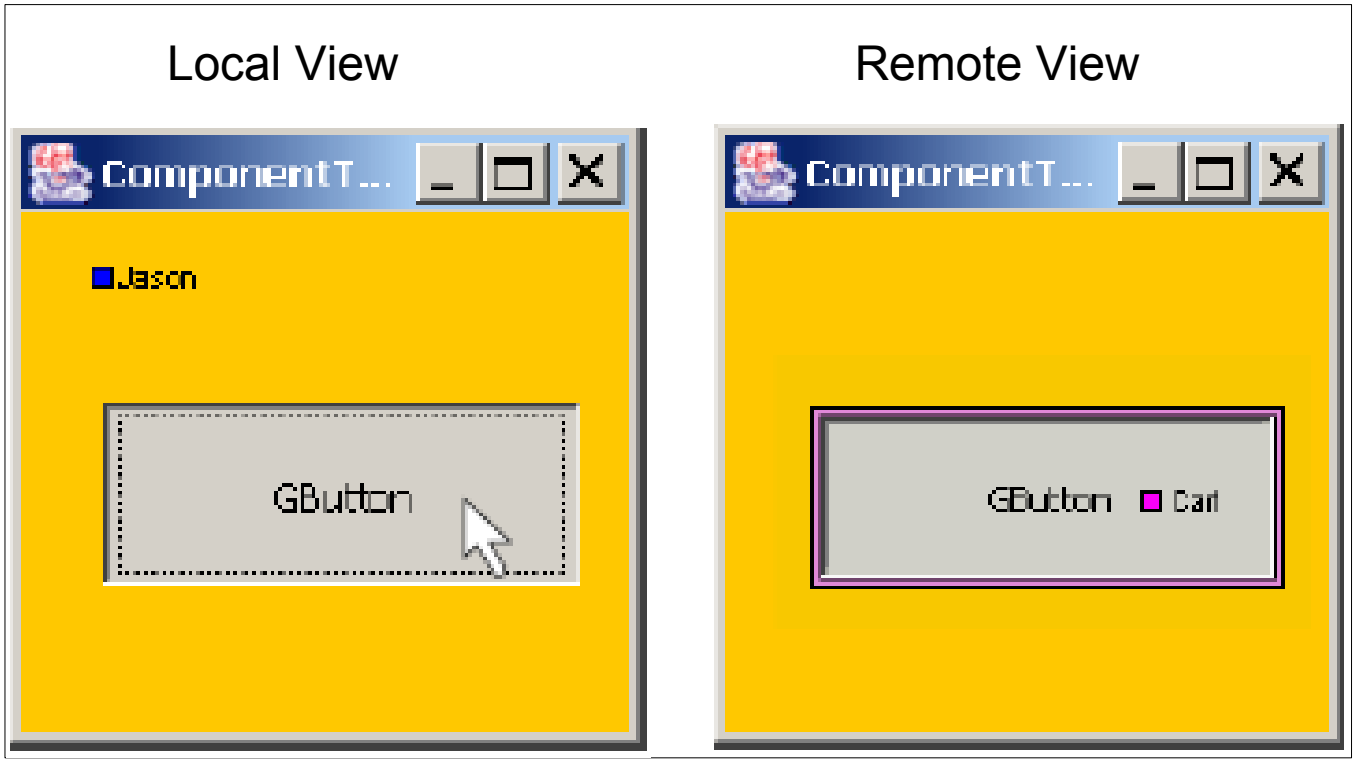
- http://hci.usask.ca/research/past/maui.shtml

MAUI Toolkit Features

- Supports both single-state and multi-state versions of most Java Swing widgets.
- Provides run-time customization that can be controlled either by the user or the application.
- Includes black-box network and session components that enable prototyping and testing.
- Packages components as Java Beans, which allows integration with standard IDEs.

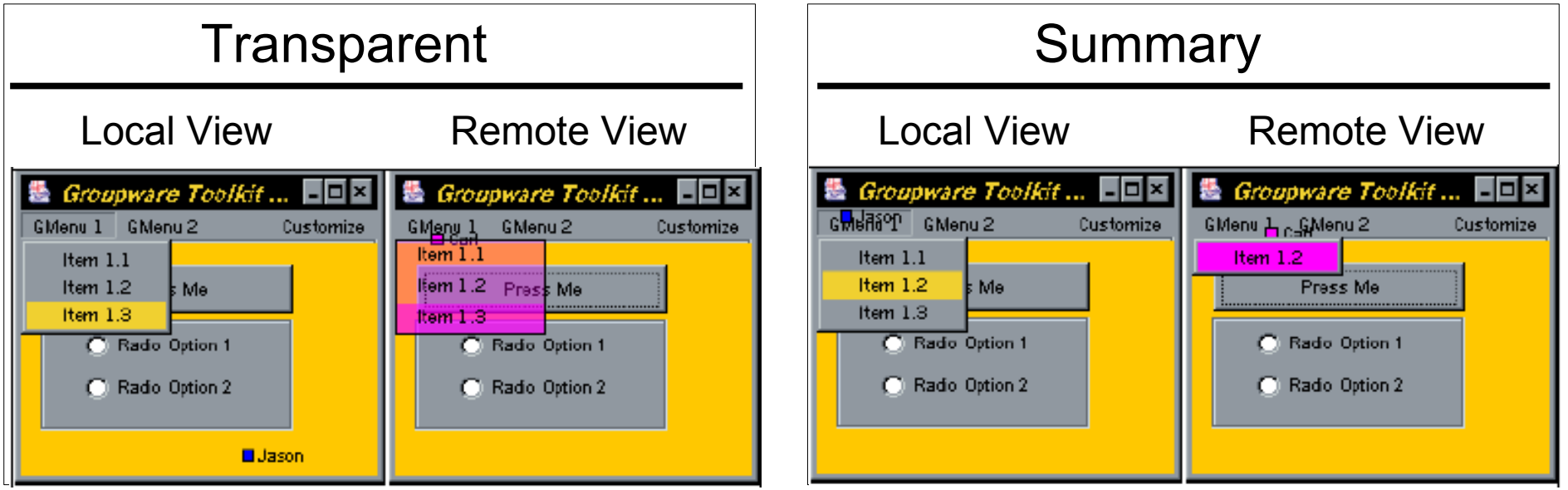
Extended Java Swing Components
GButtons

- GButtons only have shared state form



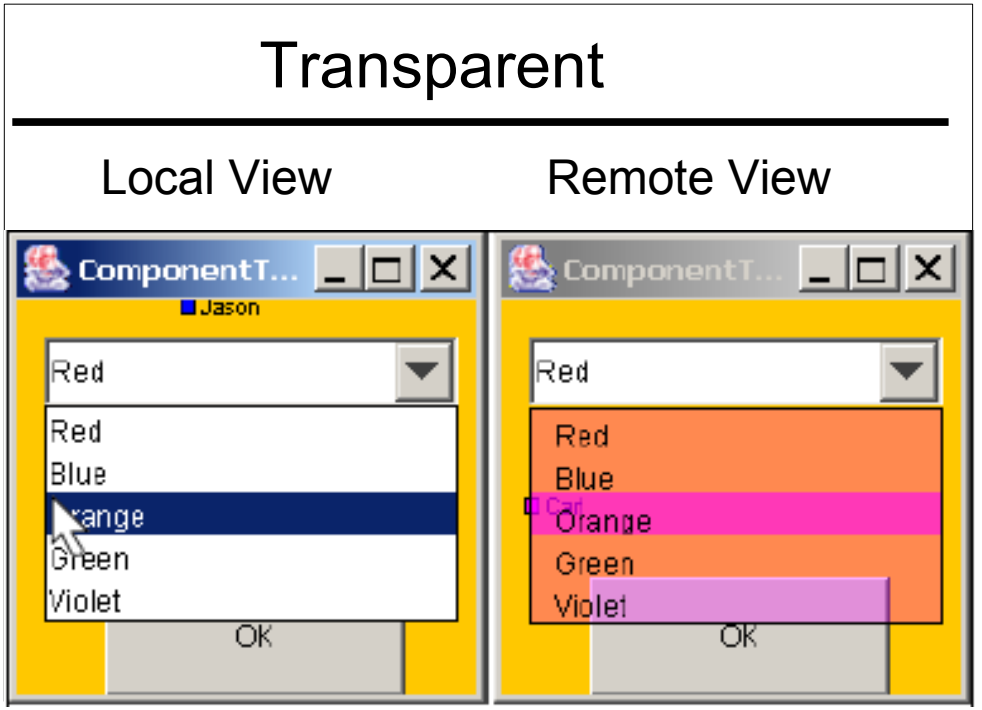
Extended Java Swing Components
GMenus

- GMenus have only shared version
  - Transparent Representation
  - Summary Representation



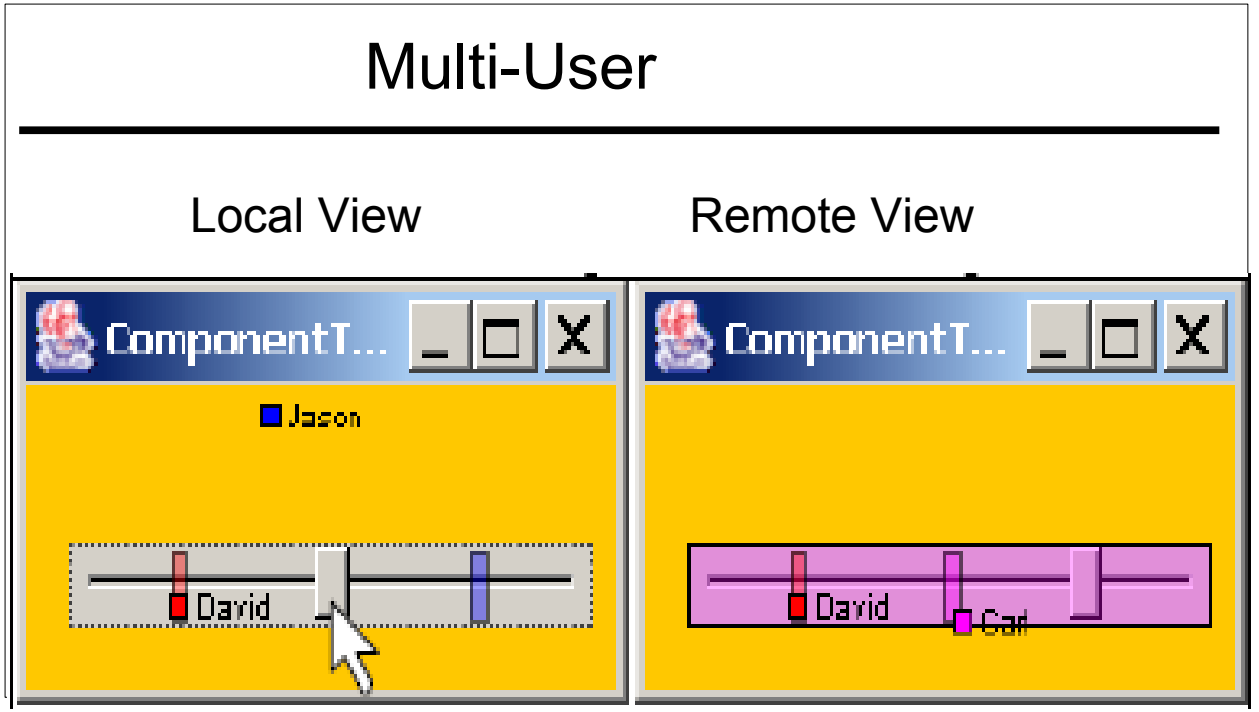
Extended Java Swing Components
GComboBoxes

- GComboBoxes have only shared versions
  - Transparent Representation
  - Summary Representation



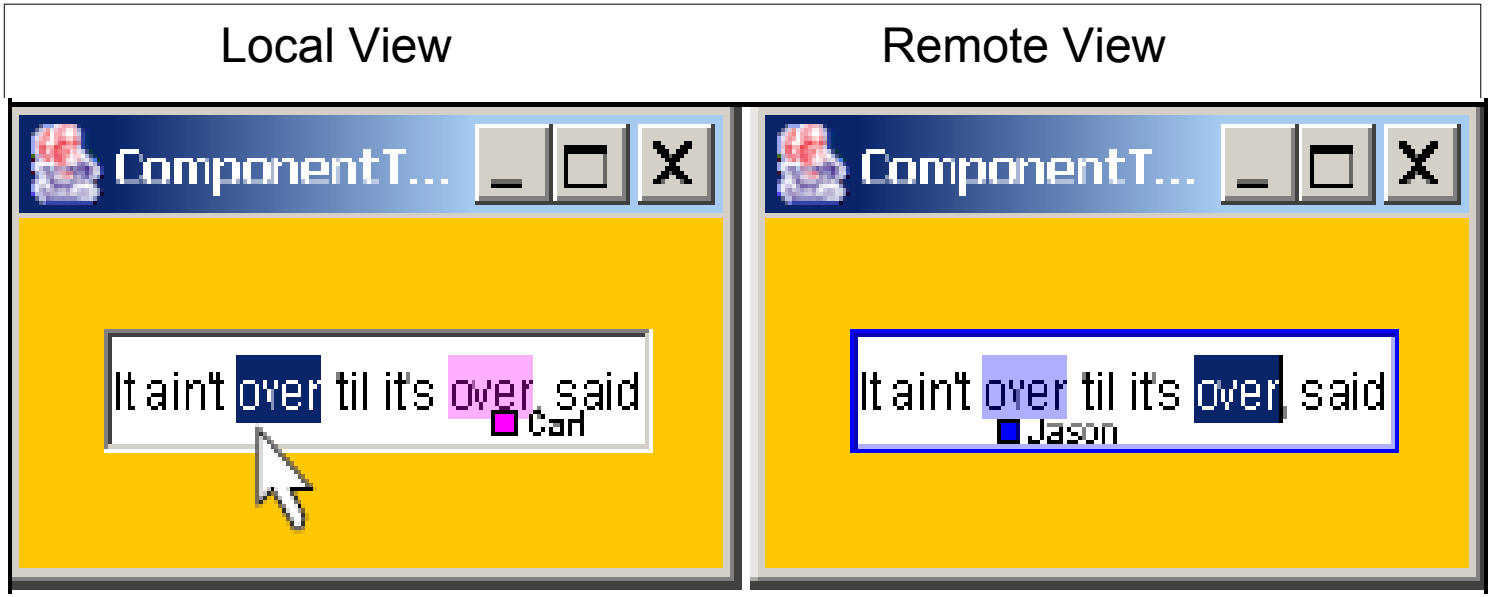
Extended Java Swing Components
GSliders

- GSliders have two versions
  - Multi-User
  - Shared



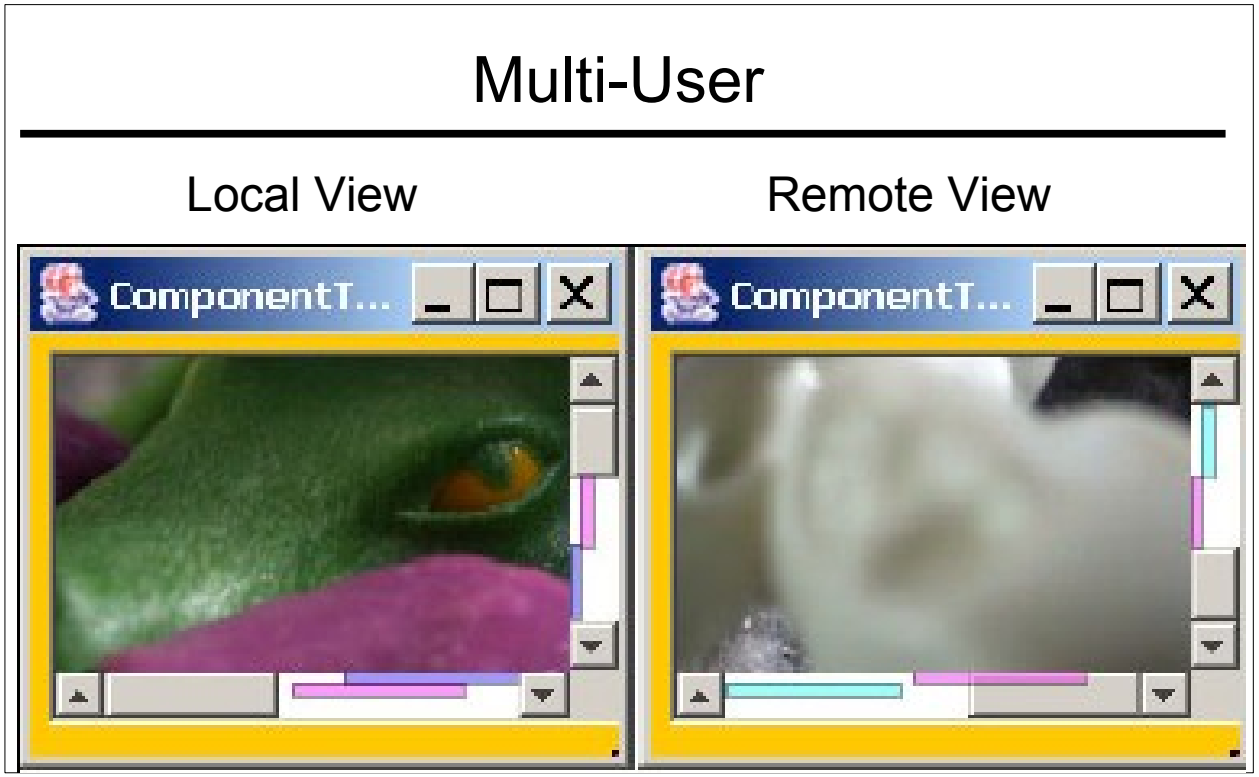
Extended Java Swing Components
GTextFields

- GTextFields use a combination of shared and multi-user properties.
  - Text is shared
  - Selections in text are multi-user



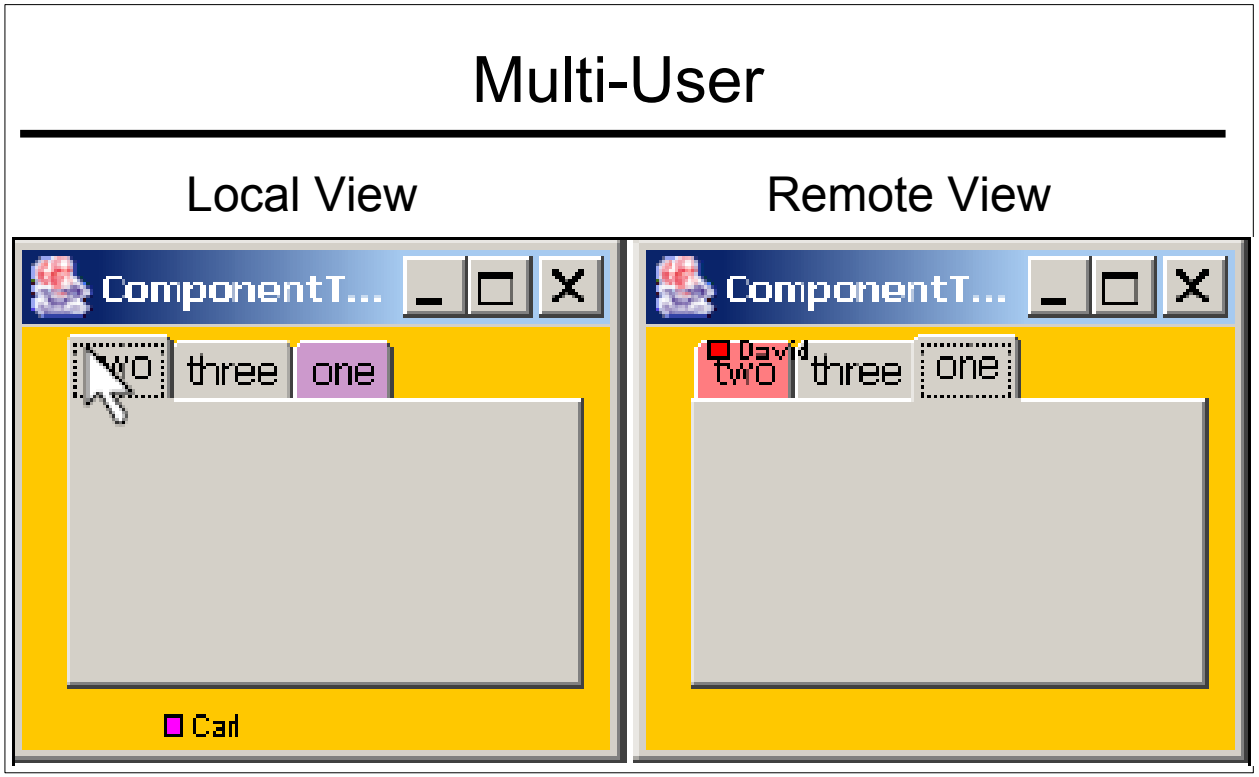
Extended Java Swing Components
GScrollPanes

- GScrollPanes have two versions
  - Multi-User
  - Shared



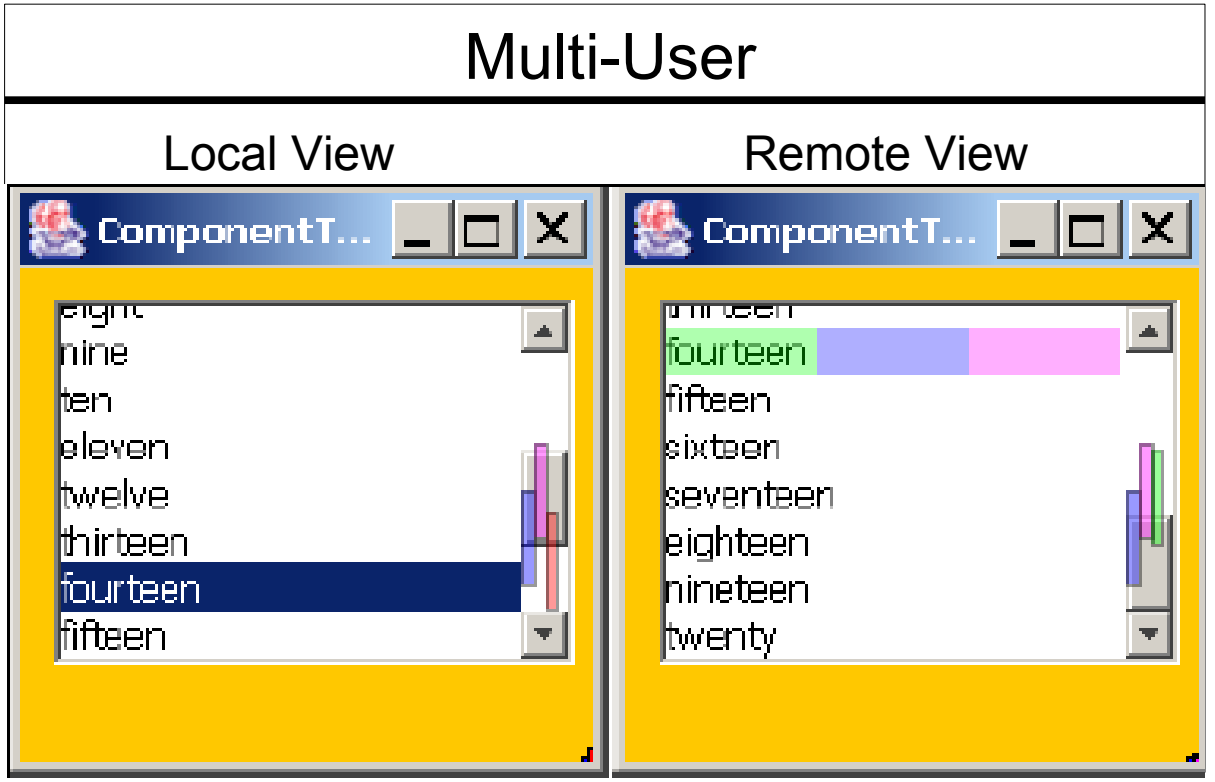
Extended Java Swing Components
GTabbedPanes

- GTabbedPanes have two versions
  - Multi-User
  - Shared



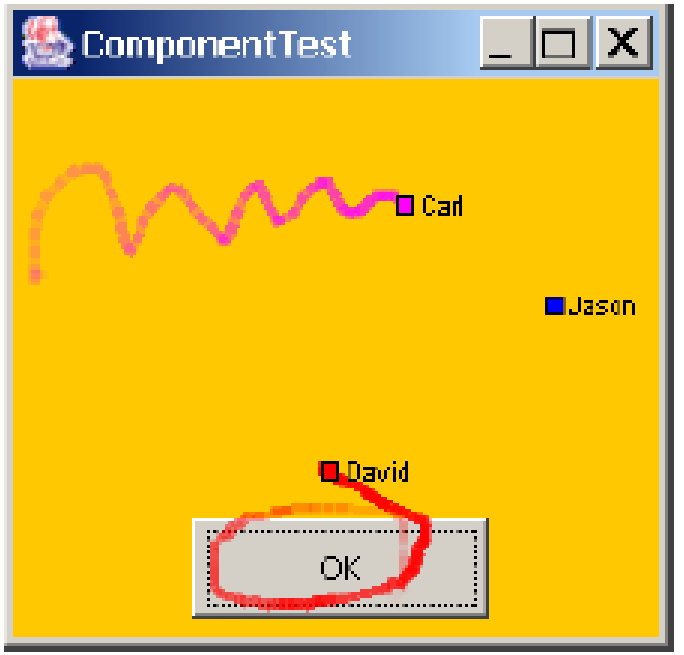
Extended Java Swing Components
GLists

- GLists have two versions
  - Multi-User
  - Shared



MAUI Groupware Widgets
Telepointers

- Participant names can be added to pointer.
- Telepointers can leave fading trails.
  - Helps provide gesture information
- Three styles available
  - Arrow
  - Block
  - Icon



MAUI Groupware Widgets
Participant List

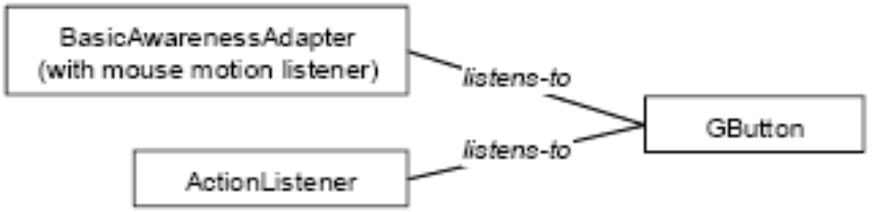
- Shows the names and colors of all connected users.
- Does not listen to standard AWT events. Only listens for connection and disconnection events.
- Can be either top-level dialog or a panel inside a frame.

MAUI Event Model

- Captures incoming AWT events from the local user.
- Generates MAUI-specific awareness events for distribution.
- Handles awareness events that have arrived from another machine.

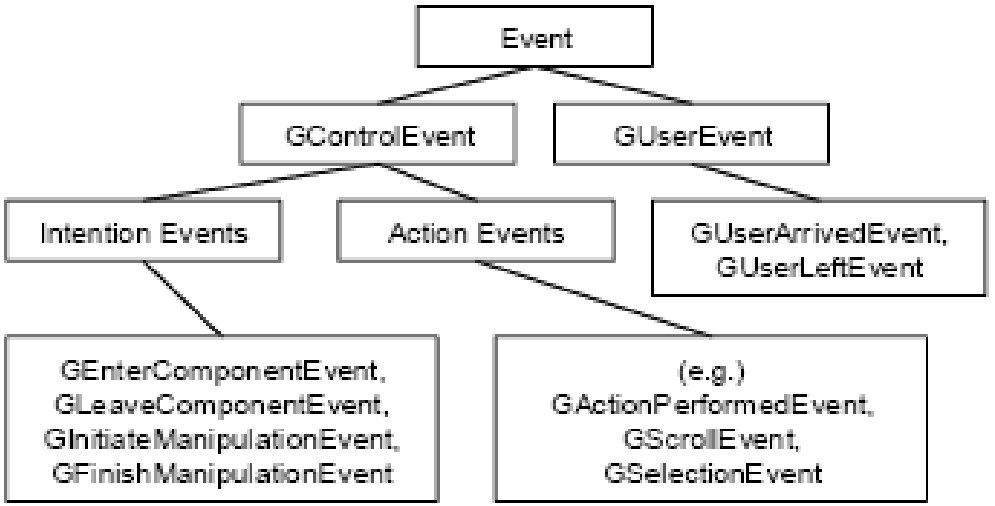
Capturing AWT and Swing User Events

- MAUI components include two types of listeners.
  - ActionListener – a widget specific listener to handle action events.
  - BasicAwarenessAdapter – a MAUI adapter to handle intention events.



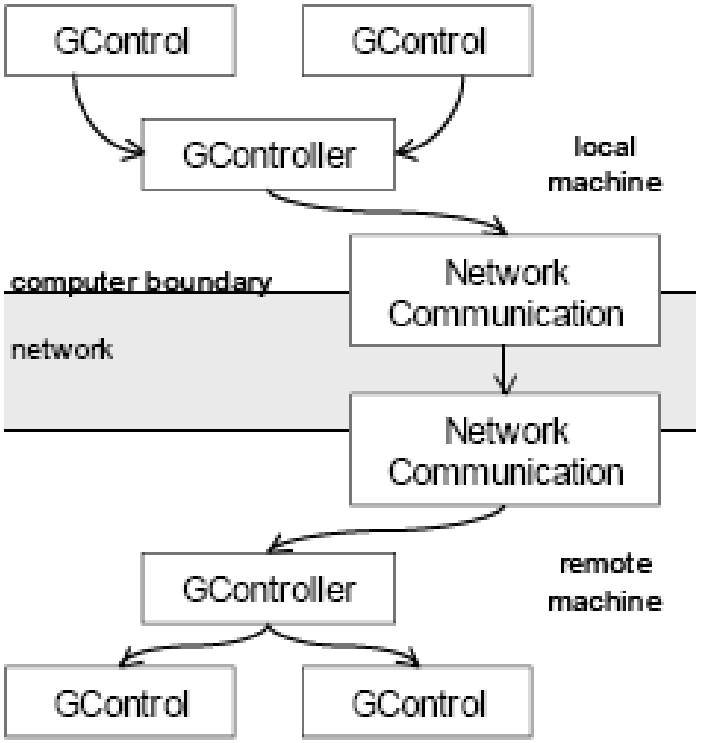
Creating and Distributing Awareness Events

- A MAUI event called a GControlEvent is created to send awareness information.
- The GControlEvent extracts information from original AWT event making it smaller and more suitable for sending over the network.

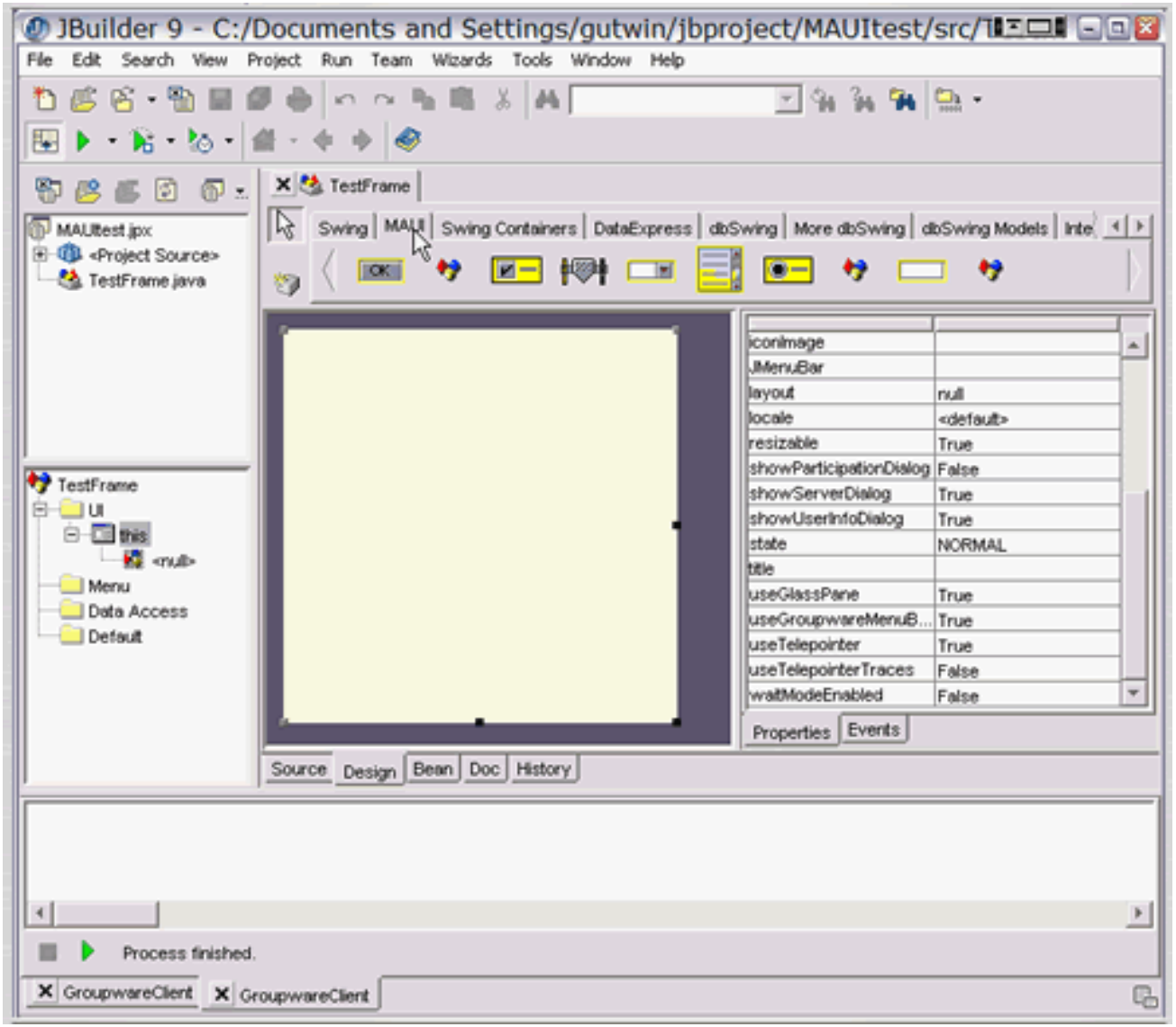


Handling Awareness Events at the Remote Component

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



MAUI Toolkit in the JBuilder IDE



Customization Dialogs

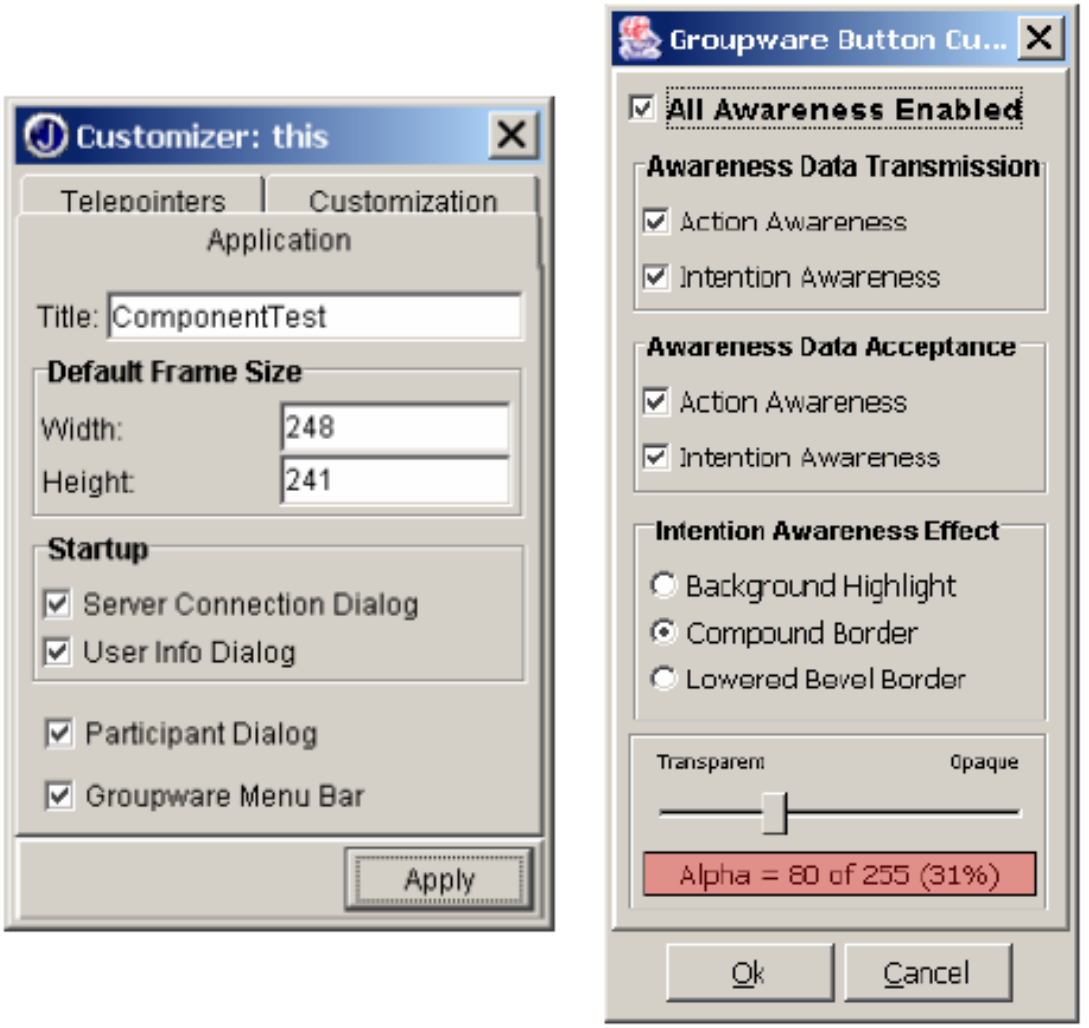


Figure 7. Customization dialogs for a GFrame (left, design-time) and for a GButton (right, run-time).

MAUI Limitations

- MAUI uses the glass pane so developers cannot use their own glass pane.
- Applications are limited to using the frame class as main window.
- Only TCP-based communication is provided and late entrants are not updated.
- Messages are processed on a first-come first-served basis.

References

- Appert, C. & Beaudoin-Lafon, M. (2006) SwingStates: Adding State Machines to the Swing Toolkit. In *Proc. ACM UIST '06*, pp 319-322.
- Hill, J. & Gutwin, C. Awareness Support in a Groupware Widget Toolkit. Computer Supported Cooperative Work. **13**(5-6):539-571, 2004.
- Hudson, S. E., Mankoff, J., and Smith, I. (2005) Extensible input handling in the subArctic toolkit. In *Proc. ACM Conference on Human Factors in Computing Systems.* CHI '05. ACM Press, pp 381-390.