Jedemo : The Environment of Event-driven Demonstration for Java Toolkit

Motoki Miura
Master's Program in Science and Engineering
University of Tsukuba
1-1-1, Tennodai, Tsukuba
Ibaraki, 305-8573, JAPAN
+81 298 53 5165
miuramo@softlab.is.tsukuba.ac.jp

Jiro Tanaka
Institute of Information Sciences and Electronics
University of Tsukuba
1-1-1, Tennodai, Tsukuba
Ibaraki, 305-8573, JAPAN
+81 298 53 5343
jiro@softlab.is.tsukuba.ac.jp

ABSTRACT

This paper describes an environment in which we can execute an event-driven demonstrations for Java applets. The event-driven demonstration means to show the behavior of an applet/application by re-executing the captured events. It can be used for providing a help regarding how an application works. Such an animated help is needed for Java applets, which are carried out by a vast majority of web users.

Jedemo recorder captures the occurring events while the applet is running. Jedemo analyzer tries to label each command and creates indices. The indices is used for editing and providing abstract of the demonstration. Then Jedemo player re-executes the captured events with displaying the indices.

Jedemo systems suit to almost all applets. Developers can integrate applets with Jedemo player without any trouble. This environment is helpful for both the developer of the applets and the person who accesses the applets.

Keywords
animated-help, internet, World Wide Web, programming by demonstration

INTRODUCTION

We can download and run a small program called “applet” on the browser. Many developers want to publicize these applets on their web-sites. For the user, it is necessary to know how to operate the applet. Of course the developers know the usage of the applets, but the task of writing explanation is too much troublesome.

Though making help documents is important, it needs a lot of time and effort. We think the major reason of the trouble is caused by the expression gap between a “text” and a “graphical object.”

Almost all applets are adopting a graphical user interface (GUI). The GUI allows the user to try the system easily. But the user can not always operate the system properly. The user needs some instructions. Usually the instructions are provided in “textual” form. Using the textual expression, it is difficult to specify the graphical objects: icons, menus, buttons and so on. The user should read the textual instructions and should look for the corresponding object.

The developer may also think it boring to express each graphical object as a plain text. This paper presents *Jedemo* which enables the developer to reduce the trouble caused by constructing help documents.

RECORDING

Components

The developer of an applet uses the class packages such as AWT (Abstract Window Toolkit) and JFC (Java Foundation Classes) [8]. In either case, there are two types of classes: a component and a container. The component receives user inputs and issues corresponded events. The container can add some components and layout them. The container can also add the other container. The container has a list of components. Referring the list, we can access each component. The developer put together components and containers for programming an applet/application. These components constitute a tree structure (Fig. 2).

Event Propagation

If the user presses a button, the button issues an instance of ActionEvent. The action-event includes a command name and a source object. The command name is the label of the pressed button. The source object links to the pressed button.

Usually, each component which can be a source object has some event-listeners which respond to the events occurred in the component. We achieve the event capturing by adding some extra event-listeners. The extra event-listener includes a facility to send the occurred events to the recorder. We have prepared extra event-listeners corresponding to each type of event-listener.

The recorder searches the component tree (Fig. 2) starting from the root. Then the recorder checks the type of extra event-listeners which can be added to the component. For example, if the component's class is “Button” which has a method *addActionListener*, the extra listener object added to the component is the action-listener.

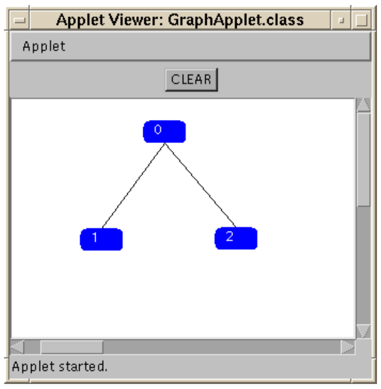


Figure 1: GraphApplet

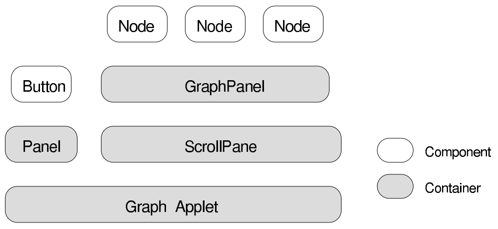


Figure 2: Component tree of the GraphApplet

Low-level Events

We have implemented a graph editor applet (Fig. 1) which adopts direct manipulation interface. In the applet, dragging is used for creating new child node, moving the node, create a link, delete a link and delete the node. When we drag down to the node, new child node is made with link. When we drag up, the node is moved. Moving the node outside of the applet means deleting it. ButtonPress in parent node and ButtonRelease in child node creates a new link. To delete the link, press on the child node and release on the parent node.

Although the direct manipulation such as dragging is an intuitive operation, the handling of events is complicated. To achieve direct manipulation, we have used low-level events which are both MouseEvent and MouseMotionEvent. The MouseEvent has following status: *press*, *release*, *click*, *enter* and *exit*. The MouseMotionEvent has a status of *move* or *drag*.

The low-level events in both clicking and dragging are started with a *mouse pressed* event, followed by some *mouse dragged* events when dragging, and ended with a *mouse released* event. The *mouse moved, mouse entered* and *mouse exited* events have almost no meaning. So we specify them as "separators." The analyzer separates the lengthy mouse events by the separator. We call each part of the separated events as a command which has meaning.

Dynamic Component

In graph editor applet, each node has designed as "component" (Fig. 2). The advantage of this design is that the event handling of the node appearance becomes easy. But the node is frequently created or deleted while editing. We have to add each node to event-listeners to obtain occurred event on the node. When new node is added to a certain container, the container issues ContainerEvent which has a link for the new node. The container-event also notifies if a node is removed. The recorder makes each container register a container-listener in advance. Utilize the container-listener, the recorder can add event-listeners to the dynamic component and update the components tree. This updating method is utilized not only by the recorder but also by the player.

Labeling

After all the occurred events are separated to some commands, the analyzer tries to label each command and creates indices. The indices is used for editing and providing abstract of the demonstration. If the command is obvious, for example action-command derives from action-event which has command name, the analyzer can automatically add an index like "[command-name] pressed" or "[command-name] selected." If the command-name is unknown, the recorder must decide it from corresponding events. The recorder may not know how the target system handles the event. For example, the graph applet handles mouse events in complicated manner. We give some rules which enable to identify the command name. These rules are particular to the target applet. The analyzer analyzes mouse events and picks out some attributes. The attributes are classname (pressed, released, passed, generated, removed), coordinates (pressed, released) and so on. The rule is written as a condition including the attributes. If all the rules are satisfied, the command is labeled. The label interpolates some instance information. The instance information is provided by method which returns string or integer value. For example *int getNodeID()* method which is defined in Node class, *String toString()* method which is standard one for returning its own state. Once these rules are created, the developer does not have to write indices for each command.

PLAYING

Method

During the playing session, target system should be changed both its state and view by the player as it is recorded. For example "press CLEAR button" in the graph editor applet performs cleaning action and denting the button.

To change both the state and the view, the player invokes a proper method. Almost all events are subclass of AWTEvent. Then they can be performed by *dispatchEvent()* method in Component class. The *dispatchEvent()* delivers the provided event to the appropriate method.

Source Component

Although the dispatchEvent method can process many types of events, the player have to specify a source component before invoking. Each event has the source component as a

path	name	label
/	GraphApplet	
/0	Panel	
/1	ScrollPane	
/0/0	Button	CLEAR
/1/0	GraphPanel	
/1/0/0	Node	0
/1/0/1	Node	1
/1/0/2	Node	2
⋮	⋮	⋮

Table 1: Component path of GraphApplet

"link." The link points to an address. While the target applet is running, the link is valid. But when it comes to play the stored event, the link becomes invalid. Because the linked source component will be loaded with another address.

To identify the link with the source component, the player must track each component. Two tracking techniques are considered: *tracking by location* and *tracking by component path*. Tracking by location is a technique by matching the event's coordinate with the object's location. Tracking by component path uses component's tree structure.

If we utilize the tracking by location, a problem arizes when the target applet is resized. The resize operation invokes re-layouting. Therefore, re-layouting may move some components. Then the source component tracking by location does not work well.

The container has a component list which keeps its components in adding order. Even if the target applet was re-layouted, the order of the list is not changed. Therefore we adopt tracking by component path. To realize this mechanism, each event keeps a component path when it is recorded. Table 1 shows a part of the component path of GraphApplet. The player searches its own component tree which is generated as well as the recorder. Then each event is executed by the component in the same way as recorded.

The system works well in almost all cases, however the stored event may not send to correct component after changing the structure of components. To prevent this, the player checks the classname of target component before sending. If it differs from the recorded one, the player returns error dialog and stops sending.

Pseudo Mouse Cursor

The demonstration should be displayed as well as someone operates. To attain this purpose, a pseudo mouse cursor should be shown. The pseudo mouse cursor moves and points current attention during the demonstration. The coordinates of the cursor is obtained by a series of mouse events. The recorder captures all mouse events while in the recording session. The player uses the coordinates unless the component position is changed.

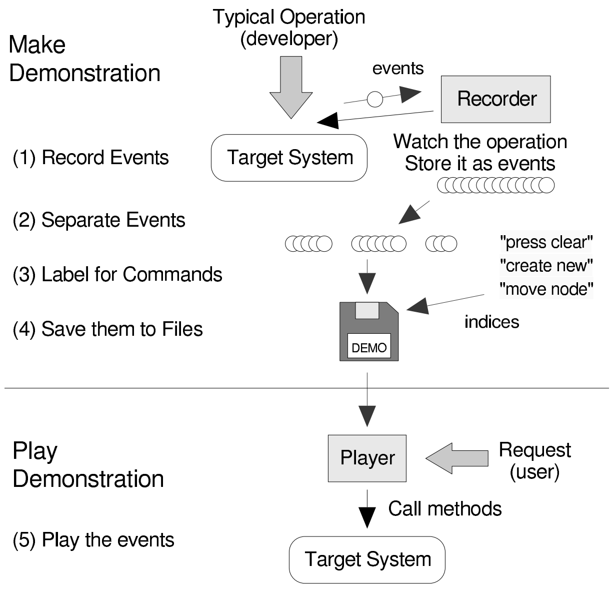


Figure 3: Flow of the help generation

IMPLEMENTATION

In Java environment, an applet runs under Java VM(Virtual Machine). Though we can implement the manager as a customized Java VM, it will not becomes popular among the users.

We have implemented Jedemo (Java Event-driven DEMOnstration) manager which is one of the applet viewer running as an applet. Using this framework, a developer can integrate the target applet with the manager easily. In addition, a user who needs the demonstration can see without any difficulty. The steps to use the demonstration system are summarized in Fig. 3.

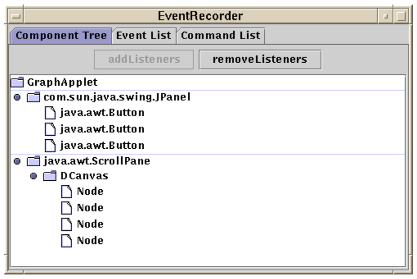
Jedemo Recorder

Fig. 4 shows the ComponentTree tab folder. Pressing "add Listener" inspects the target applet's component structure and shows updated tree view in above. Fig. 5 shows the EventList tab folder which enables developers to record events and to check its contents. Fig. 6 shows the CommandList tab folder which applies command production rules and checks generated commands.

The developer specifies GraphApplet.class as a target applet in applet's parameter as follows. The recorder is executed by appletviewer as a local applet.

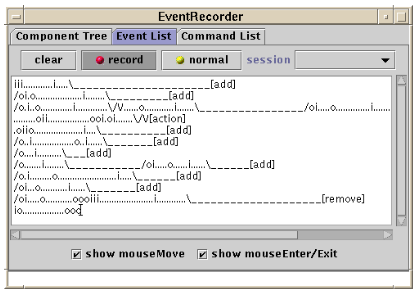
```
<applet code="Recorder.class">
  <param name="target" value="GraphApplet">
</applet>
```

The recorder loads the target applet by name and shows it. The developer pushes "addListeners" button. The recorder inspects the target applet's component tree and shows it. The developer operates the target applet. Then the operation is recorded as Fig. 5. After storing, the developer apply some production rules for separating to commands and labeling



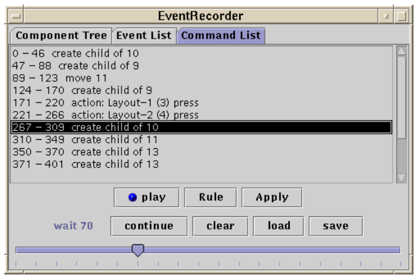
```
GraphApplet
  com.sun.java.swing.JPanel
    java.awt.Button
    java.awt.Button
    java.awt.Button
  java.awt.ScrollPane
    DCanvas
      Node
      Node
      Node
      Node
```

Figure 4: ComponentTree tab folder



```
iii...........i.....\________________________[add]
/oi.o..................i........\___________[add]
/o.i..o.............i.............\/V......o...........i.......\__________________/oi.....o.............i.......
.........oii.................ooi.oi.......\/V[action]
.oiio...................i....\____________[add]
/o..i.................o..i......\_________[add]
/o....i..........\___[add]
/o.......i........\_____________/oi.....o......i......\_______[add]
/o.i.........o........................i.....\_______[add]
/oi...o..........i......\________[add]
/oi.....o...........oooiii......................i...........\_______________________[remove]
io.................ooo
```

Figure 5: EventList tab folder



```
0 – 46  create child of 10
47 – 88  create child of 9
89 – 123  move 11
124 – 170  create child of 9
171 – 220  action: Layout–1 (3) press
221 – 266  action: Layout–2 (4) press
267 – 309  create child of 10
310 – 349  create child of 11
350 – 370  create child of 13
371 – 401  create child of 13
```

Figure 6: CommandList tab folder

each one. The analyzer creates indices from command names. Then the recorder saves both stored events and the indices to a file named "`demo1.jdm`."

Jedemo Player

To publicize the target applet with the demonstration, write an applet tag as follows.

```
<applet code="Player.class">
  <param name="target" value="GraphApplet">
  <param name="demofile" value="demo1.jdm">
</applet>
```

Then the user sees the target applet which is added a help-invoke button. If the user needs the demonstration, press the button. The user can control the demonstration from Controller. Of course the user can also operate the applet normally.

RELATED WORK

For the specific environment, many scripting tools are known. In X environment, Bharat[2] argues how to perform a certain script for X application. In Macintosh, AppleScript[1] generates a script which is executable and editable. AppleScript does not correspond to all application running on Macintosh. Jedemo systems are intended for general Java applets and applications.

There are many applications which record user's operation and use it. The most popular usage for the operation is a macro facility. Metamouse[5] and EAGER[3] have a facility of generating macros from repetitive tasks. Chimera[4] represents macros like comic strip. For code generation, Peridot[6] makes specification of direct manipulation interface from example actions. Such programming by demonstration/example systems are powerful for reducing the complicated operations, but they do not center the instruction use. Jedemo player shows the demonstration as if someone operates.

As the system which has automatic generating help, Cartoonist[7] generates an animated help from UI specification. We work towards the generation of the help demonstration without any specifications. Jedemo manager obtains target system's information from the executable class files only.

CONCLUSIONS

Jedemo recorder enables us to make the general applet's demonstration and store it as events. Jedemo player loads the events and executes target applet with a pseudo mouse cursor. Both Jedemo recorder and player run as an applet, which works like an applet viewer. The developer can show the effective demonstration that he wants to emphasize. The user can understand about the applet. This technique would benefit a lot of people. In future, we plan to improve the interface through which the user accesses the help contents.

REFERENCES

1. Apple Computer,Inc. Introduction to the Macintosh Family — Second Edition.
2. Krishna Bharat, Piyawadee "Noi" Sukaviriya, and Scott Hudson. Synthesized Interaction on the X Window System. Technical report, Graphics and Usability Center, Georgia Tech, USA, 1995.
3. Allen Cypher. Eager : Programming Repetitive Tasks by Example. In *CHI '91 Conference Proceedings*, pages 33–39, May 1991.
4. David Kurlander and Steven Feinter. A History-Based Macro By Example System. In *Proceedings UIST '92*, pages 99–106, 1992.
5. David L. Maulsby, Ian H. Witten, and Kenneth A. Kittlitz. Metamouse: Specifying Graphical Procedures by Example. In *Proceedings SIGGRAPH '89*, pages 127–136, 1989.
6. Brad A. Myers. Creating Dynamic Interaction Techniques by Demonstration. In *Proceedings CHI + GI '87*, pages 271–278, 1987.
7. Piyawadee "Noi" Sukaviriya and James D. Foley. Coupling A UI Framework with Automatic Generation of Context-Sensitive Animated Help. In *Proceedings UIST '90*, pages 152–166, 1990.
8. Sun Microsystems Inc. Java foundation classes, http://java.sun.com/products/jfc/.