A Framework for Event-driven Demonstration based on the Java Toolkit

Motoki Miura and Jiro Tanaka
Institute of Information Sciences and Electronics
University of Tsukuba
Tennodai 1-1-1, Tsukuba-shi, Ibaraki 305-8573, Japan
{miuramo, jiro}@softlab.is.tsukuba.ac.jp

Abstract

An event-driven demonstration is to show the behavior of an application by re-executing the captured events. It is used to provide help regarding how an application works because it can show typical behaviors in an effective way. Such help functions are needed in Java applets, which are carried out by everyone using a Web browser.

In this paper, we describe a framework in which we can execute the event-driven demonstrations of Java applets. We have developed "Jedemo" recorder and player. The Jedemo recorder captures the occurring events while the applet is running. To make the demonstration more efficient and understandable, the developer can add index and messages to the captured events. Then Jedemo player re-executes the captured events showing also the indexes and messages that were added to them.

Jedemo recorder and player suit for almost all applets. They are implemented as applets. The developer can integrate his applet with Jedemo player without any trouble. This framework is helpful for both the developer of the applets and the person who accesses the applets.

Key Words: World Wide Web, internet, animation, programming by demonstration

1. Introduction

The internet technology, especially WWW (World Wide Web) technology is receiving much attention in these days. We can obtain all sorts of information using a Web browser. Hotjava is a new Web browser, which is able to run applets. An applet is a small program written in Java. We can often see various kinds of applets which have interesting functions.

When we operate an applet which has a specific interface, we cannot understand its usage at first. We need some instructions for operating the applet. Usually the instructions are given in "textual" form.

However, when the applet has a graphical user interface (GUI), it is difficult to explain the GUI operations by "text," especially when GUI has the direct manipulation interface. An example of a graphic editor's operation by textual information is given below.

Move objects to other workplace

1. Select objects by mouse dragging.
2. Select [Cut] option from [Edit] menu or click [Cut] icon.
3. Set focus to the destination workplace.
4. Select [Paste] option from [Edit] menu or click [Paste] icon.

When we want to "move" an object, we read the above directions. We need to find the corresponding GUI components; such as [Cut] icon, [Paste] icon, one by one. There is a gap between the "text" and the operated "object."

There are many Web pages which have both the applet and the instructions to operate it. When we want to read the instruction and operate the applet, we need to scroll the page up and down frequently. Shneiderman[8] describes that help by animation is efficient in a limited display.

2. Event versus Video

We suggest to provide help by "demonstration" instead of using "text." By demonstration we mean to show the application as running. Demonstration can be event-driven or image-driven. In event-driven demonstration, the key-point is to record the sequence of events when the application is first executed by the developer. These events are re-executed when a user of the application wants to know how the application works. In image-driven demonstration,

actual images of the application are recorded and re-played as video.

Image driven demonstration is in-flexible because it cannot be changed according to the needs of different users. Unlike this, more information can be added to the recorded events making the event-driven demonstration quite flexible. Event-driven demonstration can be performed without any special arrangement, whereas playing video images always need special programs. In addition, image-data consumes a large disc space. Therefore we prefer event-driven approach for demonstration.

3. Design

Using the event-driven method, the **target system** is combined with the **manager**. The target system represents the application to be demonstrated. The manager, which is composed of a recorder and a player, represents the demonstration tool.

3.1. What Jedemo manager should do?

The recorder is used by the application's developer. The recorder assists the developer to make demonstrations. The player helps the user of the application. The steps to use the demonstration system are summarized in Figure 1.

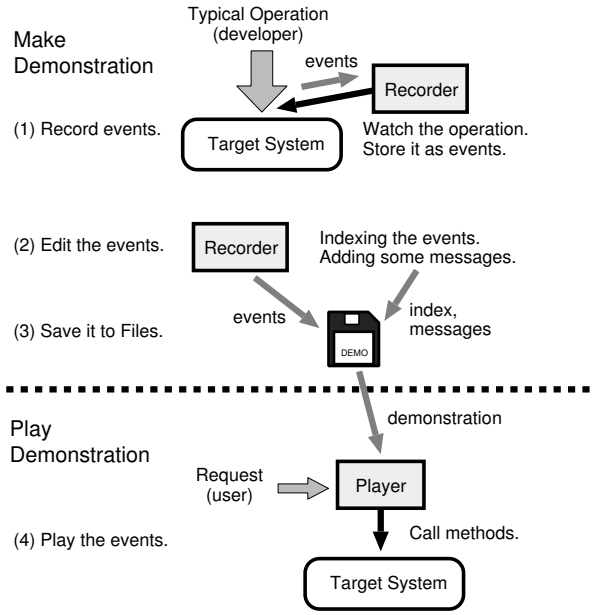

Figure 1. Outline of the system

The recorder captures the events of developer's input. There are two ways for capturing the events. First, the target system sends the events to the recorder. Each component of the target system should tell the recorder what events have occurred in the component. Second, the recorder watches the target system and catches the events. We adopt the latter approach, since the developer does not need to modify the target system. But generally it is hard to catch the events of the target system.

After capturing the events, the recorder generates indices for the events. We can edit the indices and add messages which will be displayed on the demonstration while the events are re-executed.

The player loads the demonstration which includes the stored events, the indices and the messages. While playing, for each event the player calls the method in the target system which would have been called if the event had occurred as a result of a user input. The player also displays a pointer (pseudo mouse cursor), the indices and the messages. The indices are shown as a list. For such a demonstration, displaying the pointer is important because the user cannot understand what is happened without any pointer. The user can make the player skip the demonstration using the indices.

3.2. What the target system should do?

To replay exactly the same behavior, the target system should have an initializing method. The recorder invokes the method before recording events. The player also invokes the method before playing demonstration.

The target system which has already been developed should not be modified. The developer makes the target system as usual. The only thing the target system needs is an initializing method. The target system should not concern whether it is controlled by the manager or not.

4. Event model of target applet

4.1. Components

The developer of an applet uses the class packages such as Abstract Window Toolkit(AWT) or JFC/Swing. There are two types of classes: a component and a container. The component can receive user inputs and perform corresponded actions. The container can add some components and layout them. The container can also add the other container. The container has a list of own components. Using the list, we can know the components information. The developer put together components and containers for programming an applet/application. These components constitute a tree structure. For example, components of TreeApplet (Figure 2) form a tree structure shown in Figure 3.

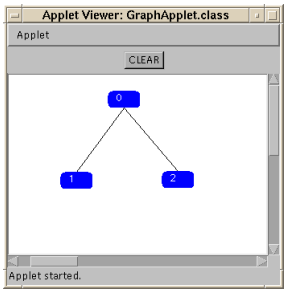

Figure 2. TreeApplet

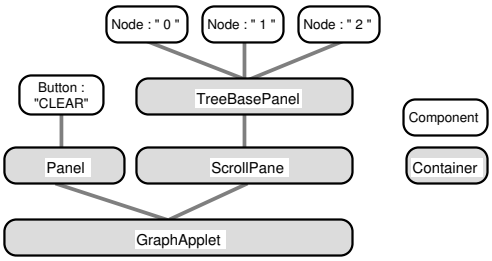

Figure 3. Component tree of the TreeApplet

4.2. Event-listener object

Mouse operation creates mouse-event instances. An instance of mouse-event class, which is capsulated as a low-level mouse operation, can be occurred by all of the components. Each instance of mouse-event is performed by either `processMouseEvent()` or `processMouseMotionEvent()` method in the component.

The component can add some mouse-listener objects which is instances of MouseListener class or MouseMotionListener class. The instance of MouseListener handles press, release, click, enter and exit mouse-event. The instance of MouseMotionListener handles move and drag mouse-event. Each mouse-listener object has methods which handle specific mouse-event. The methods of mouse-listener object are called from the component which generates the event.

The component can add more than one listeners. There are many listeners which correspond to each event-object. For example, an action-listener object performs an action-event which is created when a button is clicked. A container-listener object performs a container-event which is created at the time the container add or remove any component.

The recorder uses listener objects for catching all of the events. The listener objects sends the events to the recorder. While catching events, the recorder adds the listener objects to the components.

4.3. Handling the stored events

Most applets are manipulated with a mouse. Usually, a mouse is used by clicking or dragging for operating each component object. The clicking is used for pressing a button or selecting an object. The dragging is used for moving a scrollbar or specifying an area. These operations consists of a sequence of low-level events. 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. We call the sequence of the mouse events as a stroke. Each low-level event is too complicated to handle. Jedemo manager considers a “stroke” as a set of low-level events. Jedemo recorder creates the strokes which consist of low-level events separated by *Mouse Moved* event (see Figure 4). The advantages of such a hierarchical event are described in [9].

Each stroke is labeled by the recorder when it is decided. In editing, we can change the label for explanation. The player uses the label as an index.

5. For playing

5.1. Search the source component

The player invokes the target applet’s method of a specific component. So the player should specify the component before invoking.

Originally each generated event has a **source component**. For example, an action event which represents a button-pressed action knows which button was pressed. Each event has the source component as a “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 from the running target applet. Two tracking methods are considered: tracking by location and tracking by path.

Tracking by location (Figure 5) is a method by matching the event’s coordinate with the object’s location. Tracking by path (Figure 6) is a method by using an order which is decided when the component is added to the container.

Tracking by location has a problem in case the target applet was resized. The resize operation brings re-layouting. After re-layouting, the location of the components are changed. Then the source component tracking by

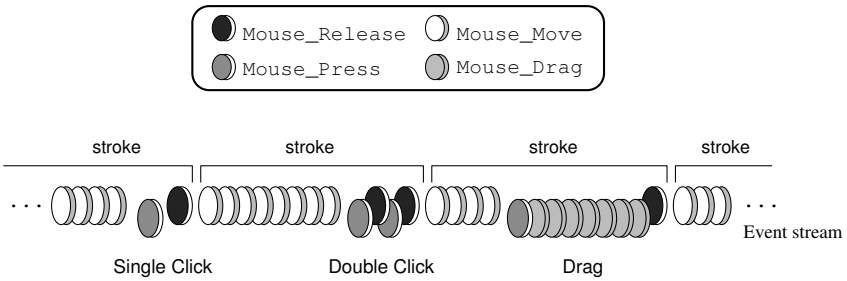

Figure 4. Mouse event-stream and strokes

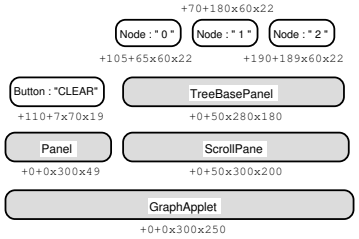

Figure 5. Tracking by location

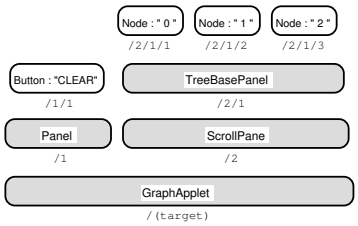

Figure 6. Tracking by path

location does not work well. Even if the target applet was re-layouted, the order of the component does not change.

Therefore we adopt tracking by path. To realize this mechanism, each event need to keep a path when it is recorded. The player can identify the path with the source component.

In the real applet, new component is often created dynamically. For example, node components in TreeApplet are created frequently while editing. The recorder and the player use ContainerListener to realize the changes of the component structure.

5.2. Re-executing

During the playing session, target applet's state should be changed. Furthermore, the target applet's view should be changed as it is recorded. For example "press CLEAR button" performs cleaning action and denting the button.

To change both of the state and the view, the player invokes `processEvent()` method with the stored event. The `processEvent()` method calls proper method in compliance with the event. MouseEvent invokes `processMouseEvent()` method. KeyEvent invokes `processKeyEvent()` method. These methods call all registered listener object. So the target applet's state and view will be changed.

The `processEvent()` method can process all types of events. However, the `processEvent()` method is a private method so that it cannot be called from the other class by default. When we want to call this method, we make subclass of the built-in component. To perform it another way, the player call high-level method. For example JButton class, which is a subclass of AbstractButton, has `doClick()` method. This method performs a "click" programmatically.

When it comes to perform a button-pressed event, the player calls such a high-level method by name. The recorder prepares proper method name from the stored events.

6. Implementing Jedemo manager

6.1. Design form of events

In the previous section, we described that the player needs to know both a source component and a method name which corresponds to stored events for invoking. The information belongs to a "primitive" class. The Primitive class which represents a low-level event has the following fields.

path represents a location of base container.

id is a component order for identification.

event stands for stored low-level event.

method keeps a name of the invoking method.

xy is a coordinate of the pointer. We can obtain the coordinate by mouse-event.

shape means a pointer's form.

skip is boolean field whether the event can skip or not.

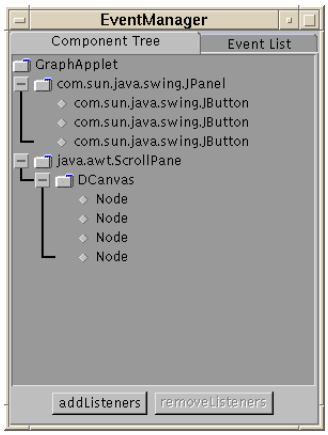

Figure 7. Component-Tree tab folder

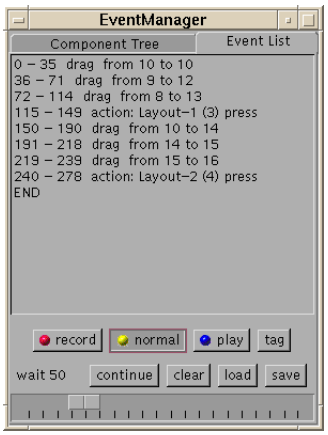

Figure 8. EventList tab folder

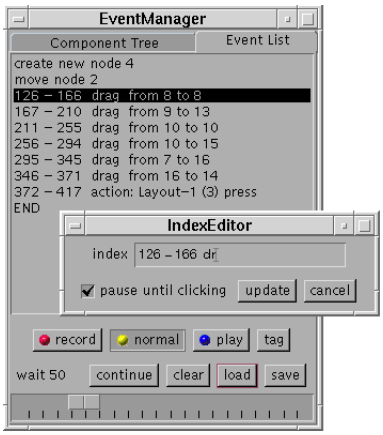

Figure 9. Index Editor

message is displayed for assistance.

Path, id and classname are used for identifying the source object. Skip is used when we want to jump to the index. Message is shown in playing.

Stroke class has an array of primitives and an index. The index can be written by the developer. The recorder writes a temporary index when the primitives are decided.

6.2. Jedemo manager

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 become popular among the users.

We have implemented Jedemo (Java Event-driven DEMOnstration) manager which is one of the applet viewer running as an applet. Jedemo manager has functions of recording, editing and playing the demonstration. 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.

Figure 7 shows the ComponentTree tab folder which can add/remove event-listeners and inspect the target applet's component structure. Figure 8 shows the EventList tab folder which can record events and check its contents and play it.

In the EventList tab folder, we can also control running speed and save/load the stored events. The event list has indexes of the strokes. By double-clicking the index, we can skip the demonstration at that point. To skip the demonstration at any point, the player initializes the target applet and processes events which cannot skip, with minimum interval. We can also edit the index by IndexEditor (Figure 9).

6.3. Target Applet

To make sure the manager works well, we have implemented a graph editor applet. The applet adopts direct manipulation interface for editing a graph. The applet can layout the nodes by clicking button. In the applet, node dragging operation 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. ButtonPress in parent node and ButtonRelease in child node brings creation of new link. To delete the link, press on the child node and release on the parent node.

6.4. Example

The developer specifies the target applet (class file is GraphApplet.class) by applet's parameters in HTML document as follows.

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

Jedemo manager 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. After the operation, the developer can confirm and edit the demonstration. When the editing is finished, saves the stored events to a file named "`example1.jdm`." To publicize the target applet with the demonstration, write an applet tag as follows.

```
<applet code="JedemoPlayer.class">
  <param name="target" value="GraphApplet">
  <param name="demofile" value="example1.jdm">
</applet>
```

Then the user sees the target applet as Figure 10. If the user needs the demonstration, press “start DEMONSTRATION” button. Of course the user can also operate the applet normally.

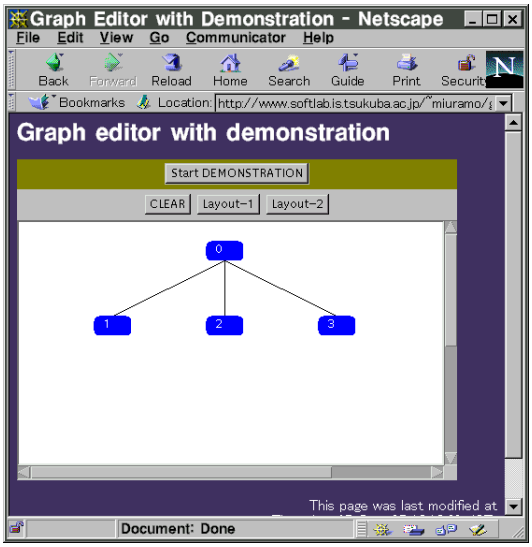

Figure 10. The publicized applet

7. Related work

We often see the applications which record the user’s action and use it. Macro is popular in editing tools which may be operated repetitive tasks. Metamouse[3] and EAGER[2] detect a repetitive task and generate macros. Chimera[5] shows the generated macros with visual representation. 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 manager shows the demonstration as if someone operated.

Bharat[4] argues what is needed for the X window system to perform a certain script language. In Macintosh, AppleScript[1] generates a script which is executable and editable. Jedemo manager is intended for general Java applets and applications.

Cartoonist[7] generate an animated help from UI specification automatically. We work towards the generation of the help demonstration without any specifications. If the target system has been developed, Jedemo manager can obtain occurred events.

8. Conclusions

We have implemented Jedemo recorder and player. 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 pointer. Both Jedemo recorder and player run as applet, which work like applet viewer. The developer can show the effective demonstration that he wants to emphasis. The user can understand about the applet. This technique would benefit a lot of people.

9. Future work

Jedemo manager is helpful for both the developer and the user. The editing function is important for the developers. We have to provide more useful environment for editing.

References

[1] Apple Computer,Inc. Introduction to the macintosh family — second edition.

[2] A. Cypher. Eager : Programming repetitive tasks by example. In *CHI ’91 Conference Proceedings*, pages 33–39, May 1991.

[3] I. H. David L. Maulsby and K. A. Kittlitz. Metamouse: Specifying graphical procedures by example. In *Proceedings SIGGRAPH ’89*, pages 127–136, 1989.

[4] P. Krishna Bharat and S. Hudson. Synthesized interaction on the x window system. In *Technical report 95-07*. Graphics and Usability Center, Georgia Tech, USA, 1995.

[5] D. Kurlander and S. Feinter. A history-based macro by example system. In *Proceedings UIST ’92*, pages 99–106, 1992.

[6] B. A. Myers. Creating dynamic interaction techniques by demonstration. In *Proceedings CHI + GI ’87*, pages 271–278, 1987.

[7] Piyawadee”Noi”Sukaviriya and J. D. Foley. Coupling a ui framework with automatic generation of context-sensitive animated help. In *Proceedings UIST ’90*, pages 152–166, 1990.

[8] B. Shneiderman. *Designing the User Interface 2nd edition*. Addison-Wesley, 1992.

[9] D. S.Kosbie and B. A. Myers. Extending programming by demonstration with hierarchical event histories. In *Proceeding of East-West International Conference on Human-Computer Interaction EWCHI ’94*, 1994.