Event Dispatch

Review: Events

- Event: noun: a thing that happens, especially one of importance. Example: the media focused on events in Egypt
- Event: a structure used to notify an application of an event's occurrence
- Examples:
  - Keyboard (key press, key release)
  - Pointer Events (button press, button release, motion)
  - Window crossing (mouse enters, leaves)
  - Input focus (gained, lost)
  - Window events (exposure, destroy, minimize)
  - Timer events

Review: Why do we need them?

- Users have lots of options in a modern interface
- Need a uniform, well-structured, way to handle them
- Need to be able to handle any event, including those that aren't appropriate given the current state of the app
  - eg: clicking on a button that is currently disabled

3

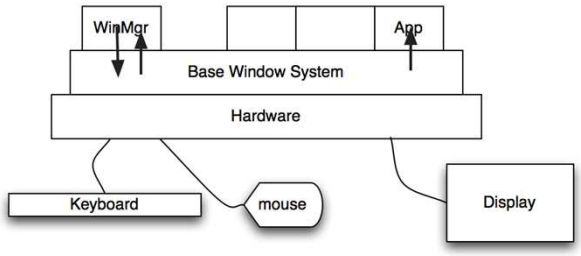
Review: Role of the BWS

- Collect event information
- Put in a known structure
- Order the events by time
- Decide to which application/window the event should be dispatched.
- Deliver the event.

4

Review: BWS: Collecting Events

- Some events come from the user via the underlying hardware; some from the window manager.



5

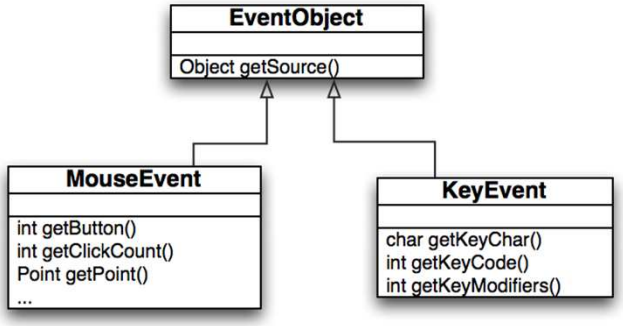
Review: Known structure : X

- X uses a C union
  - typedef union {
  - int type;
  - XKeyEvent xkey;
  - XButtonEvent xbutton;
  - XMotionEvent xmotion;
  - // etc.
  - }
- Each structure contains at least the following
- typedef struct {
  - int type;
  - unsigned long serial; // sequential #
  - Bool send_end; // from SendEvent request?
  - Display* display;
  - Window window;
  - } X___Event

6

Review: Known Structure: Java

- Java uses an inheritance hierarchy
- Each subclass contains additional information, as required (not shown)



7

Review: Event Loop

```
XEvent event;

while( true ) {
        XNextEvent( display, &event );
        switch( event.type ) {
                case ButtonPress:
                        if (event.xany.window == xInfo1.window)
                                cout << "Got button press in window 1!\n";
                        else if (event.xany.window == xInfo2.window)
                                cout << "Got button press in window 2!\n";
                        break;
                case KeyPress:
                        if (event.xany.window == xInfo1.window)
                                cout << "Got key press in window 1!\n";
                        else if (event.xany.window == xInfo2.window)
                                cout << "Got key press in window 2!\n";
                        break;
                case Expose:
                        if (event.xany.window == xInfo1.window)
                                repaintWindow1(xInfo1);
                        else if (event.xany.window == xInfo2.window)
                                repaintWindow2(xInfo2);
                        break;
        }
}
```

8

Event Dispatch

- Event Dispatch: How do we get the correct code to execute in response to an event?
- Questions:
  - Which widget?
    - Positional dispatch
      - Bottom-up dispatch
      - Top-down dispatch
    - Focus dispatch
  - How do we invoke the code?

9

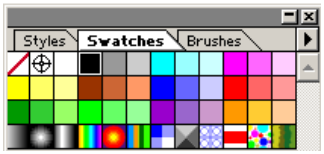
Dispatch: Positional

- Basic strategy: Send input to the component the mouse is over.
- Primary issue: Components can overlap, so which one should receive the event?
  - Bottom-up
  - Top-down

10

Positional: Bottom-up

- Leaf node (that contains the mouse) in the interactor tree receives the event
  - Can process the event itself
  - Send the event to its parent (who can process it or send to its parent...)



- Why send to its parent?
  - Example: A palette of colour swatches may implement the colours as buttons. But palette as a whole needs to track the currently selected colour. Easiest if the palette deals with the events.
- Strategy is applicable to architectures where the BWS knows everything about each component (eg heavy-weight toolkits).

11

Positional: Top-down

- Highest level node in the interactor tree (that contains the mouse) receives the event.
  - Can process the event itself.
  - Can pass it on to a child component.
- In bottom-up dispatch, the BWS does this to determine which of the leaf nodes to deliver the event to.
- Most applicable for light-weight widget toolkits

12

Top-down vs. Bottom-up

- When do these behave the same way?
- Advantages of top-down?

13

Positional Dispatch Limits

- Positional dispatch is sometimes inappropriate.
  - Send keystrokes to scrollbar if mouse over the scrollbar?
  - Mouse starts in a scrollbar, but then moves outside the scrollbar. Send the events to the adjacent component?
  - Mouse button press event in in one button component but release is in another. Each button gets one of the events?
- Conclusion: Sometimes we need to give one control the "focus".

14

Dispatch: Focus

- When a control has the focus
  - Events go to that control, regardless of mouse position.
- Need to pay attention to focus for both keyboard and mouse events:
  - Mouse down on a button, move off, release (mouse focus)
  - Click on a text field, move mouse off, start typing (keyboard focus)
- Only one widget should have control at a time
- Need to gain and lose focus at appropriate times
  - Transfer focus on a user click
  - Transfer focus on a tab

15

Dispatch: Focus

- Even though a component has focus, it should not necessarily receive every event:
  - Must be able to click on another control to change focus
  - Paint/damage events not necessarily associated with the component that has focus
    - Example: moving a slider has an effect on some other component which is repainted

16

Dispatch: Focus

- Conclusions:
  - Mouse-down events: direct to component under cursor
  - Input events go to the component with focus; other events may go elsewhere.
  - Often helpful to have an explicit focus manager in a container component to manage which component has the focus.

17

Dispatch: Accelerator Keys

- Accelerator Keys provide two ways to invoke the same functionality:
  - Via the keyboard: Seems natural to send them to the component with keyboard focus
  - Via the menu: Click on the menu and it gets the focus. So now who gets the command?
- Have two places in code that do the same thing?
- Alternative:
  - Menus register keyboard accelerators with specific menu items.
  - The GUI toolkit intercepts accelerators and forward to the appropriate menu to be handled

18

Event Delivery

- An event happens…
- The toolkit decides which component it should be dispatched to.
- Now, how do we actually deliver it? How do we structure our GUI architecture to deliver the event information to the code that should handle it?
- Lots of approaches -- X and a case statement is just one.
- Criteria to judge alternatives:
  - Easy to bind event to code
  - Clean, easy to understand what happened and why
  - Good performance

19

BWS: Delivery Options (1/3)

- Nested Case Statements (X, Original Mac)
  - Usually used nested case statements (as above). The outer case statement to select the window and the inner case to select the code to handle the event.
- Event Tables (GIGO from Sun)
  - Each window has an event table, consisting of addresses of C procedures that should be called for a specific event. Index the table based on event type and call the procedure found there. Fill tables with default values.
- Callbacks (Xtk, Motif)
  - Similar to event tables, but distributed to individual components.

20

BWS: Delivery Options (2/3)

- WindowProc: MS Windows
  - Each window has just one callback, called a WindowProc. The WindowProc uses a case statement to identify each event that it needs to handle. There are over 100 standard events. Rather than handling all of them, it can delegate to another WindowProc.
- Subclassing: Java 1.0
  - BWS directs events to the component in which it occurs. That component inherits from an abstract class, overriding any methods it needs to modify to handle the event.

21

BWS: Delivery Options (3/3)

- Listeners: Java 1.1 and later
  - Register objects implementing a specific interface with the component. Appropriate method in each object is called when event occurs.
- Delegates: .NET
  - Only one of the methods in a listener's interface is called. Why not provide just a method?

22

Inheritance Doesn't Scale Well

- Reserve subclassing for extending class's functionality.
- Muddies separation between application model and GUI because app code is integrated directly into subclassed component.
- All event types are processed through the same methods: complex, error-prone.
- No filtering of events.
- If the same subclass is used for multiple widgets, they are often distinguished with button labels: hard to localize.

23

Delivery: Listeners

- Use the Strategy design pattern to factor out the behaviour unique to a particular UI component.
- Provide the component with one or more objects implementing a particular interface (set of methods). When the event occurs, the relevant method in the listener objects are called.
- Reference for Java event/listener architecture (and critique of inheritance-based architecture):
  - http://java.sun.com/j2se/1.3/docs/guide/awt/designspec/events.html

24

Inheritance vs. Listeners 1

- Delivering events by overriding methods (inheritance) leads to a huge class tree or convoluted code
  - Every button must be subclassed to respond to clicks
  - Everything else about the button remains the same
  - Alternative: overridden methods include a switch to distinguish code for many different button instances
- Inheritance does not lend itself to maintaining a clean separation between the application model and the GUI.
- No filtering of events; every event is delivered, resulting in performance issues

25

Inheritance vs. Listeners 2

- Listener approach factors out the behaviour that is unique to each application
  - Application provides an object implementing the particular listener interface and the code needed for a particular button
- This is a common approach in UI toolkits:
  - Delegate customizable, application-specific functionality to configurable run-time objects.
  - Next step?

26

Listeners: Adapter pattern

- Many listener interfaces have only a single method; others have more.
  - WindowListener has 7, including
    - windowActivated(WindowEvent e)
    - windowClosed(WindowEvent e)
    - windowClosing(WindowEvent e)
- Typically interested in only a few of these methods. Leads to lots of "boilerplate" code.
- Each listener with multiple methods has an adapter with null implementations of each method. Simply extend the adapter, overriding only the methods of interest.

27

Adapter

```
JFrame f = new JFrame();

f.addWindowListener(new WindowListener() {
        public void windowClosed(…) {}
        public void windowClosing(…) {
                System.exit(0);
        }
        public void windowActivated(…) {}
        public void windowDeactivated(…) {}
        // and 6 others
});
```

```
// Compare to:
f.addWindowListener(new WindowAdapter() {
        // Just override the method
    // we're interested in
        public void windowClosing(…) {
                System.exit(0);
        }
});
```

28

BWS: Delivery: Delegates & .NET

- .NET designed by Microsoft
- Allegedly intended to be cross-platform, but architecture, conventions clearly rooted in Windows
  - Example: Very easy to use native libraries compared to Java (using P/Invoke), but mechanisms not designed with cross-platform use in mind (no generic method of loading dynamic libraries)
- But, still a number of significant improvements in basic architecture of the VM, core system, and C# language
  - Many improvements noteworthy for building GUIs

29

C# and .NET

- C# and .NET architecture very, very Java-esque, but with more syntactic sugar
  - And more liberal use of Capital Letters!
- Once you know Java and Swing, C# is easily learned
- Example:
  - Java: System.out.println("CS is the best program ever!");
  - .NET: System.Console.WriteLine("No, SE is the best ever!");

30

.NET + Mono

- Mono an open source implementation of C# and .NET by Novell and recently taken over by Xamarin
  - GPL, LGPL, and MIT licenses
- Mono 2.0.1 includes WinForms compatibility (basic GUI system in .NET)
  - Most basic .NET GUIs will work in Mono
- Now at version 2.10

31

Responding to Events in .NET

- Rather than listeners, C#/.NET uses delegates
- Delegates an elegant form of broadcasting/subscribing to events

32

Delegates

- Three components:
  - Definition of a delegate type
  - Declaration of a delegate instance
  - One or more methods assigned to the delegate
- Definition of delegate type defines a method signature
- Delegate instance maintains a list of references to methods with that method signature
- Delegate instance can then be invoked to call those methods

33

Delegates Example

```
using System;
using System.IO;
public delegate void Logger(string s);
public class DelegateDemo
{ static StreamWriter LogFile;
  public static void FileLogger(string s){
        LogFile.WriteLine("Error: " + s);
  }
  public static void StdErrLogger(string s){
   System.Console.Error.WriteLine
            ("Error: " + s);
}
```

```
public static void Main() {
    LogFile = new
          FileInfo("Log.txt").AppendText();
    Logger log = null;
    log += FileLogger;
    log += StdErrLogger;
    log += (s) => System.Console.WriteLine
                  ("Error: " + s);
    log("Oops!");
    log("Oh, no!");
    LogFile.Close();
  }
}
```

34

Delegates Example

- (s) => System.Console.WriteLine(s); is a lambda expression
- log will refer to FileLogger, StdErrLogger, and the lambda expression
- Will invoke all of the methods when called
- The result (if there is one) returned is the result of the last method added to the delegate
- Methods can be removed using the -= syntax

35

Events in .NET

- Events in .NET are an extension of delegates
- Declare an "event" instance instead of a "delegate" instance:
- public event Logger d;
- "event" keyword allows enclosing class to use delegate as normal, but outside code can only use the -= and += features of the delegate
  - Gives enclosing class exclusive control over the delegate
  - Outside code can't wipe out delegate list (e.g., "myObject.d = null")

36

```
using System;
public delegate void Logger(string s);

public class MyClass {
   // add/remove "event" to prove the point
   public event Logger log = null;
}

public class OtherClass
{
  public static void StdErrLogger(string s)
      { System.Console.Error.WriteLine("Err:"+s); }
   public static void Main() {
      MyClass c = new MyClass();
      c.log += StdErrLogger;  // allowed
      c.log = StdErrLogger;   // not allowed
      c.log("Oops!");        // not allowed
   }
}
```

37

```
using System;
using System.Windows.Forms;
```

Example

```
public class HelloWorld : Form
{
   private void HandleClick(object source, EventArgs args)
   {  System.Console.WriteLine("Got button click.");
   }

   public HelloWorld()
   {  Button b = new Button();
      b.Text = "Click Me!";
      b.Click += HandleClick;
      this.Controls.Add(b);
   }

   public static void Main()
   {  Application.Run(new HelloWorld());
   }
}
```

38

Event Queues Revisited

- Java uses listeners to inform components
  - Does it still have an event loop?

39

Java Event Queue

- Available from java.awt.Toolkit:
  - Toolkit.getDefaultToolkit().getSystemEventQueue()
- java.awt.EventQueue
  - Methods for:
    - Getting current event, next event
    - Peeking at an event
    - Replacing an event (push())
    - Checking whether current thread is dispatch thread
    - Placing an event on the queue for later invocation

40

Awareness Systems

- Latching into event queue allows applications which are more “aware”: Can change behaviour based on degree of activity in the interface
- Provides more nuanced types of interaction
- Some examples of this?

41

Awareness Systems

- IM clients, screensavers can make use of raw event queue by monitoring “activity”
- When activity drops, can do something
  - IM client: Set state to “away”
  - Screensaver: Start screensaver
- Issue: Windows allows any application access to global event queue through windows hooks
  - Implications with this?

42

Security

- Open access to global event queue is an enormous security risk
- Enables keyboard loggers, without user's awareness
  - Trivial to implement

43

Event Queues and New Input

- Event queue best suited for discrete, low frequency events
  - Breaks down for rich sensor input of high frequency or high bandwidth
  - Example: Pen input

44

Recap

- How events get delivered to application
- How application delivers events to components
- How components receive and act on events
- Next up:
  – Design Process/Custom Controls

45