CS 160: Lecture 11

Professor John Canny
Spring 2004
March 3
Finish windows and events

Platforms:
- PC (Viz Basic, Java)
- Web (HTML, Javascript)
- Windows CE
- BREW
- SVG
All generated events go to a single event queue
  * provided by operating system
  * ensures that events are handled in the order they occurred
  * hides specifics of input from apps
Widgets

- Reusable interactive objects
- Handle certain events
  * widgets say what events they are interested in
  * event queue/interactor tree sends events to the “right” widget
- Update appearance
  * e.g. button up / button down
Widgets (cont.)

- Generate some new events
  - “button pressed”
  - “window closing”
  - “text changed”

But these events are sent to interested listeners instead
  - custom code goes there
Main Event Loop

while (app is running) {
    get next event
    send event to right widget
}

Mouse Software
Keyboard Software

Events

Display Screen
- "F:cs160\Public" window
  - title bar
  - horizontal scroll bar
  - contents area
  - "CDJukebox" folder
  - "Home Ent..." folder
  - ...
  - "Web Newspaper" window
  - ...

Source Code
---
package edu.berkeley.guir.lib.satin;
import java.awt.*;
import java.awt.event.*;
import edu.berkeley.guir.lib.satin.objects.*;

/**
 * Satin constants.
 *
 * This software is distributed under the
 * <A HREF="http://guir.cs.berkeley.edu/projects/COPYRIGHT.txt">...
 *
 * @version SATIN-v2.1-1.0.0, Aug 11 2000
 */
public interface SatinConstants {

  //=================================================
  //==   GLOBAL SATIN PROPERTIES   =====================
  //=================================================
  /**
   * The name of Satin's properties file. Assumed to be in the current
   * directory, from which Satin is started (via the java interpreter).
   */
  public static final String SATIN_PROPERTIES_FILENAME = "satin.properties";

  //=================================================
  //==   STYLE PROPERTIES   ===========================
  //=================================================
  // If you add any new Style properties, be sure to update the
  // Style.java file too.
  public static final String KEY_STYLE_FILLCOLOR        = "FillColor";
  public static final String KEY_STYLE_FILLTRANSPARENCY = "FillTransparency";
  public static final String KEY_STYLE_MITERLIMIT       = "MiterLimit";
  public static final String KEY_STYLE_DASHARRAY        = "DashArray";
  public static final String KEY_STYLE_DASHPHASE        = "DashPhase";

  //=================================================
  //==   STYLE PROPERTIES   ===========================
  //=================================================

} // of interface

//==============================================================================

Copyright (c) 2000 Regents of the University of California.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditionsare met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
   Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
   Neither the name of the copyright holders nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.
   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS"
Callbacks

callbacks: every widget registers itself to receive certain events with the OS.

dangers: UI code using callbacks is usually single-threaded. Control does not return to the event loop until the callback returns. Therefore in a callback routine, never:

* Wait for another event.
* Sleep.
* Perform I/O or other actions that may block.
Threaded code

- **Multi-threaded code** uses distinct threads for distinct widgets.
- It allows blocking operations (e.g. I/O) to be confined to particular threads that don’t affect interactivity.
- If your program has any time-intensive parts, they should run in different threads from the UI code.
Threaded code

Use separate threads for any operations that can occur asynchronously:

* UI interactions.
* File operations - use separate threads if you need to be updating several files at the same time.
* Inter-process communication (sockets): use one thread for each connection.
* Use a thread for each other I/O device, e.g. one each for reading from or writing to the sound card.
Inter-thread communication

- The window system and running process in the OS communicate using message passing:
  * The event queue and sockets are examples of message-passing primitives.

- Processes in the same address space (i.e. within your program) can use shared memory (i.e. shared variables) which is much more efficient.
Synchronization

- Shared-memory communication poses challenges. If you rely on “mailbox” primitives, things can go wrong:

Flag to show this thread is writing new data

Data
Synchronization

Intuitively, threads that want to write should:

0
<blank>

Flag to show this thread is writing new data

Data

wait until thread_id = 0;
write thread_id;
write data;
Synchronization

But thread switching can happen anytime, e.g.

wait until thread_id = 0;
write thread_id;
write data;

Flag to show this thread is writing new data
Synchronization

Flag to show this thread is writing new data

Data

A switch between checking the flag and setting it allows both threads to (incorrectly) write the flag and their data.

To prevent this, we define **critical sections** of the code that cannot be interrupted.
Synchronization

Flag to show this thread is writing new data

Data

0 <blank>

E.g. the critical section in the example is:

....
wait until thread_id = 0;
write thread_id;
write data;

{ Critical section, thread can’t be pre-empted. }
Rather than many critical sections in your code, you can use a single semaphore primitive.

A semaphore is initialized to an integer n, and has just two operations:

- test(); wait until semaphore > 0, then decrement it.
- increment(); increment the semaphore.
Semaphore example

Then the example code becomes (with sem1 initialized to 1):

```java
    ....
    sem1.test();
    write thread_id;
    write data;
    sem1.increment();
    { This section cannot be pre-empted by the other process that is using this semaphore. }
```
Break
Interactor Tree

- Decompose interactive objects into a tree
  * interactive objects also known as “widgets”
  * based on screen geometry of objects
  * nested rectangles
- Used for dispatching events
  * events are dispatched (sent) to code in widget
  * the code then handles the event
- Variety of methods for dispatching events
  * return to this later
Interactor Tree 1

Display Screen

- “F:\cs160\Public” window
  - Inner Window
    - title bar
    - horizontal scroll bar
    - contents area
      - “CDJukebox” folder
      - “Home Ent…” folder
      - ...
    - size control
      - ...
  - “Web Newspaper” window
    - ...

3/3/2004
Interactor Tree 2

Display Screen

- Outer Win [black]
- Inner Win [green]

Result Win [tan]
- Result String

Keypad [Teal]
- = button
- - button
- + button
- 0 button

93.54
Who gets the events?

- To catch events, a widget registers a “listener” for that event
  - Mouse click
  - typed input
  - drag...

- Events go to a widget containing the pointer
Who gets the events?

But there are often several widgets containing the pointer.

Events go down the “stack” of visible widgets at the pointer until there is an active widget that has registered a listener for that event.
Interactor Tree (Java)

Display Screen

Frame [black]
Panel [green]

Mouse click listener

Text Entry [tan]
Result String

Keypad Panel [Teal]

Button("=") Mouse click listener
Button("-") Mouse click listener
Button("+") Mouse click listener
Button("0") Mouse click listener

93.54
Platforms - PC

For regular PC development, the options are:

- C++/VBasic (Visual Studio)
- Java
- Flash
- SVG
- Rapid prototyping: Suede, Silk, Satin (see gui.r.berkeley.edu/projects)
Platforms - Web

For web development one of the main issues is portability. Before designing your app, think about browsers for your user group.

There is a lot more than IE and Netscape:

- Mozilla/Opera
- AOL: huge community, many versions with limited browsers
- Old versions of IE and Netscape
Web standards

- Unfortunately, HTTP is a non-standard. The current version is HTML 4 (1997), but no browsers fully support it.

- Microsoft seems to have given up on HTML 4 in 1998.

- Reasonable support for HTML 4 in Netscape 7 and Mozilla. (but tables are different in most browsers)
Web standards

- For portability, it’s best to stay with HTML 3.2

- Javascript is the most portable script. But you’ll probably still need browser-specific code (e.g. CS160 lecture page).
PDAs

- Two options for native development-
  - MS Embedded Visual Tools 3.0 (VB and C++) - includes emulators for all platforms (download).
  - MS Visual Studio .NET (huge!) includes tools for XML exchange. - Ask william@eecs for CDs
- Java: Chai VM for HP Jornadas etc.
  - Usually well behind PC Java - no Jini support
- Flash: Interesting choice for small devices, better use of limited screen space, but check functionality (esp. script execution).
Cell phones - BREW

- BREW is Qualcomm’s “Binary Runtime Environment for Wireless”

- Something like the WIN32 API, but smaller. BREW 2.0 includes support for:
  - GPS-one - though no providers yet
  - Can get GPS info through serial port on emulator PC.
New standards - XML

Fortunately, the situation looks better in future. XML is going to become the standard for web info exchange.

XML provides data exchange, and complementary standards control formatting - XSL and XHTML.

Good support in Mozilla, also IE and Netscape.
There are two standards for 2D graphics:

- VML (old) promoted by Microsoft - static 2D graphics, available in MS IE now. (example)

- SVG (new) dynamic 2D graphics, the latest W3C standard. No browser support natively, but plug-in available (Adobe), and custom builds of Mozilla support it.
SVG

SVG is already a big part of mobile UI development: the W3C requires it for multi-media messaging (MMS) on phones in their latest standard.

SVG doesn’t have “widgets” per se, but it’s easy to write them in SVG.

Very good for thin clients, or for DENIM-style interactive prototypes.
SVG Widgets

- See
  http://www.mycgiserver.com/~amri/widget.cocoon.xml
  for the kinds of widgets you can build or use.

- Many functions in SVG are directly available using the “set” attribute.

- Others are available via Javascript.
SVG Authoring

- Limited set of tools:
- Authoring static SVG is possible with many tools, especially Adobe Illustrator.
- Check your favorite tool for a “save as” SVG option.
- For dynamic SVG, Webdraw is available as a free, limited-time trial, and supports animation.
- The best tool is XStudio, which has a limited-function trial.