The SquidKit Windowing Architecture

Ronald J. Mann
Sr. Staff Engineer
Sun Microsystems Inc.
ron@east.sun.com
Window Woes

- The WORA promise
- Human Interface Design
  - View drives architecture (Policy vs. Mechanism)
  - Developers are people too!
- A single framework would ease use, maintenance
- Is a single GUI paradigm for Java Possible?
  - AWT: Too dependant on native functionality
  - Swing: Lots of Features, too big
  - Profiles: Just give up on WORA
  - Html/WAP/Etc: Not feature rich
  - Toolkit Model: Procedural, fixed, largely static
- New devices + Internet = Need for new Paradigm
SquidKit

Scalable Customizable GUI Development Kit

- Portability
  - WORA

- Flexibility
  - Eliminate the procedural remnants of the past; Provide a true OO model
  - Window DNA

- Scalability
  - Fixed collections of objects can’t so the job…Kill the Toolkit!
Most visual interactive devices are created equal (kind of)
  – Provide for Events
  – Are Pixel Addressable
  – Have at least one user-accessible clipped Rectangle

Exploit Commonality, Stop dwelling on individuality

HI should not drive infrastructure design

Toolkit model is inflexible; “Mechanism, Not Policy!”
OTOH...

- Targeted Toolkits can compete with native facilities
- Performance can be higher
- Footprint (in the ME space) can be lower
- We need both, at least until resources are limitless
Squid Architecture

- SquidKit est omnis divisa in partes tres
- Core Layer (Portability)
  - Native Bridge Interface
  - BasicWindow Abstraction
  - Event Model
- Window Construction Kit (Flexibility)
  - Windowing DNA
- Octopus (Scalability)
  - IDE Interface Layer
History

- A Mistake Repeated
  - Expedience forces reliance on high-level constructs
    - WABI
    - AWT
  - Mann’s Law ;-)  
    - “The level of cross platform compatibility achievable by any application is inversely proportional to its reliance on native functionality.”
Core Layer

- Abstracts Fundamental Types
- Eliminates reliance on Native State
- Standardizes a common Native Interface
- Standardizes Events
  - Delivery Order
  - Hide events necessary to platform
Core Layer: Native Bridge

- Event Abstraction
- Window Abstraction
- Screen Abstraction
- Prototype has 16 Entry Points!
  - Init
  - GetEvent, DispatchEvent, Event Masking
  - Window Manipulation (Create, Destroy, Geometry)
  - Need to add D&D, Font
Core Layer: Proto Entry Points

boolean initializeNative();
boolean getEventNative( CoreEvent e );
void dispatchEventNative( CoreEvent e );
void setEventMaskNative( long mask );
long getEventMaskNative();
long createWindowNative( WindowAttributes wa );

void destroyWindowNative( long window );
public void minimizeWindowNative( long window );
public void showWindowNative( long window );
void setGeometryNative( long winHandle, int x, int y, int w, int h);
void setStackingOrderNative( long win, long winStack, int flag );
void setVisibleNative( WindowAttributes wa );
void setCursorNative( Cursor c );
boolean openDisplayNative( ScreenAttributes sa );
boolean openDisplayNative( ScreenAttributes sa, String s );

void getDesktopRectNative( Rect r );
Core Layer: Windows

- BasicWindow (Abstract Class)
  - CoreWindow
  - Screen (Device)
  - Parenting
  - Default Event Handling
  - Region, Geometry Manipulation
  - Thread Issues

- WindowAttributes
  - Encapsulates Persistent State
  - Geometry, Visibility, State (Enabled, modality) etc.
  - Listeners

- Décor (Decorations)
Core Event Class
- System, Info, Action, User, Native
- System contains abstracted native info
- Data retrieved on message receipt

Traditional Get/Dispatch Methodology
- Eliminate Async delivery on random threads
- Events delivered ONLY to Window’s owner thread (can migrate)
Core Layer: Event Queues

- **System Event Queue**
  - Repository for Events
  - Guarantee Delivery Order
  - Preprocess Event for Delivery

- **Application Event Queue**
  - One per thread SquidKit Instance
  - Each Window has an owner thread
  - Get/Peek/Dispatch/Send constructs
  - Pre/Post Event Listeners
Core Layer: Graphics

- Graphics Interface
  - Bridge to graphics services
  - Minimalist approach
  - Java vs. Native implementation

- Tiered Model
  - Renderers 1, 2 & 3
  - Multiple versions, capabilities
import squidkit.core.*;
import squidkit.util.Color;
public class Sample {
    public static void main( String[] args ) {
        SquidKit s = new SquidKit();
        CoreWindowAttributes wa = new CoreWindowAttributes();
        wa.setX(100);
        wa.setY(200);
        wa.setWidth(300);
        wa.setHeight(400);
        wa.setVisible(true);
        wa.setBackgroundColor(Color.yellow);
        wa.setTitle("Hello World");
        wa.parent = s.getDefaultScreen();
        CoreWindow cw = new CoreWindow(wa);

        while (true) {
            CoreEvent e = s.getEvent();
            if (e != null)
                s.dispatchEvent(e);
        }
    }
}
Window Result
Core Layer: Summary

- Builds on common native constructions
- Provides base windowing environment
- Ensures portability
- Enables the heart of Squidy: The WCK
The Window Construction Kit

- WCK est omnis divisa in partes quattour
  - Singular State Change Object (Buttons)
  - Data Manipulation Object (Edits)
  - Singular Value Out of Many Object (Dials)
  - Manager Object (Containers)

- Music Composition: Motif & Development
- Customization via Configurable Personalities
- Customization via Event Handling
- Customization via Painting
WCK Features

- Complex constructions are derived
  - Build on the Fly
  - Use only what you need
  - Total transformation of stock objects

- Customization
  - Model, Controller determine behavior
  - View is arbitrary
    - Intercept Events
    - Override Painting
    - Install new personalities
Dial Personality Transforms

Dial Object: Conventional Horizontal Slider

Dial object: Slider becomes pulldown menu, Draw background, render text on thumb.

Dial Object: Simple Knob becomes Pie Menu.
Complex Object Transforms

- Conventional Horizontal Scrollbar
- Scrollbar with Modified Layout.
- Same Scrollbar, new L&F through personality transforms
- Vertical Scrollbar converted to Scrollable Menu
WCK Objects

● Buttons
  – Label, Push, Radio, Arrows, CheckBox, Briefcase, Tree, Lock
  – Proto Class size: 17K

● Dials
  – Slider, Scroll, Progress, Knob
  – Box, Triangular, Round Thumbs
  – Proto Class size: 20K
WCK Objects

- **Edit**
  - Event Framework (6K)
  - Editable

- **Container**
  - Event Framework (1K)
  - LayoutManager
Button Object Types
Dial Object Types

---

The image contains a computer interface with multiple dial object types displayed. The interface includes various circular and rectangular shapes, each representing different dial types. The labels 'HSLDER' and 'TRIANGULAR' are also visible, indicating the names of the dial types.
WCK Summary

- New Perspectives on Common Constructs
- Small Footprint
- True Reuse
- Flexible Runtime generation of Objects
- Easily manipulated, Customizable
- Not A Toolkit!!!!
Builders

- Eliminate the Toolkit
  - ‘Scalability’ lost on traditional Toolkit designs
  - Reduce programmer error
- IDEs
- Enables ‘Just in Time’ Windowing
- But WAIT!
  - How do we develop a protocol when objects can be customized?
Octopus: Object Conversion TO Protocol for Use by SquidKit

- Java is XML friendly
  - Reflection
  - Invocation

- WCK Factory Layer
  - XML Input for Window Construction
  - XML Serialization for ‘On the Fly’ Protocol generation
Octopus: Rules/Assumptions

- All Objects ultimately decompose to Primitives
- Objects ↔ Elements
- Primitives ↔ Attributes
- Persistence Achievable via Set/Get Methods
- Object must supply null Constructor
- Naming Limitations
  - XML limits Element naming
XMLSerialization

- Seven Methods
  - `xmlAttributeMethods();`
  - `xmlAttributeReturns();`
  - `xmlAttributeDefaults();`
  - `xmlElementMethods();`
  - `xmlElementReturns();`

- Static construction of lists
- Default Array built in constructor
- Values retrieved via reflection at runtime
- Object Repository
XML Commands

- Currently
  - <Clone>
  - <Update>
  - <Remove>
  - <Multi>

- A few to follow for Event Handling
Sample Static Initialization:

static{
    Method[] m;
    attributeMethods = new ArrayList();
    m = WindowAttributes.class.getDeclaredMethods();

    for ( int i = 0; i < m.length; i++ ) {
        name = m[i].getName();
        if ( name.startsWith("get") == false ) { //check for get/set methods
            continue;
        } ...
        argClass[0] = m[i].getReturnType();
        retType = argClass[0].getName();
        WindowAttributes.class.getMethod(testname.toString(), argClass);
        if ( retClass.isPrimitive() ) {
            attributeMethods.add( m[i] );
            attributeReturns.add( new StringBuffer( retType ));
        } else if ( retClass.isAssignableFrom( String.class ) ) {
            attributeMethods.add( m[i] );
            attributeReturns.add( new StringBuffer( retType ));
        } else {
            elementMethods.add( m[i] );
            elementReturns.add( new String( retType ));
        }
    }
}
Sample XML Output for Clone Operation:

<Clone>
  <squidkit.core.Screen elementID="1" />
</squidkit.core.Screen>

<squidkit.core.CoreWindow elementID="2" >
  <squidkit.core.CoreWindowAttributes elementID="3" parentAccessor="setWindowAttributes"
    parentType="squidkit.core.WindowAttributes" parentID="2"
    setTitle="(java.lang.String)&lt;&quot;&amp;quot;&amp;quot;&gt;
    setX="(int)100" setY="(int)200" setWidth="(int)700" setHeight="(int)400" >
    <squidkit.core.Screen elementID="1" parentAccessor="setParent"
      parentType="squidkit.core.BasicWindow" parentID="3" />
  </squidkit.core.CoreWindowAttributes>

  <squidkit.util.Color elementID="4" parentAccessor="setForegroundColor"
    parentType="squidkit.util.Color" parentID="3" setRGB="(int)-16777216" >
  </squidkit.util.Color>

  <squidkit.util.Color elementID="5" parentAccessor="setBackgroundColor"
    parentType="squidkit.util.Color" parentID="3" setRGB="(int)-1" >
  </squidkit.util.Color>

  <squidkit.util.Rect elementID="6" parentAccessor="setMargins"
    parentType="squidkit.util.Rect" parentID="3" >
    <_5BI elementID="7" parentAccessor="setRect" parentType="[I"
      parentID="6" dim="4" value="0,0,0,0" />
  </squidkit.util.Rect>

  <squidkit.core.DefaultDecor elementID="8" parentAccessor="setDecor"
    parentType="squidkit.core.Decor" parentID="3" setStyle="(int)447" >
  </squidkit.core.DefaultDecor>
</squidkit.core.CoreWindow>
</Clone>
Sample XML Update

<Mult>
  <Clone>
    <squidkit.util.Rect elementID="95">
      <_5BI elementID="96" parentAccessor="setRect"
        parentType="[I" parentID="95"
        dim="4" value="100,200,800,600"/>
    </squidkit.util.Rect>
  </Clone>
  <Update>
    <squidkit.core.CoreWindow elementID="2" setGeometry="(squidkit.util.Rect)95"/>
  </Update>
  <Remove>
    <squidkit.util.Rect elementID="95"/>
  </Remove>
</Multi>
Distributed Execution Issues

- Updating via Get/Set Pairs
- Invoke of non-Get/Set pairs?
- Object Removal
  - Object Reference Problem (Ref Counts?)
  - Repository serves as placeholder
  - Arrays…argh
  - Just remove from Object Repository
- Event handling (All events vs. some)
- Synchronization
Key Points

- Layered Architecture
- A more OO-centric model
- Less specialization in Objects means they are easier to understand
- A reasonable shot at allowing developers to ignore most toolkit issues via IDE/builders
- Universality over Feature set
Summary

- **SquidKit:**
  - More Portable
  - More Dynamically customizable
  - Layered
  - Eliminates the toolkit concept
  - Simpler IDE to Windowing Model
  - Smaller footprint

- **Further needs**
  - Graphics, Font Solutions
  - Core personality definitions
  - Protocol Work
Can We At Least Agree?

- Java should provide:
  - Mechanism Vs. Policy
  - Build Infrastructure
- Attempt to achieve a single framework
  - Aid developer understanding
  - Reduce industry level of effort
- Provide targeted solutions where uGUI can’t go
Testimonials