RemoteJFC: A Graphical User Interface Toolkit Approach to Thin-Client Computing

Computer Graphics and User Interfaces Laboratory

Columbia University
Motivation

• Multi-device UIs are difficult to implement
• Client side code approaches do not work
  – Distributed memory, deployment difficulties, reliability issues…
• Thin-client systems solve some issues
  – Shared state, rapid prototyping…
• Existing thin-client architectures are ad-hoc
Existing Thin-Client Techniques

- **Virtual Frame Buffer Transport**
  - App Logic
  - Virtual FB
  - Viewer
  - Client FB
  - Compressed Pixel Data over network

- **Web-based Application**
  - App Logic
  - App Server
  - Browser
  - Client FB
  - Render
  - HTML over network

Goals

• Performance
  – Low bandwidth
  – Low latency

• Rapid prototyping
  – Shared/single state, no explicit memory synch
  – Standard/familiar API
    • Robust tool support
    • User buy-in
Goals

• Push capability
  – Time critical interaction
  – Consistent information

• Generic “viewer” software
  – No need to update client software when functionality changes
Our Approach

- Most apps are built on a UI toolkit
- Why not have a distributed UI toolkit?
Related Systems

• **X-Windows** [Scheif86], **NeWS** [Gosling89]
  – Transport low level drawing commands (e.g. draw line from x to y) across the network

• **Repo3D** [MacIntyre98], **DIV** [Hesina99]
  – Distributed 3D graphics systems
  – Shared scenegraph but not thin-client
RemoteJFC Implementation

• Built on the JAVA platform
  – JFC/Swing is the baseline UI toolkit
  – RMI as the network transport mechanism

• System consists of…
  – Server-side library (~1.1M SLOC)
  – Client-side viewer (~2.5K SLOC)
Code Generation

• For each class in the UI toolkit API…
  – Create a equivalent “distributed” class
  – For each method in the original class…
  – Create a method of the same name
    • In the body, call the method in the original API

• Majority of the protocol is implicitly defined by code generator
Support Classes

- **RJFCFactory**
  - Exposed interface for creating UI components
  - Lives on client, remote reference on server

- **Viewer**
  - Allow user to choose a server and application
  - Passes remote references of client-side objects to the server
  - Executes “real” UI toolkit code
RJFC API

• Programmer creates class SomeApp
  – extends ChildApplication
  – Must have method \texttt{start}

• Create the UI in body of \texttt{start}
  – Retrieve objects from a factory instead of instantiating them directly
  – Frame is passed to ChildApplication at instantiation
public class HelloWorld extends ChildApplication {
    public void start() {
        try{
            RJFConnectionFactory f = server.getFactory(clientInfo);
            JFrame display = server.getDisplay(clientInfo);
            JLabel label = f.getRJLabel(“Hello World”);
            display.getContentPane().add(label);
            frame.show();
        } catch (RemoteException e) {
            …
        }
    }
}
Comparing RJFC and Swing

public void start() throws RemoteException {
    RJFCFactory f = server.getFactory(clientInfo);
    RJFrame d = server.getDisplay(clientInfo);
    RJTextArea TA = f.getRJTextArea(20,20);
    TA.addKeyListener(new TAKeyListener());
    RJScrollPane P = f.getRJScrollPane();
    P.setViewportView(TA);
    RJTextField TF = f.getRJTextField();
    TF.setEditable(false);
    RContainer c = d.getContentPane();
    c.setLayout(new BorderLayout());
    c.add(TF, BorderLayout.SOUTH);
    c.add(CreateMenu(), BorderLayout.NORTH);
    c.add(P, BorderLayout.CENTER);
}

public MyJFrame() {
    JTextArea TA = new JTextArea(20,20);
    TA.addKeyListener(new TAKeyListener());
    JScrollPane P = new JScrollPane();
    P.setViewportView(TA);
    JTextField SB = new TextField();
    SB.setEditable(false);
    Container c = this.getContentPane();
    c.setLayout(new BorderLayout());
    c.add(SB, BorderLayout.SOUTH);
    c.add(CreateMenu(), BorderLayout.NORTH);
    c.add(P, BorderLayout.CENTER);
}
Behind the Scenes

- User launches Viewer, connects to server
- Viewer executes the registerDisplay method
  - Remote references to RJFCFactory and RJFrame are passed to the RJFCServer
  - All operations on RJFrame and RJFCFactory are actually executed on the Viewer
  - Actual JFC components are instantiated by the RJFCFactory in the Viewer’s memory space
Where Everybody Lives

• Server-side:
  – All application logic
  – Remote references to…
    • RJFCFactory
    • RJComponents (including RJFrame)

• Viewer (client) -side
  – RJFCFactory and all RJComponents
  – Instances of all user created JFC components
References to the “Real” UI

• RJComponents cannot contain references to the actual UI toolkit components!
  – RJComponents are constantly passed around
  – Create RJButton, pass RJButton to add() method in content pane of RJFrame
• Viewer-side Hashtable to keep track of RJComponent -> JComponent links
Hello World
## Performance - Bandwidth

<table>
<thead>
<tr>
<th>Event</th>
<th>To Srv</th>
<th>To Client</th>
<th>To Srv</th>
<th>To Client</th>
<th>To Srv</th>
<th>To Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>724613</td>
<td>12553</td>
<td>12261</td>
<td>27343</td>
<td>69181</td>
<td>56623</td>
</tr>
<tr>
<td>Log In</td>
<td>0</td>
<td>0</td>
<td>82152</td>
<td>4687</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Open Application</td>
<td>39607</td>
<td>9364</td>
<td>24613</td>
<td>4509</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Idle (1 min, static mouse)</td>
<td>0</td>
<td>0</td>
<td>12660</td>
<td>6200</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Idle (1 min, anim. mouse)</td>
<td>0</td>
<td>0</td>
<td>24810</td>
<td>9217</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Idle (1 min, no mouse)</td>
<td>0</td>
<td>0</td>
<td>6390</td>
<td>2200</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Idle (1 min, full screen)</td>
<td>0</td>
<td>0</td>
<td>1709</td>
<td>2960</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Typing (1 min, 382 chars)</td>
<td>377159</td>
<td>135392</td>
<td>79617</td>
<td>74304</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cut Paragraph</td>
<td>125969</td>
<td>38786</td>
<td>79618</td>
<td>74304</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Paste Paragraph</td>
<td>91811</td>
<td>29430</td>
<td>1437</td>
<td>1899</td>
<td>658</td>
<td>461</td>
</tr>
<tr>
<td>Copy Paragraph</td>
<td>153062</td>
<td>41224</td>
<td>2508</td>
<td>2979</td>
<td>295</td>
<td>460</td>
</tr>
<tr>
<td>Find in Paragraph</td>
<td>154306</td>
<td>40750</td>
<td>5157</td>
<td>2312</td>
<td>1390</td>
<td>1965</td>
</tr>
<tr>
<td>Save Document</td>
<td>187768</td>
<td>49384</td>
<td>10621</td>
<td>5684</td>
<td>1238</td>
<td>2004</td>
</tr>
<tr>
<td>New Document</td>
<td>60940</td>
<td>19498</td>
<td>1360</td>
<td>2123</td>
<td>689</td>
<td>875</td>
</tr>
<tr>
<td>Open Document</td>
<td>114686</td>
<td>25120</td>
<td>6590</td>
<td>3144</td>
<td>1423</td>
<td>1396</td>
</tr>
<tr>
<td>Resizing from full screen</td>
<td>741322</td>
<td>60056</td>
<td>180576</td>
<td>16185</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Drag 1/4 size window</td>
<td>697433</td>
<td>64530</td>
<td>134016</td>
<td>212275</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Drag mouse across screen</td>
<td>308324</td>
<td>99300</td>
<td>1471</td>
<td>3726</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tear down</td>
<td>9618</td>
<td>3006</td>
<td>1779</td>
<td>2097</td>
<td>1667</td>
<td>2210</td>
</tr>
</tbody>
</table>

WWW 2002

VNC

RDP

RJFC
Performance - Latency

- Excellent performance once connected
- Startup latency is relatively high
  - Viewer calls registerDisplay() on RJFCServer
  - RJFCServer repeatedly calls methods on RJFCFactory (which resides on Viewer)

Solutions:
- “bank” / “bundle” calls to RJFCFactory
  - More difficult than it’s worth due to JAVA language
- Pipeline calls to RJFCFactory with threads
  - This is currently implemented in the latest RJFC
Additional Features

• Optional shared application footprint between multiple clients
  – Well-formed API for client connect/disconnect
  – Provides shared memory facilities
  – Enables community oriented shared applications (e.g. “chat” and “auction”)

WWW 2002
CGUI Lab - Columbia University
Screen Shots

RJFC Viewer

RJFC Notepad

RJFC Chat
RJFC In Action

RJFC implementation of 3D AR view management control system [Bell, UIST01]
Conclusions

• Distributed UI research is under-utilized
• Most web-based applications would benefit from this using approach!
Possible Future Directions

• RJFC compiler
  – Automatically generate RJFC from JFC code

• Viewer side caching of objects
  – Allow the viewer to dynamically “download” objects that have executable code in them
  – Draw on JIT and other compiler techniques
  – Violates thin-client principles to some extent

• Framework for analyzing the IBC of UIs
Acknowledgements

• William Chiong and Yoav Hirsch
  – They did all of the “real” work
• Blaine Bell
  – First user, primary driver behind improvements
• Supported by:
  – NSF Grant IIS-98-17434 under DLI2
  – Donations from Microsoft and Intel.
Website

• For downloads and more information:
  – http://rjfc.cs.columbia.edu