An Examination of the Run-time Performance of GUI Creation Frameworks

Christopher J. Howell
Gregory M. Kapfhammer
Robert S. Roos

Principles and Practices of Programming in Java
Kilkenny City, Ireland  June 16-18, 2003
Presentation Outline

• Introduction: importance of graphical user interfaces (GUIs)
• What is a GUI?
• Event handling latency and GUI manipulation event difficulty
• Overview of GUI creation frameworks: Swing and Thinlet
• Experimental design and justification
• Empirical results:
  → Event handling latency
  → CPU and memory consumption
• Related and future work
• Conclusion
Introduction

• Source code for GUIs: Past- 48%, Current- 60%
• GUI creation frameworks: correctness and performance
• Analysis of Java programs
  → Statically, at source code and bytecode levels
  → Dynamically, at bytecode level and on specific virtual machine(s)
• Our focus: performance of GUI creation frameworks for specific applications and Java virtual machines
• GUI toolkit showdown: Thinlet vs. Swing
• User-perceived performance for a case study application
GUI Fundamentals

- A GUI is simply a set of widgets
- The state of the GUI is the state of all the widgets
- Our model ignores widget layout constraints
- Event handling latency: \( L(E) = L_A(E) + L_G(E) \)
- Difficulty of GUI manipulation event: \( D(E) = D_A(E) + D_G(E) \)
  - Formulation of \( D_A(E) \) requires analysis of algorithms in the underlying application and JVM
  - Formulation of \( D_G(E) \) requires understanding of the GUI widgets that are updated and added to the GUI
Comparing Swing and Thinlet

- **Swing:**
  - Extension of AWT
  - Approximately 50 components
  - **Advantages:**
    - Lightweight – more efficient use of resources
    - Written in Java – cross-platform and very consistent look and feel
  - **Disadvantages:**
    - Inherent abstraction level
    - Excessive object creation

- **Thinlet:**
  - Created by Robert Bajzat
  - Currently 22 components
  - **Advantages:**
    - Application Separation: GUI in XML and underlying code in Java
    - Relatively simple GUI development
  - **Disadvantages:**
    - Limited number of components
    - Limited threading model
Visual Database Querying Tool

- Ideal candidate application - enables the variation of GUI manipulation event handling difficulty
- Difficulty was varied by changing table sizes to 25, 250, and 2500 tuples
- User can select tables, attributes, and comparison operators
- Query results displayed in the form of a table
- One version of the tool was developed with Swing and another with Thinlet
- Each tool uses the same Java Database Connectivity (JDBC) driver to connect to a PostgreSQL database
Experiments

• Systems Used
  • Pentium III, 533 Mhz with 128 MB RAM
    • Debian/GNU Linux – JVM 1.4.1
    • Ms Windows NT – JVM 1.4.0
  • UltraSPARC-5 Sun4u, 366 Mhz with 128 MB RAM
    • Solaris 8 – JVM 1.4.1

• Five Distinct Experiments
  • Initial startup
  • Opening of Screens (Selection of tables, attributes, relational operators)
  • Viewing of final query results with 3 different table sizes
Latency Results: Overview

- Measured average event handling latency for single addition to textarea
- First four experiments measure event handling latency when table size is not a factor

<table>
<thead>
<tr>
<th>Latency Time (ms)</th>
<th>Thinlet</th>
<th>Swing</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solaris</td>
<td>4</td>
<td>6.33</td>
</tr>
<tr>
<td>Linux</td>
<td>3.16</td>
<td>3.66</td>
</tr>
<tr>
<td>Windows</td>
<td>3.33</td>
<td>3.33</td>
</tr>
</tbody>
</table>

- Fifth experiment varied the table sizes:
  - Thinlet outperforms Swing for smaller number of updates/adds
  - Swing outperforms Thinlet for larger number of updates/adds
Latency Results: Graphs

(a) Latency at varied difficulty levels:
- $D_0(E_1) = 12$
- $D_0(E_2) = 6$
- $D_0(E_3) = 15$
- $D_0(E_4) = 15$

(b) Latency for GUI manipulation events:
- $D_0(E_5) = 25$
- $D_0(E_6) = 250$
- $D_0(E_7) = 2500$

Graphs showing latency in milliseconds with different event conditions.

Legend:
- Thinlet
- Swing
CPU and Memory Results

- Swing uses less CPU when rendering large amounts of data.
- Memory usage consistent throughout applications with more use at startup and querying final results.
- Memory usage for single addition to textarea.

<table>
<thead>
<tr>
<th>OS</th>
<th>Thinlet</th>
<th>Swing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>392</td>
<td>992</td>
</tr>
<tr>
<td>Linux</td>
<td>312</td>
<td>748</td>
</tr>
<tr>
<td>Windows</td>
<td>334.66</td>
<td>846.66</td>
</tr>
</tbody>
</table>
Related and Future Work

- **Related research:**
  - Memon et al.: testing and analysis of programs with GUIs
  - Endo et al.: interactive system performance analysis
  - Horgan et al: Java - centric performance analysis

- **Future research:**
  - The impact of different JVM modes (HotSpot client, HotSpot server, interpreted) on user-perceived performance
  - Memory usage patterns for applications that use Swing and Thinlet
  - New case study applications
  - New Java GUI creation frameworks: Eclipse SWT, SWIXML
  - General methodology for GUI toolkit performance analysis
Conclusion

- Thinlet is better for less difficult GUI manipulation events
  - Easier to implement due to XML interface
  - Currently, only 22 widgets
  - Threading model needs to be improved
- Swing is better for more difficult GUI manipulation events
  - Harder to implement
  - Approximately 50 widgets in toolkit
- GUI toolkit choice depends on the application being created for which to choose
- Our goal: to provide GUI-driven application developers with heuristics for choosing the appropriate GUI creation framework
Resources

• **Java GUI Creation Framework Performance Research:**
  ➔ [http://cs.allegheny.edu/~gkapfham/research/jgp/](http://cs.allegheny.edu/~gkapfham/research/jgp/)

• **Java Performance Tuning (J. Shirazi):**
  ➔ [http://www.javaperformancetuning.com](http://www.javaperformancetuning.com)

• **Performance Documentation for Java HotSpot VM:**

• **Performance Documentation for Java Platform:**