Simplifying Cyber Foraging for Mobile Devices

Andrew Zafft
Computer Science Dept
azafty@wpi.edu

Worcester Polytechnic Institute
Agenda

- The Problem
- Building on Past Research
- The Solution
- Testing and Results
- Personal Observations

Anything in red are my comments and questions

“Cry ‘Havoc!’ and let slip the hogs of war”
The Problem

- Portable devices and application are an area of great potential for growth.
- These devices must balance processing power and battery life.
- Applications must be ported to new hardware in very short timeframes to be commercially successful.
- Many situations tailor themselves to applications that are used while away from fixed location computers:
  - Written and verbal language translation
  - Optical character recognition
  - Face recognition
  - Any general information storage & recall of real world events and people in real time.
The Problem

- If mobile applications could tap into the power of fixed location computers, they could pass off some of the expensive processing – **Cyber Foraging**

- Cyber Foraging can be broken down into 3 parts
  1. Locating a server
  2. Establish a trusted connection with the server
  3. Partition the application between remote and local execution

*Topic of this paper*
Goals

• **Stated goal of the paper**
  – Enable rapid modification of applications for cyber foraging

• **Desired tangible outcomes**
  – Help novice developers partition large, unfamiliar applications in just a few hours
  – Show that the quality of novice-modified and expert-modified applications are comparable in most cases
Building on Past Research

• **Concept of Little Languages**
  – Used to tackle a very small problem set (hence the ‘Little’)
  – Narrowly defined scope and functionality (domain specific)

• **Some examples**
  – \( \text{\LaTeX} \): Typesetting
  – Excel: Spreadsheets
  – Make: Compiler execution
  – HTML: Hypertext web pages
Building on Past Research

Info on the authors

• R.K. Balan – PhD student at CMU who originally worked on *Vivendi* (for dissertation) and *Chroma*

• M. Satyanarayanan – PhD Advisor for Balan at CMU

• D. Gergle – Assistant Professor at Northwestern University (PhD student at CMU at time of paper)

• J. Herbsleb – Professor at CMU
Building on Past Research

- **Chroma** – an operating system lite which provides feedback to applications on system resources

- **Vivendi** – a small programming language used to designate procedures that can be handled remotely

- **RapidRe** – Rapid Retarget, the rapid deployment of existing technology on old hardware onto new hardware
Building on Past Research

Design Considerations

• Language independent & coarse grained
  – No fully automated code analysis
  – Manually modified for runtime support
  – Coarse grained module based approach

• Support for runtime adaptation
  – Dynamic use of fidelity ("application specific output quality")

• Focus on quick portability
  – Rapid deployment of next generation devices

• An application for the “middle ground”
  – A.K.A. no Strawman solutions
The Solution

Their keys to a successful application

- Design a complex application which requires little training
- Enable the modification of complex applications quickly
- Actively reduce the chance of errors while editing complex applications
- Produce applications comparable in quality to expert designed systems

Four Keys By Which They Directly Measure Their Success
The Solution

The architecture will be three-fold:

- A static description of meaningful partitions (*Vivendi*)

- A powerful runtime system that allows dynamic configuration on changing system needs (*Chroma*)

- A stub generation tool with application specific interfaces to the runtime system (*Vivendi* stub generator)
The Solution

How it all works:

• A programmer uses the application source code to build a tactics file

• The programmer runs the stub generator

• The programmer modifies the application by inserting the stub calls at appropriate points in the source code

• The programmer compiles and links the modified application, stubs and Chroma
The Solution

A Little Language named *Vivendi*

- Serves as the static description file
- Contains code short hand similar in structure to C
- Holds code components that can benefit from remote execution (REMOTEOP)
- Order of magnitude is small; few REMOTEOPs exist for each application (coarse grained)
- Contains simple handling for consecutively and concurrent remote procedure calls
The Solution

A Little Language named *Vivendi*

- Code components are broken down into individual RPC steps

- RPC functions that will be remotely executed are organized into remote execution tactic, or *tactic*

- The same RPC functions can be organized in different ways in different *tactics*
  - allows the system to handle REMOTEOPs in different ways depending on system resources

```plaintext
APPLICATION graphix;
REMOTEOP render;

IN int size DEFAULT 1000; // parameters
OUT float quality FROM 0.0 TO 1.0; // fidelities

// TACTIC definitions
// do step 1 followed sequentially by step 3
TACTIC do_simple = step_1 & step_3;

// do steps 1 & 2 in parallel, then do step 3
TACTIC do_all = (step_1, step_2) & step_3;

// RPC definitions
RPC step_1 (IN string input, OUT string buf1);
RPC step_2 (IN string input, OUT string buf2);
RPC step_3 (IN string buf1, IN string buf2, OUT string final);
```

Figure 1: Example Tactics File in Vivendi
The Solution

**Chroma**

- Provides resource measurement, prediction and fidelity selection complimentary to *Vivendi*

- Acts as a layer between applications and the underlying operating system, device characteristics and resource availability

- Mechanics largely not discussed in this paper
The Solution

Chroma

- The core of Chroma is the Solver
  - Responds to queries from Vivendi regarding tactics and fidelities for REMOTEOPs
  - Optimizes tactic-fidelity combinations for the best result given an overarching utility function (logical goal)
  - Gauges resource availability through a resource monitor
    - Software sensors on the server and the mobile device that measure network bandwidth, CPU utilization, memory usage, battery level etc.
  - Predict resource demands through historical based usage
    - Self correcting

Worcester Polytechnic Institute
The Solution

Stub Generator

• Provides an inherent server listener loop with opcode demultiplexing (standard stuff)

• Generates two kinds of stubs
  – Standard RPC stubs
  – Wrapper stubs
    • Simplifies application modifications by customizing the Chroma interface to the application

• These stubs must be manually incorporated into the source code

Why can’t most of this be automated?
The Solution

- Testing to be done through RapidRe
- Testing follows the same path as their standard development process
Testing and Results

• Baseline/Control (without *Vivendi & Chroma*) test difficult to incorporate

• Project deemed to be successful if
  – Development time was cut by an order of magnitude or more with regards to an expert developed system (without *Vivendi & Chroma*) AND
  – Novice built system performed similar to expert built system (using *Vivendi & Chroma*)

• Quantitative measurements taken on soft topics
  – Ease of use of system
  – Ease of learning
  – Personal gauge of performance

• And on readily calculated topics
  – Errors committed by users
  – Application latency
  – Additional lines of code
Testing and Results

Testing Applications

• Focused on those key applications identified at the top of the article

<table>
<thead>
<tr>
<th>Application</th>
<th>Lines of Code</th>
<th>Number of Files</th>
<th>Language</th>
<th>Fidelity</th>
<th>Parameter RPCs</th>
<th>Total RPC Args</th>
<th>Tactics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>20K</td>
<td>105</td>
<td>Ada w/C interface</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Flite</td>
<td>570K</td>
<td>182</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>GLVU</td>
<td>25K</td>
<td>155</td>
<td>C++, OpenGL</td>
<td>1</td>
<td>15</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>OCR</td>
<td>30K</td>
<td>71</td>
<td>C++</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Janus</td>
<td>126K</td>
<td>227</td>
<td>C, Tcl/Tk, Motif</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Music</td>
<td>9K</td>
<td>55</td>
<td>C++, Java</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Panlute</td>
<td>150K</td>
<td>349</td>
<td>C++</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Radiator (3D Lighting)</td>
<td>65K</td>
<td>213</td>
<td>C++, OpenGL</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Overview of the Test Applications
Testing and Results

Testing Approach

- Train a user on Vivendi using a sample application
- Separate the tasks into 3 activities (Stages A, B & C)

<table>
<thead>
<tr>
<th>Stage A</th>
<th>Stage B</th>
<th>Stage C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tactics file</td>
<td>Client component</td>
<td>Server component</td>
</tr>
<tr>
<td>Read docs</td>
<td>Read docs</td>
<td>Read docs</td>
</tr>
<tr>
<td>Application</td>
<td>Include file</td>
<td>Include file header</td>
</tr>
<tr>
<td>In</td>
<td>Register</td>
<td>service_init API call</td>
</tr>
<tr>
<td>Out</td>
<td>Cleanup</td>
<td>Create RPCs</td>
</tr>
<tr>
<td>RPC</td>
<td>Find Fidelities</td>
<td>run_server API call</td>
</tr>
<tr>
<td>Tactic</td>
<td>Do Tactics</td>
<td>Compile and fix</td>
</tr>
<tr>
<td></td>
<td>Compile and fix$^1$</td>
<td></td>
</tr>
</tbody>
</table>

This table shows the task stages and the subtasks within each stage. $^1$ Note that in Stages B and C, the participants compiled their code, but did not run it.

Table 2: Task Stages
Testing and Results

Testing Approach

- Randomly select one individual task for 1 of 7 applications (focusing on applications of opportunity)
  - Why not have a tester run through all 3 stages for the same application?

- Testers not able to actually run their application, only make sure it compiles
  - I’m somewhat baffled by this. I imagine it makes good academic sense, but this is rarely the case in reality

- A trained expert independently builds all tasks for all applications to serve as a comparison
Testing and Results

Testing Approach
• Applications were then tested under 6 scenarios (Q highest fidelity, T least latency)

<table>
<thead>
<tr>
<th>ID</th>
<th>Load</th>
<th>BW</th>
<th>User Prefs</th>
<th>Typical Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Low</td>
<td>High</td>
<td>Highest quality result</td>
<td>Conducting an important business meeting using a language translator</td>
</tr>
<tr>
<td>T</td>
<td>Low</td>
<td>High</td>
<td>Lowest latency result</td>
<td>Field engineer just wanting to navigate a quick 3D model of a building to understand the building’s dimensions</td>
</tr>
<tr>
<td>LH</td>
<td>Low</td>
<td>High</td>
<td>Highest quality result within X s</td>
<td>Sitting in an empty cafe with plentiful bandwidth and unused compute servers</td>
</tr>
<tr>
<td>HH</td>
<td>High</td>
<td>High</td>
<td>Highest quality result within X s</td>
<td>Bandwidth is available in cafe but long lived resource intensive jobs are running on the compute servers</td>
</tr>
<tr>
<td>LL</td>
<td>Low</td>
<td>Low</td>
<td>Highest quality result within X s</td>
<td>Cafe’s compute servers are unused but other cafe users are streaming high bitrate multimedia content to their PDAs</td>
</tr>
<tr>
<td>HL</td>
<td>High</td>
<td>Low</td>
<td>Highest quality result within X s</td>
<td>The cafe is full or people either streaming multimedia content or using the compute servers for resource intensive jobs</td>
</tr>
</tbody>
</table>

Load is the compute server load. BW is the available bandwidth. User Prefs are the User Preferences. X is 20s for Face, 25s for Radiator, and 1s for the rest.

Table 3: Scenario Summary
Testing and Results

Results – Little Training Required
• While the ultimate test is performance, the secondary test was impression by users

• The limited training was found helpful by all users (avg 1.33 out of 7)

• Users felt very certain about their performance
Testing and Results

Results – Quick Modifications
• Can novices modify applications quickly?
  – Average completion time was just over 2 hours

• Self evaluation of how easy they found the task
  – Heavily weighted towards the easy end
Testing and Results

Results – Low Error Rate

- Divided into Stage A and Stage B/C error groupings

- **Stage A Errors**
  - A total of 18 errors
  - A success rate of about 72% (no harmful errors)

<table>
<thead>
<tr>
<th>Apps</th>
<th>Params</th>
<th>RPCs</th>
<th>Tactics</th>
<th>Harmless</th>
<th># Apps</th>
<th>Okay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Flite</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>GLVU</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Janus</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Music</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Panlite</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Radiator</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>25</td>
<td>18</td>
</tr>
</tbody>
</table>

The # Apps column lists the no. of tactics files created for each app. Okay lists how many tactic files had no harmful errors.

Table 6: Total Errors for Stage A Across All Participants
Testing and Results

Results – Low Error Rate

• Stage B/C Errors
  – Trivial: commonly occurring errors in assignments
    • A total of 25 trivial errors
    • An average of 1 trivial error per modification attempt
  – Non-Trivial: materially effective errors
    • A total of 10 non-trivial errors
    • An average of 0.4 non-trivial errors per modification event
  – A success rate of about 64% (no harmful errors)
Testing and Results

Results – Good Quality

• Is novice quality comparable to expert quality?
  – Application is indistinguishable if 36 test cases (6 scenarios each) came within 1% of expert (for fidelity and latency)

<table>
<thead>
<tr>
<th>Participant</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>100%</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>67%</td>
<td></td>
</tr>
<tr>
<td>Flite</td>
<td>100%</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>67%</td>
<td></td>
</tr>
<tr>
<td>GLVU</td>
<td>44%</td>
<td>44%</td>
<td>44%</td>
<td>100%</td>
<td>44%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Janus</td>
<td>100%</td>
<td>100%</td>
<td></td>
<td></td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Music</td>
<td>100%</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Panlite</td>
<td>100%</td>
<td>83%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radiator</td>
<td>94%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>78%</td>
<td></td>
</tr>
</tbody>
</table>

A score of 100% indicates that the participant’s client version matched the performance of the expert in all 36 experiments. A blank entry indicates that the participant was not asked to create a modified version of that application.

Table 8: Relative Performance of Novice-Modified Client Component

• 16 of 25 cases were indistinguishable
• GLVU accounted for the most anomalies
  – Most of these were related to different participants repeating the same error

Worcester Polytechnic Institute
Testing and Results

• Results – Source Code Modifications

<table>
<thead>
<tr>
<th>App</th>
<th>Lines of Code</th>
<th>File Count</th>
<th>Tactic File Size</th>
<th>Stage B: Client Modifications</th>
<th>Stage C: Server Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Lines Added</td>
<td>Lines Removed</td>
</tr>
<tr>
<td>Face</td>
<td>20K</td>
<td>105</td>
<td>10</td>
<td>31 – 68</td>
<td>12 – 15</td>
</tr>
<tr>
<td>Flite</td>
<td>570K</td>
<td>182</td>
<td>10</td>
<td>29 – 39</td>
<td>1 – 5</td>
</tr>
<tr>
<td>Janus</td>
<td>126K</td>
<td>227</td>
<td>25</td>
<td>28 – 47</td>
<td>2 – 7</td>
</tr>
<tr>
<td>Music</td>
<td>9K</td>
<td>55</td>
<td>11</td>
<td>61 – 77</td>
<td>4 – 6</td>
</tr>
<tr>
<td>Radiator</td>
<td>65K</td>
<td>213</td>
<td>15</td>
<td>41 – 51</td>
<td>1 – 47</td>
</tr>
</tbody>
</table>

Any $a$–$b$ value indicates a lower bound of $a$ and an upper bound of $b$. Lines of Code and File Count show the size and number of files in the application. Tactic File Size gives the number of lines in the application’s tactics file. The Lines Added and Removed columns show how many lines were added and removed when performing the task. Stub Lines gives the number of stub-generated lines of code. Files Changed gives the maximum number of files that were actually modified by the participants.

Table 5: Application Modifications

Why are the number files and lines changed not proportional?
Testing and Results

Conclusions

• The system performed well given the limited training and experience of the participants

• Post results analysis showed that many applications were already logically divided in such a way that was easy for Vivendi & Chroma to exploit
  – Logical separation of resource intensive procedures from interactive code components inherently exists in “good programming”
  – Additionally, almost no state was stored between these elements
Testing and Results

Conclusions
• Taking this a step further, most applications had the following logical order of operations
  1. Obtain an input (speech utterance, natural language fragment, etc.)
  2. Resource intensive processing
  3. Output presented to the user

• Vivendi & Chroma enhance this design!
Future Work

• Work on further eliminating errors

• Work on the difficult areas
  – Finding Fidelities
Personal Observations

- Paper appears to be well written (sections logically titled and organized)
- The concepts were complex but presented in a straightforward manner
- Much of the paper was over explained, especially the definition and usage of the testing criteria (very belabored)
Questions?