A Usable Approach to Retargeting Software for Mobile Devices

Rajesh Krishna Balan

Computer Science Department
Carnegie Mellon University

Motivation: mobile interactive applications

• speech recognition, language translation, augmented reality, ...
  – Resource-heavy, but need bounded response time
  • Unfortunately, handhelds are weak!!

Motivation: Handhelds are weak!

- Resource intensive App
- Huge Data Sets

2 GHz, 1 GB, 3-D graphics 2 GB of data

200 MHz, 32 MB, no 3-D, no FPU 32 MB Flash

Resource-poor wearable

Poor performance!

Solution: Cyber Foraging

• "To live off the land"
  – Use resources in environment
• Remote execution
  – Augment device capabilities
• Fidelity Adaptation
  – Reduce resource usage
• Part of Aura Project

Problem #1

• How do we get large applications to work on small devices?

• Made more difficult by the mobile domain
  – Bandwidth fluctuates wildly
  – Battery concerns require variable operating modalities
    • Conserve battery at expense of quality and latency
    • Vice versa

Problem #2

• Seems likely that Middleware can be built
  – With enough sweat, engineering, and plumbing

• But, what about the applications??

Columbia U. MARS project
Problem: Retargeting Applications

- Large number of useful apps already exist
  - Complicated and large (> 100K lines of code)
  - Infeasible to write them from scratch

- Must be able to quickly retarget apps
  - Solution should be language independent
  - State of the art is 2 - 4 weeks per app

Strawman Solutions

- Run all applications on remote server
  - No application modifications needed
  - Use ssh or vnc to access server
  - RTT is a problem
  - Requires bandwidth
  - No degraded / corrective failure mode

- Run all applications locally
  - Not enough resources

Roadmap

- Motivation & Background
- Requirements of Solution
- Systems + Software Engineering Solution
- Evaluation
- Conclusion

Requirement No. 1

- Support as many applications as possible
  - No restriction on language
    - Must support C, C++, Java, .NET, etc.
  - No restriction on programming style
    - Procedural, functional, event-based (OpenGL), etc.
  - No restriction on type of application
    - Language translators, face detectors, graphics apps

Requirement No. 2

- Target Entry-level Application Developers
  - Effectively fresh hires right out of college
    - Representative of who actually does this work
  - Assume no expert knowledge of anything
    - Basic programming skills assumed

Requirement No. 3

- Solution must be good
  - Retargeting time must be low
    - < 6 hrs per application
    - Current time is ~ 2 to 4 weeks per application
  - Retargeted app is isolated
    - From runtime / OS / hardware changes
    - Vital for fast moving mobile market
  - Performance must be excellent
    - Comparable to hand-retargeted applications
Summary of Requirements

• Any application
• Any developer
• Fast
• Good

⇒ Sounds Impossible

Key Insight

For a large number of applications

• Application aspects relevant for remote execution and fidelity adaptation can be expressed in a short declarative form

– Expected Application Resource Usage
  • Use concept of parameters

– Runtime adaptive aspects of application
  • 2 aspects
    – Application runtime settings ⇒ fidelity variables
    – Remote execution partitionings ⇒ tactics

Roadmap

• Motivation & Background
• Requirements of Solution
• Systems + Software Engineering Solution
• Evaluation
• Conclusion

Solution Design

• Coarse-grained remote execution
  – Modular level (RPCs)
  – Finer grain requires language support

• Cannot wrap around existing APIs
  – Application / language independent
  ⇒ Applications must be hand-modified
    – Cannot use generic code analysis techniques

Components of Solution

1. Language for describing applications
   • Vivendi

2. Adaptive Runtime System
   • Chroma
   • Handles common adaptation requirements

3. Smart Stub Generator
   • Generates most of the app to runtime interface code

4. Well-defined procedure for modifying app

Vivendi

APPLICATION pangloss-lite;

IN int num_words;
OUT float quality;

Dictionary
Example based

// RPC Specifications */
RPC server_dict (IN string line, OUT string dict_out);
RPC server_ebmt (IN string line, OUT string ebmt_out);
RPC server_lm (IN string gloss_out, IN string dict_out, IN string ebmt_out, OUT string translation);

// Tactic (Useful Ways to Combine the RPCs) */
TACTIC dict = server_dict & server_lm;
TACTIC ebmt = server_ebmt & server_lm;
TACTIC dict_ebmt = (server_dict, server_ebmt) & server_lm;
Overview of Chroma

Function

Tactics selection engine

Utility Function

Selected tactics

Tactics description

User-specific knowledge

Core functionality

Input

Application

Output

Operation executor

Resource Demand

Resource Monitors

Resource availability

Predicted resource usage

Log file

(tactic, resource consumption)

Porting Applications

(1) Collaborate

Domain expert

Application

(2) Stub generator

Application source code

Adaptation expert

Application

(3) Modify

Chroma stub code

Application executable

(4) Compiler

Modified source code

App-Inserted API Calls (Client)

Basic

- Register :: Register app with runtime
- Cleanup :: Remove app from runtime

Core Functionality

- Find_fidelity :: Asks runtime to decide appropriate runtime settings
- Do_tactics :: Perform the operation

Benefits of Solution

- Developer describes app using Vivendi
  - Knowledge of small part of app needed
- Stub creates code to interface with Chroma
  - Knowledge of Chroma unnecessary
  - Easy well defined process to insert APIs into app
- App APIs are runtime / OS agnostic
  - Only stub generator needs to be modified
    - Apps just need to be recompiled with new stub code
    - Similar to glibc changes

Roadmap

- Motivation & Background
- Requirements of Solution
- Systems + Software Engineering Solution
- Evaluation
- Conclusion

Methodology

- Conduct rigorous software usability study
  - Large number of real apps
  - Large number of representative developers
    - 13 Senior undergrads
    - Measure time taken
    - Measure number of errors in solutions
- Conduct systems evaluation
  - Compare user study apps with expert apps
    - Expert apps are hand retargeted for the same runtime
**Test Applications**

<table>
<thead>
<tr>
<th>App</th>
<th>Lines of Code</th>
<th>File Count</th>
<th>Language</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>20K</td>
<td>105</td>
<td>Ada / C</td>
<td>Face Recognizer</td>
</tr>
<tr>
<td>Flite</td>
<td>570K</td>
<td>182</td>
<td>C</td>
<td>Speech Synthesis</td>
</tr>
<tr>
<td>GLVU</td>
<td>25K</td>
<td>155</td>
<td>C++ / OpenGL</td>
<td>3D Model Viewer</td>
</tr>
<tr>
<td>GOCR</td>
<td>30K</td>
<td>71</td>
<td>C++</td>
<td>Char Recognition</td>
</tr>
<tr>
<td>Janus</td>
<td>126K</td>
<td>227</td>
<td>C / Tcl / Motif</td>
<td>Speech Recognizer</td>
</tr>
<tr>
<td>Music</td>
<td>9K</td>
<td>55</td>
<td>Java / C++</td>
<td>Music Recognizer</td>
</tr>
<tr>
<td>Panlite</td>
<td>150K</td>
<td>349</td>
<td>C++</td>
<td>Language Translator</td>
</tr>
<tr>
<td>Radiator</td>
<td>65K</td>
<td>213</td>
<td>C++ / OpenGL</td>
<td>3D Lighting Models</td>
</tr>
</tbody>
</table>

**Procedure**

- Train each participant for 1 hr
  - Using GOCR
- Measure the time needed for them to perform each stage of the process
  - Stage 1: create tactics file
  - Stage 2: create client component
  - Stage 3: create server component

**Summary**

- Reduced modification times from weeks to under 4 hrs
- Very few errors in solutions (not shown)
- But what about performance of modified apps?
  - Test modified apps against expert created apps
    - Under various mobile computing scenarios (not listed)

**Modification Times**

<table>
<thead>
<tr>
<th>App</th>
<th>Completion Time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>20</td>
</tr>
<tr>
<td>Flite</td>
<td>570</td>
</tr>
<tr>
<td>Janus</td>
<td>126</td>
</tr>
<tr>
<td>GLVU</td>
<td>25</td>
</tr>
<tr>
<td>GOCR</td>
<td>30</td>
</tr>
<tr>
<td>Music</td>
<td>9</td>
</tr>
<tr>
<td>Panlite</td>
<td>150</td>
</tr>
<tr>
<td>Radiator</td>
<td>65</td>
</tr>
</tbody>
</table>

**Difficulty and Confidence**

<table>
<thead>
<tr>
<th>App</th>
<th>Difficulty Score</th>
<th>Uncertainty Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Flite</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Janus</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GLVU</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GOCR</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Music</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Panlite</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Radiator</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Performance of Modified Apps**

<table>
<thead>
<tr>
<th>App</th>
<th>Participant Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>100% 100%</td>
</tr>
<tr>
<td>Flite</td>
<td>100% 100%</td>
</tr>
<tr>
<td>GLVU</td>
<td>44% 44% 44% 100%</td>
</tr>
<tr>
<td>Janus</td>
<td>100% 100% 100%</td>
</tr>
<tr>
<td>Music</td>
<td>100% 100% 100%</td>
</tr>
<tr>
<td>Panlite</td>
<td>83% 100% 100% 100%</td>
</tr>
<tr>
<td>Radiator</td>
<td>94% 100% 78%</td>
</tr>
</tbody>
</table>

- Tactics files had few errors (not shown)
- Client App performance shown
- All server apps had equivalent performance
Summary

- Performance is also good
  - 25 / 25 server apps correct
  - 16 / 25 client apps correct
    - Not perfect but this was expected

- Performance problems due to two factors
  - Failure to specify application resource usage
  - Failure to use returned fidelity values
    - Measures can be taken to reduce these errors

Summary of Solution

- Any application ➔ Yes!
- Any developer ➔ Yes!
- Fast ➔ Yes!
- Good ➔ Mostly
  ➔ There must be a catch

Reasons for Success

- Exploiting previously unknown similarities common to all the applications
  - Commonly used software development methods encourage proper modularization

- Naturally causes the “thin waistband” needed for this method
  - Facilitates high level description
  - Makes it easy for novices to add APIs
    - Interactive app model facilitates this as well

Additional Benefits

- Code is easier to test and debug
  - Very few developer-added lines
  - Most of the new code is stub generated

- Experts can now do the retargeting
  - Should require less time than novices
  - Resulting applications should be excellent

Only Part of the Solution

- 3 big missing pieces still needed
  - Integrating user preferences
    - Work by Sousa and Poladian (Aura project)
  - Application hint modules
    - Map fidelities / resources to application quality
  - Create appropriate UIs for a mobile device
    - Work by Eisenstein et al. and Myers et al.

Some Related Work

- Dynamic runtime systems
  - Odyssey [Noble], Puppeteer [deLara], Rover [Joseph], Coign [Hunt]

- Little Languages
  - Make [Feldman], QML [Frolund], QDL [Loyd], Aspect Languages [Kiczales]

- Retargeting Frameworks
  - IBM’s WebSphere, Microsoft’s .NET Framework
Didn't Talk About

• Chroma Runtime System in detail
• Other research
  – Distributed infrastructure for supporting massively multiplayer games
  – Virtual machine techniques to improve scientific applications
  – Changes to TCP's protocol
    • Improve wireless network performance
    • Reduce power consumption

Summary

• Big challenge in modern computing
  – Design high performance systems that are provably usable
• Systems are usually fast enough
• However, they are frequently not usable
• This is a concerted attempt to bridge the gap
  – Using techniques from very diverse areas

The End

• Questions?

App-Inserted API Calls (Server)

Basic
  – Service_init :- Register app with runtime

Core Functionality
  – Run_server :- Start the stub-generated event loop

Other Modifications
  – Create RPC stubs required by server
    • The RPCs specified in the vivendi syntax file

Applications

<table>
<thead>
<tr>
<th>App</th>
<th>Stage A</th>
<th>Stage B</th>
<th>Stage C</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>10.3 (1.7)</td>
<td>36.6 (4.5)</td>
<td>33.6 (17.8)</td>
<td>80.5 (22.7)</td>
</tr>
<tr>
<td>Flite</td>
<td>12.6 (7.8)</td>
<td>37.7 (6.7)</td>
<td>20.6 (16.4)</td>
<td>70.9 (20.4)</td>
</tr>
<tr>
<td>James</td>
<td>29.3 (14.0)</td>
<td>31.0 (6.5)</td>
<td>42.1 (10.2)</td>
<td>102.4 (26.2)</td>
</tr>
<tr>
<td>GLVU</td>
<td>66.3 (20.8)</td>
<td>65.1 (22.5)</td>
<td>40.3 (7.7)</td>
<td>171.7 (33.8)</td>
</tr>
<tr>
<td>Music</td>
<td>49.6 (15.7)</td>
<td>68.2 (17.1)</td>
<td>83.0 (23.0)</td>
<td>200.8 (45.4)</td>
</tr>
<tr>
<td>Panlite</td>
<td>36.2 (7.7)</td>
<td>48.7 (20.2)</td>
<td>32.8 (14.7)</td>
<td>117.8 (36.6)</td>
</tr>
<tr>
<td>Radiator</td>
<td>17.2 (6.6)</td>
<td>45.3 (8.7)</td>
<td>39.4 (7.0)</td>
<td>101.9 (11.7)</td>
</tr>
</tbody>
</table>
Solution: Remote Execution

- Augment capabilities of handhelds by using nearby servers
- But how can good performance be achieved in mobile environments?
- And easily allow legacy applications to use remote execution?

Performance of Modified Apps

- Tactics files had few errors (not shown)
- Client App performance shown

<table>
<thead>
<tr>
<th>Participant Number</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>
</tr>
</thead>
<tbody>
<tr>
<td>Flex</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>CPU</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Mem</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Perf</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Failure</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

- All server apps had equivalent performance

Performance of Modified Apps

- Tactics files had few errors (not shown)
- Client App performance shown

<table>
<thead>
<tr>
<th>Participant Number</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>Flex</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>CPU</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Mem</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Perf</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Failure</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

- All server apps had equivalent performance

Key Insight

For a large number of applications

- Number of useful remote partitions is small
  - Largest so far is 7 partitions
  - Modular level coarse-grained partitions
- Application developer specifies these partitions (static partitioning)
  - At runtime, pick the optimal partition and locations (dynamic partitioning)

Solution: Tactics

- Concise description of application’s remote execution capabilities
  - Only the useful remote partitions are described
  - Can be captured in a compact declarative form
  - Each tactic performs the required operation
  - Operation ➔ application specific notion of work

- Tradeoff between dynamic and static partitioning
  - RPC model
  - Assume servers have been discovered and are able to handle any RPC call (no code migration)
  - Coarse-grained remote execution

Too good to be true?

- Results are incredibly good
  – < 3½ hours for novices to retarget complex applications with good performance

- What’s the Catch?
Validation of Ease of Use

- Conduct Software Usability Study
  - Using senior undergrads
- Modify a large number of real applications
  - 7 applications + 1 training application
  - Created by other groups
  - Written in many languages
    - Java, C, C++, Ada, TCL/TK

Example Tactic

APPLICATION pangloss-lite;
/* RPC Specifications */
RPC server_dict (IN string line, OUT string dict_out);
RPC server_ebmt (IN string line, OUT string ebmt_out);
RPC server_lm   (IN string gloss_out, IN string dict_out,
                 IN string ebmt_out, OUT string translation);
/* Tactics (Useful Ways to Combine the RPCs) */
TACTIC dict    = server_dict & server_lm;
TACTIC ebmt    = server_ebmt & server_lm;
TACTIC dict_ebmt = (server_dict, server_ebmt) & server_lm;

Background

- Proliferation of Mobile Devices
  - Cellphones, PDAs, watches
  - Laptops are not really mobile devices
- However, smaller size \( \Rightarrow \) compromise CPU and battery
  - Unable to run large applications
- But, these applications are highly useful
  - Language translation
  - Speech recognition
  - Pattern recognition (signs, text etc.)