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!!

Columbia U. MARS project

2

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!

3

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

4

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

5

Problem #2

- Seems likely that Middleware can be built
  - With enough sweat, engineering, and plumbing
- But, what about the applications??

6

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

7

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

8

Roadmap

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

9

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

10

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

11

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

12

Summary of Requirements

- Any application
- Any developer
- Fast
- Good

➔ Sounds Impossible

13

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*

14

Roadmap

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

15

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

16

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

17

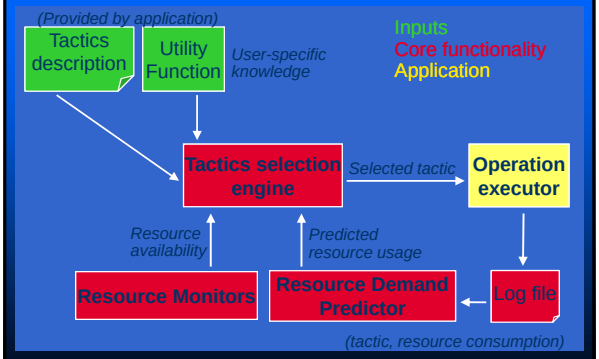
Vivendi

```
APPLICATION   pangloss-lite;

IN int num_words;
OUT float quality;

/* 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;
```

Dictionary
Example based
Language modeler

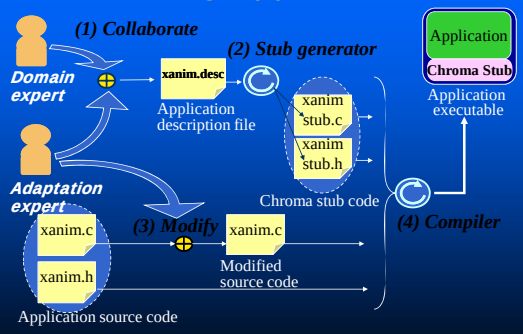
18

Overview of Chroma



19

Porting Applications



20

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

21

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

22

Roadmap

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

23

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

24

Test Applications

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

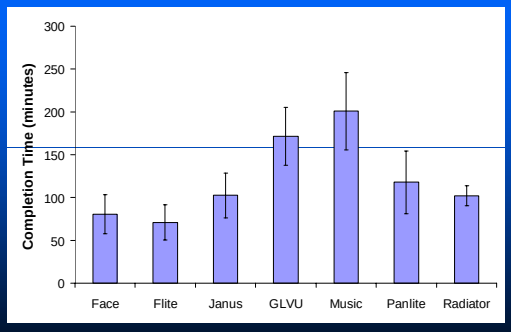
25

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

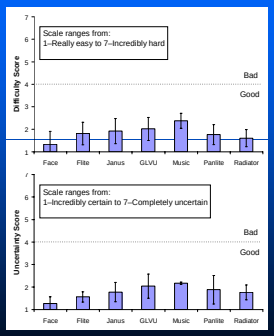
26

Modification Times



27

Difficulty and Confidence



Difficulty against Time
Correlation coeff = 0.82
Probability = 0. 02

Certainty against Time
Correlation coeff = 0.88
Probability = 0.009

Certainty and Difficulty
Correlation coeff = 0.88
Probability = 0.01

28

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)

29

Performance of Modified Apps

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

Participant Number

	1	2	3	4	5	6	7	8	9	10	11	12	13
Face	100%			100%							67%		
Flite								100%			100%	67%	
GLVU	44%		44%		44%		100%		44%				
Janus		100%	100%				100%						
Music			100%		100%								100%
Panlite		100%		83%		100%		100%		100%			
Radiator				94%						100%		78%	

- All server apps had equivalent performance

30

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

31

Summary of Solution

- Any application ➔ Yes!
- Any developer ➔ Yes!
- Fast ➔ Yes!
- Good ➔ Mostly

➔ There must be a catch

32

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

33

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

34

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.

35

Some Related Work

- Dynamic runtime systems
  - Odyssey [Noble], Puppeteer [deLara], Rover [Joseph], Coign [Hunt]
- Little Languages
  - Make [Feldman], QML [Frolund], QDL [Loyal], Aspect Languages [Kiczales]
- Retargeting Frameworks
  - IBM's WebSphere, Microsoft's .NET Framework

36

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

37

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

38

The End

- Questions?

39

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

40

Applications

App	Lines of Code	File Count	Tactic File Size	Stage B: Client Modifications				Stage C: Server Modifications			
				Lines Added	Lines Removed	Stub Lines	Files Changed	Lines Added	Lines Removed	Stub Lines	Files Changed
Face	20K	105	10	31 – 68	12 – 15	556	2	26 – 45	15 – 24	185	2
Flite	370K	182	10	29 – 39	1 – 5	556	2	13 – 30	3 – 87	185	2
GLVU	25K	155	38	62 – 114	3 – 21	1146	2	88 – 148	12 – 32	324	2
Janus	126K	227	25	28 – 47	2 – 7	1538	3	59 – 130	7 – 70	434	4
Music	9K	55	11	61 – 77	4 – 6	1127	2	131 – 269	23 – 147	203	2
Panlite	150K	349	21	50 – 66	1 – 39	1481	3	12 – 73	18 – 39	406	3
Radiator	65K	213	15	41 – 51	1 – 47	643	2	49 – 106	17 – 32	202	2

41

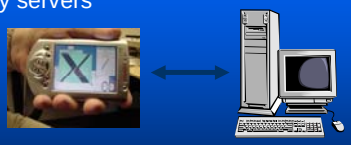
Application Modification Times

App	Stage A	Stage B	Stage C	Total
Face	10.3 (1.7)	36.6 (4.5)	33.6 (17.8)	80.5 (22.7)
Flite	12.6 (7.8)	37.7 (6.7)	20.6 (16.4)	70.9 (20.4)
Janus	29.3 (14.0)	31.0 (6.5)	42.1 (10.2)	102.4 (26.2)
GLVU	66.3 (20.8)	65.1 (22.5)	40.3 (7.7)	171.7 (33.8)
Music	49.6 (15.7)	68.2 (17.1)	83.0 (23.0)	200.8 (45.4)
Panlite	36.2 (7.7)	48.7 (20.2)	32.8 (14.7)	117.8 (36.6)
Radiator	17.2 (6.0)	45.3 (8.7)	39.4 (7.0)	101.9 (11.7)

42

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?

43

Performance of Modified Apps

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

	Participant Number												
	1	2	3	4	5	6	7	8	9	10	11	12	13
Face	100%			100%							67%		
Flite								100%			100%	67%	
GLVU	44%		44%		44%		100%		44%				
Janus		100%	100%				100%						
Music			100%		100%								100%
Panlite		100%		83%		100%		100%		100%			
Raciator				94%						100%		78%	

- All server apps had equivalent performance

44

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)

45

Performance of Modified Apps

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

	Participant Number												
	1	2	3	4	5	6	7	8	9	10	11	12	13
Face	100%			100%							67%		
Flite								100%			100%	67%	
GLVU	44%		44%		44%		100%		44%				
Janus		100%	100%				100%						
Music			100%		100%								100%
Panlite		100%		83%		100%		100%		100%			
Raciator				94%						100%		78%	

- All server apps had equivalent performance

46

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

47

Too good to be true?

- Results are incredibly good
  - < 3 ½ hours for novices to retarget complex applications with good performance
- What's the Catch?

48

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

49

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;
```

Dictionary
Example based
Language modeler

50

Background

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

51