J2EE architecture analysis using relational algebra

Michael W. Godfrey
SWAG, University of Waterloo
migod@uwaterloo.ca

From RAMP to SALSA:
A success story of migrating research ideas into industry

Michael W. Godfrey
SWAG, University of Waterloo
migod@uwaterloo.ca

Lossy program analysis or Lies my extractor told me

Michael W. Godfrey
SWAG, University of Waterloo
migod@uwaterloo.ca

Research pre-history

- 2002 a research project called RAMP is born
  - RAMP == Rapid Assisted Migration Project
  - An industrial research collaboration with Sun Microsystems
  - Principal investigators:
    - UW: Profs Holt / Malton / Godfrey
    - Sun: Brian Down, Wai-Ming Wong
  - Part of the CSER research consortium:
    `http://www.cser.ca`

RAMP goals

- Investigate aiding *assisted* software migration
  - Quick & dirty architecture modelling and analysis
  - Building a KB of discovered problems
  - Analysis of sw construction processes and artifacts
  - …

RAMP to Jackpot

- Contacts thru Sun / RAMP / conferences led to a sabbatical invitation
  - Sept-03 to Aug-04 in Sun's Research Lab in Mountain View, CA
- Jackpot: An AST-based analysis tool
  - Team members:
    - Michael Van De Vanter, James Gosling, Tom Ball, Tim Prinzing

Sun's Jackpot Tool

- AST-based analysis + transformation tool
  - Metrics summaries
  - "Bad smell" detection
  - Semi-automated source transformation
    - J++-to-Java migration, bad smell removal, …
  - Code visualization
  - "Smart" editor support

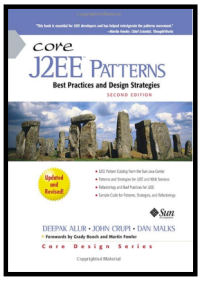
- Basic idea:
  1. Suck up *whole program* into memory
  2. "Play" with the AST
  3. Output transformed source code

Jackpot

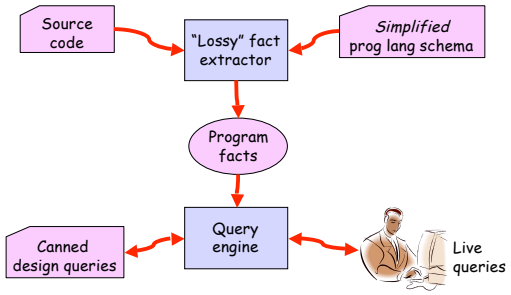
- When I arrived in Sept 2003:
  - Basic infrastructure works
  - Several bad smells can be detected automatically
  - Several automated transformations work
  - …
- But
  - While the technology is very promising, it's hard for outsiders to pick up and adapt easily
    - Must understand both Jackpot and `javac` internals
    - Work is slow going and very detailed (AST hacking)

"Solve a Real Problem"

- Van De Vanter introduces me to John Crupi, who has a problem:
  - *"We wrote the book on J2EE patterns (good and bad), but we're still using* `grep` *and* `perl` *to fix them !"*
- I meet with Van De Vanter, Crupi, several times to sketch out the design of a prototype J2EE architecture analysis tool based around Jackpot



Lossy program analysis



Kinds of program analysis tools

1. Special purpose, batch static analysis tools
   - Read in code, analyze, spit out (relatively small) result set
     - Result set typically makes no sense on its own; need refs back to source code
   - Analysis goals hard-coded into tool
     - New goals? Write a new tool!

Kinds of program analysis tools

2. Whole earth / big bang analysis tools:
   - Perform generic analysis (*e.g.* compilation) and keep all of compilation "facts" in store
     - Then allow AST walkers to generate desired info
     - Source-to-source code transformation also possible
   - Analysis results can be customized via new tree walkers
     - Slow and detailed work
     - … but you can do just about anything to the source code
   - Each run requires a new compilation (or reading in saved AST / symbol table)

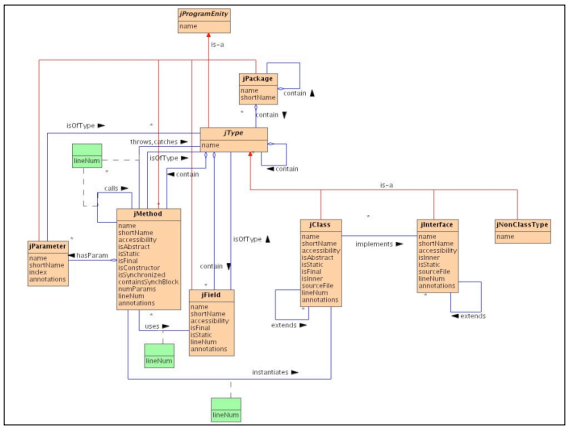
Kinds of program analysis tools

3. "Lossy" program analysis
   - Generates a set of "facts" about the program
     - An abstracted ("lossy") view of the system, according to a defined schema
     - The facts are complete, relative to their defined abstraction level
       e.g., can spot global variable uses across packages, but no information about how for loops are used
     - Source code examined only once
       - New run of the tool means only loading the "facts" into the query engine
       - Can add / refine queries using same factbase (since the facts don't change unless the code does)

"Lossy" program analysis

- Advantages:
  - Much easier to write canned queries, GUIs for navigation, experiment / go fishing with results
  - Model is self-contained, complete so no need to consult or link back to source code
  - Source code examined only once!
    - Loading factbase usually much faster than compilation
- Disadvantage:
  - Source-to-source code transformation not possible
    - But can feed results back into a whole-earth analsyis tool

    e.g., find known bad smells, feed the fixes to a transformation engine



Jackpot-to-SALSA

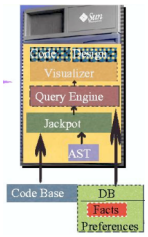
- I "finish" my extractor (still part of Jackpot), and give a demo for Crupi's group
  - I show how to define and run pattern queries they specify (using grok/QL) on source code they've provided

```
// Want to find all SessionBeans that call EntityBeans

extendsRTC = extends*
subtypeof = extendsRTC + extendsRTC o implements o extendsRTC
sessionsBeanClasses = classes ^
                      subtypeof . {"javax.ejb.SessionBean"}
entityBeanClasses = classes ^ subtypeof . {"javax.ejb.EntityBean"}
sessionBeansCallingEntityBeans = sessionsBeanClasses o calls
                                 o entityBeanClasses
```

SALSA goals *[Crupi]*

- Crupi pitches the idea to several big clients
  - It is very enthusiastically received!
  - The SALSA project (*Sun Appliance for Live Software Analysis*) is born!
- Main goal:
  - (Semi-) automate architectural assessment as much as possible
    - Aim for remote, collaborative, client-driven, early feedback
    - Ship with a library of known "bad patterns" + allow application/domain knowledge to be added
    - Feedback into the code (comments, annotations, transformed source code)



Current Status

- I finished the fact extractor
  - Now, a standalone Java 1.5 application
    - If `javac` can compile your code, I can extract it!
  - Extracts info about generic classes/methods, inner (non-local) classes, exceptions, initialization clauses, parameters, …
- Ongoing work at UWaterloo
  - A co-op student who worked on Jackpot has been working with me on extending this work; will start an MMath in Fall.
  - Recently completed: byte-code extractor using same schema
  - Next step: characterizing NFRs (e.g., security concerns) using the extracted facts
- Work on SALSA continues at Sun
  - Patterns library
  - GUI
  - Infrastructure enhancements