

Software Visualizations

Rolf Biehn

What is Software Visualization?

- Visualization of a software systems based on their structure, history, or behavior
- Today's presentation:
 - Program Execution Traces
 - Source Code History
 - Program Optimization

Execution Patterns in Object-Oriented Visualization

David Lorenz et al.

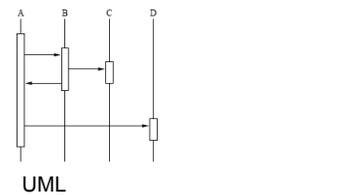
What is it?

- Techniques to visualize the execution flow and execution patterns
- Input is call traces from instrumented code

Motivation

- Understand program execution flow in order to program or debug it

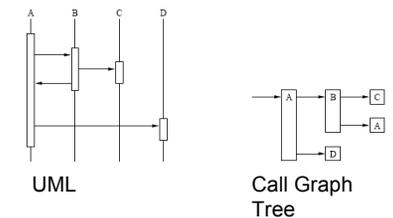
UML Visualization



UML Discussion

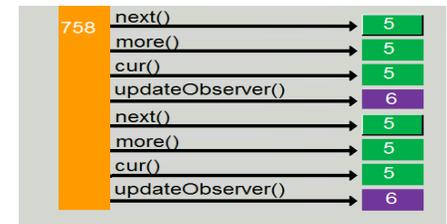
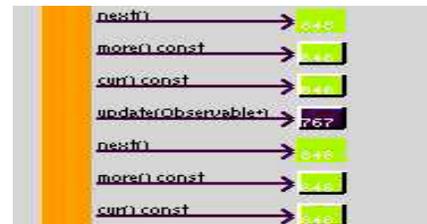
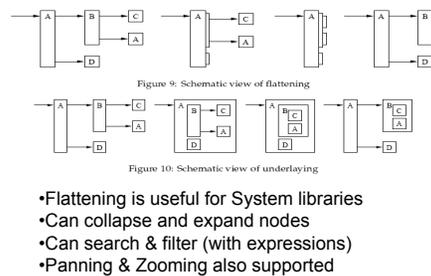
- +Scales better than directed graphs
- Vertical Space is consumed quickly
- Somewhat difficult to read

UML Visualization

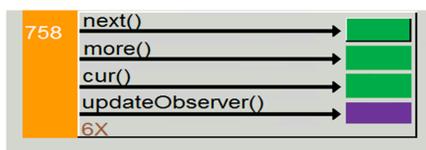


Execution Pattern Discussion

- +Easier to read than an UML diagram (no "bouncing between axis")
- +Horizontal & Vertical space is used more efficiently
- +Enables better user interaction



- 3D box indicates a collapsed node
- Colors correspond to a class
- ID #s represent identity of the object



- 3D box used to show pattern
- Saves lots of space in call traces
- Can expand/contract
- Number (6X) shows number of repetitions
- Also applies to recursion

How to detect pattern?

- Bunch of dimension:
 - Identity, Class Identity, Message Structure, Depth Limiting, Repetition, Polymorphism, Associatively, Commutatively
- Create a hash function for each leaf node which considers these dimensions
- Create a recursive hash function which considers its children in the call graph
- Put all nodes into a dictionary
- How long does it take? Memory concerns?

Evaluation

- Understand program execution flow in order to program or debug it
 - (B) Looks like it should work, if implemented carefully
 - How to navigate from high-level if I don't know precisely what I want to see?
 - What about multi-threading?
 - How well does it scale? What if number of Classes exceeds distinguishable colours?

CVSscan: Visualization of Code Evolution

Alex Telea, et al.

What is it?

- CVSScan is part of a larger suite of tools called Visual Code Navigator
- Provides information of the history of check-ins

Motivation

- Answer the following questions
 - Who performed these modifications of the code
 - Which parts of the code are unstable?
 - How are changes correlated?
 - How are the development tasks distributed?
 - What is the context in which a piece of code appeared?

Dimensions to Show

- All encoded using colors
 - Author
 - Content (block, comment, references)
 - Evolution (add/remove/delete/unchanged)

Global Line Position

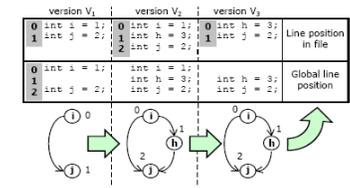
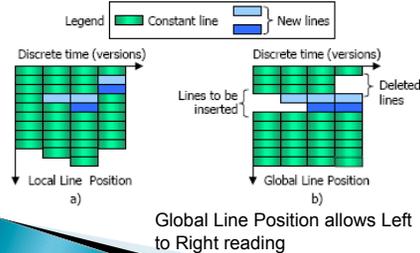


Figure 2 Global line position and corresponding graph analogy

Global Line Position (2)



Multiple Views

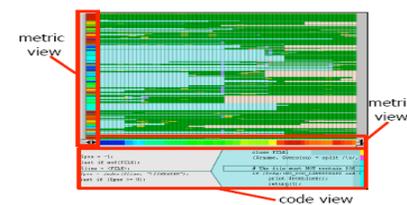


Figure 9: Multiple code views in CVSScan

2 Ways to Display Code

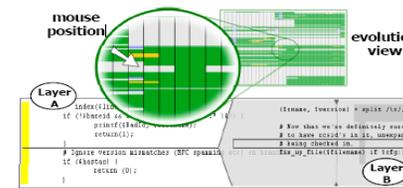


Figure 11: Two-layered code view

Use Case Validation

- Informal Studies (not targeted)
- 15 minutes of training
- Silent Observer
- Why not use a real-world case? (i.e. trying to fix a bug)
- No control
- No negative/constructive comments

Use Case #1

- Script file from the FreeBSD
- "Here they tuned the regular expressions"
- "Apparently a major change took place in the middle of the project. It mainly affected the check_version procedure"
- Rated as a success

Use Case #2

- C file socket implementation of the X Transport service layer
- The user recognized 2 authors performed most of the changes and the area of heavy modification
- Overall, the user did not have a very clear image of the file's evolution

Demo

Evaluation

- Who performed these modifications of the code?
 - (E) Hard to Track exactly "who is pink?"
- Which parts of the code are unstable?
 - (B) Seems o.k. for this purpose
- How are changes correlated?
 - (F) Correlation to other files in same check-in?
 - Correlation to other changes in the same file?

Evaluation

- How are the development tasks distributed?
 - (D) Although we can see distribution, precisely who wrote what is difficult to figure out
- What is the context in which a piece of code appeared?
 - (F) Hard to link back to changelist
 - Branching history?

Visualizing Application Behavior on Superscalar Processors

» Chris Stolte et al.

What is it?

- Program called Rivet
- Help optimization on multi-processor architectures

Motivation

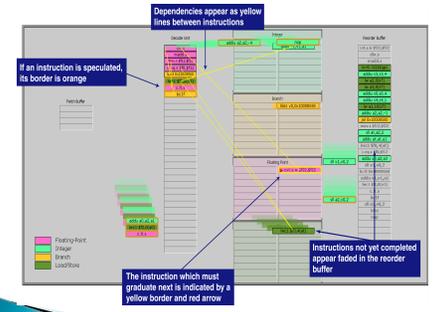
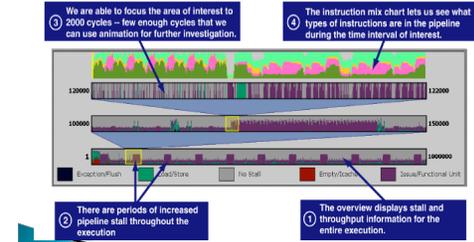
- Optimize
 - Know where to look
 - Drill into the details
 - Know the context – map back to the source code somehow

Main Optimization Techniques

- ▶ **Pipelining:** overlap the execution of multiple instructions within a functional unit
- ▶ **Multiple Functional Units:** exploit instruction level parallelism (ILP)
- ▶ **Out-of-Order Execution:** increase possibility of ILP
- ▶ **Speculation:** guess and fetch ahead

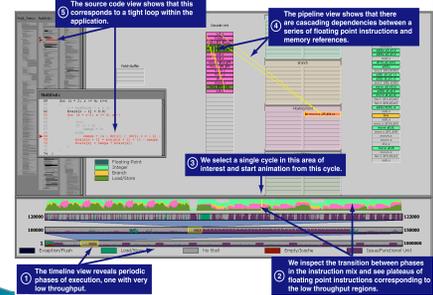
What the program tracks

- ▶ **Empty/Cache:** An instruction cache miss
- ▶ **Exception/Flush:** An instruction requires sequential execution
- ▶ **Load/Store:** Waiting for memory
- ▶ **Issue/Functional Unit:** Waiting for a functional unit to complete execution



```

63  i = brPoint - x + 1;
64  brVal[s] = 1 + 0.0;
65  for (s = 0; s <= 0; s--)
66  {
67      ++i;
68      omega = 0;
69      else
70          omega = (e - br[i]) / (br[i + e - 1] - br[i]);
71      brVal[s + 1] = brVal[s] + 1 + i - omega * brVal[s];
72      brVal[s] = omega * brVal[s];
73  }
74  }
75  /* Compute derivatives of the basis functions B_i(x) */
76  printf "\n";
77  BasisDerivsEval float G, long brPoint, float * br, long k, float * brVal
81  |
    
```



Evaluation

- ▶ Know where to look.
 - (B) Great use of overview-plus detail display
 - But is this really the best entry point?
 - What about filters?
- ▶ Look at the details
 - (A) Looks good
- ▶ Know the context – map back to the source code somehow
 - (A) Looks good
 - Next step link to IDE?

Questions?