PCODA Session 1

Patterns & behavior

Andy Zaidman, Delft University of Technology

PCODA'06 (co-located with WCRE'06)

Benevento, Italy

October 23rd, 2006

Summary

1. A Hybrid Analysis Framework to Evaluate Runtime Behavior of OO Systems
   Azin Ashkan, Ladan Tahvildari

2. Summarizing Traces as Signals in Time
   Adrian Kuhn, Orla Greevy

3. An Environment for Pattern based Dynamic Analysis of Software Systems
   Kamran Sartipi, Hossein Safyallah

A Hybrid Analysis Framework to Evaluate Runtime Behavior of OO Systems

Azin Ashkan, Ladan Tahvildari

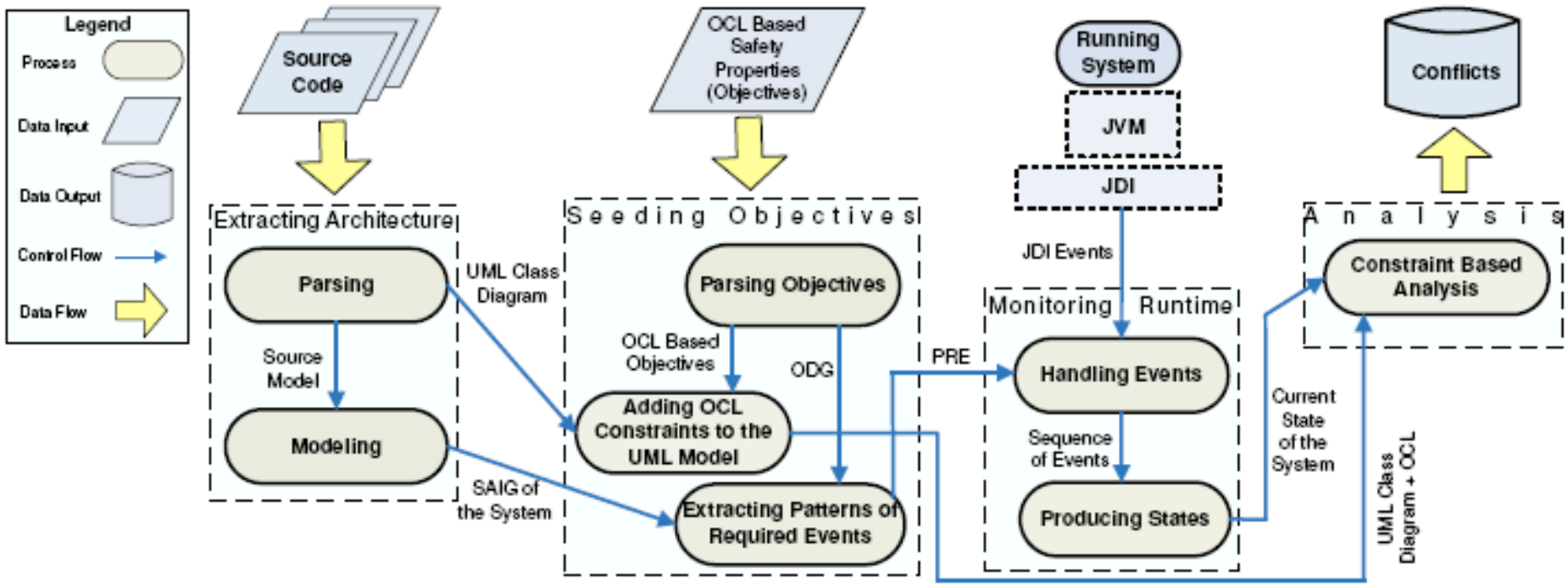
Context & goals

- **Context:** Safety critical systems
  - Constant need to establish whether a system is behaving correctly and reliably
  - Testing and verification are not enough or do not scale well
- **Goal:** Combine techniques from reverse engineering and runtime monitoring
  - OCL specifications to ensure safety properties

Framework overview (1)

- Extracting architecture
  - Extract meta-model (using graph representation)
- Seeding objectives
  - Safety properties defined in OCL
- Monitor runtime
- Analysis
  - Verify runtime data against safety properties

Framework overview (2)



Case study

- “Trader system”, 300 LOC
- OCL constraints
  - inv **Transaction**: Trader.allInstances $\rightarrow$ forAll (t_1, t_2: Trader | (t_1 <> t_2) implies $t_1.itemID$ <> $t_2.itemID$)
  - inv **TradingPolicy**: self.tradePrice <= self.basePrice
- Trace application based on OCL constraints (omit entities outside OCL specifications)

Analysis phase

Safety Property	Traders	States	Conflicts
Transaction	2	174	64
	4	223	97
TradingPolicy	2	174	0
	4	223	0

- *Transaction* safety constraint shows conflicts
  - Due to transactional problem: interference caused by making critical sections mutually exclusive

To conclude

- *Reverse engineering* & *runtime monitoring* techniques combined
- Safety critical systems, small case study shown
- Andy's thoughts
  - You talk about scalability. Experience with this? When OCL intensive, what happens?
  - "*Completeness*" of OCL safety specifications?
  - What would be needed to implement this industrially? (e.g. car, ATM, …)

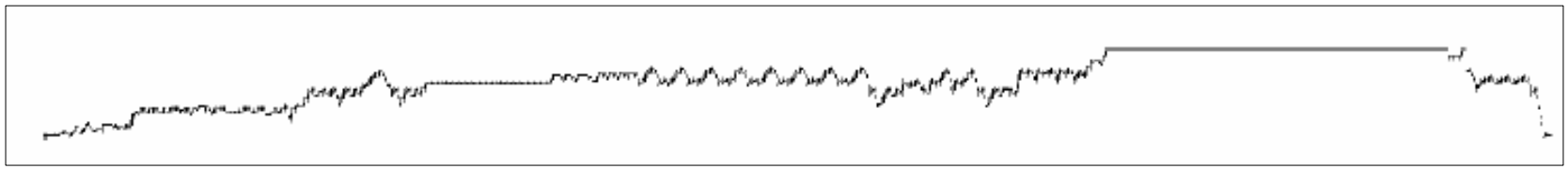
Summarizing Traces as Signals in Time

Adrian Kuhn, Orla Greevy

Paper overview

- Research questions
  - How to visualize dozens of feature traces efficiently?
  - Which (feature-)traces are similar?
  - Which execution patterns are common to all features, which are unique to a single feature?
  - How does architecture map to feature traces?

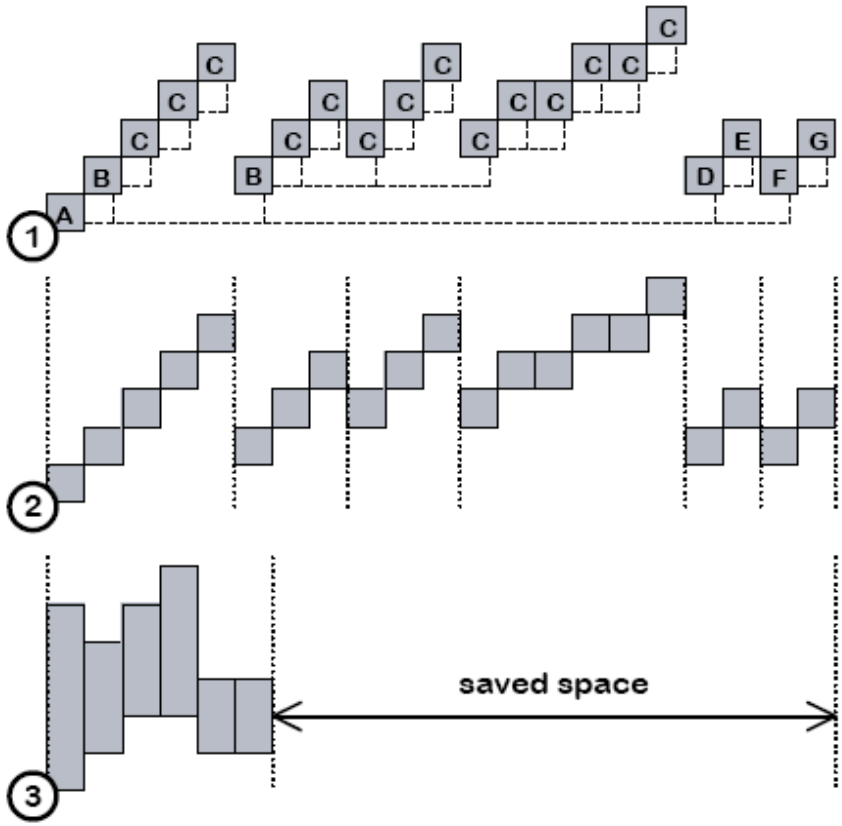
Monotone Subsequence Summarization



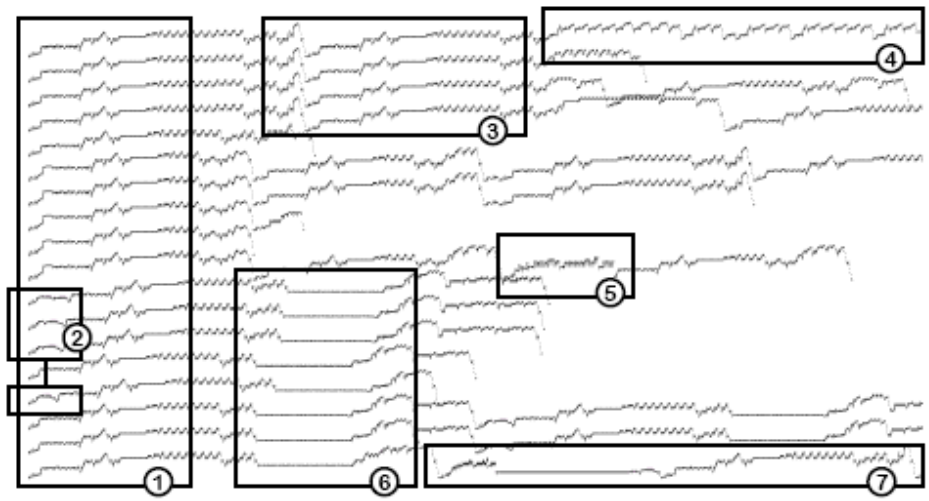
Based on nesting level

- Increase *y* value as an event calls its first child
- Stay constant as subsequent children are called
- Event without children → nesting level drops

Technique basics



Combining feature traces



- Identify similar regions
- Identify unique regions

To conclude

- Traces as signal: reduce trace by 50 to 90%.
- 200 000 events → 1 screen
- Applied on *SmallWiki* case study

- Open according to Andy:
  - What's the next step?
    Can you use trace similarity as input for other techniques?
  - Polymorphism: visible in visualization?

An Environment for Pattern based Dynamic Analysis of Software Systems

Kamran Sartipi, Hossein Safyallah

Goal

- “Identify the implementation of different software features without any prior knowledge about the implementation of the subject system”
- Important subgoals:
  - Deal with large traces
  - Discovery of patterns in traces

Basic building blocks

1. Data mining: to extract hidden patterns of execution
2. Concept lattice analysis: visualize relations among execution traces and their patterns
3. String matching algorithms: to find redundancy in traces

Patterns

- **Execution pattern** is a candidate group of functions implementing a common feature

- First step: find patterns with *sequential pattern mining*

Pattern organization

- Step 2: categorize patterns
  - Feature-specific pattern
  - Omnipresent pattern
- To categorize, use
  - *concept lattice analysis*:
  - or re-apply *sequential pattern mining* to find omnipresent patterns

To conclude

- Retrieve patterns
- Inject the obtained information in structure recovery proces (static technique(s))

- Andy’s view:
  - What are the (overall) results
  - What’s your next step?
  - What are the results of injecting the data in recovery proces?

General Q&A topics

Are patterns interesting to know?

1. When would *you* use them during program comprehension?
2. Do they typically explain high-level interactions or low-level functionality?
3. Is the runtime check of these patterns sufficient to guarantee behavior?