Conceptual Code Mining

Pr. Kim Mens
INGI / UCL

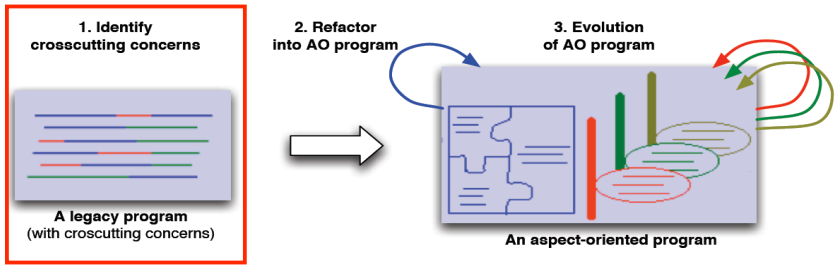
Dr. Tom Tourwé
SEN / CWI

Friday July 23, 2004

Overview

- Research context
- A crash course in formal concept analysis
- Mining for crosscutting concerns with FCA
- Overall approach
- The substring experiment in detail
- The parse tree experiment
- Conclusion

Software Evolution and Aspect-Oriented Programming



Research Idea

- Original goal
  - *Mining Aspects* using Formal Concept Analysis
  - also mining for architectural and other patterns
- First step (± completed)
  - Checked feasibility of approach with simple properties
    - By relying on naming conventions
  - Managed to discover relevant source code regularities
    - Coding conventions
    - Coding idioms and design patterns
    - Crosscutting features
- Next step (ongoing)
  - Improve approach to do "real" aspect mining
    - By relying on source-code similarities
  - Hope to discover real aspects or join points

Overview

- Research context
- A crash course in formal concept analysis
- Mining for crosscutting concerns with FCA
- Overall approach
- The substring experiment in detail
- The parse tree experiment
- Conclusion

A note on terminology

FCA terminology	Our terminology
- (Formal) Context	- Context
- (Formal) Concept	- Concept
- (Formal) Object	- Element • to avoid confusion with "objects" in the OO sense
- (Formal) Attribute	- Property • to avoid confusion with "attributes" in OO / UML sense

Formal Concept Analysis (*FCA*)

- Starts from
  - a set of elements
  - a set of properties of those elements
- Determines concepts
  - Maximal groups of elements and properties
  - Group:
    - Every element of the concept has those properties
    - Every property of the concept holds for those elements
  - Maximal
    - No other element (outside the concept) has those same properties
    - No other property (outside the concept) is shared by all elements

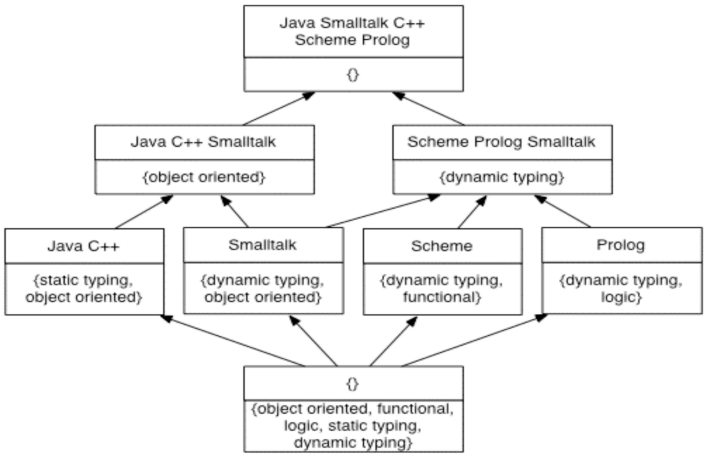
Example : Elements and Properties

	object-oriented	functional	logic	static typing	dynamic typing
C++	X	-	-	X	-
Java	X	-	-	X	-
Smalltalk	X	-	-	-	X
Scheme	-	X	-	-	X
Prolog	-	-	X	-	X

Example : Concepts

	object-oriented	functional	logic	static typing	dynamic typing
C++	X	-	-	X	-
Java	X	-	-	X	-
Smalltalk	X	-	-	-	X
Scheme	-	X	-	-	X
Prolog	-	-	X	-	X

Concept Lattice



Overview

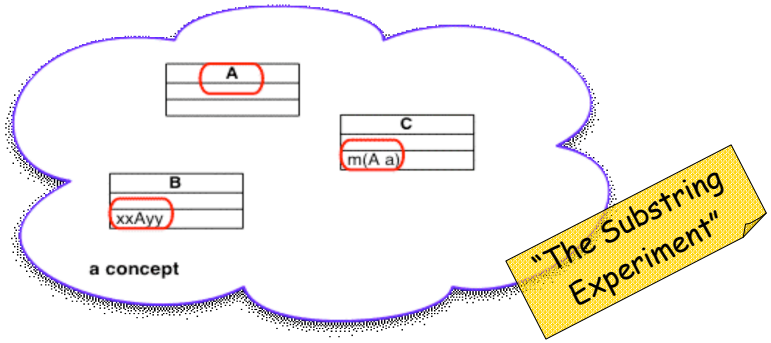
- Research context
- A crash course in formal concept analysis
- Mining for crosscutting concerns with FCA
- Overall approach
- The substring experiment in detail
- The parse tree experiment
- Conclusion

Mining for crosscutting concerns with formal concept analysis

- First Step
  - Use common substrings of class, method & parameter names to group related source code elements
  - Relies on coding conventions
  - Assumes that elements corresponding to a same concern will have a *similar name*
- Next step (ongoing)
  - Use "regular parse tree expressions" to find source code fragments that implement similar behaviour
  - Looks for recurring patterns in the source code
  - Similar to clone detection, but more advanced
  - Assumes that elements corresponding to a same concern will have *similar code*

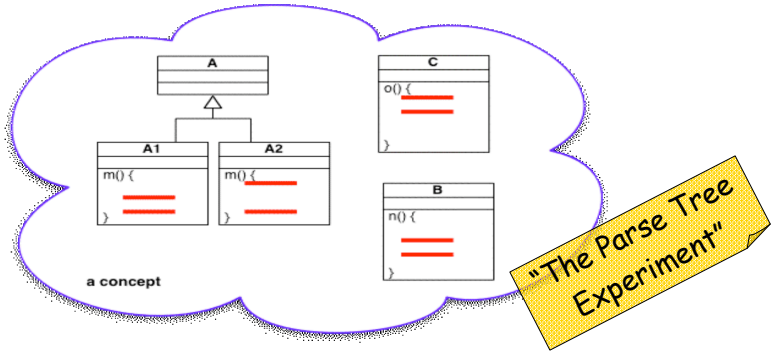
Schematically : Substring Concepts

- Elements : classes, methods, parameters
- Properties : substrings of classes, methods, …



Schematically : Parse tree Concepts

- Elements : methods
- Properties : regular parse tree expressions



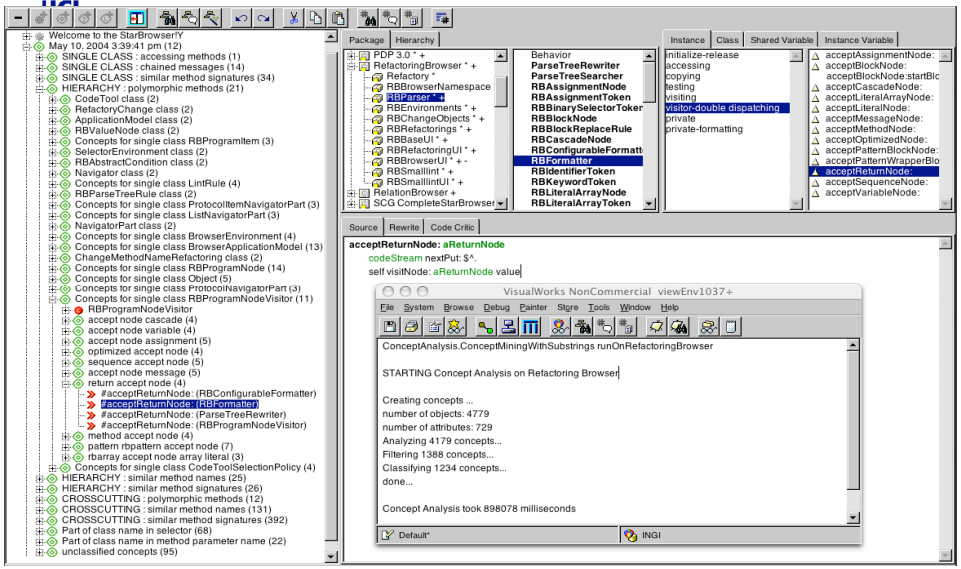
Overview

- Research context
- A crash course in formal concept analysis
- Mining for crosscutting concerns with FCA
- Overall approach
- The substring experiment in detail
- The parse tree experiment
- Conclusion

Overall approach

1. Generate the formal context
   - ✓ Elements, properties & incidence relation
2. Concept Analysis
   - ✓ Calculate the formal concepts
   - ✓ Organize them into a concept lattice
3. Filtering
   - ✓ Remove irrelevant concepts (false positives, noise, useless, …)
4. Classification
   - ✓ Classify results according to relevance for user
5. Analyse unclassified concepts
   - ✓ Manually analyse concepts that were not classified automatically
6. Completion of concepts
   - ✓ Some concepts are relevant but need to be completed to represent reality correctly

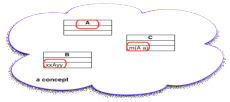
DelfSTof : our Conceptual Code Mining Tool



Overview

- Research Context
- A crash course in formal concept analysis
- Mining for crosscutting concerns with FCA
- Overall approach
- The substring experiment in detail
- The parse tree experiment
- Conclusion

The substring experiment



1. Generate formal context

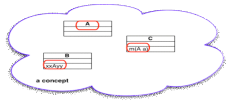
- We want to group elements that share a substring
- As elements we collect
  - all classes, methods and parameters
  - in some package(s) of interest
- As properties we compute
  - All "relevant" substrings of the names of those elements
    - Based on where uppercases occur in an element's name
      - QuotedCodeConstant → { quoted, code, constant }
    - Filter substrings that produce too much noise
- Incidence relation : An element has a certain property if
  - It has the substring in its name

The substring experiment

2. Concept Analysis (1)

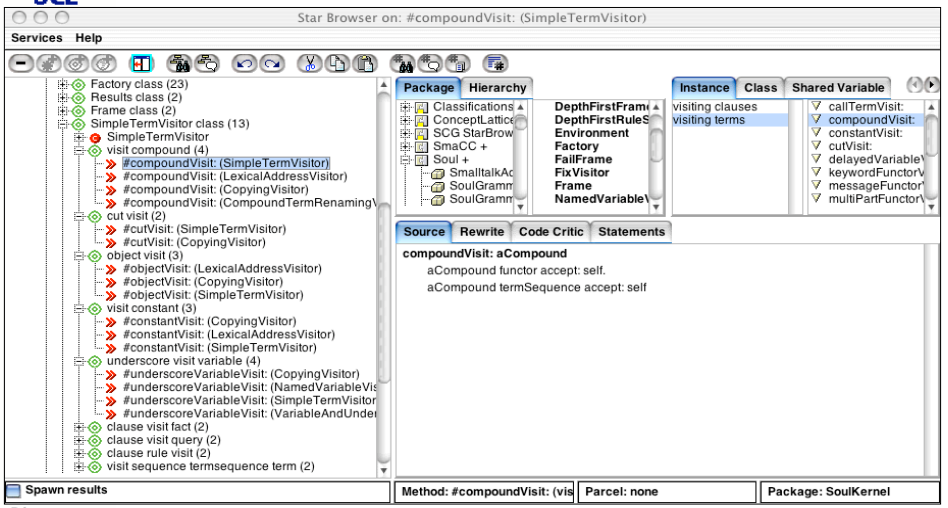
	unify	index	env	source	message	functor	variable	...
Object>>unifyWithObject: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-		-	...
Variable>>unifyWithMessageFunctor: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	X	X	-	...
AbstractTerm>>unifyWith: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-	-	-	...
AbstractTerm>>unifyWithVariable: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-	X	X	...
...	X	X	X	X	...	...	...	...

The substring experiment
2. Concept Analysis (2)

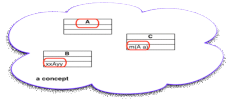


	unify	index	env	source	message	functor	variable	...
Object>>unifyWithObject: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-		-	...
Variable>>unifyWithMessageFunctor: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	X	X	-	...
AbstractTerm>>unifyWith: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-	-	-	...
AbstractTerm>>unifyWithVariable: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-	X	X	...
...	X	X	X	X	...	...	...	...

2. Concept Analysis - a concept



The substring experiment
Some quantitative results



Case study	#elements	#properties	#raw	#filtered	time (sec)
Soul	1469	434	1188	281	22
StarBrowser	527	266	491	73	4
CodeCrawler	1370	477	1419	327	24
DelfSTof	756	237	617	126	5
Ref.Browser	4779	729	4179	1234	414

- Remarks :
  - | properties | < | elements | is a good sign
  - Time to compute = a few seconds / minutes
  - Still too much concepts remain after filtering

The substring experiment
3. Filtering

- Irrelevant substrings are already filtered
  - with little meaning : "do", "with", "for", "from", "the", "ifTrue", ...
  - too small (< 3 chars)
  - ignore plurals, uppercase and colons
- Extra filtering
  - Drop top & bottom concept when empty
  - Drop concepts with two elements are less
- More filtering needed (ongoing work)
  - Recombine substrings belonging together
  - Require some minimal coverage of element name by properties
  - Concepts higher in the lattice may be more relevant
    - More shared properties
  - Avoid redundancy in discovered concepts
    - Make better use of the lattice structure (now it is "flattened")

The substring experiment

Discovered aspectual views (Soul)

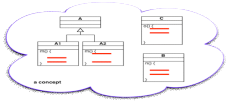
- Programming idioms
  - Accessor methods (*accessors*)
  - Polymorphism (*hierarchy methods*)
- Design patterns (*hierarchy methods*)
  - Visitor, Abstract Factory, Observer
- Crosscutting features
  - "Unification" (*hierarchy methods*)
  - Crosscutting class-related behaviour (*class name in keyword* & *class name in parameter*)
  - "Bindings", "Horn clauses", "resolution" (*unclassified*)
- Opportunities for refactoring
  - Mainly code duplication

An aspectual view is a set of source code entities, such as classes, methods and parameters, that are structurally related and often crosscut the entire source code.

Overview

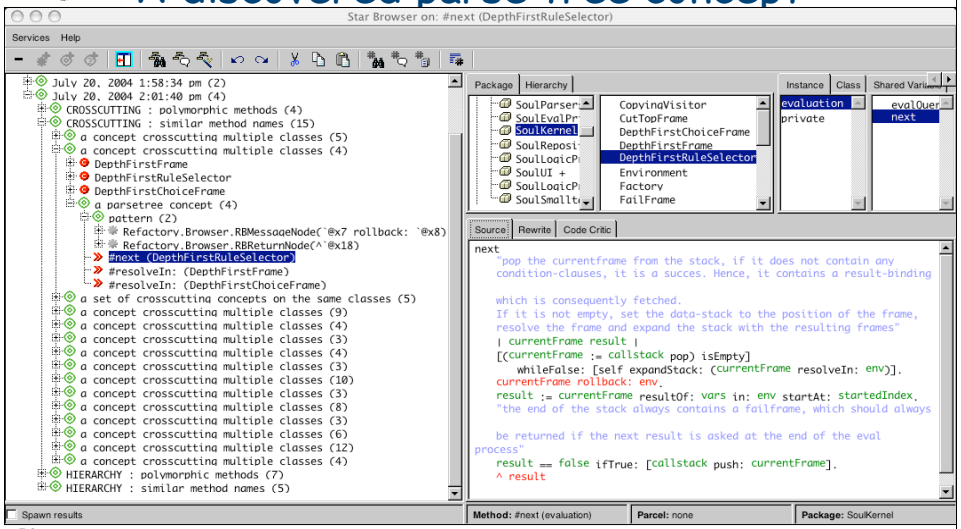
- Research Context
- A crash course in formal concept analysis
- Mining for crosscutting concerns with FCA
- Overall approach
- The substring experiment in detail
- The parse tree experiment
- Conclusion

Parse Tree Experiment (1)

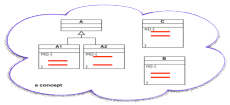


- Use FCA to group methods according to structural similarities in their parse trees
  - Elements = methods
  - Properties = "regular parse tree expressions"
- Regular parse tree expressions
  - We "abuse" some functionality provided by the *rewrite rule editor* of the *Refactory Browser*
  - Allows us to describe parse tree nodes, parameterized with an @ for those subtrees that we want to leave generic
  - Example
    - Refactory.Browser.RBMessageNode(`@x7 rollback: `@x8)
    - Refactory.Browser.RBReturnNode(^`@x18)

A discovered parse tree concept



Parse Tree Concepts (2)



- Experiment ongoing
- Currently a kind of advanced clone detection technique
  - Slightly more expressive (e.g., order of statements unimportant)
- Discovers
  - Lots of cases of code duplication
  - Interesting opportunities for refactoring
- Enough to detect aspects / join points?
  - (How) do we detect potential aspects or join points?
    - What exact combination of regular parse tree expressions to use?
  - How to filter uninteresting concepts?
    - E.g., cases of code duplication that are not really aspects
- More experiments / fine-tuning needed

Overview

- Research Context
- A crash course in formal concept analysis
- Mining for crosscutting concerns with FCA
- Overall approach
- The substring experiment in detail
- The parse tree experiment
- Conclusion

Conclusion

- Current status
  - Substring experiment performed
    - Discovers interesting source-code regularities just based on names
    - Some refinement needed : mainly more advanced filtering
  - Parse tree experiment seems promising complement / extension to already existing experiment
  - Enough to detect aspects?
- Future work
  - Work out parse tree experiment
  - Check it on a real aspect program : are the weaved aspects discovered by the approach?
  - Consider more dynamic information
    - E.g., examining the execution trace of the program
    - Perhaps in combination with examining the static structure

Some Publications

- Mining Aspectual Views using Formal Concept Analysis
  - Tom Tourwé (CWI) & Kim Mens (UCL)
  - Accepted for publication / presentation at SCAM2004 (+ journal ?)
- Conceptual Code Mining – Mining for Source-Code Regularities with Formal Concept Analysis
  - Kim Mens (UCL) & Tom Tourwé (CWI)
  - Accepted for publication / presentation at ESUG2004 research track
  - Accepted for publication in a special issue of the Elsevier international journal "Computer Languages, Systems and Structures"
- Aspect-Oriented Software Evolution
  - Tom Mens (UMH), Kim Mens (UCL) & Tom Tourwé (CWI)
  - Published in ERCIM News No. 58, Special theme on Automated Software Engineering