Conceptual Code Mining

Mining for Source Code Regularities With Formal Concept Analysis

Pr. Kim Mens
INGI / UCL

Dr. Tom Tourwé
SEN / CWI

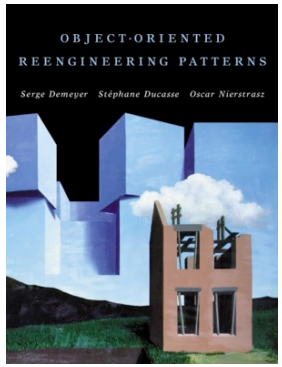
Thursday, May 13 2004

Overview

- Research idea
- A crash course in formal concept analysis
- Mining for source-code regularities with FCA
- The experiments in detail
- Conclusion

Research Context

- Software understanding and reengineering
  - Where to start?
- Book on "*Object-oriented engineering patterns*"
  - *Chapter 3: First Contact*
  - a set of patterns that may be useful when you encounter a legacy system for the first time.
- Forces:
  - Time is scarce
  - Legacy is large and complex



- "First contact" patterns
  - Chat with the maintainers
  - Interview During Demo
  - Read all code in one hour
  - Skim the documentation
  - Do a mock installation

Research Goal

- Research goal :
  - Automated tool support to help you "get started"
- Formal concept analysis (FCA)
  - A mathematical technique
  - With known applications in data analysis and knowledge processing
- Can we use FCA to "mine" the source code?
  - For relevant structural regularities in the source code
    - Coding conventions
    - Coding idioms and design patterns
    - Crosscutting features

Overview

- Research idea
- A crash course in formal concept analysis
- Mining for source-code regularities with FCA
- The experiments in detail
- Conclusion

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

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

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

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

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

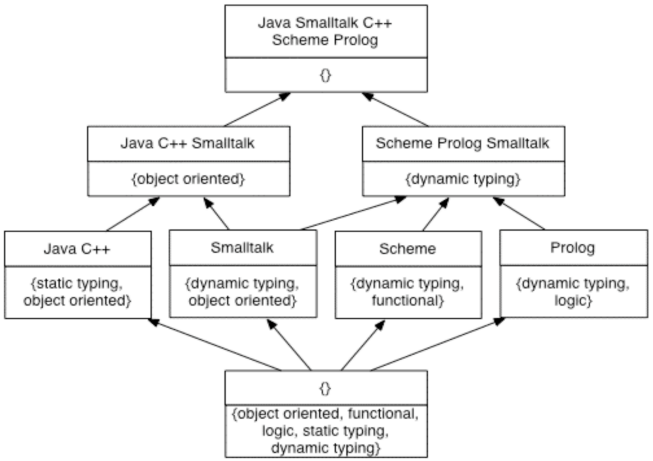
Example : Concepts

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

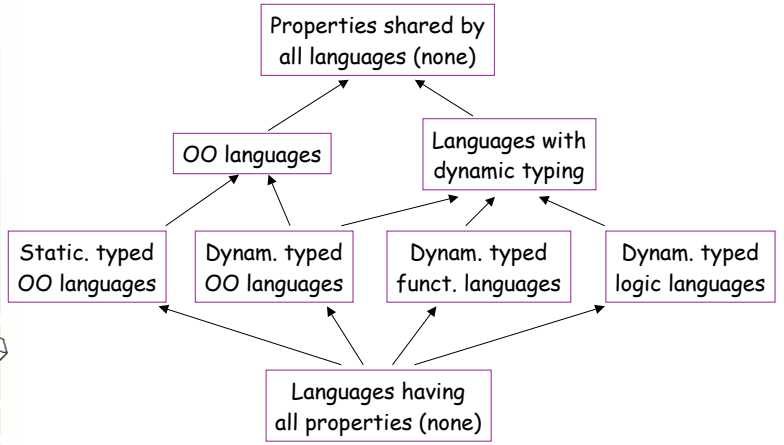
Example : Concepts

	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



Discovered Concepts

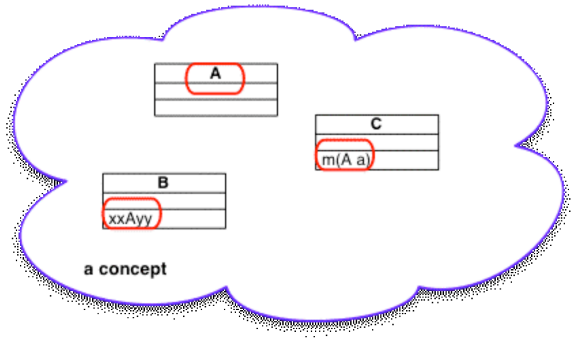


Overview

- Research idea
- A crash course in formal concept analysis
- Mining for source-code regularities with FCA
- The experiments in detail
- Conclusion

Mining for source-code regularities with formal concept analysis

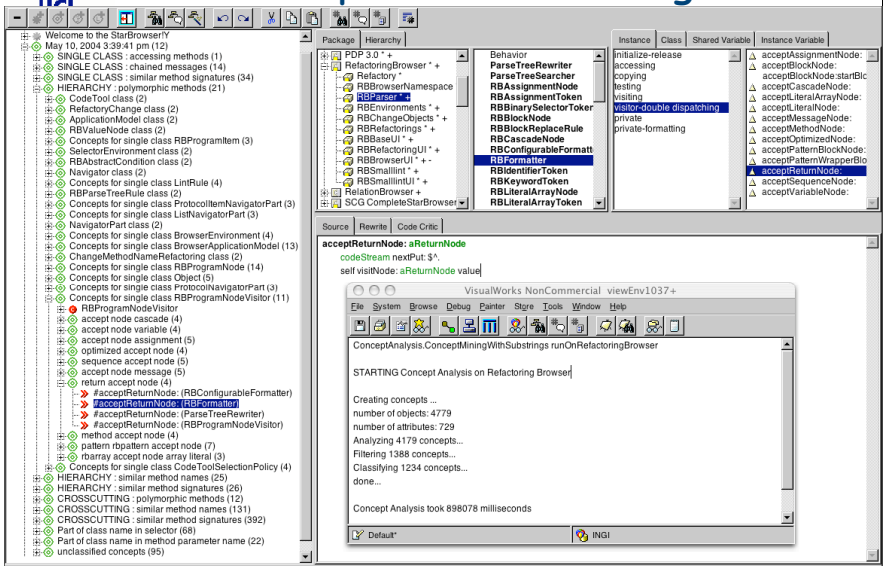
- Elements : classes, methods, parameters
- Properties : substrings of classes, methods, …



Overall approach

1. Generate elements & properties for FCA algorithm
   - ✓ Pre-filter irrelevant ones
2. Concept Analysis
   - ✓ Find relevant groupings of elements in source code
3. Filtering
   - ✓ Remove irrelevant concepts (false positives, noise, useless, …)
4. Classification
   - ✓ Classify results according to relevance for user
5. Completion of concepts
   - ✓ Some concepts are relevant
     but need to be completed to represent reality correctly

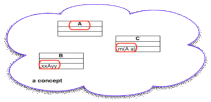
Our Conceptual Code Mining Tool



Overview

- Research idea
- A crash course in formal concept analysis
- Mining for source-code regularities with FCA
- The experiments in detail
- Conclusion

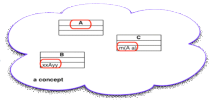
The substring experiment



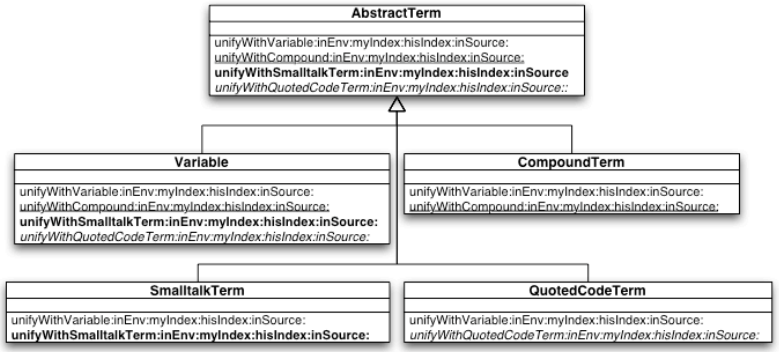
1. Generate elements & properties

- We want to group elements that share a substring
- Problem :
  - "Having a substring in common" is *binary*
  - FCA properties are *unary*
    - Does an element satisfy the property or not?
- Solution :
  - Every substring corresponds to an FCA property
    - Does an element have this substring in its name?
  - Generate relevant substrings
    - Based on where uppercases occur in an element's name
      - QuotedCodeConstant → { quoted, code, constant }
    - Filter substrings that produce too much noise

The substring experiment



2. Concept Analysis (1)



The substring experiment

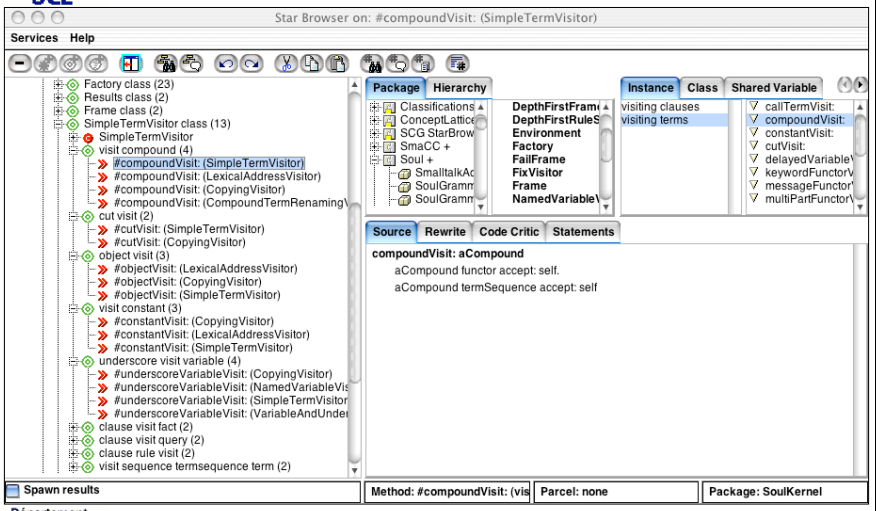
2. Concept Analysis (2)

	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	...	...	...	...

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 (3)



The substring experiment
Some quantitative results

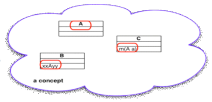
Case study	#elements	#properties	#raw	#filtered	time (sec)
Soul	1469	434	1188	281	22
StarBrowser	527	266	491	73	4
CodeCrawler	1370	477	1419	327	24
DelfSTof	756	237	617	126	5
Ref.Browser	4779	729	4179	1234	414

- 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")

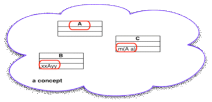
The substring experiment
4. Classification



- In single class
  - Accessors
  - Chained messages
  - Delegating methods
  - Similar signatures
- In same hierarchy
  - Polymorphic methods
  - Substring shared by method name & parameter name
  - Similar signatures
  - Similar class names

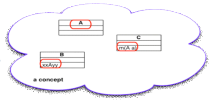
- Crosscutting
  - Polymorphic methods
  - Substring shared by method name & parameter name
  - Similar signatures
  - Similar class names
- Substring shared by method name & class name
- Substring shared by class name & parameter name
- Unclassified

The substring experiment
5. Completion (ongoing work)



- Discovered classifications may require completion
  - E.g., we may discover an interesting set of polymorphic methods
  - But some methods are missing because, e.g.,
    - Their implementing class does not adhere to the right naming convention
    - One of their parameters they had was named differently
  - These classifications should be completed "a posteriori"
    - Can this be done (semi) automatically?

The substring experiment
Discovered "regularities"



- Code duplication
- Design patterns
  - Visitor, Abstract Factory, Builder, Observer
- Programming idioms
  - Accessor methods
  - Polymorphism
- Relevant domain concepts
  - Correspond to frequently occuring properties
  - "Unification", "Bindings", "Horn clauses", "resolution"
- Opportunities for refactoring
- Crosscutting concerns

Overview

- Research idea
- A crash course in formal concept analysis
- Mining for source-code regularities with FCA
- The experiments in detail
- Conclusion

Conclusion

- Current status
  - Substring experiment already performed, but needs refinement
    - Mainly more advanced filtering
  - Parse tree experiment seems promising complement / extension to already existing experiment
    - Use "generic parse trees" as properties (ongoing work)
- Future work
  - Can we use FCA to mine the source-code for "aspects"?
  - Current results do seem promising enough
    - Using substrings assumes that elements corresponding to a same concern will have a similar name
    - Using generic parse trees assumes that elements corresponding to a same concern will have similar code