Delving (Smalltalk) Source Code

with Formal Concept Analysis

Pr. Kim Mens
INGI / UCL

Dr. Tom Tourwé
SEN / CWI

Monday, September 6, 2004

Overview

- Research goal
- A crash course on formal concept analysis
- Delving Smalltalk source code with FCA
- Experiments
- Results
- Conclusion

Research Goal

- A lightweight source-code mining tool
  - get "initial understanding" of structure of software system
  - detect recurring patterns in the source code
- Formal concept analysis (FCA)
  - A mathematical technique
  - Known applications in data analysis and knowledge processing
- Can we use FCA to delve code for indications of patterns?
  - Coding conventions
  - Programming idioms and design patterns
  - Opportunities for refactoring
  - Relevant domain concepts

A crash course on FCA — example

	object-oriented	functional	logic	static typing	dynamic typing
C++	Find relevant taxonomy of programming languages based on their common properties				
Java					
Smalltalk					
Scheme					
Prolog					

A crash course on FCA — theory

A. Starts from
- a set of elements
- a set of properties of those elements
- incidence table

B. 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

C. Organizes these concepts in a lattice structure

A. Incidence table

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

B. Concept 1

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

B. Concept 2

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

B. Concept 3

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

B. More concepts …

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

B. More concepts …

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

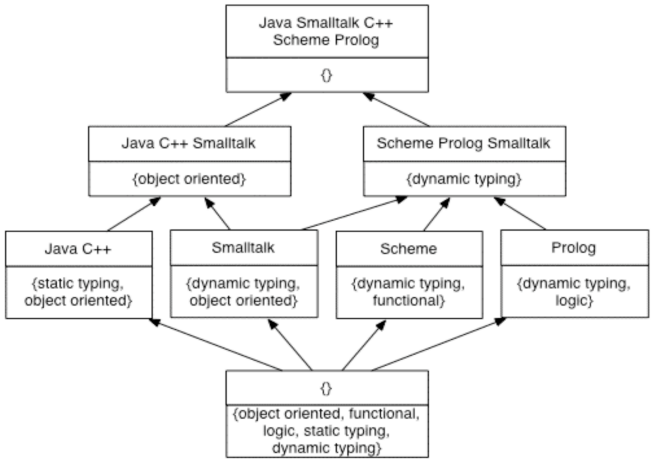
B. More concepts …

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

B. Concepts

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

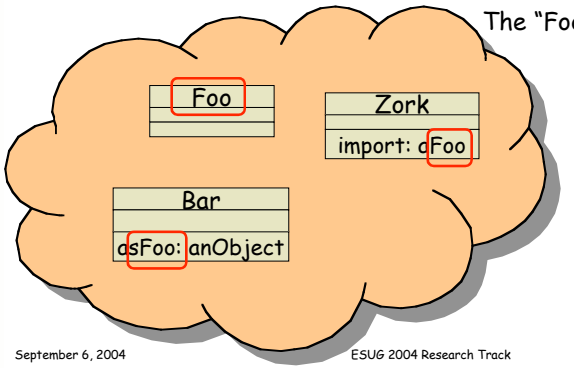
C. Concept Lattice



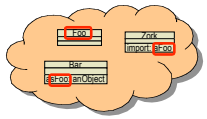
Delving ST source code with FCA

- Elements : classes, methods, argument names
- Properties : substrings of classes, methods, …

The "Foo" concept

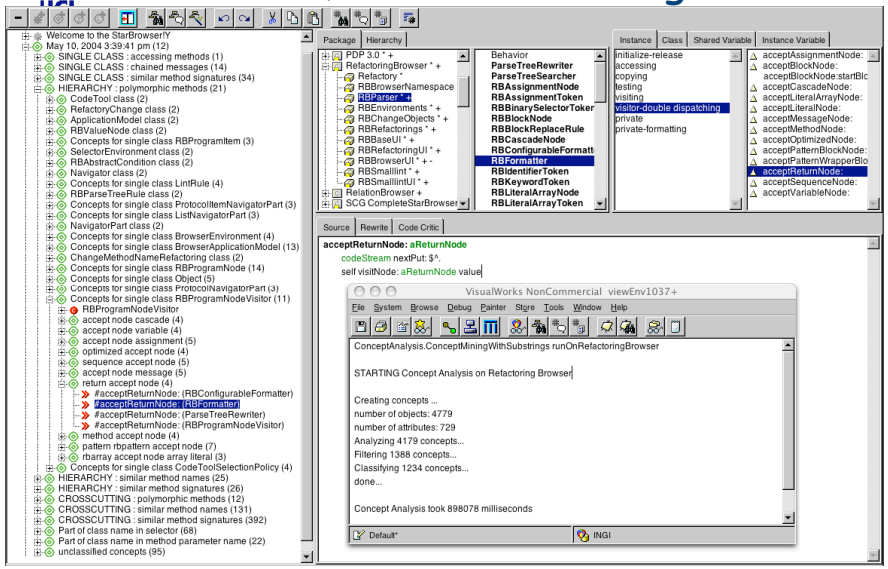


Delving source code

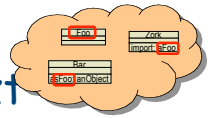


1. Generate the formal context
   - Elements, properties & incidence relation
2. Concept Analysis
   - Calculate the formal concepts
   - Organize them into a concept lattice
3. Filtering
   - Remove irrelevant concepts (false positives, noise, useless, …)
4. Classify, combine and annotate concepts
   - In a way that is more easy for a software engineer to interpret

DelfSTof, our Code Delving Tool

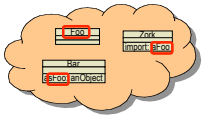


1. Generate formal context



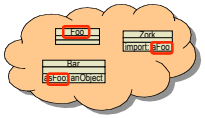
- We want to group elements that share a substring
- As elements we collect
  - all classes, methods and parameters
  - in some package(s) of interest
- As properties : "relevant" substrings of element names
  - Normalisation :
    - extract terms based on where uppercases occur
    - convert to lower case and remove special characters like ':'
    - QuotedCodeConstant → { quoted, code, constant }
  - Elimination of stopwords : with, do, object
  - Stemming : reduce words to their root
- Incidence relation : An element has a certain property if
  - It has the substring in its name

2. Concept Analysis



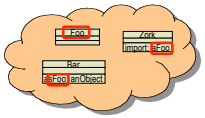
	unify	index	env	source	message	functor	variable	...
Object>>unifyWithObject: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-		-	...
Variable>>unifyWithMessageFunctor: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	X	X	-	...
AbstractTerm>>unifyWith: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-	-	-	...
AbstractTerm>>unifyWithVariable: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

2. Concept Analysis



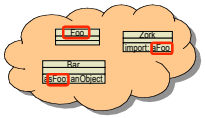
	unify	index	env	source	message	functor	variable	...
Object>>unifyWithObject: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-		-	...
Variable>>unifyWithMessageFunctor: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	X	X	-	...
AbstractTerm>>unifyWith: inEnv: myIndex: hisIndex: inSource:	X	X	X	X	-	-	-	...
AbstractTerm>>unifyWithVariable: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

3. Filtering



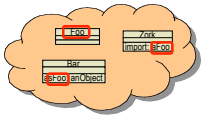
- Preprocessing to filter irrelevant properties :
  - with little meaning : "do", "with", "for", "from", "the", "ifTrue", …
  - too small (< 3 chars)
  - ignore plurals, uppercase and colons
- Extra filtering
  - Drop top & bottom concept when empty
  - Drop concepts with two elements are less
  - Drop concepts that group only classes
- 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 properties)
  - Avoid redundancy in discovered concepts
    - Make better use of the lattice structure (Now it is "flattened")

4. Classification, Combination & Annotation



- Annotate concepts with their properties
  - i.e. with the substring(s) shared by their elements
- Classification
  - Single class concepts
    - Elements are methods (or their parameters) in that class
  - Hierarchy concepts
    - Group classes, methods and parameters in same class hierarchy
    - Annotate concept with root of hierarchy
    - Annotate methods with implementing class
  - Crosscutting concepts
    - When two different class hierarchies are involved
- Combine concepts
  - that belong together (subconcept relationship)
- Group methods
  - belonging to the same class

Quantitative results

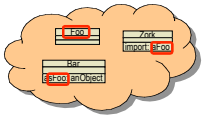


Case study	#elements	#properties	#raw	#filtered	time (sec)
DelfSTof	802 (135)	247	650	131	5
StarBrowser	731 (52)	352	740	115	7
Soul	1488 (111)	438	1206	284	22
CodeCrawler	1370 (93)	477	1419	327	24
Ref.Browser	4834 (271)	736	4228	1243	447

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

  Upperlimit: theoretical < $2^{min(\#elements, \#properties)}$; experimental < #elements

Discovered "indications" of patterns



- Code duplication
- Design patterns
  - Visitor, Abstract Factory, Builder, Observer
- Programming idioms
  - Accessing methods, chained messages, delegating methods, polymorphism
- Relevant domain concepts
  - Correspond to frequently occurring properties
  - "Unification", "Bindings", "Horn clauses", "resolution"
- Opportunities for refactoring
- Some crosscutting concerns

Conclusion

- Current status : feasibility study
  - Approach produced relevant results
  - Efficiency is acceptable
  - Tool needs refinement
    - More advanced filtering ; extra checking a posteriori
- Future work : applying FCA to delve source code for
  - aspects and crosscutting concerns
    - based on "generic parse trees"
    - by using an incidence relation that represents "message sends"
  - refactoring opportunities
  - Both Smalltalk and Java source code