SOURCE CODE ARCHEOLOGY:

USING LOGIC PROGRAM QUERIES ACROSS VERSION REPOSITORIES

Andy Kellens
Coen De Roover
Carlos Noguera
Reinout Stevens

Software Languages Lab
Vrije Universiteit Brussel

Friday 8 April 2011
Source code history comprehension:

- “Who changed this method?”
- “Why was this class introduced? When was it introduced?”
- “Why is this test broken?”
- Find out what happened in the past
VERSION REPOSITORIES

Wealth of information; how to access?
Focus on two aspects of the tool:

- How do we represent the history?
- How can we query the history?
REPRESENTING THE HISTORY
REQUIREMENTS

- Scalability
  - Lots (!) of information
- Efficient access
- Online use
- Level of detail
Model for each version

- E.g., HISMO [Girba et al.]

Scalability
DESIGN SPACE (II)

- Store the delta
  - E.g., SVN

- Efficient access
OUR SOLUTION

- Internally: store only changed entities
- Externally: offer interface to full model
- Inspired by OO implementation techniques:
  - V-table
  - Late binding while querying
- Inspired by Orion [Laval et al.]
Each entity has a unique ID
Each entity has a unique ID
Store structure up until methods; cache some interesting info
Each entity has a unique ID
Store structure up until methods; cache some interesting info
Keep a pointer to original entity (AST nodes!)
LOOKUP OF ENTITIES

Version 1

- Package P1 (ID1)
  - classes -> {ID2, ID3}
- Class C1 (ID2)
  - fields -> {ID4}
  - methods -> {ID5, ID6}
  - Method M1 (ID5)
    - refersTo -> {ID2}
  - Field F1 (ID4)
  - Method M2 (ID6)
    - readsFields -> {ID4}

Version 2

- Package P1 (ID1)
  - classes -> {ID2}
- Class C1 (ID2)
  - fields -> {ID4}
  - methods -> {ID5, ID6}
  - Method M1 (ID5)
    - refersTo -> {ID2}
  - Field F1 (ID4)
  - Method M2 (ID6)
    - readsFields -> {}

Removed class C2

Method M2 no longer reads field F1
LATE BINDING OF VERSION

pack := version1 packageWithName:'P1'.

cl := pack classWithName:'C1'.

m1 := cl methodWithName:'M1'.

m1 refersTo.
   => {C1}

pack := version2 packageWithName:'P1'.

cl := pack classWithName:'C1'.

m1 := cl methodWithName:'M1'.

m1 refersTo.
   => {}}
IMPORTING

Build model from version control system:

<table>
<thead>
<tr>
<th></th>
<th>IntensiVE</th>
<th>SOUL</th>
</tr>
</thead>
<tbody>
<tr>
<td>versions</td>
<td>175</td>
<td>137</td>
</tr>
<tr>
<td>classes</td>
<td>42.730</td>
<td>26.447</td>
</tr>
<tr>
<td>changed classes</td>
<td>5301</td>
<td>3380</td>
</tr>
<tr>
<td>methods</td>
<td>414.745</td>
<td>302.103</td>
</tr>
<tr>
<td>changed methods</td>
<td>6483</td>
<td>4144</td>
</tr>
<tr>
<td>LOC</td>
<td>1.903.215</td>
<td>1.727.777</td>
</tr>
<tr>
<td>avg #classes</td>
<td>244</td>
<td>193</td>
</tr>
<tr>
<td>avg #methods</td>
<td>2369</td>
<td>2205</td>
</tr>
<tr>
<td>avg LOC</td>
<td>10875</td>
<td>12611</td>
</tr>
<tr>
<td>Memory use</td>
<td>79,9Mb</td>
<td>30Mb</td>
</tr>
</tbody>
</table>
QUERYING THE HISTORY
Logic program querying using SOUL

Reification of concepts in representation

Explored two alternatives:

- Temporal logic (LTL)
- Regular path queries
## Reification of Entities

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>?c isClass : ?version</code></td>
<td>Entity is a class in a particular version</td>
</tr>
<tr>
<td><code>?c isClassInPackage: ?p : ?version</code></td>
<td>Class belongs to package in a particular version</td>
</tr>
<tr>
<td><code>?c isClassWithName: ?n : ?version</code></td>
<td>Class in version has name</td>
</tr>
<tr>
<td><code>?i isInterface : ?version</code></td>
<td>Entity is an interface in a particular version</td>
</tr>
<tr>
<td><code>?i isInterfaceInPackage: ?p : ?version</code></td>
<td>Interface belongs to package in a particular version</td>
</tr>
<tr>
<td><code>?i isInterfaceWithName: ?n : ?version</code></td>
<td>Interface in version has name</td>
</tr>
<tr>
<td><code>?m isMethod : ?version</code></td>
<td>Entity is a method</td>
</tr>
<tr>
<td><code>?m isMethodInClass: ?c : ?version</code></td>
<td>Entity is a class (static) method</td>
</tr>
<tr>
<td><code>?m isMethodWithName: ?n inClass: ?c : ?version</code></td>
<td>Method belongs to class</td>
</tr>
<tr>
<td><code>?p isPackage : ?version</code></td>
<td>Method with particular name in class</td>
</tr>
<tr>
<td><code>?p isPackageWithName: ?n : ?version</code></td>
<td>Entity is a package in a particular version</td>
</tr>
<tr>
<td><code>?v isInstanceVariableWithName: ?n inClass: ?c : ?version</code></td>
<td>Package with name</td>
</tr>
<tr>
<td><code>?v isClassVariableWithName: ?n inClass: ?c : ?version</code></td>
<td>Entity is instvar (field) with name in class</td>
</tr>
<tr>
<td><code>?e isOriginalEntityOf: ?x : ?version</code></td>
<td>Retrieve the original AST-node from the repository for an entity</td>
</tr>
</tbody>
</table>

### Entity relations

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>?c classInHierarchyOf: ?super : ?version</code></td>
<td>Class belongs to hierarchy of superclass in a particular version</td>
</tr>
<tr>
<td><code>?c isSubclassOf: ?super : ?version</code></td>
<td>Class is a direct subclass of superclass</td>
</tr>
<tr>
<td><code>?c isSuperclassOf: ?sub : ?version</code></td>
<td>Class is a direct superclass of a subclass</td>
</tr>
<tr>
<td><code>?c classImplementsInterface: ?i : ?version</code></td>
<td>Class implements a particular interface</td>
</tr>
<tr>
<td><code>?i interfacesImplementedBy: ?c : ?version</code></td>
<td>Interface is implemented by a particular class in a version</td>
</tr>
<tr>
<td><code>?i isSubinterfaceOf: ?super : ?version</code></td>
<td>Interface is a subinterface of a particular interface</td>
</tr>
<tr>
<td><code>?m methodReferencesClass: ?c : ?version</code></td>
<td>Method refers to a particular class</td>
</tr>
<tr>
<td><code>?m methodWrites: ?v : ?version</code></td>
<td>Method writes to a variable in a particular version</td>
</tr>
<tr>
<td><code>?e wasChanged : ?version</code></td>
<td>Entity was altered in a particular version</td>
</tr>
</tbody>
</table>

### Versions

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>?v isVersion</code></td>
<td>Entity is a version</td>
</tr>
<tr>
<td><code>?v isOldestVersion</code></td>
<td>Entity is the oldest version in the model</td>
</tr>
<tr>
<td><code>?v isMostRecentVersion</code></td>
<td>Entity is the latest version in the model</td>
</tr>
<tr>
<td><code>?v isVersionAtDate: ?d</code></td>
<td>Find the version at a particular date</td>
</tr>
<tr>
<td><code>?v isVersionBetweenDates: ?start and: ?end</code></td>
<td>Find all versions during a particular time interval</td>
</tr>
<tr>
<td><code>?d isTimestampOfVersion: ?v</code></td>
<td>Retrieve the time stamp of a version</td>
</tr>
<tr>
<td><code>?a isAuthorOfVersion: ?v</code></td>
<td>Retrieve the author of a version</td>
</tr>
</tbody>
</table>
Parameterize predicates with version

Smalltalk style syntax
REIFICATION OF ENTITIES

?version isOldestVersion
?author isAuthorOfVersion: ?version
?timestamp isTimestampOfVersion: ?version

First-class versions

Basic version information
Linear temporal logic (LTL)
E.g., all methods that never got changed

```
?version isOldestVersion,
?method isMethod : ?version,
not(finally(?method wasChanged) : ?version)
```
EVALUATION

- Very expressive
- Sometimes rather difficult
- Clumsy to express particular patterns
  - E.g., sequence of events in the history

```
1   ?method isMethod : ?version,  
2   ?class isClassWithName:MyClass : ?version,  
3   ?method methodReferencesClass: ?class : ?version,  
4   ?removed isSuccessorOf: ?version,  
5   ?method isMethod : ?removed,  
6   not(?method methodReferencesClass: ?class : ?removed),  
7   globally(not(?method methodReferencesClass: ?class)) : ?removed
```
REGULAR PATH QUERIES

cfr. De Moor et al.

Similar to regular expressions (with variables)

Match regular expression over different paths

Quantifiers

- Universal: holds for all paths (branches)
- Existential: exists a path (branch) for which holds

\[ e((?\text{class isClass})^+, \neg(?\text{class isClass})) \]
EXAMPLE OF PATH QUERY

Find all references to a class **MyClass** that was deprecated at some point

```prolog
?class isClassWithName: MyClass,
   e(
   (and(?method isMethod, ?method methodReferencesClass: ?class))+, 
   ¬(and(?method isMethod, ?method methodReferencesClass: ?class))+, 
   (and(?method isMethod, ?method methodReferencesClass: ?class)))
```

1. ?method isMethod : ?version,
2. ?class isClassWithName:MyClass : ?version,
3. ?method methodReferencesClass: ?class : ?version,
4. ?removed isSuccessorOf: ?version,
5. ?method isMethod : ?removed,
6. not(?method methodReferencesClass: ?class : ?removed),
7. globally(not(?method methodReferencesClass: ?class)) : ?removed
FUTURE WORK

- Regular expression syntax
- Examples
- Integration with tabling
- Performance evaluation
- Find a cool name :-(
SOURCE CODE ARCHEOLOGY:

USING LOGIC PROGRAM QUERIES ACROSS VERSION REPOSITORIES

HTTP://SOFT.VUB.AC.BE/SOUL