Comparative Study of Code Query Technologies

Tiago Alves

March 25th, 2009
Company

- Spin-off from CWI in 2000, self-owned, independent
- Management consultancy grounded in source code analysis
- Innovative, strong academic background, award-winning, profitable

Services

- Software Risk Assessments (snapshot) and Software Monitoring (continuous)
- Toolset enables to analyze source code in an automated manner
- Experienced staff transforms analysis data into recommendations
- We analyze over 50 systems annually
- Focus on technical quality, primarily maintainability / evolvability
Services

**DocGen**
- Automated generation of technical documentation

**Assessments**
- In-depth investigation of software quality and risks

**Monitoring**
- Continuous measurement and decision support

**Certification**
- Five levels of technical quality
Who is using our services?
Background

- **Extract**
  - Facts and relations and represent them an intermediary structure

- **Abstract/Enrich**
  - Add new facts and relations

- **Present**
  - Visualize or extract information

Eindhoven, March 25th, 2009
Comparative Study of Code Query Technologies
Tiago Alves

Software Improvement Group
Motivation

• **Current implementation of SIG tooling**
  - Extraction for several languages
    - Graph as intermediary data structure
  - Abstraction implemented in Java
    - Use of visitors
  - Presentation
    - Tables and Charts

• **Problems**
  - Implementation verbose and imperative
  - Reuse among analysis difficult
  - Error prone

• **Use of code query technologies to improve SIG productivity**
  - Replace current imperative implementation for a more declarative one
Code Query Technologies - Timeline

- **Grok, JGrok**
  - Ric Holt, Canada
  - Implemented in Turing

- **Rscript**
  - Paul Klint, Netherlands
  - Implemented in ASF+SDF

- **JRelCal**
  - Tijs van der Storm, Netherlands
  - Implemented in Java

- **GraLab, JGraLab**
  - Jürgen Ebert, Germany
  - Implemented in Java

- **SemmleCode**
  - Oege de Moor, UK
  - Implemented in Java

- **CrocoPat**
  - Dirk Beyer, Germany
  - Implemented in C

- **JTransformer**
  - Günter Kniesel, Germany
  - Implemented in Java

Eindhoven, March 25th, 2009
Comparative Study of Code Query Technologies
Tiago Alves
Comparison

- Code query example
  - Experience the language and tool

- Language criteria
  - Overview of the language features

- Tool criteria
  - Overview of the tool features
Language criteria

- **Style/Paradigm**
  - Compare implementation conciseness

- **Types**
  - Support for Integers,Chars, Strings, ...

- **Parametrization**
  - Behavior depends on a parameter value

- **Polymorphism**
  - Abstract over the fact types

- **Modularity**
  - Reuse of queries to construct other queries

- **Libraries**
  - Support of libraries of queries
Tool criteria

- **Output formats**
  - Text, preformatted text, tables, charts, others?
- **Interactive interface**
  - Command line interface (CLI), Graphical user interface (GUI), Eclipse plug-in
- **API support**
  - Invocations of the functionality from a host program
- **Interchange format**
  - To store facts from the extraction and results of abstraction
- **Extraction support**
  - None, Java, C/C++, XML, others?
- **Licensing**
  - Free, Open-source, Proprietary
Scenarios

• **Interactive use**
  - The tool is used directly by the software analyst (exploratory)
  - The user specifies and executes the queries, and extracts results

• **Tool integration**
  - The tool is used by a programmer as a component to build other tools
  - Reimplementation of existent functionality
## Criteria vs. Scenarios

<table>
<thead>
<tr>
<th>Criteria vs. Scenarios</th>
<th>Interactive use</th>
<th>Tool integration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Style/Paradigm</td>
<td>Important</td>
<td>Important</td>
</tr>
<tr>
<td>Types</td>
<td>Important</td>
<td>Relevant</td>
</tr>
<tr>
<td>Parametrization</td>
<td>Important</td>
<td>Relevant</td>
</tr>
<tr>
<td>Polymorphism</td>
<td>Important</td>
<td>Relevant</td>
</tr>
<tr>
<td>Modules</td>
<td>Important</td>
<td>Relevant</td>
</tr>
<tr>
<td>Libraries</td>
<td>Important</td>
<td>Relevant</td>
</tr>
<tr>
<td><strong>Tool</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output formats</td>
<td>Important</td>
<td>Important</td>
</tr>
<tr>
<td>Interactive use</td>
<td>Important</td>
<td>Not important</td>
</tr>
<tr>
<td>API support</td>
<td>Not important</td>
<td>Important</td>
</tr>
<tr>
<td>Interchange format</td>
<td>Important</td>
<td>Important</td>
</tr>
<tr>
<td>Extraction support</td>
<td>Important</td>
<td>Relevant</td>
</tr>
<tr>
<td>Licensing</td>
<td>Relevant</td>
<td>Important</td>
</tr>
</tbody>
</table>
**Package instability**

**Afferent Coupling**

\[ Ca = \text{# of classes outside the package that depend upon classes within the package} \]

\[ Ca = \{ (P1, C5), (P1, C7), (P2, C2) \} \]

**Efferent Coupling**

\[ Ce = \text{# of classes inside the package that depend upon classes outside the package} \]

\[ Ce = \{ (P1, C2), (P2, C5), (P2, C7) \} \]

**Package Instability**

\[ I = Ce / (Ca + Ce) \]

\[ I = \{ (P1, 1 / (2 + 1)), (P2, 2 / (1 + 2)) \} = \{ (P1, 0.33), (P2, 0.67) \} \]
Input Relations

Relations

PackageOf : Package x Class

ClassOf : Class x Method

MethodCall : Method x Method
Implementation guidelines

Defined relations

ClassDepInterPackg : Class x Class

AffCoupling : Package x Class
EffCoupling : Package x Class

AfferentCoupling : Package x N
EfferentCoupling : Package x N

PackageInstability : Package x N
Lifting implementation example

**Input relations**

- `ClassOf : Class x Method`
- `MethodCall : Method x Method`

**Compute lifting**

- `ClassDep : Class x Class`
  - `= ClassOf o MethodCall o inv (ClassOf)`

- `PackageDep : Package x Package`
  - `= PackageOf o ClassDep o inv (PackageOf)`
Grok

PackageDep := PackageOf o ClassDep o (inv PackageOf)

PackgDepInterPackg := PackageDep - (id dom PackageOf)

ClassFowardRel := (inv PackageOf) o PackgDepInterPackg o PackageOf

ClassDepInterPackg := ClassForwardRel ^ ClassDep

AffCoupling := PackageOf o (inv ClassDepInterPackg)

AfferentCoupling := indegree (AffCoupling)
rel[str,str] PackageOf
rel[str,str] ClassOf
rel[str,str] MethodCall

rel[&T1, int] indegree(rel[&T1,&T2] R)
   = { <D, #(domainR(R, {D}))> | &T1 D, &T2 U : R }

rel[str,str] ClassDepInterPackg
   = { <C1,C2> | <str P1, str C1> : PackageOf,
             , <str P2, str C2> : PackageOf,
             , <str C3, str C4> : ClassDep,
             , C1 == C3, C2 == C4, P1 != P2 }

rel[str,str] AffCoupling = PackageOf o inv(ClassDepInterPackg)

rel[str,int] AfferentCoupling = indegree(AffCoupling)

rel [str,int] PackageInstability
   = { <P1, (100*N1)/(N1+N2)> | <str P1, int N1> : EfferentCoupling,
             , <str P2, int N2> : AfferentCoupling, P1 == P2 }
Relation<String, String> packageDep
    = packageOf.compose(classDep.compose(packageOf.inverse()));

Relation<String, String> packgDepInterPackg
    = packageDep.difference(packageOf.domain().id());

Relation<String,String> classForwardRel
    = (packageOf.inverse()).compose(packgDepInterPackg).compose(packageOf);

Relation<String,String> classDepInterPackg
    = classForwardRel.intersection(classDep);

Relation<String,String> affCoupling
    = packageOf.compose(classDepInterPackg.inverse());

Relation<String,Int> afferentCoupling = affCoupling.indegree();
predicates classDepInterPackg(Class c1, Class c2) {
    c1.getPackage() != c2.getPackage() and classDep(c1, c2)
}
class MyPackage extends Package {
    MyPackage() { this.fromSource() }
    predicate affCoupling(Class c) {
        exists(Class c1 | this.contains(c1) and classDepInterPackg(c1, c))
    }    int afferentCoupling() {
        result = count(Class c | this.affCoupling(c))
    }
    float packageInstability() {
        result = (1.0 * this.efferentCoupling()) / (this.afferentCoupling() + this.afferentCoupling())
    }
}
from p : V {JavaPackage}
reportMap p,
    from outerClass : V {JavaClass}
    with
        (not p --> {PackageOf} outerClass) and
        (p --> {PackageOf} <-- {ClassDep} outerClass)
    report outerClass end
end store as AffCoupling

using AffCoupling:
from p : V {JavaPackage}
reportMap p, count(get(AffCoupling,p)) end
store as AfferentCoupling

using AfferentCoupling, EfferentCoupling:
from p : V {JavaPackage}
reportMap p, get(EfferentCoupling,p) /
    ( get(EfferentCoupling,p) + get(AfferentCoupling,p))
end store as PackageInstability
ClassDepInterPackg(c1,c2)
  := EX( p1, PackageOf(p1, c1) & EX( p2, PackageOf(p2, c2) &
  !=(p1,p2) & ClassDep( c1, c2)));

AffCoupling(p,c)
  := EX( c1, PackageOf( p, c1) & ClassDepInterPackg( c, c1));

Package(x) := PackageOf(x,_);

FOR p IN Package(x) {
  ca := #(AffCoupling(p,c));
  PRINT "AfferentCoupling ", p, " ", ca, ENDL;

  ce := #(EffCoupling(p,c));
  PRINT "EfferentCoupling ", p, " ", ce, ENDL;

  i := ce / (ca + ce);
  PRINT "Instability ", p, " ", i, ENDL;
}
classDepInterPackg(C1, C2) :-
    packageOf(P1, C1), packageOf(P2, C2),
    not(P1 = P2), classDep(C1, C2).

affCoupling(P, C) :-
    packageOf(P, C1), classDepInterPackg(C, C1).

afferentCoupling(P, N) :-
    setof(C, affCoupling(P, C), AffClasses),
    length(AffClasses, N).

packageInstability(P, I) :-
    efferentCoupling(P, Ec),
    afferentCoupling(P, Ac),
    I is Ec/(Ec + Ac).
## Language comparison

<table>
<thead>
<tr>
<th>Criteria vs. tools</th>
<th>Grok</th>
<th>Rscript</th>
<th>JRelCal</th>
<th>SemmleCode</th>
<th>CrocoPat</th>
<th>JGraLab</th>
<th>JTransformer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Style/paradigm</strong></td>
<td>Relational</td>
<td>Relational &amp; comprehensions</td>
<td>API Relational</td>
<td>OO &amp; SQL-like</td>
<td>FO-logic &amp; imperative</td>
<td>SQL-like &amp; Path expr.</td>
<td>FO-logic</td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Int</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Real</td>
<td>x</td>
<td>-</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bool</td>
<td>-</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Other</td>
<td>-</td>
<td>Composite &amp; location</td>
<td>Java</td>
<td>Class</td>
<td>-</td>
<td>Edge &amp; Node</td>
<td>Logic terms</td>
</tr>
<tr>
<td><strong>Parametrization</strong></td>
<td>-</td>
<td>x</td>
<td></td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td><strong>Polymorphism</strong></td>
<td>-</td>
<td>x</td>
<td></td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td><strong>Modules</strong></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td><strong>Libraries</strong></td>
<td>-</td>
<td>-</td>
<td></td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
</tbody>
</table>
## Tool comparison

<table>
<thead>
<tr>
<th>Criteria vs. tools</th>
<th>Grok</th>
<th>Rscript</th>
<th>JRelCal</th>
<th>SemmleCode</th>
<th>CrocoPat</th>
<th>JGraLab</th>
<th>JTransformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output formats</td>
<td>Text</td>
<td>Rstore</td>
<td>Sets &amp; Relations</td>
<td>Text, Charts, maps, graphs</td>
<td>Text, RSF</td>
<td>Text, HTML</td>
<td>Text</td>
</tr>
<tr>
<td>Interactive interface</td>
<td>CLI</td>
<td>CLI, GUI</td>
<td>-</td>
<td>CLI, Eclipse</td>
<td>CLI</td>
<td>CLI</td>
<td>Eclipse</td>
</tr>
<tr>
<td>API support</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Interchange format</td>
<td>RSF, TA</td>
<td>Rstore</td>
<td>RSF</td>
<td>-</td>
<td>RSF</td>
<td>TGraph</td>
<td>Prolog</td>
</tr>
<tr>
<td>Extraction support</td>
<td>C++</td>
<td>-</td>
<td>-</td>
<td>Java, XML</td>
<td>-</td>
<td>Java, C</td>
<td>Java</td>
</tr>
<tr>
<td>Licensing</td>
<td>-</td>
<td>BSD</td>
<td>LGPL</td>
<td>Proprietary</td>
<td>LGPL</td>
<td>GPL 2</td>
<td>EPL</td>
</tr>
</tbody>
</table>

Eindhoven, March 25th, 2009
Comparative Study of Code Query Technologies
Tiago Alves
Summary

• **Language criteria**
  - No significant differences found
  - It is not possible to implement Package Instability in Grok

• **Tool criteria**
  - Significant differences: interchange format, extraction, licensing
  - Poor support for extraction

• **Interactive use**
  - Only JRelCal is less suitable.

• **Tool integration**
  - JRelCal, SemmleCode, CrocoPat, JGraLab, JTransformer
  - Grok, Rscript only through interchange format
Conclusion

• **Compared seven code query technologies**
  • Package instability example
  • Six language criteria
  • Six tool criteria

• **Comparison not evaluation**

• **Presented findings**
  • Allow an informed decision about which tool to choose
Future work & challenges

• **Future work**
  - Add more tools / formalisms
  - Performance comparison

• **Challenges**
  - Adoption of each tool stronger points
  - Better support for libraries, interchange format and extractors
  - Availability of API
  - Interfacing through IDE

• **Research directions**
  - Analyze several versions of software
  - Architecture checking
Questions?

Software Improvement Group

W: www.sig.nl
T: +31 20 3140950
t.alves@sig.nl