Static Source Code Analysis using OCL

Toulouse, 30.09.2008

Mirko Seifert, Roland Samlaus
Outline

- **Motivation**
- Prerequisites
  - Metamodeling textual languages
- Source code analysis
- Approach to solution
  - The RestrictEd Plugin
  - Examples
  - Conclusion
  - Future work
Motivation

Need for automated Tools to analyse programs for:

- Correct behaviour
- Adhere to given coding conventions

Many tools exist to analyse source code, like SemmleCode or Checkstyle, but:

- They are language specific
- Use self defined languages for rule definition

Usage with other or new languages like DSL's not possible
Motivation

If we could use OCL to define analysis we could:

- Analyse programs for before mentioned problems
- Define rules with a standardized language
- Analyse arbitrary languages with the help of their meta model
Motivation
Contents

- Motivation
- **Prerequisites**
  - Metamodeling textual languages
- Source code analysis
- Approach to solution
  - The RestrictEd Plugin
  - Examples
  - Conclusion
  - Future work
Metamodelling textual languages – In general

- Traditionally textual languages are “metamodeled” using EBNF.
- This allows to generate lexers and parsers.
- Editors can use these tools to generate an Abstract Syntax Tree (AST) from the source code.
- Correctness of programs can be checked with the help of the AST.
Metamodelling textual languages – Using EMFText

```
compilationUnit : packageDeclaration? importDeclaration* typeDeclaration*;
```

```
public interface CompilationUnit extends EObject {
    EList<String> getPackage();
    EList<Import> getImports();
    EList<Classifier> getClassifierDeclarations();
}
```

EBNF

ECORE

Implementation
Contents

- Motivation
- Prerequisites
  - Metamodelling textual languages
- **Source code analysis**
- Approach to solution
  - The RestrictEd Plugin
  - Examples
  - Conclusion
  - Future work
Source code analysis – When?

Some facts on static source code analysis:

- Some things are checkable at development time
  - Type safety
  - Naming conventions
  - Metrics

- Others can be checked only at runtime

    public void addMoney(int n){}    // int < 1000 ?
Source code analysis – What?

Language specific constructs can prevent from runtime-errors ...

- Const, final
- Visibilities (public, private, protected)

... but they can not prevent program constructs from being changed ...

```java
class Example{
    public void addMoney(int n, float m){}  // only one Parameter of type int!
}
```

... changing the structure could lead to errors when generated code is involved
Source code analysis – What?

Coding conventions should be adhered to, e.g.:

- Attributes names should start with an underscore

  ```java
  private float _balance;  // OK
  private int taxClass;   // ERROR
  ```

- Enforcement of certain metrics like maximum number of methods per class

  ```
  # methods < 15?
  ```
Source code analysis – How?

Rules for conventions must be defined.

Question: How can we define rules?

• One could define a new language for querying code like SemmleCode
  -> This leads to language dependence (Java and XML in SemmleCode)

• A standard language like OCL can be used for definition
  -> With the help of meta models this approach leads to language independence
Source code analysis – Where?

The rules have to be saved somewhere.

- Internal (within the source code) vs. External (somewhere else)

<table>
<thead>
<tr>
<th>Location</th>
<th>Localization</th>
<th>Language independence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>External</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

OCL queries provide navigation that must be coded by hand when analysis is performed with standard program code
Contents

- Motivation
- Prerequisites
  - Metamodeling textual languages
- Source code analysis
- **Approach to solution**
  - The RestrictEd Plugin
  - Examples
  - Conclusion
  - Future work
RestrictEd

- Eclipse-plugin that checks constraints on ecore-models
- Therefore it requires editor with ecore-based AST
- Generation of lexer and parser by Reuseware EMFTextEdit from concrete syntax
- Generation of the meta model from abstract syntax
- Uses OCL as constraint language on meta model
- Each non-empty query result indicates an error in the source code
- Marks and shows errors

Toulouse, 30.09.2008

Static Source Code Analysis using OCL
Example SQL

- Some SQL-commands should be prohibited
- Generated code uses table 'bankaccounts' so deletion should be prevented

```
DROP TABLE bankaccounts;
```

- The owner of a bankaccount must not be altered

```
update accounts set (ownerName='Myself') where value>50000;
```
Example SQL

```sql
self.sqlStatements->
    select(dp | dp.oclIsKindOf(Drop) and (dp.oclAsType(Drop).v.
        value = 'bankaccount'))

self.sqlStatements->select(us|
    us.oclIsKindOf(Update)).oclAsType(Update).list->select(col|
    col.columnName.value = 'owner')
```
Example SQL

```sql
1: DROP TABLE bankaccounts;
2: UPDATE accounts SET (value = 500000) WHERE owner = 'myname';
3: UPDATE accounts SET (owner = 'myname') WHERE value > 50000;
```

Toulouse, 30.09.2008

Static Source Code Analysis using OCL
Example Java

Examples for restrictions:

• Forbid calls to specific methods

    java.sql.Connection _con;
    java.sql.Statement _st = _con.createStatement();

• Attribute names must begin with an underscore

    List<String> subjects;
Example Java

```java
public class BankAccount {
    List<String> subjects;
    Float _balance;
    Connection _con;
    Statement _st;

    public void update() {
        _st = _con.createStatement();
        _balance = 0;
    }

    public void isActive() {
        _balance = 0;
    }
}
```
Example DSL

- Language specific for a domain
- For example: A language for state charts can be defined
- There should be checks for deadlocks

```plaintext
chart Account Init: regular End: deleted {
  state regular {activity regularBanking}
  state gold    {activity goldBanking}
  state frozen  {activity wait}

  from regular to gold when upgradeToGold do {}
  from regular to frozen when freezeCustomer do {}
  from gold    to frozen when freezeCustomer do {}
```
Example DSL

```java
class Account {
    state regular {
        activity regularBanking
    }
    state gold {
        activity goldBanking
    }
    state frozen {
        activity wait
    }
    state deleted {
        activity closeAccount
    }

    trigger upgradeToGold
    trigger freezeCustomer
    trigger closure

    from regular to gold when upgradeToGold do ()
    from regular to frozen when freezeCustomer do ()
    from gold to frozen when freezeCustomer do ()
    from regular to deleted when closure do ()
}
```
## Conclusion

<table>
<thead>
<tr>
<th><strong>Pros</strong></th>
<th><strong>Cons</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>language independence</td>
<td>meta model must be created for each language</td>
</tr>
<tr>
<td>solution not restricted to one editor</td>
<td>need for ecore-based AST</td>
</tr>
<tr>
<td>rules are based on standard language (OCL)</td>
<td>OCL queries may become very large</td>
</tr>
<tr>
<td></td>
<td>restricted by the expressive power of OCL</td>
</tr>
</tbody>
</table>
Future work

- Evaluation using new languages, especially textual DSLs form interesting use cases
- Determination where OCL reaches its limits when used for source code analysis
- Getting a clear understanding about the possible types of analysis
- Compare our method with specialized tools for existing languages (can it compete?)
- Handling the complexity of OCL queries
Questions?

Thanks for your attention!
Example Java – java.sql.Connection

```plaintext
self.typeDeclarations->select(class |
 class.oclIsKindOf(Class)).oclAsType(Class).members->select(method |
 method.oclIsKindOf(Method)).oclAsType(Method).body->statements-
>select(ass | ass.oclIsKindOf(Assignment)).oclAsType(Assignment)
 .value->select(vr | vr.oclIsKindOf(VariableReference))
 .oclAsType(VariableReference)->select(v | v.variable.type
 .oclAsType(Class).name = 'Connection')
```
Example Java – enforce underscore

```java
self.typeDeclarations->select(class |
    class.oclIsKindOf(Class)).oclAsType(Class).members->select(v |
    v.oclIsKindOf(Variable)).oclAsType(Variable)->select(var|
    var.name.substring(1,1)<>'_')
```