Gra2MoL: A domain specific transformation language for bridging grammarware to modelware in software modernization

Javier Luis Cánovas Izquierdo

University of Murcia
Spain
I am nervous!
Outline

- Introduction
- Our proposal
- Gra2MoL language
- Example
- Conclusion
Introduction

- Extraction of knowledge from the source code of the system.
- Dealing with source code conforming to a grammar
Introduction

- Approaches for bridging grammarware and modelware
  - Grammar-based: xText, Wimmer et al.
  - Metamodel-based: TCS
Introduction

- Main problems detected in grammar-based approaches
  - Poor quality of metamodels extracted from grammar
  - Duplication of information
  - Modernization specific concepts
  - Reusing grammar definitions
Our proposal

- Providing a DSL for bridging grammarware and modelware technical spaces

Main features
- Considering source code as a model
- Query language
Our proposal. Main features

- Considering source code as a model
  - Grammar definition acts as metamodel
  - A syntax tree is constructed from the source model
    - The syntax tree “conforms to” the grammar definition
  - Nodes are typed in the syntax tree
  - Navigable typed syntax tree
Our proposal. Main features

- The query language
  - Models (explicit references) must be generated from syntax trees (references are identifiers)
  - An identifier-based reference must be transformed into an explicit reference
  - Grammar-to-model transformations involve intensive queries over the whole syntax tree
  - Structure-shy language to allow navigation over the syntax tree

```java
Public ClassA() {
    ClassB myClass;
    myClass.method1;
    ...
}
```

```java
Public class ClassB {
    public void method1() {
        ...}
    ...
}
```
Gra2MoL language

- A transformation definition consists of a set of rules
- Each rule is composed of four parts:
  - From
    - Specifies the source grammar symbol
    - Declares a variable bounded to a syntax tree element
    - Optionally can include a filter expression
  - To
    - Specifies the target element metaclass
  - Queries
    - Contains a set of query expressions
    - Used in the mapping part
  - Mappings
    - Contains a set of bindings
    - Bindings have an equivalent syntax and semantics of RubyTL bindings
The query language

- Two operators:
  - `/`: Immediate children of a node
  - `//`: Navigate along all node’s children retrieving all nodes of a given type

- Filter expressions
- `#` character to indicate the type of the query
The query language

- Examples in Java syntax tree

- Extracting Java class declarations

- Filtering the query
Example

Extracting KDM models from Java code

```
classBody
    : '{' classBodyDeclaration* '}'
    ;

classBodyDeclaration
    : modifier* memberDecl
      | ...
      ;

memberDecl
    : methodDeclaration
      | ...
      ;

methodDeclaration
    : type methodName=Identifier
        methodDeclaratorRest
    ;

normalClassDeclaration
    : 'class' classId=Identifier (typeParameters)?
      ('extends' type)?
      ('implements' typeList)?
      classBody
    ;

classOrInterfaceDeclaration
    : modifier* (classDeclaration | ...)
    ;

typeDeclaration
    : classOrInterfaceDeclaration
    ;

packageDeclaration
    : annotations? packageDeclaration?
      importDeclaration* typeDeclaration*
    ;

compilationUnit
    : classOrInterfaceDeclaration
      classDeclaration
      ;
```

<table>
<thead>
<tr>
<th>kdm</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>model</th>
<th>codeModel</th>
<th>codeElement</th>
<th>classUnit</th>
<th>codeElement</th>
<th>methodUnit</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>name</td>
<td>String</td>
<td>name</td>
</tr>
</tbody>
</table>
Example

```plaintext
rule 'createSegment'
    from compilationUnit cu
    to kdm::Segment
    queries
      class : /cu//#normalClassDeclaration;
    mapping
      model = new code::CodeModel;
      model.name = "codeModel";
      model.codeElement = class;
end_rule

rule 'createClass'
    from normalClassDeclaration nc
    to code::ClassUnit
    queries
      ms : /nc//#methodDeclaration[@methodName.exists];
    mapping
      name = nc.classId;
      codeElement = ms;
end_rule
```
Example

rule 'createMethod'
    from methodDeclaration md
to code::MethodUnit
queries
mapping
    name = md.methodName;
end_rule

public class BeanBulletinBoard extends ActionForm{
    private String selectedTitle = "";
    private String title = "";
    private String content = "";
    private String author = "";
    public ActionErrors validate {...}
    private boolean isNotValidTitle {...}
    public void setSelectedTitle(...) {...}
    public String getSelectedTitle() {...}
    public void setTitle(...) {...}
    public String getTitle() {...}
    public void setContent(...) {...}
    public String getContent() {...}
    public void setAuthor(...) {...}
    public String getAuthor() {...}
}
Our proposal

Diagram:

- **Grammar**
  - conforms
  - *grammarware*

- **Code**
  - *Gra2MoL Query language*

- **Model**
  - *modelware*

- **Metamodel**
  - conforms
Moreover…

- Poor quality of metamodels extracted from source
- Duplication of information
- Modernization specific concepts
- Reusing grammar definitions
Conclusion

- Proposal
  - Transformation language for bridging grammarware to modelware
    - Modernization specific features
      - Powerful query language
    - Reusing grammars

- Future work
  - Improving the navigation over the syntax tree
  - Dealing with other grammar definitions
  - Source-driven vs. target-driven language
Thanks!
Implementation

- Pre-processing phase:
  - Language defined by ANTLR grammar
  - Grammars are enriched to build an in-memory syntax tree
    - Concrete Syntax Tree (CST) metamodel
  - ANTLR generates a parser from the enriched grammar

- Main phase:
  - Gra2MoL definition
  - Transformation engine executes the Gra2MoL definition over the syntax tree
  - A target model is created
Implementation