Towards a Language Independent Refactoring Framework

Authors: Carlos López clopezno@ubu.es
Raúl Marticorena rmartico@ubu.es
Yania Crespo yania@infor.uva.es
Francisco Javier Pérezjperez@infor.uva.es
Outline

- Introduction
- Related Works
- Refactoring Framework
  - Framework Core
  - Framework Extension
- Example: UML 2.0 as Metamodel
  - Statement
  - UML Mapping
- Conclusions
Towards a Language Independent Refactoring Framework

Introduction

- Refactoring [Fowler, 2000]
  - “Process of changing a software system in such a way that it does not alter external behavior of the code yet improve its internal structure”

- Open Research Trends [Mens et al., 2004]
  - Define new refactorings
  - Identify code defects *(Bad Code Smells)*
  - Apply refactorings
  - Tool support with language independence
  - etc...
Introduction

Techniques

- Abstract Syntax Tree (AST)
- Metamodel
  - Language Independence

Goals

- Display our proposal using a metamodel
- Study the suitability of the UML 2.0 metamodel [OMG, 2004]
  - new “action” concept as support to refactoring tools
Related Works

- **FAMIX Metamodel** [Tichelaar et al., 2000]
  - Store information
    - *Class, Method, Attribute, Inheritance, etc.*
    - *Invocation*
      - represents the definition of a *Method* calling another *Method*
    - *Access*
      - represents a method body accessing an *Attribute*.
  - Aimed at the integration of several CASE tools
    - MOOSE as Refactoring tool [Ducasse et al., 2000]
  - Not contains features related to
    - Advance inheritance
    - Genericity
Related Works

- **GrammyUML** [Van Gorp et al., 2003]
  - Focus on UML 1.4.
    - From a point of view of
      - tool compatibility
      - understandability
  - Argue that UML 1.4 metamodel is inadequate
    - For maintaining the consistency between a refactored design model and the corresponding program code
  - Propose a UML 1.4 Metamodel Extension
    - Add `LocalVariable` as a specialization of `ModelElement`
    - Add `SingleTargetAction` as a specialization of `Action`
    - 6 more...
  - Not considering the following UML packages
    - Action Semantic Package
    - Genericity Package
Towards a Language Independent Refactoring Framework

Related Works

- **MOON** [Crespo 2000]
  - Minimal Object-Oriented Notation
    - minimal abstractions for refactoring
  - Storing:
    - Classes, relationships, correctness rules to inheritance, genericity, variants on the type system, etc
  - Entity
    - Any concept in source code that has a type
      - self reference, super reference, local variable, method formal argument, class attribute and function result
  - Instruction
    - creation, assignment, call and compound instructions
Towards a Language Independent Refactoring Framework

**Refactoring Framework**

- Proposed framework in previous works
  - To define, to detect and to execute refactorings

**Framework Core**

- **Functionality**
  - A. Code information
  - B. Transformation Actions
  - B. Compose Refactorings
  - C. Metric Collector
  - C. Code Defect Detection
  - D. Query System

---

Setúbal, Portugal September 2006
Towards a Language Independent Refactoring Framework

Framework Java Extension

- Introduction
- Related Works
- Refactoring Framework
  - Framework Core
  - Framework Extension
  - Example: UML 2.0 Metamodel
  - Statement
  - UML Mapping
- Conclusions
Towards a Language Independent Refactoring Framework

Example: UML 2.0 as Metamodel

- Evaluate UML 2.0 as candidate metamodel in the framework (Module A)
  - Understandability
  - Exchange model

- Include new abstractions
  - Actions
  - Genericity

- Example
  - Factory Method Design Pattern
  - Generic implementation variant
    - Programming Language Java 1.5
Example: UML 2.0 as Metamodel

Example: Class Diagram
Example: UML 2.0 as Metamodel

public class GenericFactory <P extends ProductIF>
    implements FactoryIF{

    private Class<P> c;

    public GenericFactory (Class<P> c) {
        this.c = c;
    }

    public P createProduct(){
        P product = null;
        try{product=c.newInstance();}
        catch(InstantiationException e){  }
        catch(IllegalAccessException e){  }
        return product;
    }
}
Example: Partial Code

- Generic instantiations of `GenericFactory`

- Piece of code associated to `FactoryTest`

```java
// Generic instantiation of Concrete Product1 Factory
FactoryIF factory1 = new GenericFactory<ConcreteProduct1>(ConcreteProduct1.class);

// Generic instantiation of Concrete Product2 Factory
GenericFactory factory2 = new GenericFactory<ConcreteProduct2>(ConcreteProduct2.class);
```
Example: UML 2.0 as Metamodel

- **UML Mapping**: `GenericFactory.createProduct`

- **Mapping questions**

  1. **Exception Handlers**
     - `ProtectedNode` and `ExceptionHandler` classes
     - In Behavior concretely in Action section [OMG 2004]

  2. **Instruction Sequences**
     - `Activity Diagrams` [Booch et al., 1999]
     - In Behavior concretely in Action section [OMG 2004]

  3. **Call Instructions**
     - Actions can be contained in activities that provide a context

  4. **Parametric Types**
1 Exception handlers

- **ProtectedNode**
  - Group an activity set that could throw one or more exceptions

- **ExceptionHandler**
  - Specify the action sequence to be executed in case that exceptions happen

2 Instruction Sequences

- **Activity Diagrams can model an operation** [Booch et al., 1999]
- An action flow is represented
Example: UML 2.0 as Metamodel

Mapping

```
GenericFactory.createProduct

protected Node

product = null

product = c.newInstance()

return product

ExceptionHandlers

Display Error

IllegalAccessException

InstantiationException

Display Error
```
3 Call Instructions

- Actions
  - A fundamental unit of behavior specification
  - Each activity is defined by a set of actions that provide precise semantics
  - Take input set \((\text{InputPin})\) and transform it to an output set \((\text{OutputPin})\)
  - 54 classes in UML 2.0 Action Subsystem
Example: UML 2.0 as Metamodel

Mapping

Call Instruction: `c.newInstance()`
4 Parametric Types

- Support to parameterize classifiers (Classifiers), packages (Packages) and operations (Operations)
- Support to generic instantiations
- Some subsystem classes
  - TemplateableElement
  - TemplateSignature
  - TemplateParameter
  - ParameterableElement
  - TemplateParameterSubstitution
  - TemplateBinding
Example: UML 2.0 as Metamodel

Generic class signature

class GenericFactory <P extends ProductIF>
Towards a Language Independent Refactoring Framework

Example: UML 2.0 as Metamodel

- Mapping

Generic instantiation

```
GenericFactory<ConcreteProduct1>

P : TemplateParameter
ConcreteProduct1 : Classifier : Class : ParameterableElement
```

- Introduction
- Related Works
- Refactoring Framework
  - Framework Core
  - Framework Extension
- Example: UML 2.0 as Metamodel
  - Statement
  - UML Mapping
- Conclusions
Towards a Language Independent Refactoring Framework

Conclusions

- **UML can be used to store source code**
- **High complexity in the metamodel structure**
  - Structure is three times higher than in the MOON metamodel

<table>
<thead>
<tr>
<th>MOON</th>
<th>Number of classes</th>
<th>UML</th>
<th>Number of classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>module</td>
<td>24</td>
<td>classes</td>
<td>55</td>
</tr>
<tr>
<td>inheritance</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>genericity</td>
<td>5</td>
<td>templates</td>
<td>20</td>
</tr>
<tr>
<td>instructions</td>
<td>20</td>
<td>actions</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>activities</td>
<td>52</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>56</strong></td>
<td><strong>Total</strong></td>
<td><strong>181</strong></td>
</tr>
</tbody>
</table>
Conclusions

- **Experiment is limited**
  - Not include all Object-Oriented code abstractions

- **UML structure cannot represent:**
  - Typecast
  - Multiple bounds parametric type
  - Etc.

- **MOON cannot represent:**
  - Conditionals
  - Loops
  - Etc.

- **Proposed solution:**
  - Based on minimal core (MOON metamodel)
  - UML metamodel extension as future direction
    - extending the current MOON metamodel in the same way as we have done with programming languages
Towards a Language Independent Refactoring Framework

Thank you very much