Using aspect-orientation techniques to improve the reuse of metamodels

A. M. Reina Quintero, J. Torres Valderrama
Department of Languages and Computer Systems
University of Seville
Seville, Spain
{reinaqu, jtorres}@lsi.us.es
Outline

- Introduction
- Motivation
- Background
  - Eclipse Modelling Framework
- Problem Statement: Metamodel Extension
  - Extending by means of inheritance
  - Extending by means of associations
- Improving the reuse of metamodels
  - 1\textsuperscript{st} approach: Inheritance
  - 2\textsuperscript{nd} approach: Inter-type declarations
  - 3\textsuperscript{rd} approach: Relationship Aspect
- Conclusions and further work
Introduction

Context:
Metamodel reuse

Problem:
Some of the current technologies for implementing metamodels do not always maintain obliviousness

OBLIVIOUSNESS

“AOP is quantification and obliviousness”
(R. Filman)

- 1st approach: OOP. Inheritance
- 2nd approach: AOP. Inter-type declaration
- 3rd approach: AOP. Relationship Aspect.

Let’s look at AOP

2nd Int. WS on Aspect-Based and Model-Based Separation of Concerns in Software Systems - 10th July 2006. Bilbao, Spain
Motivation: Why is it interesting the reuse of metamodels?
Motivation (I)

Model Driven Development (MDD)

Software Factories (SF)

- By Microsoft.
- Software Product Lines.
- Extensible and configurable tools.
- It integrates different activities and techniques.
- Development of different modelling languages and specific tools.

Model Driven Architecture (MDA)

- By the OMG.
- Modelling standards (UML, MOF, XMI, ...).
- Levels of modelling (CIM, PIM, PSM).
- Transformations between levels.
- Models & transformations ➔ First class citizens.
Motivation (II)

Software Factories (SF)  
Model Driven Architecture (MDA)

Modelling languages

Metamodel (MM)

Reuse of MMs  
Mechanisms for reuse introduced in Catalysis

Package Extension  
Package Template

2nd Int. WS on Aspect-Based and Model-Based Separation of Concerns in Software Systems - 10th July 2006. Bilbao, Spain
Background
Metamodel Repositories

Model Repository: Software tool that let us manage models independently of their physical representation.

- MOF-based.
  - MOF ➔ metamodelling framework by the OMG.
  - Vendor & language independent.
  - XMI ➔ XML model representation
  - JMI ➔ abstract syntax for meta-data in Java.
    Implementations: MDR from NetBeans, NSMDF from Novosoft.

- Ecore-based.
  - Ecore ➔ metamodelling framework included in the Eclipse Modelling Framework (EMF).
  - Low cost tool.
  - Language dependent.
Eclipse Modelling Framework (I)

2nd Int. WS on Aspect-Based and Model-Based Separation of Concerns in Software Systems - 10th July 2006. Bilbao, Spain
Eclipse Modelling Framework (II)

*java → EMF → *.ecore → EMF → *.java
(annotated Java) → *.genmodel → (annotated Java)

Book : EClass

library

library.impl

Book

BookImpl

2nd Int. WS on Aspect-Based and Model-Based Separation of Concerns in Software Systems - 10th July 2006. Bilbao, Spain
Problem Statement
Extending by means of inheritance (I)

package library;
import java.util.List;
/**
 * @model
 */
public interface Library
{
/**
 * @model
 */
String getName();
/**
 * @model type="Writer" containment="true"
 */
List getWriters();
/**
 * @model type="Book" containment="true"
 */
List getBooks();
}

package library;
/**
 * @model
 */
public class BookCategory
{
/**
 * @model name="Mystery"
 */
public static final int MYSTERY = 0;
/**
 * @model name="ScienceFiction"
 */
public static final int SCIENCE_FICTION = 1;
/**
 * @model name="Biography"
 */
public static final int BIOGRAPHY = 2;
Extending by means of inheritance (II)

```java
package schoollibrary;
import library.Library;
/**
 * @model
 */
public interface SchoolLibrary extends Library {
    /**
     * @model
     */
    String getLocation();
}
```

```java
package schoollibrary;
import library.Book;
/**
 * @model
 */
public interface SchoolBook extends Book, Asset {
}
```
Extending by means of associations (I)
Extending by means of associations (II)

package com.metamodel.java2;
import org.eclipse.emf.common.util.EMap;
/**
 * @model
 * @model
 * public interface Class extends Classifier{
 * @model
dataType="com.metamodel.java2.Block"
 * @model
dataType="com.metamodel.java2.Block"
 * Block getStaticInit();
 * Block getStaticInit();
 * /**
 * @model
dataType="com.metamodel.java2.Block"
 * dataType="com.metamodel.java2.Block"
 * Block getInstanceInit();
 * Block getInstanceInit();
 * /**
 * @model
type="com.metamodel.java2.Block"
 * opposite="thrownExceptions"
 * EList getBehavioralFeatures();
 * EList getBehavioralFeatures();
 */
}
Extending by means of aspectcut

```
package com.metamodel.aspectj;
import org.eclipse.emf.common.util.EList;
import com.metamodel.aspectj.
import com.metamodel.java2.Class;
import com.metamodel.java2.Parameter;

/**
* @model
* /
public interface Pointcut extends Feature {
    /**
     * @model
     * type="com.metamodel.java2.Class"
     * lowerBound="1" upperBound="1"
     * /
    Class getDeclarer();

    /**
     * @model
     * type="com.metamodel.java2.Parameter"
     * containment="true"
     * /
    EList getParameters();

    /**
     * @model type="PointcutExpression"
     * containment="true" lowerBound="0"
     * upperBound="1"
     * /
    PointcutExpression getPce();

    /**
     * @model type="PointcutExpression"
     * opposite="pcOperand"
     * /
    EList getExpression();
}
```
Conclusions

- Inheritance does not cause any problem regarding to the extension of metamodels.
- Associations and compositions may cause difficulties.
Improving the reuse of metamodels
1st Approach: Inheritance

Idea:
- To leave inheritance as the only possible relationship between package elements ⇒ Introduction of a new fictitious class.

Advantages:
- Simplicity.

Disadvantages:
- Metamodel is scattered.
2nd Approach: Inter-type declarations

Idea:
- To introduce a new aspect constructor.
- To introduce inter-type declarations.

```java
import com.metamodel.java2.Class;
import org.eclipse.emf.common.util.EList;
/**
 * @model
 */
public aspect ClassAsp {
    /**
     * @model type="com.metamodel.aspectj.Pointcut"
     * containment=true
     */
    public EList<Class>.getPointcuts();
}
```

Advantages:
- It uses the AspectJ syntax.

Disadvantages:
- The AspectJ compiler does not allow the introduction of inter-type declaration in interfaces.
3rd Approach: Relationship Aspect (I)

Idea:
- To treat relationships as first-class citizens at implementation.

"Relationships Aspects" (Pearce&Noble, Proc. AOSD06)

import java.util.HashSet;
public class Student {
    String name;
    Integer number;
    HashSet<Course> attends;
}

import java.util.HashSet;
public class Course {
    String code;
    String title;
    Integer workload;
    HashSet<Student> attendees;
}
3rd Approach: Relationship Aspect (II)

```java
import java.util.HashSet;

public class Course {
    String code;
    String title;
    Integer workload;
}

public class Student {
    String name;
    Integer number;
}

public aspect Attends extends StaticRel<Student, Course> {
}
```
3rd Approach: Relationship Aspect (III)

```java
import com.metamodel.java2.Class;
/**
 * @model
 **/
public aspect PointcutDeclaration extends CompositionRel <Class, Pointcut>{
}
```

Advantages:
- Clarity and cohesion of the implementation of each metaclass are increased.
- Relationships are better localized.
- The syntax is totally compatible with the AspectJ compiler.

Disadvantages:
- Current implementation of the RAL library does not let us define two different relationships between the same classes due to name problems.
- The RAL library should be extended with new kinds of relationships.
Conclusions

- A problem for extending metamodels has been shown.
- Three different approaches have been introduced to address the problem.
  - 1st approach: Inheritance.
  - 2nd approach: Inter-type declarations.
  - 3rd approach: Relationship aspect.
- The pros and cons of the three approaches have been analyzed, and as a result, we think that the 3rd approach is the better one, because:
  - Relationships are better localized, defined and the coupling of the metamodel implementation is reduced.
  - We can take advantage of the AspectJ compiler.
  - Relationships themselves can be reused.
Further work

- Adaptation of the RAL library to our needs.
  - New kinds of relationships.
  - To solve the name problem.

- Add aspect-oriented features to current modelling and metamodelling frameworks.
Using aspect-orientation techniques to improve the reuse of metamodels

A. M. Reina Quintero, J. Torres Valderrama
Department of Languages and Computer Systems
University of Seville
Seville, Spain
{reinaqu, jtorres}@lsi.us.es

Thanks for your attention!