Systematic Analysis of Crosscutting Concerns in the Model-Driven Architecture Design Approach

Mehmet Aksit
CTIT and Department of Computer Science
P.O. Box 217
7500 AE Enschede, The Netherlands
aksit@ewi.utwente.nl
trese.cs.utwente.nl/~aksit

Table of contents

- Principles of software development
- Constructability challenges
- Composition versus transformation
- Understanding aspect-oriented modeling using an example scenario
- Assessments
- Conclusions
Principles software development

Fundamental processes in software engineering

Simple case
- Requirements
- Problems
- Solutions (Software)

More realistic case
- Requirements
- Problems
- Solutions
- Problems
- Solution (Software)
Definitions

**Computability:**
is defined as a property of a program (function) that can be computed by a Turing machine.

**Constructability:**
is an ability to create a computable solution model that retains its desired quality values.

The MDA Perspective

- **CIM** (Computation Independent Model)
- **PIM** (Platform Independent Model)
- **PSM** (Platform Specific Model)
- **Code**

transformation
Abstractness constraint:
The elements of a PSM must be abstract enough to match the elements of PIM but also must be concrete enough to match the elements (or components) of implementation (processor) architectures.

This constraint aims at minimizing the effort in refining PIM models!
Standardization constraint:
Elements of implementation architectures must be standardized to ease sharing among multiple PIMs but must be different enough to match the needs of multiple PIMS.

This constraint aims at reducing costs through sharing implementations of PIMs.

Constructability design space

Abstractness

Solution

Ideal: difficult to realize

Computation

Standard

Dedicated

Standardization

Minimum effort

Maximum cost

Minimum cost

Maximum effort

© Mehmet Aksit

Systematic analysis of crosscutting concerns in the MDA design approach
Systematic analysis of crosscutting concerns in the MDA design approach

Composition versus transformation

Traditional view on reuse models (integration and evolution)

- Compositional reuse
- Transformational (generative) reuse

How do they relate to each other?
Composition versus transformation (cont’d)

- Any composition operator can be expressed as a transformation

- It is harder to express an arbitrary transformation as a composition
Aspect-oriented modeling

Example with an evolution scenario

- Introduce an example: Concurrent file versioning system
- PIM of the concurrent file versioning system
- Relational PSM of the concurrent file versioning system
- Java PSM of the concurrent file versioning system
- Evolution of PIM by adding security of logging
- Evolution of the relational PSM
- Evolution of the Java PSM
- Observations
Example:

Concurrent file versioning system

Henninger, MSc thesis 2004

Check in File

Check Out File

Merge Changes

Commit Changes

PIM of the concurrent file versioning system
Relational PSM of the concurrent file versioning system

<table>
<thead>
<tr>
<th>File</th>
<th>name</th>
<th>contents</th>
<th>size</th>
<th>deleted</th>
<th>timestamp</th>
<th>checkin()</th>
<th>checkout()</th>
<th>difference()</th>
<th>commit()</th>
<th>remove()</th>
<th>update()</th>
<th>tag()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>name</td>
<td>number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Directory</td>
<td>name</td>
<td>number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repository</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example of repository and file entities:
- Repository: ID, name, create(), delete(), getName(), getNumber(), add(), find(), update()
- File: ID, name, contents, size, deleted, timestamp, checkin(), checkout(), difference(), commit(), remove(), update(), tag()
- Version: ID, number, timestamp, checkin(), checkout(), difference(), commit(), remove(), update(), tag()
- Tag: ID, name, add(), find(), update(), delete()
- Directory: ID, name, number

Java PSM of the concurrent file versioning system

Example of repository and branch entities:
- Repository: name, create(), delete(), getName(), getNumber()
- Branch: branchID, name, create(), delete(), parent, getBranch(), add(), find(), update()
Systematic analysis of crosscutting concerns in the MDA design approach

PIM of the CFVS with security and logging concerns

PIM of the CFVS with security and logging concerns (changed logging)
Systematic analysis of crosscutting concerns in the MDA design approach

Relations PSM of the CFVS with security and logging concerns

Hidden crosscutting concerns: aspects are in the queries!

Java PSM of the CFVS with security and logging concerns
Overview

**Concern:** Considered important by the stakeholder(s)

**Aspects** (crosscutting concerns) may **complicate models** and **hinder evolution**;

Aspects are model dependent:

- Modular in aspect-oriented modeling
- Crosscutting (UML and Java based models)
- Hidden (Relational PSM)
- Aspects are **subjective**; depend on the interest of the stakeholders

Composition versus transformation
(Revisited)
Modeling composition and transformation

<table>
<thead>
<tr>
<th>Number</th>
<th>Primitive Transformation Pattern</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>A \xrightarrow{T_A} A'</td>
<td>This is the most basic transformation, where A is transformed to A' via T_A. This is likely the most used transformation, especially when a model is transformed from a higher to a lower level modeling.</td>
</tr>
<tr>
<td>P2</td>
<td>A \xrightarrow{T_A} A', B</td>
<td>This is the absorbing transformation, where B is transformed, together with A to a single model A'. In the resulting model it is not possible or easy to identify the different elements of the transformed model B.</td>
</tr>
<tr>
<td>P3</td>
<td>A \xrightarrow{T_A} A', B'</td>
<td>This is the opposite transformation as the above one, the extracting transformation. In this case A is transformed to a composed model of A' and B'. It is also possible that T_A introduced a model or requirement that gets expressed by B' composed with A'.</td>
</tr>
<tr>
<td>P4</td>
<td>A \xrightarrow{T_A} A', B', C</td>
<td>This primitive is used to transform the composition operator. It is necessary, as the composition needs to be maintained after the transformation of the models that are composed with each other.</td>
</tr>
</tbody>
</table>

Assessment

- Aspects can be preserved in both source and target models (like our examples of PIM and java PSM)
- Aspects can be modeled modularly (if AO models are used in PIM and/or PSMs)
- Aspects in the source model but disappear in the target model (Due to expressive models or not interested in them anymore)
- Aspects are introduced in the target model (because they are relevant in the target model or target model is not expressive)
Assessment (2)

- What is an aspect depends on the interest of the stakeholders at a given level of abstraction (model)
- The expression power of modeling language is important for representing aspects modularly.
- An entity may not be considered as an aspect after transformation (or vice versa)
- We know that transformation languages are inevitably depend on source and/target (meta) models
- To explicitly manage all with respect to the desires of the stakeholders, should the transformational languages consider aspects explicitly?
- What are the crosscutting concerns at the level of transformation languages?

Assessment (3)

- For minimizing the effort in transforming models (abstractness constraint) the abstractions (modules) used in the target model must be semantically closer to the abstractions (modules) in the source model.
  - For example, if aspects are modeled explicitly in the source model they must be also modeled explicitly in the target model
  - Requires domain specific aspect languages
- For minimizing costs through sharing (standardization constraint) sharing the target models is essential. Since non aspect models are special case of aspect models, all target and source models must be aspect-oriented.
Conclusions

- It is better to have **aspect-oriented source and target models**
- It is better to have **domain specific aspect modeling languages**
- It looks like that transformation languages must be **aspect aware**; we should also represent aspects at the level of transformation languages.
- MDA methods (**MD Engineering**) must be able to manage aspects, since what is an aspect is stakeholder dependent