Integration of UML Profiles into the SiDiff and SiLift Tools

Based on a SysML case study

Master’s Thesis (24.09.2013)

Dennis Reuling
1 Introduction
2 SysML Case Study
3 Environment and Tools
4 Integration of UML Profiles
5 Results
6 Conclusion
7 Future Work
8 Literature
There are two important steps in the creation of a tool:

1. Create the tool itself
2. Test and analyze the tool using case studies

Case studies can be obtained through

a) „Real World“ studies
   - Hard to obtain and most authentic
b) Generated studies (e.g. via SMG)
   - Somewhat hard to obtain and less authentic
c) Manually created studies (e.g. via EcoreEditor)
   - Easy to obtain but not authentic
Getting a „Real World“ study to work in the SiDiff/SiLift ecosystem has several advantages:

- Proof the **practice-oriented** focus of these tools
- Analyze the tools in a **real** environment
- Test the tools in a (most often) more **complex** environment
- Errors in this study can be used as **worst-case-scenario** for testing the tools
- A **generic** integration of UML Profiles extends the supported modeling domains **massively**
- All **future** tools can make use of these advantages
UML Profiles Introduction

- Are used to **extent** (subsets of) UML
- They **comply** to all UML standards
- Define specialized and semantically more understandable DSMLs
- No need for **new** modeling tools

**Figure**: Profile Application Example [UML13]
SysML Introduction

Systems Modeling Language
- Defined as extension to a subset of **UML** using the **profiling** mechanism
- Domain-specific for **Systems Engineering** applications
- Developed for better **accessibility** in this particular areas
- Used in automotive and embedded areas

Figure: UML and SysML [Obj13]
Figure: The SysML Diagram Taxonomy [Obj12]
A real industrial Pick-and-Place unit, constructed by TU Munich [LFVH13], with the following features:

- Based on **SysML** as modeling language
- Constructed via **Papyrus**
- In Revision 0 the PPU picks up an **Workpiece** and places it onto the **Slide**
- 13 Revisions with changes are available
- Makes use of unique identifiers

**Figure:** PPU [LFVH13]
Every revision has been analyzed regarding the type of issue:

<table>
<thead>
<tr>
<th>Technical Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issues which can cause problems of technical nature, for example in model processing tools.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pragmatical Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issues which can cause problems of semantical nature, for example the understanding and accessibility of the model.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minor Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issues which can cause problems of minor importance, such as bad variable naming schemes.</td>
</tr>
</tbody>
</table>
Wrong UUIDs between revisions

Revision 0

Revision 1
Wrong UUIDs between revisions

Revision 0

Revision 1
Wrong UUIDs between revisions

**UML Profiles Integration**

Dennis Reuling

**Introduction**

**SysML Case Study**

**Introduction**

**Analysis**

**Technical Issues**

**Pragmatical Issues**

**Environment and Tools**

**Integration of UML Profiles**

**Results**

**Conclusion**

**Future Work**

**Literature**

---

**Revision 0**

**Revision 1**

---

**Master's Thesis (24.09.2013)**
Wrong UUIDs between revisions
Wrong UUIDs between revisions
Wrong UUIDs summary

How detected? ▶ Later!
Usage of EAnnotations

**Figure**: Papyrus-specific EAnnotations
Global elements

■ Assumption: Are declared globally
■ No need for declaration in any diagram

Undefined elements

■ Assumption: Are declared locally
■ Have to be declared in at least one diagram
Undefined Element Types

SysML PP-Unit report

Types of undefined elements

- Pseudostate
- State
- FinalState
- Transition
- Enumeration
- Port
- FlowPort
- Association
- Operation
- Block

Scene00  Scene01  Scene02  Scene03  Scene04a_and_b  Scene05  Scene07  Scene08  Scene09  Scene10  Scene11  Scene12  Scene13
Introduction

- **Graph transformation** tool with graphical syntax and editor
- Can be used for **matching** and/or for transformation of (sub)graphs
- Rules for matching and/or transforming are called **Henshin Rules**
- The **SiLift** framework is based on Henshin Rules as input „language“
Introduction

Meta model-independent **comparison** approach

Has three main matching services:

1. ID-based matcher
2. Signature-based matcher
3. Similarity-based matcher

Highly **customizable** via XML configurations

Can be extended via new **SiDiff Services**
Overview

UML Profiles Integration

Dennis Reuling

Introduction
SysML Case Study
Environment and Tools
Henshin
SiDiff
SiDiff
SiLift
Integration of UML Profiles
Results
Conclusion
Future Work
Literature
Introduction

- Presents differences in a **meaningful** way
- **Lifts** low-level operations/changes semantically
- End-users can comprehend the changes as they are presented as edit operations
- Supports two categories of edit rules:
  1. Generated **atomic** edit rules (required)
  2. Manually created **complex** edit rules (optional)
Overview

UML Profiles Integration

Introduction
SysML Case Study
Environment and Tools
Henshin
SiDiff
SiLift
Integration of UML Profiles
Results
Conclusion
Future Work
Literature
For full UML Profiles integration every pipeline step must be taken into consideration:

1. Profile support in **Matching** service
2. Profile support in **Lifting** service
3. Profile support in **Patching** service

Important aspects for integration:

- **Generic** integration for profiling mechanism
- **Adapt** as many tools from the ecosystem as possible
- Create **new** services and tools if necessary
Integration overview
Matching approaches

To match **stereotyped** elements, there are two possible approaches:

**SiDiff Profile configuration**
- No new matching service has to be created
  - Profiled elements do not own much "semantic“ meaning useful for similarity-based matching
  - Each UML Profile needs its own configuration

**Additional Matcher**
- No configuration necessary
- All UML Profiles are generically supported
- UML configuration is used for unprofiled elements
  - New matching service has to be created
Profiles matching example

Basic example for matching profiled elements:

Model A

Microswitch1 : Class
- active = true
- visibility = public
- xmlID = _3fX12...

: Block
- isEncapsulated = false
- xmlID = _-X31Fg...

base_Class

Model B

Microswitch2 : Class
- active = true
- visibility = public
- xmlID = _3fX12...

: Block
- isEncapsulated = false
- xmlID = _-X31Fg...

base_Class
Profiles matching example

**UUID-Matcher: Generic matching, no difference:**

- **Model A**
  - **Microswitch1**: Class
    - active = true
    - visibility = public
    - xmlID = _3fX12...
  - **: Block**
    - isEncapsulated = false
    - xmlID = _-X31Fg...

- **Model B**
  - **Microswitch2**: Class
    - active = true
    - visibility = public
    - xmlID = _3fX12...
  - **: Block**
    - isEncapsulated = false
    - xmlID = _-X31Fg...
Profiles matching example

Similarity-Matcher: Matching base elements:

Model A

Microswitch1 : Class
active = true
visibility = public
...
xmIID = null

: Block
isEncapsulated = false
xmIID = null

Model B

Microswitch2 : Class
active = true
visibility = public
...
xmIID = null

: Block
isEncapsulated = false
xmIID = null
Profiles matching example

Similarity-Matcher: How to match profiled elements?

Model A

Microswitch1 : Class
  active = true
  visibility = public
  ...
  xmlID = null

: Block
  isEncapsulated = false
  xmlID = null

Model B

Microswitch2 : Class
  active = true
  visibility = public
  ...
  xmlID = null

: Block
  isEncapsulated = false
  xmlID = null

Dennis Reuling
Profile-Matcher: Match according to base elements:

Model A

Model B

ProfileMatcher approach
ProfileMatcher

- Implemented as a **SiDiff-Service**
- Can be **integrated** easily
- Used as **final** matching service for best results
- Makes use of **semantic** information of base elements of preceded matching services
- Depends on UML SiDiff **configuration**
Wrong UUID example
**UUID-Fixer**

- Implemented as a **SiDiff-Service**
- Can be **integrated** easily
- Used as **final** SiDiff-service
- SiDiff has only to be run once, UUID-Matcher is **sufficient** afterwards
- UUID-Fixing makes models **compatible** to other tools depending on right UUIDs
Edit Rule Profile integration approaches

UML Profiles Integration

Introduction
SysML Case Study
Environment and Tools
Integration of UML Profiles
Overview
SiDiff
SiLift
Patching
Results
Conclusion
Future Work
Literature

Variant 1
Variant 2
Variant 3
ProfileApplicator Overview

- Implemented as standalone **OSGi-Application**
- **Transforms** all files in given folder according to defined configuration via Higher-Order-Transformations
- **Minimal** configuration needed, more configuration options (Whitelist, . . .) available
- Configuration and execution oriented after **SERGe**e paradigms
Higher-Order-Transformations

- Transforming Henshin Rules with Henshin Rules is called **Higher-Order-Transformation (HOT)**
- The ProfileApplicator is based on this feature of Henshin
- Henshin rules are defined as usual, but the elements to transform are elements of Henshin rules **itself**:

```
Rule createClass_IN_PackageSelected, New, Name, Visibility, IsLeaf, IsAbstract...

Rule createBlock_IN_PackageSelected, New, Name, Visibility, IsLeaf, IsAbstract, IsFinalSpecialization, IsActive, New_B...
```

![Diagram of Higher-Order-Transformation](image)
HOT for Create-Nodes
HOT for Preserve-Nodes

UML Profiles Integration

Dennis Reuling

Introduction
SysML Case Study
Environment and Tools
Integration of UML Profiles
Overview
SiDiff
SiLift
Patching
Results
Conclusion
Future Work
Literature

Rule Preserve:StereotypeEditRule(stereotypePackage, stereotype, stereotypeName, baseType, baseTypeName, baseReference, MN, MD, RN, RD)
For better lifting results the **domain** engineer can define complex edit rules:

- A complex edit rule **consists** of 2 or more atomic edit rules
- Defines a **common** edit operation
- Are **optional** on the contrary to atomic edit rules
- Require very good **domain-specific** knowledge
Complex edit rule example

Rule createBlockInteractingVia_FlowPorts(Selected1, Selected2, NewClass, NewBlock, NewPort, NewFlowPort, NewPort2, NewFlowPort2, isa...
For **creation** of patches all low-level changes must be (at least) lifted to atomic edit operations

For **application** of patches all parameters of the patch must be set correctly
Model $B'$ needs to contain all changes between $A_1$ and $A_2$ for full correctness:

If $B = A_1$ then $B' = A_2$
General Results

- Between **all** revision changes can be:
  - Matched
  - Lifted
  - Patched

- „Real World“ study is **considerably** larger and complexer than previous studies

- This leads to:
  - Very **time consuming** calculations through all pipeline steps
  - **Not** very accessible and easy to debug for edit rule engineer
  - Good testing possibilities for **all** tools
Results for SysML case study

Integration of UML Profiles

Results for SysML case study
Results for SysML case study

<table>
<thead>
<tr>
<th>Revision Change</th>
<th>Correspondences</th>
<th>Differences</th>
<th>Operations</th>
<th>Equal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00→01</td>
<td>545</td>
<td>58</td>
<td>16</td>
<td>Yes</td>
</tr>
<tr>
<td>01→02</td>
<td>545</td>
<td>203</td>
<td>86</td>
<td>Yes</td>
</tr>
<tr>
<td>02→03</td>
<td>575</td>
<td>764</td>
<td>231</td>
<td>Yes</td>
</tr>
<tr>
<td>03→04</td>
<td>774</td>
<td>9</td>
<td>9</td>
<td>Yes</td>
</tr>
<tr>
<td>04→05</td>
<td>756</td>
<td>654</td>
<td>201</td>
<td>Yes</td>
</tr>
<tr>
<td>05→07</td>
<td>904</td>
<td>165</td>
<td>102</td>
<td>Yes</td>
</tr>
<tr>
<td>07→08</td>
<td>927</td>
<td>103</td>
<td>28</td>
<td>Yes</td>
</tr>
<tr>
<td>08→09</td>
<td>943</td>
<td>298</td>
<td>94</td>
<td>Yes</td>
</tr>
<tr>
<td>09→10</td>
<td>1008</td>
<td>367</td>
<td>111</td>
<td>Yes</td>
</tr>
<tr>
<td>10→11</td>
<td>1099</td>
<td>83</td>
<td>28</td>
<td>Yes</td>
</tr>
<tr>
<td>11→12</td>
<td>1107</td>
<td>436</td>
<td>143</td>
<td>Yes</td>
</tr>
<tr>
<td>12→13</td>
<td>1216</td>
<td>95</td>
<td>40</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Summary

UML Profiles integration into every pipeline step:

- **SERGe**
  - Atomic Edit Rules
  - Difference Detection

- **Unprofiled Edit Rules**
  - Complex Edit Rules

- **Profile Applicator**
  - Profiled Edit Rules
  - Low-Level Changes

- **Recognition Rule Generator**
  - Recognition Rules
  - Operation Detection

- **SiDiff**
  - Correspondences

- **SiLift**
  - HL Edit Operations

- **Patch-Tool**
  - Apply Patch
  - Asymmetric Diff.
  - Create Patch

**Integration of UML Profiles**

**UML Profiles Integration**
Dennis Reuling

**Introduction**
**SysML Case Study**
**Environment and Tools**
**Integration of UML Profiles**
**Results**
**Conclusion**
**Future Work**
**Literature**
Summary

UML Profiles integration into every pipeline step:
Summary

UML Profiles integration into every pipeline step:

- **SERGe**
  - Atomic Edit Rules
  - Difference Detection

- **Unprofiled Edit Rules**
  - Complex Edit Rules

- **Profile Applicator**
  - Profiled Edit Rules

- **Recognition Rule Generator**
  - Recognition Rules

- **SiDiff**
  - Correspondences

- **SiLift**
  - HL Edit Operations

- **Patch-Tool**
  - Asymmetric Diff.

**UML Profiles Integration**

Dennis Reuling

**Introduction**

**SysML Case Study**

**Environment and Tools**

**Integration of UML Profiles**

**Results**

**Conclusion**

**Future Work**

**Literature**

Master's Thesis (24.09.2013)
Summary

UML Profiles integration into every pipeline step:

- **SERGe**
- **Unprofiled Edit Rules**
- **Profile Applicator**
- **Profiled Edit Rules**
- **Recognition Rule Generator**

**Low-Level Changes**

- **Difference Detection**

**Correspondences**

- **SiDiff**
- **Profile Matcher**
- **UUID Fixer**
- **Similarity Matcher**
- **UUID Matcher**

**Correspondences**

- **SiLift**
- **Create Patch**
- **Asymmetric Diff.**

**HL Edit Operations**

- **Patch-Tool**
- **Apply Patch**

**Model B**
Summary

UML Profiles integration into every pipeline step:

- SERGe
- Unprofiled Edit Rules
- Profile Applicator
- Profiled Edit Rules
- Recognition Rule Generator
- Recognition Rules
- Operation Detection
- Low-Level Changes
- Correspondences
- Model A
- Model A
- UUID Matcher
- Similarity Matcher
- Profile Matcher
- UUID Fixer
- Create Patch
- Apply Patch
- Asymmetric Diff.
- Model B
- Model B
- Patch-Tool
- HL Edit Operations
- Correspondences
- Correspondences
- Correspondences
- Correspondences
- Correspondences
Summary

UML Profiles integration into every pipeline step:

![Diagram showing the integration of UML Profiles into every pipeline step]
UML Profiles integration into every pipeline step:
The following aspects could be considered in future work:

- Extensive testing of implemented tools and services regarding other UML profiles like MARTE
- Construct more complex edit rules for better lifting results
- Integrate approach 1 of profiling edit rules into SERGe
- Implement approach 2 of profiling edit rules as standalone tool with following features:
  - Combine base type edit rules with profiled ones
  - Combine atomic edit rules into complex edit rules without the need of manually creating the latter.
- Performance optimization for large and real models in all tools and pipeline steps

