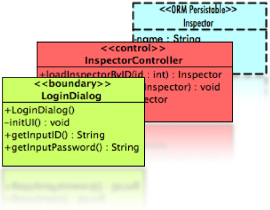
Integration of UML Profiles into the SiDiff and SiLift Tools

Based on a SysML case study



Master's Thesis (24.09.2013)

Dennis Reuling

Agenda

1 Introduction

2 SysML Case Study

3 Environment and Tools

4 Integration of UML Profiles

5 Results

6 Conclusion

7 Future Work

8 Literature

Motivation (1)

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

Motivation (2)

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 **DSML**s
- No need for **new** modeling tools

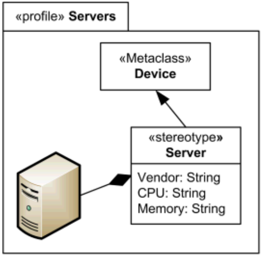


Figure : Profile Application Example [UML13]

SysML Introduction

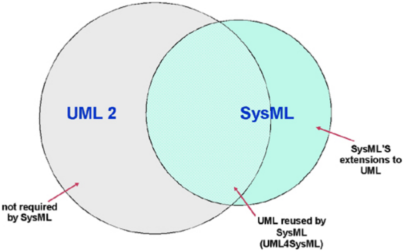


Figure : UML and SysML [Obj13]

Systems **M**odeling **L**anguage

- 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

SysML Overview

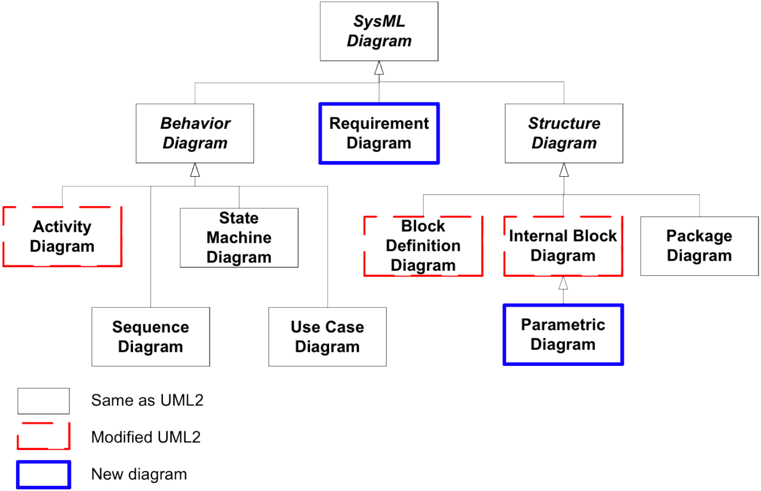


Figure : The SysML Diagram Taxonomy [Obj12]

AIS case study(SPP1593)

A **real** industrial Pick-and-Place unit, constructed by TU Munich [LFVH13], with the following features:

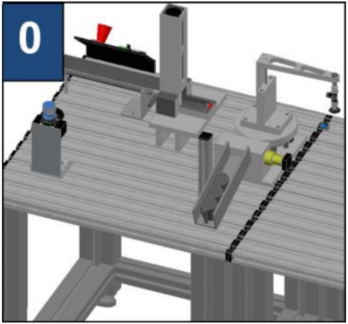


Figure : PPU [LFVH13]

- 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

Types of Issues

Every revision has been analyzed regarding the type of issue:

Technical Issues

Issues which can cause problems of technical nature, for example in model processing tools.

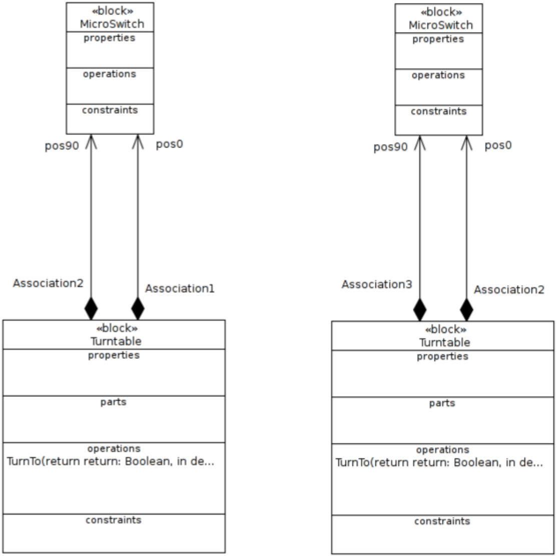
Pragmatical Issues

Issues which can cause problems of semantical nature, for example the understanding and accessibility of the model.

Minor Issues

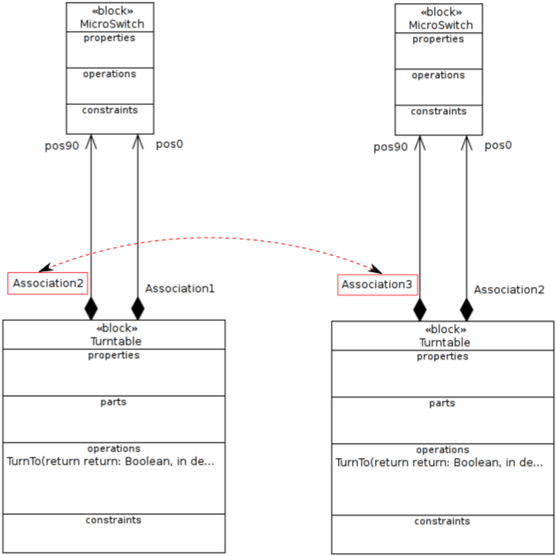
Issues which can cause problems of minor importance, such as bad variable naming schemes.

Wrong UUIDs between revisions



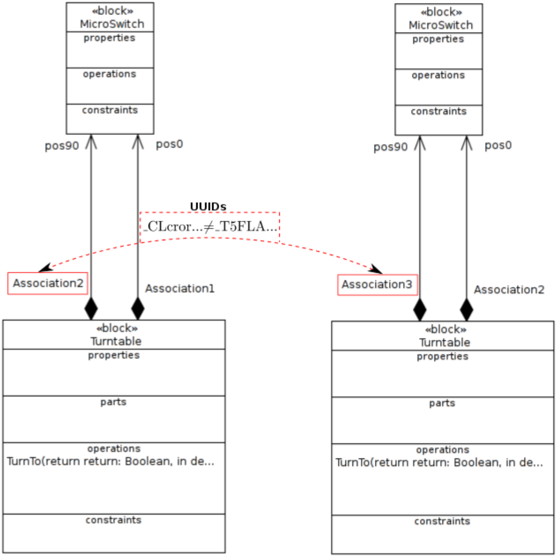
Revision 0 **Revision 1**

Wrong UUIDs between revisions



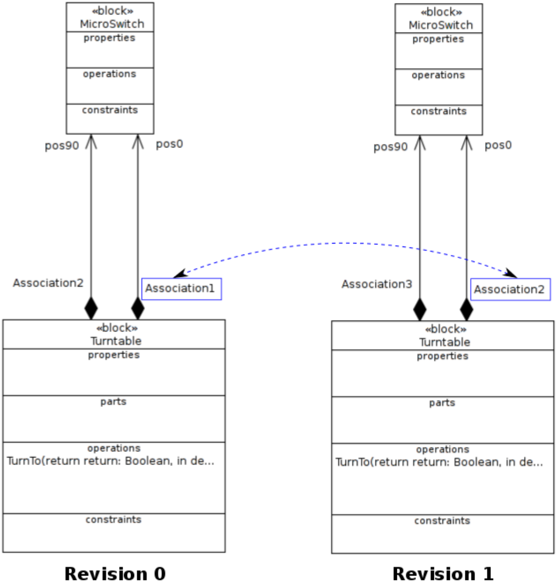
Revision 0 **Revision 1**

Wrong UUIDs between revisions

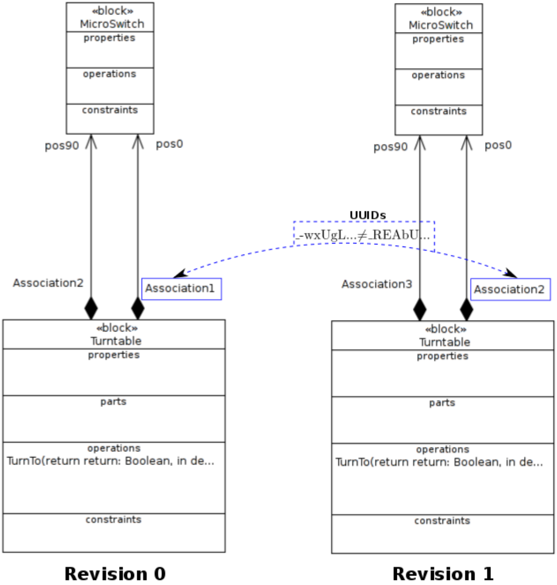


Revision 0 **Revision 1**

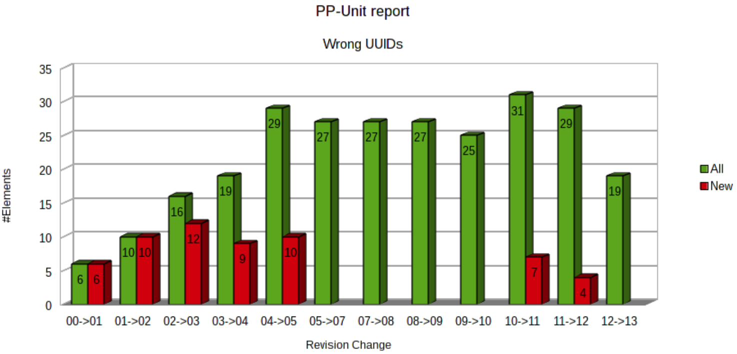
Wrong UUIDs between revisions



Wrong UUIDs between revisions

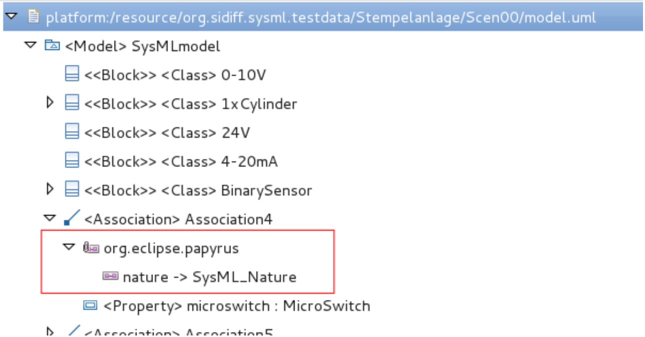


Wrong UUIDs summary



How detected? ▷ **Later!**

Usage of EAnnotations



```
platform:/resource/org.sidiff.sysml.testdata/Stempelanlage/Scen00/model.uml
  <Model> SysMLmodel
    <<Block>> <Class> 0-10V
    <<Block>> <Class> 1xCylinder
    <<Block>> <Class> 24V
    <<Block>> <Class> 4-20mA
    <<Block>> <Class> BinarySensor
    <Association> Association4
      org.eclipse.papyrus
        nature -> SysML_Nature
      <Property> microswitch : MicroSwitch
    <Association> Association5
```

Figure : Papyrus-specific EAnnotations

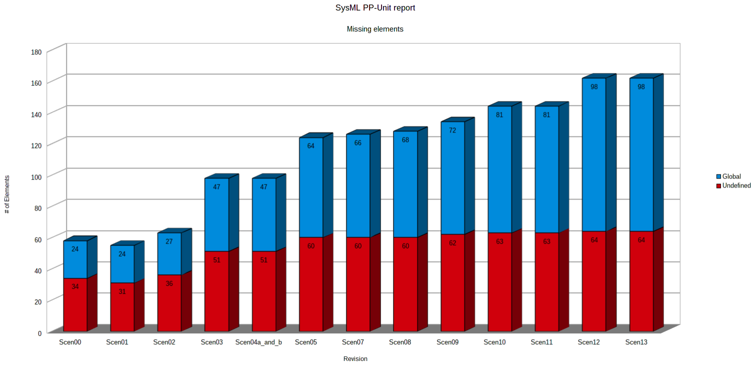
„Missing“ Elements

Global elements

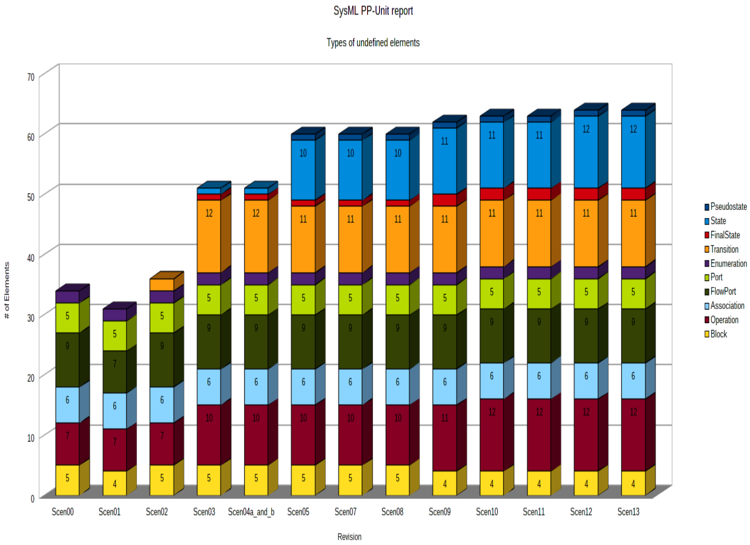
- Assumption: Are declared globally
- No need for declaration in any diagram

Undefined elements

- Assumption: Are declared locally
- Have to be declared in at least one diagram



Undefined Element Types



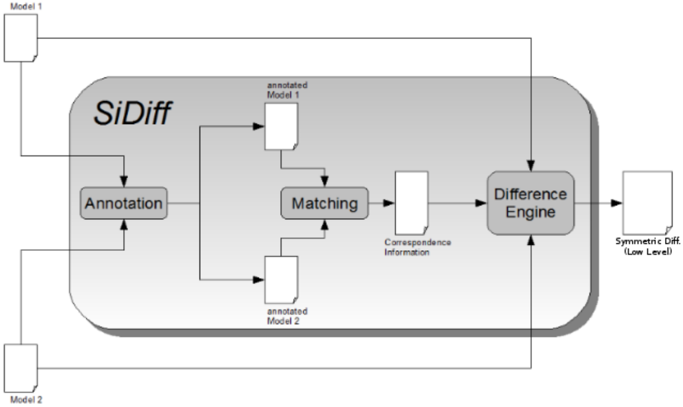
Introduction

- **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 transformating 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


Model 1
SiDiff
annotated Model 1
Annotation
Matching
Difference Engine
Correspondence Information
Symmetric Diff. (Low Level)
annotated Model 2
Model 2

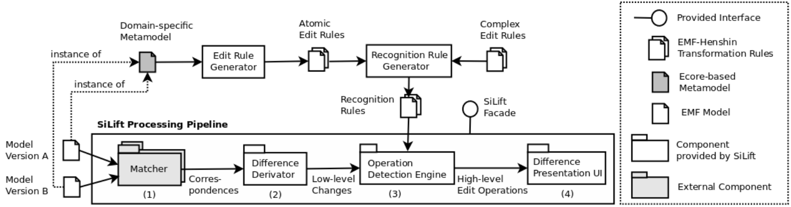
Introduction

lift me up

SiLift

- 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


Domain-specific Metamodel
instance of
instance of
Edit Rule Generator
Atomic Edit Rules
Recognition Rule Generator
Complex Edit Rules
Recognition Rules
SiLift Facade
SiLift Processing Pipeline
Model Version A
Model Version B
Matcher
(1)
Corres-pondences
Difference Derivator
(2)
Low-level Changes
Operation Detection Engine
(3)
High-level Edit Operations
Difference Presentation UI
(4)
Provided Interface
EMF-Henshin Transformation Rules
Ecore-based Metamodel
EMF Model
Component provided by SiLift
External Component

General aspects

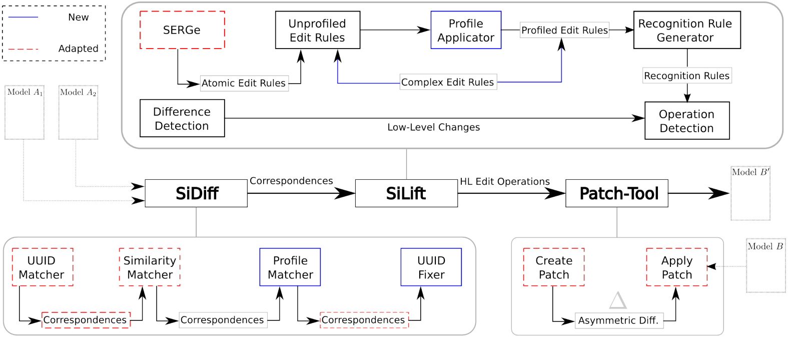
For full UML Profiles integration **every** pipeline step must be taken into consideration:

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



New

Adapted

Model A_1

Model A_2

SERGe

Unprofiled Edit Rules

Profile Applicator

Profiled Edit Rules

Recognition Rule Generator

Atomic Edit Rules

Complex Edit Rules

Recognition Rules

Difference Detection

Low-Level Changes

Operation Detection

SiDiff

Correspondences

SiLift

HL Edit Operations

Patch-Tool

Model B'

UUID Matcher

Similarity Matcher

Profile Matcher

UUID Fixer

Correspondences

Correspondences

Correspondences

Create Patch

Apply Patch

Model B

Asymmetric Diff.

Matching approaches

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

SiDiff Profile configuration

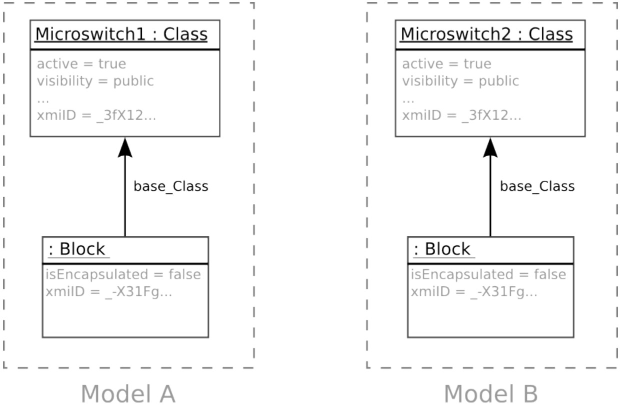
+ No new matching service has to be created
– Profiled elements do not own much „semantic“ meaning useful for similarity-based matching
– Each UML Profile needs its own configuration

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:



Microswitch1 : Class

active = true
visibility = public
...
xmiID = _3fX12...

base_Class

: Block

isEncapsulated = false
xmiID = _-X31Fg...

Model A

Microswitch2 : Class

active = true
visibility = public
...
xmiID = _3fX12...

base_Class

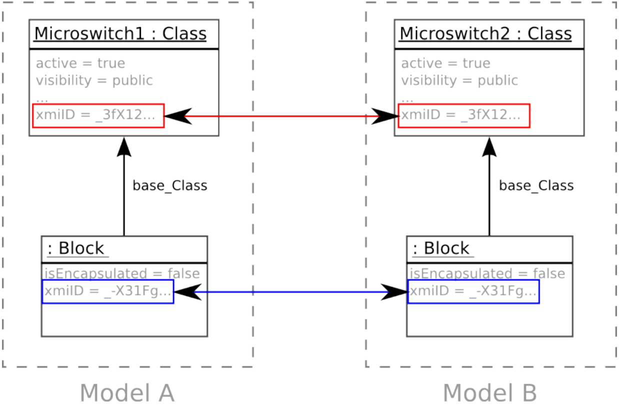
: Block

isEncapsulated = false
xmiID = _-X31Fg...

Model B

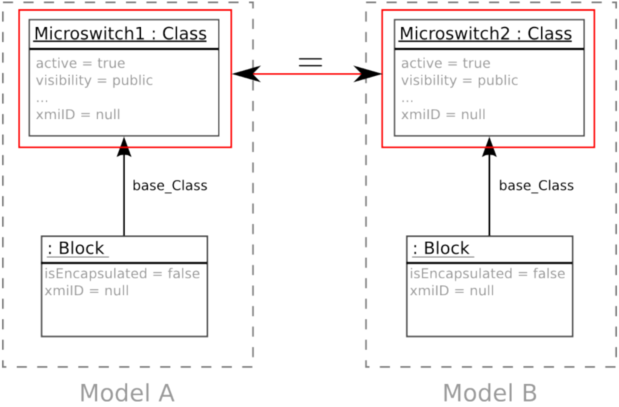
Profiles matching example

UUID-Matcher: Generic matching, no difference:



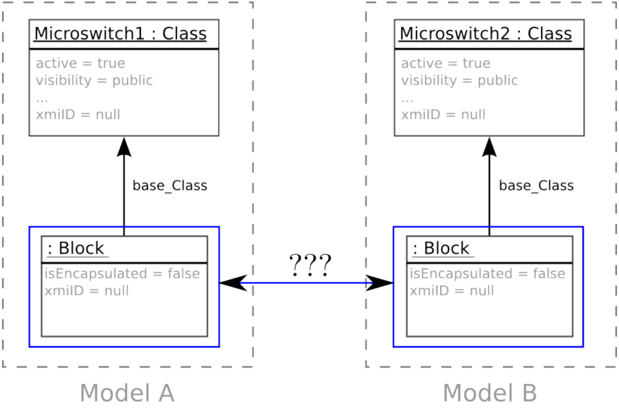
Profiles matching example

Similarity-Matcher: Matching base elements:



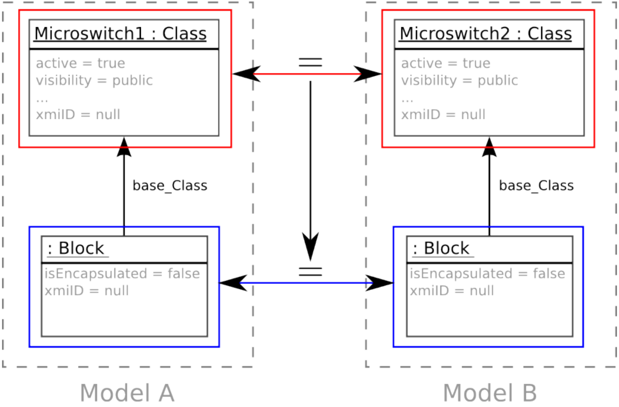
Profiles matching example

Similarity-Matcher: How to match profiled elements?

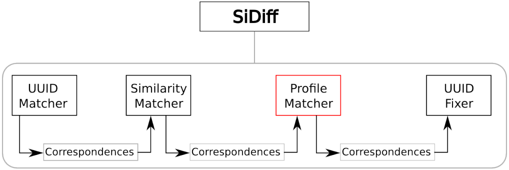


ProfileMatcher approach

Profile-Matcher: Match according to base elements:



ProfileMatcher


SiDiff

UUID Matcher

Similarity Matcher

Profile Matcher

UUID Fixer

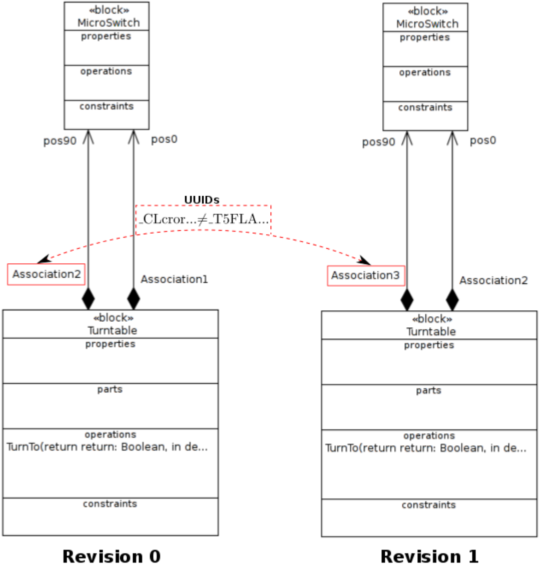
Correspondences

Correspondences

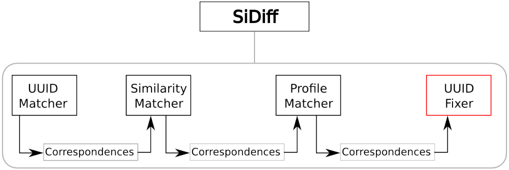
Correspondences

- 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

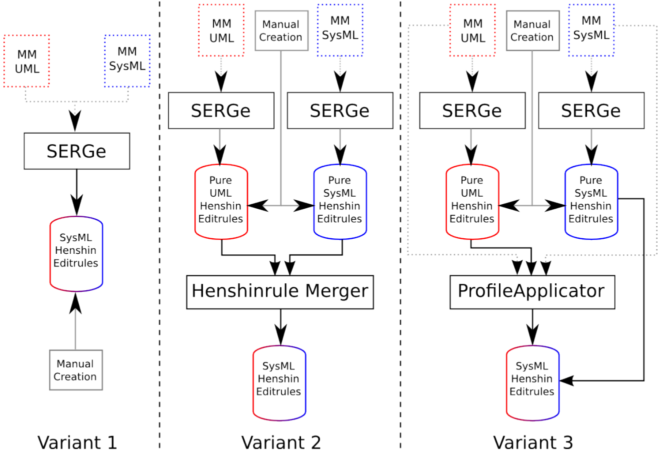


UUID-Fixer

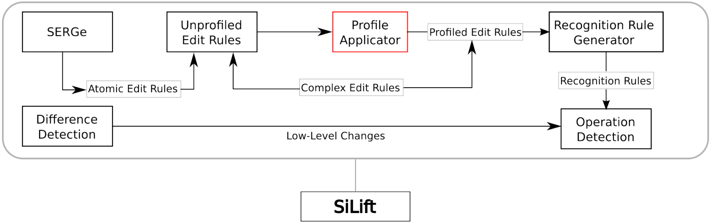


- Implemented as a **SiDiff-Service**
- Can be **integrated** easily
- Used as **final** SiDiff-service
- SiDiff has only to be run once, UUID-Matcher is **sufficient** afterwards
- UUID-Fixing makes models **compatible** to other tools depending on right UUIDs

Edit Rule Profile integration approaches



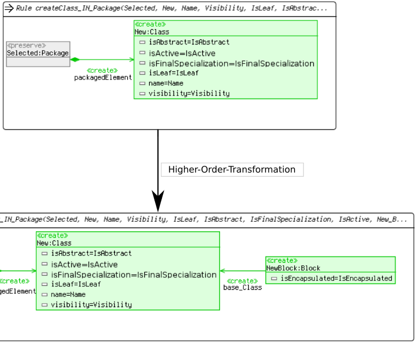
ProfileApplicator Overview



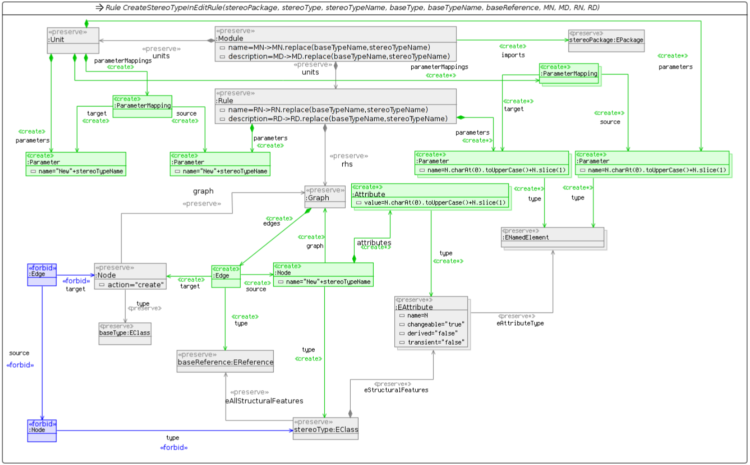
- Implemented as standalone **OSGi-Application**
- **Transforms** all files in given folder according to defined configuration via Higher-Order-Transformations
- **Minimal** configuration needed, more configuration options (Whitelist,. . .) available
- Configuration and execution oriented after **SERGe**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

Higher-Order-Transformations

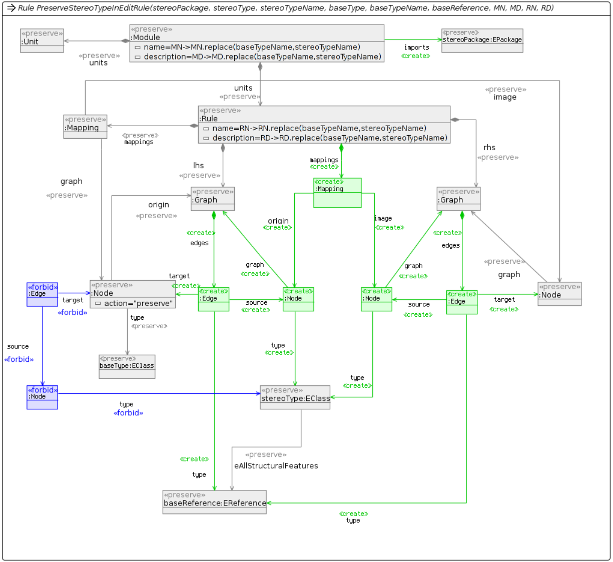
- Transforming Henshin Rules with Henshin Rules is called **Higher-Order-Transformation(HOT)**
- The ProfileApplicator is based on this feature of Henshin
- Henshin rules are defined as usual, but the elements to transform are elements of Henshin rules **itself**:


Rule createClass_IN_Package(Selected, New, Name, Visibility, IsLeaf, IsAbstrac...
«preserve»
Selected:Package
«create»
packagedElement
«create»
New:Class
isAbstract=IsAbstract
isActive=IsActive
isFinalSpecialization=IsFinalSpecialization
isLeaf=IsLeaf
name=Name
visibility=Visibility
Higher-Order-Transformation
Rule createBlock_IN_Package(Selected, New, Name, Visibility, IsLeaf, IsAbstract, IsFinalSpecialization, IsActive, New_B...
«preserve»
Selected:Package
«create»
packagedElement
«create»
New:Class
isAbstract=IsAbstract
isActive=IsActive
isFinalSpecialization=IsFinalSpecialization
isLeaf=IsLeaf
name=Name
visibility=Visibility
«create»
base_Class
«create»
NewBlock:Block
isEncapsulated=IsEncapsulated

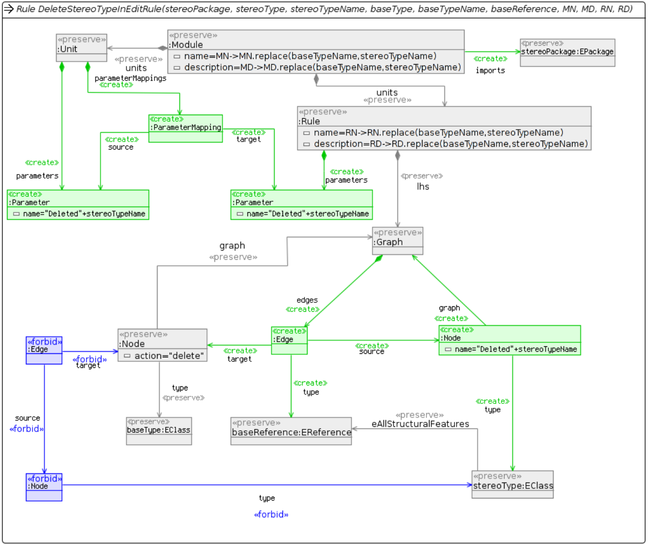
HOT for Create-Nodes


Rule CreateStereoTypeInEditRule(stereoPackage, stereoType, stereoTypeName, baseType, baseTypeName, baseReference, MN, MD, RN, RD)
«preserve» :Unit
«preserve» :Module
name=MN->MN.replace(baseTypeName,stereoTypeName)
description=MD->MD.replace(baseTypeName,stereoTypeName)
«preserve» units
«preserve» units
«preserve» stereoPackage:EPackage
«create» imports
«create*» parameters
parameterMappings «create»
parameterMappings «create*»
«create*» :ParameterMapping
«create» :ParameterMapping
target «create»
source «create»
«preserve» :Rule
name=RN->RN.replace(baseTypeName,stereoTypeName)
description=RD->RD.replace(baseTypeName,stereoTypeName)
«create*» target
«create*» source
«create» parameters
parameters «create»
parameters «create*»
«create» :Parameter
name="New"+stereoTypeName
«create» :Parameter
name="New"+stereoTypeName
«preserve» rhs
«create*» :Parameter
name=N.charAt(0).toUpperCase()+N.slice(1)
«create*» :Parameter
name=N.charAt(0).toUpperCase()+N.slice(1)
graph «preserve»
«preserve» :Graph
«create*» :Attribute
value=N.charAt(0).toUpperCase()+N.slice(1)
«create*» type
«create*» type
«preserve*» :ENamedElement
«create» edges
«create» graph
attributes «create*»
type «create*»
«forbid» :Edge
«forbid» target
«preserve» :Node
action="create"
«create» target
«create» :Edge
«create» source
«create» :Node
name="New"+stereoTypeName
type «preserve»
«preserve» baseType:EClass
«create» type
«preserve*» :EAttribute
name=N
changeable="true"
derived="false"
transient="false"
«preserve*» eAttributeType
source «forbid»
«preserve» baseReference:EReference
type «create»
«preserve» eAllStructuralFeatures
«preserve*» eStructuralFeatures
«forbid» :Node
«preserve» stereoType:EClass
type «forbid»

HOT for Preserve-Nodes



HOT for Delete-Nodes



Rule DeleteStereoTypeInEditRule(stereoPackage, stereoType, stereoTypeName, baseType, baseTypeName, baseReference, MN, MD, RN, RD)

«preserve» :Unit

«preserve» :Module
name=MN->MN.replace(baseTypeName,stereoTypeName)
description=MD->MD.replace(baseTypeName,stereoTypeName)

«preserve» stereoPackage:EPackage

«create» imports

«preserve» units

parameterMappings «create»

units «preserve»

«create» :ParameterMapping

«create» source

«create» target

«preserve» :Rule
name=RN->RN.replace(baseTypeName,stereoTypeName)
description=RD->RD.replace(baseTypeName,stereoTypeName)

«create» parameters

«create» :Parameter
name="Deleted"+stereoTypeName

«create» parameters

«create» :Parameter
name="Deleted"+stereoTypeName

«preserve» lhs

«preserve» :Graph

graph «preserve»

edges «create»

graph «create»

«forbid» :Edge

«forbid» target

«preserve» :Node
action="delete"

«create» target

«create» :Edge

«create» source

«create» :Node
name="Deleted"+stereoTypeName

type «preserve»

«create» type

«create» type

source «forbid»

«preserve» baseType:EClass

«preserve» baseReference:EReference

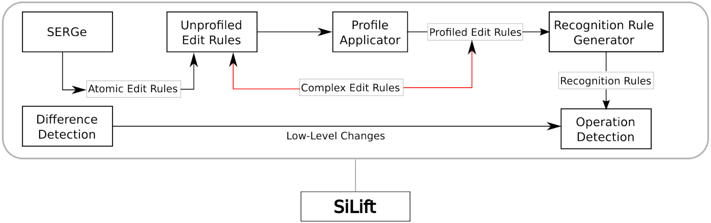
«preserve» eAllStructuralFeatures

«forbid» :Node

type «forbid»

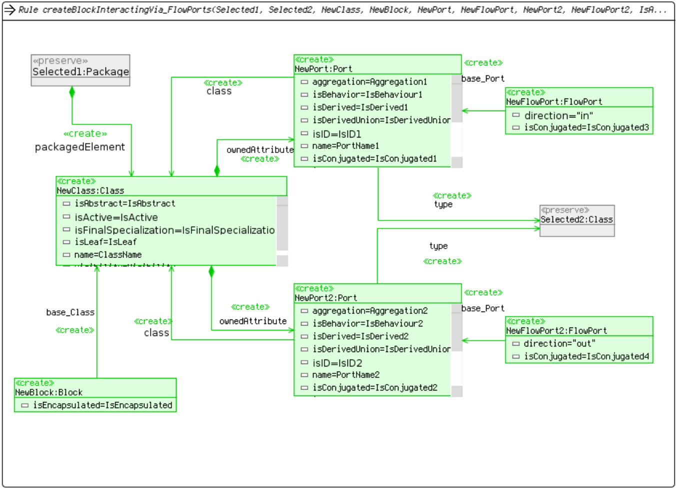
«preserve» stereoType:EClass

Complex edit rules

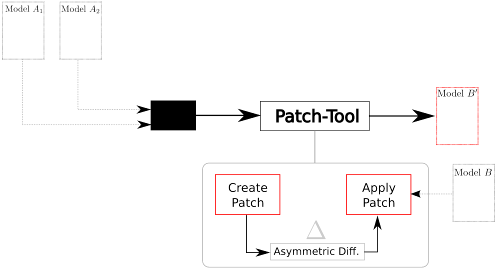


- For better lifting results the **domain** engineer can define complex edit rules:
  - A complex edit rule **consists** of 2 or more atomic edit rules
  - Defines a **common** edit operation
  - Are **optional** on the contrary to atomic edit rules
  - Require very good **domain-specific** knowledge

Complex edit rule example

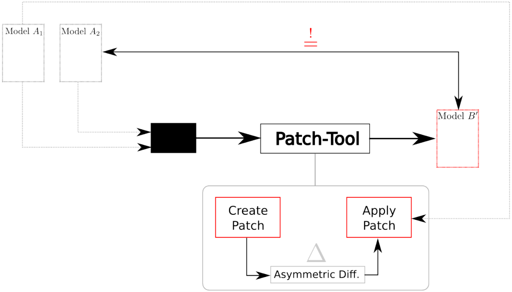


Patching integration



- 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

Patching integration



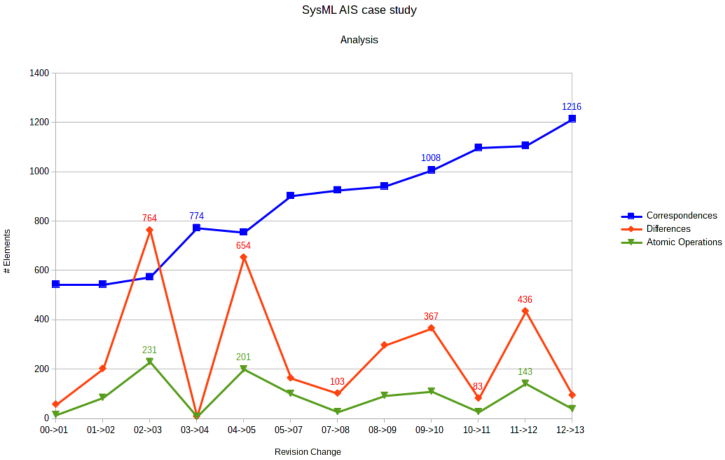
- Model B' needs to contain all **changes** between A_1 and A_2 for full **correctness**:

$$\text{If } B = A_1 \text{ then } B' \overset{!}{=} A_2$$

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

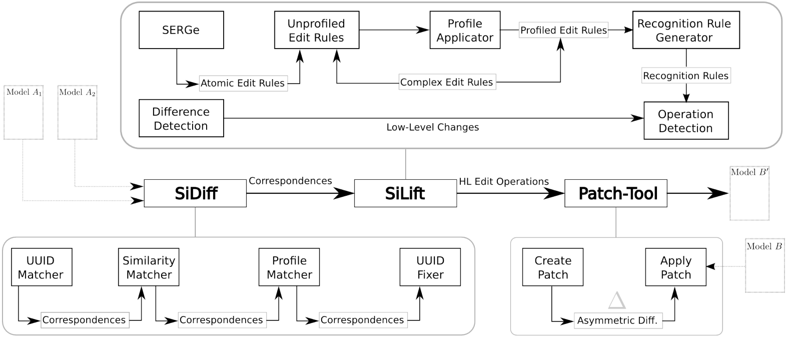


Results for SysML case study

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

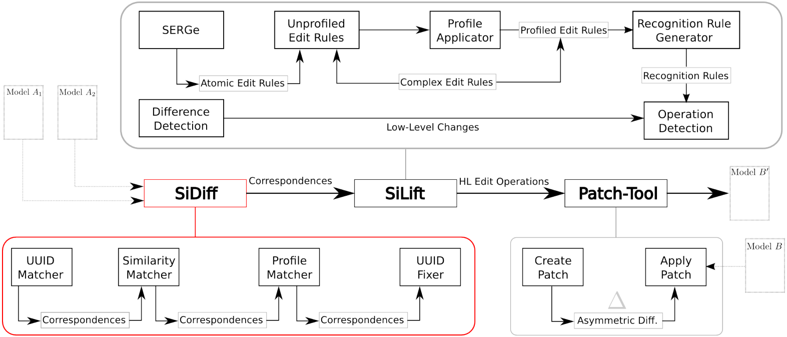
Summary

UML Profiles integration into every pipeline step:



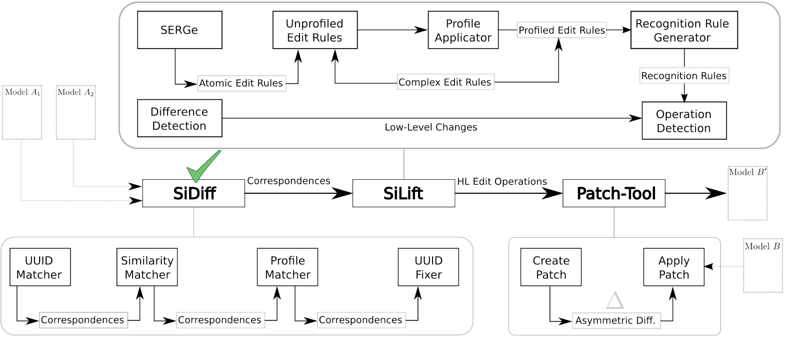
Summary

UML Profiles integration into every pipeline step:



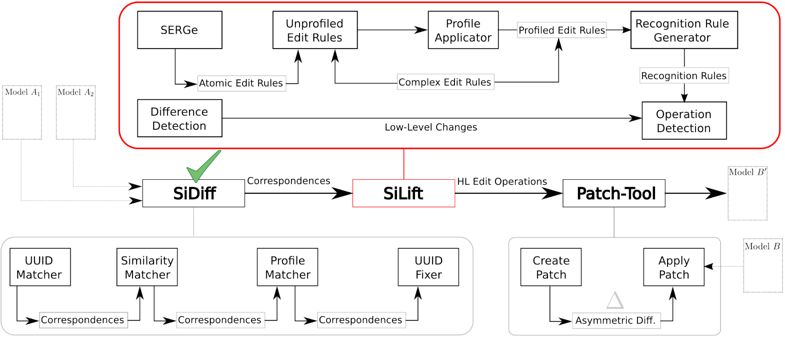
Summary

UML Profiles integration into every pipeline step:



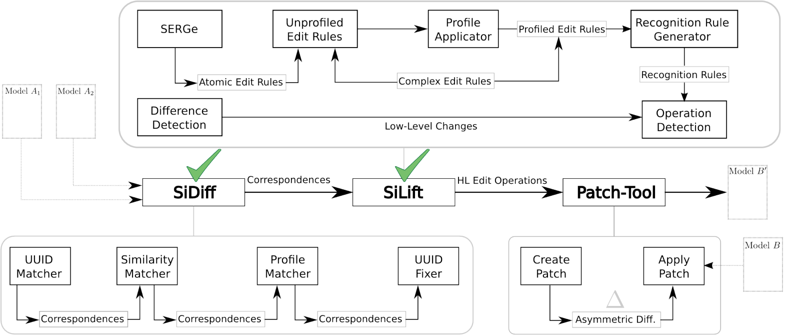
Summary

UML Profiles integration into every pipeline step:



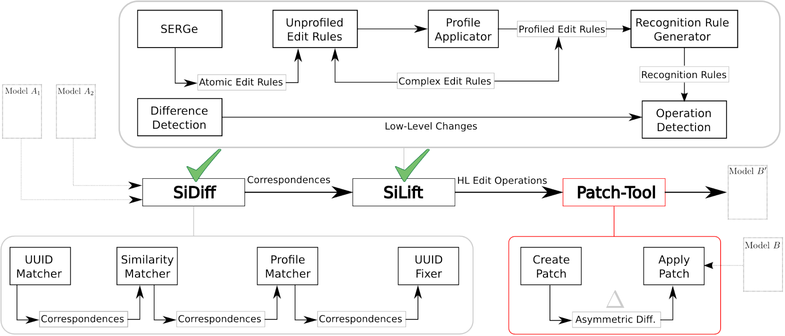
Summary

UML Profiles integration into every pipeline step:



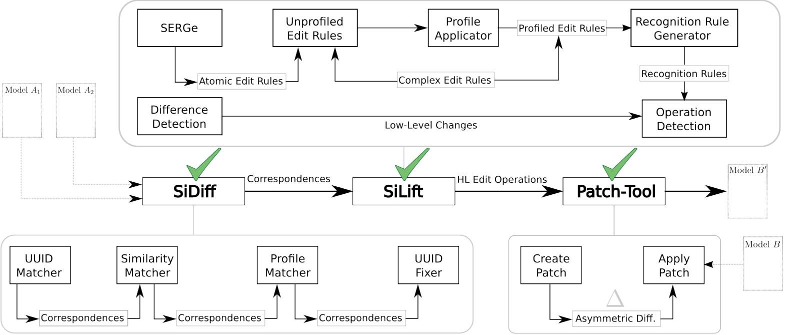
Summary

UML Profiles integration into every pipeline step:



Summary

UML Profiles integration into every pipeline step:



Outlook

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

Bibliography

C. Legat, J. Folmer, and B. Vogel-Heuser.
Evolution in industrial plant automation: A case study.
In *Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society*, IECON '13, 2013.

Object Management Group.
Systems Modeling Language Specification.
http://www.omg.org/spec/SysML/1.3/, April 2012.

Object Management Group.
Systems Modeling Language.
http://www.omgsysml.org/, September 2013.

UML Diagrams.
UML Profile Diagrams.
http://www.uml-diagrams.org/profile-diagrams.html, September 2013.