

# Model Consistency ensured by Metamodel Integration MoConseMI

6th GEMOC 2018, Copenhagen

Johannes Meier    Andreas Winter

Software Engineering Group  
Department of Computing Science  
Carl von Ossietzky University, Oldenburg, Germany

15. October 2018

# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

# Motivation

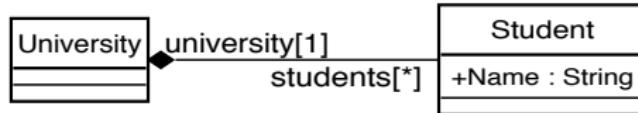
- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.

# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

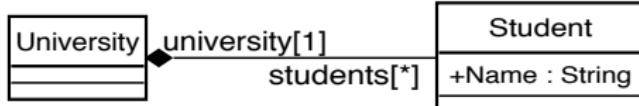
ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.



# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.



The screenshot displays an IDE interface with several windows:

- Package Explorer:** Shows the project structure with packages like 'ASGFramework', 'commons-lang3', 'commons-math4', 'guava', 'guava-gwt', 'guava-testlib', 'guava-tests', 'JavaASG', 'JavaTestProject', 'ProviderJDT', and 'TestProject'. The 'src' folder under 'TestProject' contains three files: 'Lecture.java', 'Student.java', and 'University.java'.
- Lecture.java:** Contains the following code:
 

```

package university;
import java.util.List;
public class Student {
    private String name;
    private List<Lecture> lectures;
    public void register(Lecture lecture) {
        lectures.add(lecture);
    }
}
      
```
- Student.java:** Contains the following code:
 

```

package university;
public class Lecture {
    ...
}
      
```
- University.java:** Contains the following code:
 

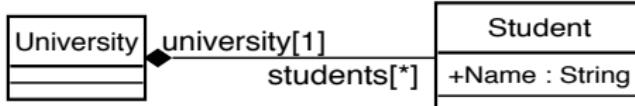
```

package university;
public class Student {
    ...
}
      
```

# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.



The screenshot displays an IDE interface. On the left, the 'Package Explorer' shows a project structure with several Java and utility libraries. In the center, the 'Lecture.java' file is open, displaying the following Java code:

```

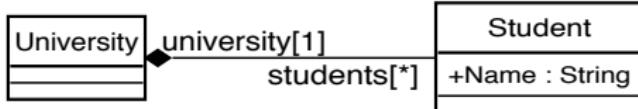
package university;
import java.util.List;
public class Student {
    private String name;
    private List<Lecture> lectures;
    public void register(Lecture lecture) {
        lectures.add(lecture);
    }
}
  
```

The code defines a 'Student' class with fields for 'name' and 'lectures' (a list of 'Lecture' objects). A method 'register' adds a new 'lecture' to the list. The 'University' class from the previous diagram is also visible in the code, indicating its use in the application.

# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.



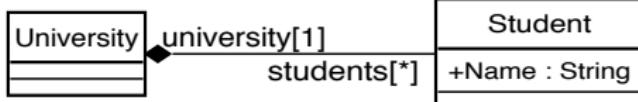
The screenshot displays an IDE interface. On the left, the 'Package Explorer' shows a project structure with packages like 'ASGFramework', 'commons-lang3', 'commons-math4', 'guava', 'guava-gwt', 'guava-testlib', 'guava-tests', 'JavaASG', 'JavaTestProject', 'ProviderJDT', and 'TestProject'. Inside 'TestProject', there is a 'src' folder containing three files: 'Lecture.java', 'Student.java', and 'University.java'. On the right, the 'Editor' window shows the content of 'Lecture.java':

```
1 package university;
2
3 import java.util.List;
4
5 public class Student {
6     private String name;
7     private List<Lecture> lectures;
8
9     public void register(Lecture lecture) {
10         lectures.add(lecture);
11     }
12 }
13
```

# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.



The screenshot shows an IDE interface with the 'Package Explorer' view on the left and the 'Editor' view on the right.

**Package Explorer:**

- ASGFramework
- commons-lang3
- commons-math4
- guava
- guava-gwt
- guava-testlib
- guava-tests
- JavaASG
- JavaTestProject
- ProviderJDT
- TestProject
- src
  - university
    - Lecture.java
    - Student.java
    - University.java

**Editor:**

```
1 package university;
2
3 import java.util.List;
4
5 public class Student {
6     private String name;
7     private List<Lecture> lectures;
8
9     public void register(Lecture lecture) {
10         lectures.add(lecture);
11     }
12 }
```

The code in the editor shows a 'Student' class with a private attribute 'lectures' of type 'List<Lecture>' and a public method 'register(Lecture lecture)' that adds the lecture to the list. A circled red mark highlights the 'lectures' attribute in the code.

# Motivation

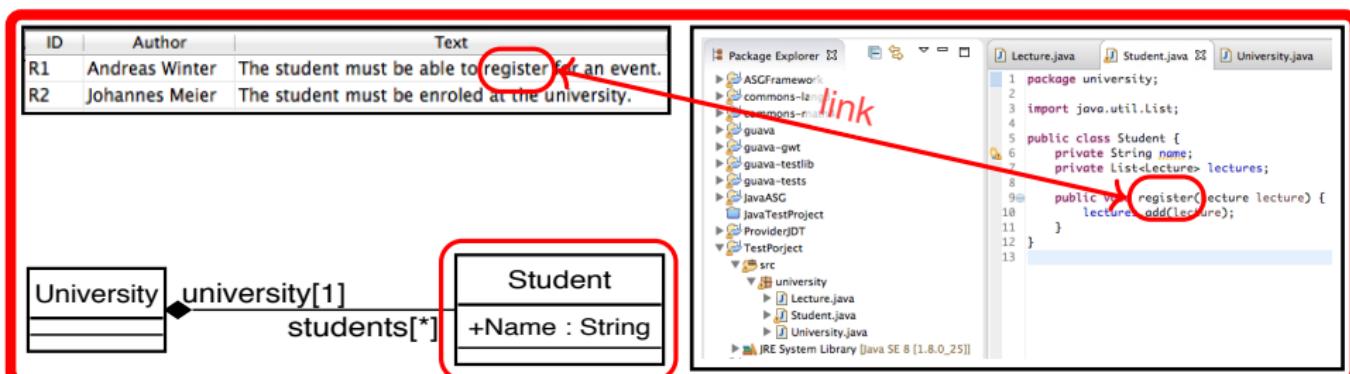
- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

The diagram illustrates the integration of various software development artifacts:

- Table:** A screenshot of a database table showing two rows of requirements (R1, R2) with their descriptions.
- EER Diagram:** An Entity-Relationship (EER) diagram showing entities: University and Student. The University entity has one attribute: +Name : String. There is a relationship named "students[\*]" from University to Student.
- Java Code:** A screenshot of the Eclipse IDE's Package Explorer and code editor. The code shows a class `Lecture.java` with a method `register(Lecture lecture)` that adds the lecture to a list of lectures.
- Red Circles:** Red circles highlight specific parts of the table and code, connected by a red arrow, indicating a consistency check or link between the requirements and the implemented functionality.

# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically



# Motivation

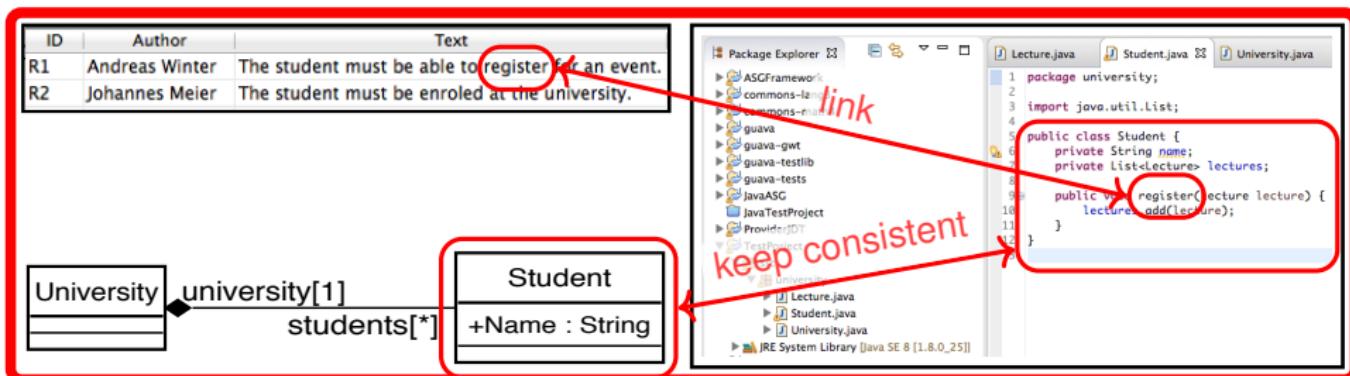
- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically

The diagram illustrates the integration of various software artifacts to ensure consistency:

- Text Table:** A table showing requirements (R1, R2) and their descriptions.
- UML Class Diagram:** A class diagram showing a relationship between **University** and **Student**. The **University** class has an association named **university[1]** pointing to the **Student** class, which has a multiplicity **students[\*]**.
- Java Code:** A screenshot of the Eclipse IDE's Package Explorer and Editor view. The code in **Lecture.java** contains a method **register(Lecture lecture)** that adds the lecture to a list of **lectures** in the **Student** class.
- Annotations:** Red annotations highlight the word **link** in the text table, the multiplicity **[\*]** in the UML diagram, and the code in **Lecture.java**, indicating the connection between these different representations.

# Motivation

- various Artifacts in Software Development:
  - ▶ Diagrams, DSLs, Tools, ...
  - ▶ Artifacts are technically separated
  - ▶ Artifacts are interrelated regarding content
- Ensure Consistency between Artifacts automatically



## Problem

There are further Software Development Projects:

- e.g. with formal Specifications, C++, Test Cases, Documentation, Project Management, Build Tools
- Traceability
- → further Consistency issues

General Problem:

- Artifacts are technically separated, but interrelated contentwise
- specific Consistency Rules have to be fulfilled automatically

## Goal

Ensure Consistency between Artifacts automatically!

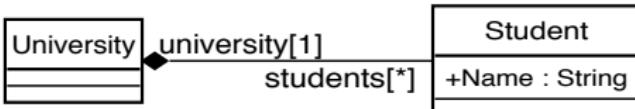
- Artifact == Model + Metamodel (structural formalization [CNS12])
- → Model Integration

# Challenges

# Challenges

## 1. Formalize Consistency Rules

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.



The screenshot displays the Eclipse IDE interface. The left pane, 'Package Explorer', shows the project structure with several Java and third-party library folders. The right pane, 'Editor', displays the Java code for the 'Student' class:

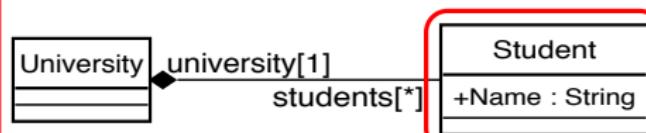
```

package university;
import java.util.List;
public class Student {
    private String name;
    private List<Lecture> lectures;
    public void register(Lecture lecture) {
        lectures.add(lecture);
    }
}
  
```

# Challenges

## 1. Formalize Consistency Rules

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an event.
R2	Johannes Meier	The student must be enroled at the university.



Sourcecode and Class Diagrams describe the same set of classes, identified by their class name.

```

package university;
import java.util.List;

public class Student {
    private String name;
    private List<Lecture> lectures;

    public void register(Lecture lecture) {
        lectures.add(lecture);
    }
}
  
```

The screenshot shows the Eclipse IDE's Package Explorer and Navigator views. The Package Explorer shows packages like ASGFramework, commons-lang3, commons-math4, guava, guava-gwt, guava-testlib, and guava-tests. The Navigator view shows three files: Lecture.java, Student.java, and University.java. A callout points from the text 'Sourcecode and Class Diagrams describe the same set of classes, identified by their class name.' to the Student.java file in the Navigator.

# Challenges

## 1. Formalize Consistency Rules

ID	Author	Text
R1	Andreas Winter	The student must be able to register for an eve
R2	Johannes Meier	The student must be enroled at the university.



Each Requirement is linked with all Methods whose Name is contained in the Text of the Requirement.

Sourcecode and Class Diagrams describe the same set of classes, identified by their class name.

```

ic class Student {
    private String name;
    private List<Lecture> lectures;
    t java.util.List;
    public void register(Lecture lecture) {
        lectures.add(lecture);
    }
}

JRE System Library [Java SE 8 [1.8.0_25]]
  
```

# Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)

# Challenges

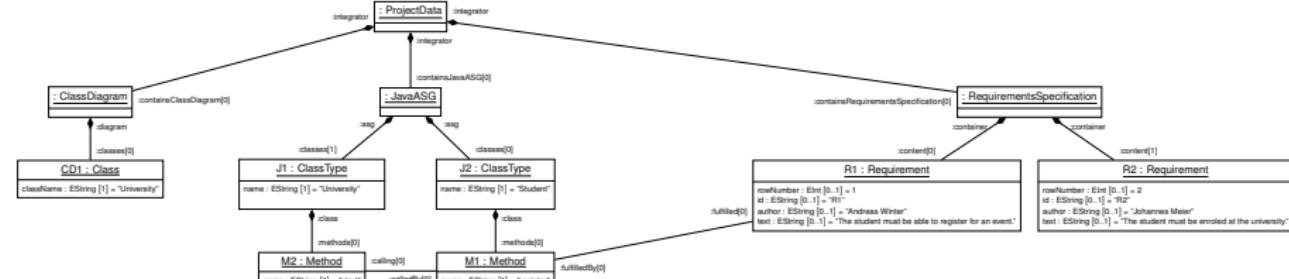
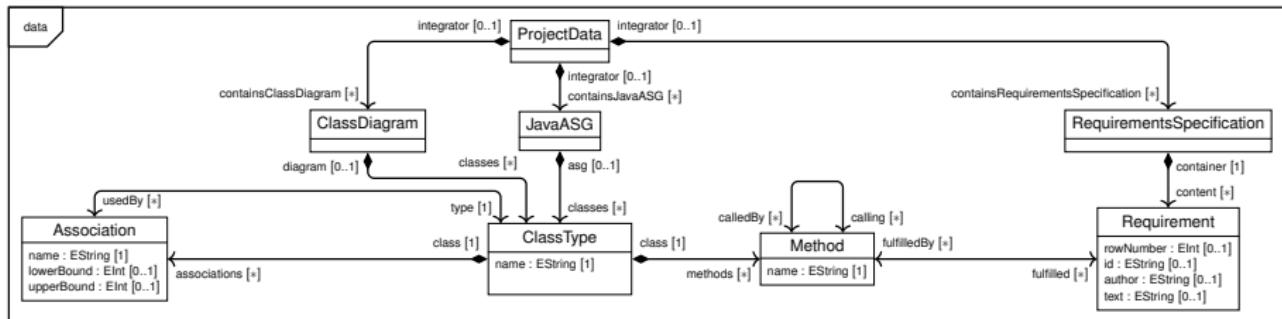
1. Formalize Consistency Rules
2. Create explicit SUM(M)

- reuse Model Techniques which work only with one Model
- used as single Point of Truth
- Single Underlying Model [ASB09]
- SUMM and SUM are explicit

# Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)

- reuse Model Techniques which work only with one Model
- used as single Point of Truth
- Single Underlying Model [ASB09]
- SUMM and SUM are explicit



# Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)

# Challenges

1. Formalize Consistency Rules
2. Create explicit  $SUM(M)$
3. Support initial (Meta)Models:

## Challenges

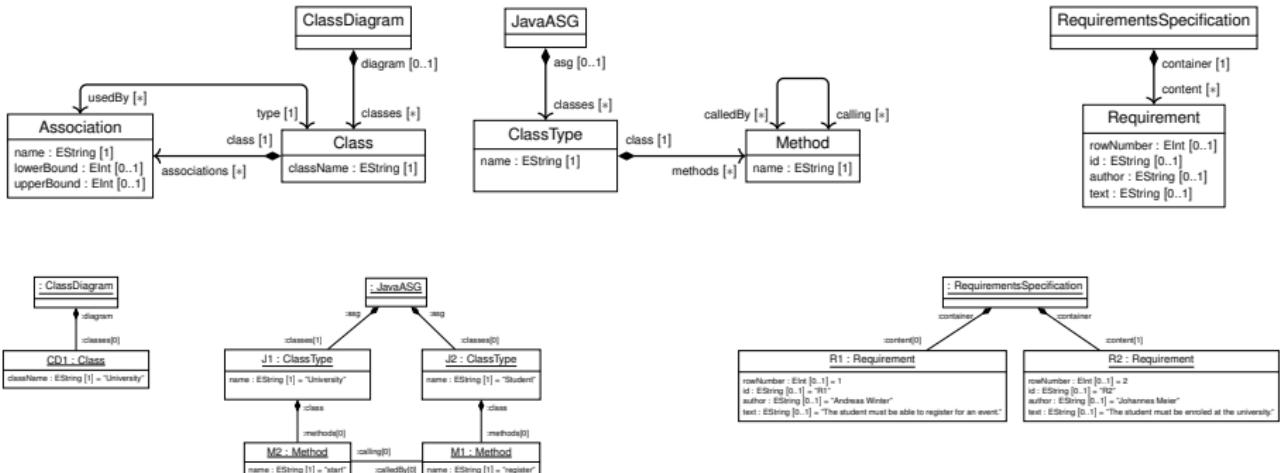
1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models

- existing Metamodels:  
DSLs, Environments, Tools, ...
- existing Models:  
ongoing projects, legacy data, ...

# Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models

- existing Metamodels:  
DSLs, Environments, Tools, ...
- existing Models:  
ongoing projects, legacy data, ...



## Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies

# Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models

# Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models
4. Ensure Model Consistency

## Related Work

	synthetic	projectional		
		OSM	Vitruvius	GEMOC
1. Formalize Consistency Rules	✓	✓	✓	✓
2. Create explicit SUM(M)	✗	✓	✗	✗
3a. Reuse initial Models	✓	✗	✓	✓
3b. Fix initial Inconsistencies	✓	–	✗	?
3c. Consistent initial Models	✓	–	✓	✓
4. Ensure Model Consistency	✓	✓	✓	✓

- ISO Standard 42010:2011 [IEE11]: synthetic vs. projectional
- synthetic: TGGs [SK08], QVT-R [RJV09], explicit correspondences [EEC<sup>+</sup>14]
- OSM: Single Underlying (Meta)Model (SUM(M)) [ASB09]
- Vitruvius [KBL13, BHK<sup>+</sup>14]
- GEMOC Approach [LDC18]

# Metamodel Integration

## Challenges

## Activities

1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models
4. Ensure Model Consistency

# Metamodel Integration

## Challenges

## Activities

1. Formalize Consistency Rules
  2. Create explicit  $SUM(M)$
  3. Support initial (Meta)Models:
    - a. Reuse initial Models
    - b. Fix initial Inconsistencies
    - c. Consistent initial Models
  4. Ensure Model Consistency
- SUMM

# Metamodel Integration

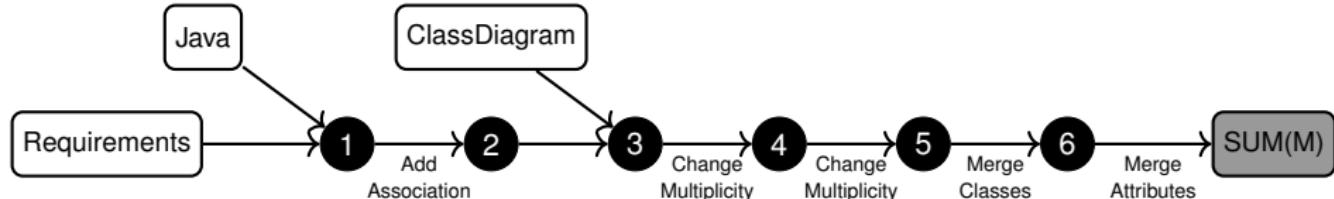
## Challenges

## Activities

1. Formalize Consistency Rules
2. Create explicit  $SUM(M)$
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models
4. Ensure Model Consistency

SUMM

1. Configuration of Operators



# Metamodel Integration

## Challenges

## Activities

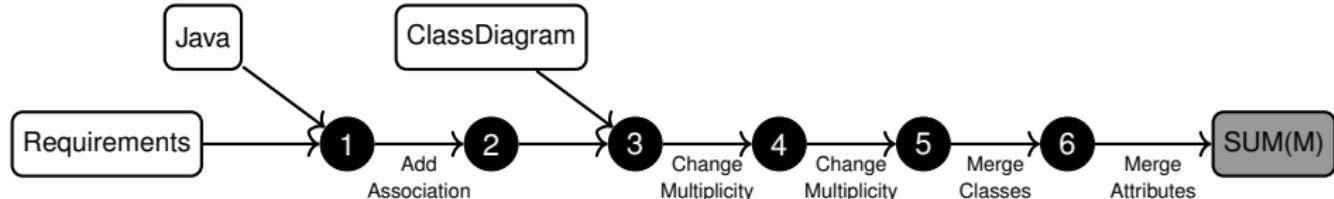
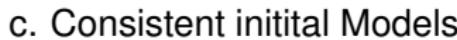
1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models
4. Ensure Model Consistency

SUMM

SUM

1. Configuration of Operators

2. Initialization of SUM



# Metamodel Integration

## Challenges

## Activities

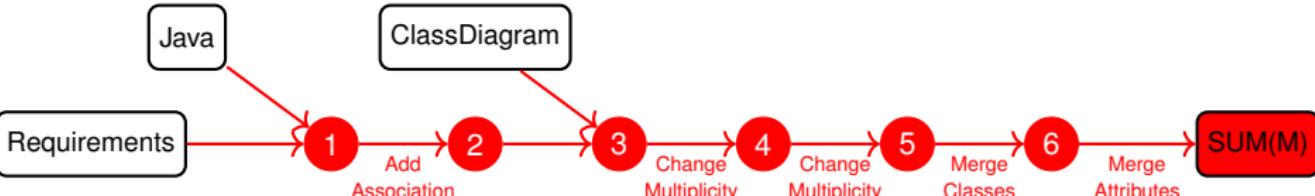
1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models
4. Ensure Model Consistency

SUMM

SUM

1. Configuration of Operators

2. Initialization of SUM



# Metamodel Integration

## Challenges

## Activities

1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models
4. Ensure Model Consistency

SUMM

SUM

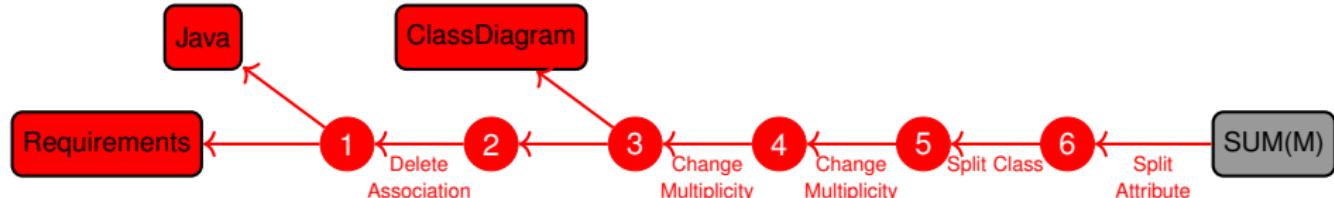
1. Configuration of Operators

2. Initialization of SUM

a. Reuse initial Models

b. Fix initial Inconsistencies

c. Consistent initial Models

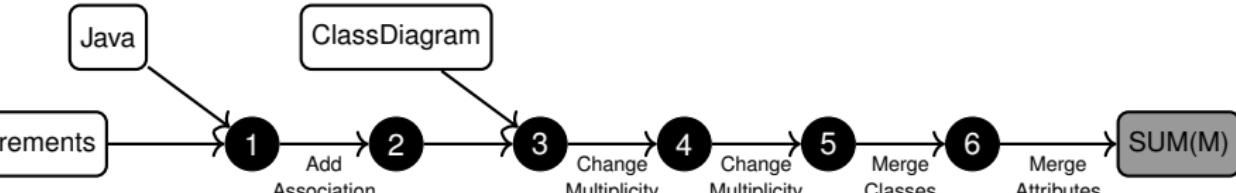


# Metamodel Integration

## Challenges

## Activities

1. Formalize Consistency Rules
  2. Create explicit SUM(M)
  3. Support initial (Meta)Models:
    - a. Reuse initial Models
    - b. Fix initial Inconsistencies
    - c. Consistent initial Models
  4. Ensure Model Consistency
- SUMM
- SUM
1. Configuration of Operators
  2. Initialization of SUM
  3. Consistency Assurance



# Metamodel Integration

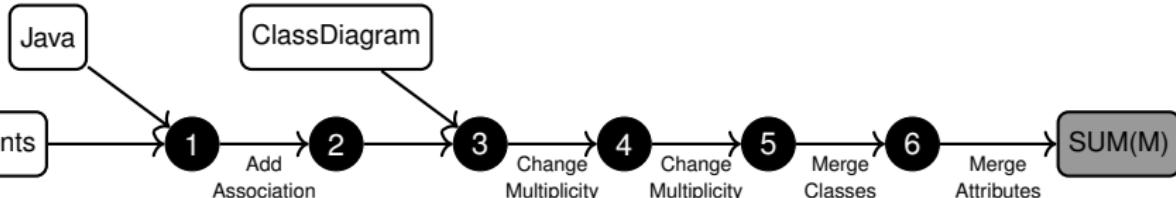
## Challenges

1. Formalize Consistency Rules
2. Create explicit SUM(M)
3. Support initial (Meta)Models:
  - a. Reuse initial Models
  - b. Fix initial Inconsistencies
  - c. Consistent initial Models
4. Ensure Model Consistency

## Activities

1. Configuration of Operators
2. Initialization of SUM
3. Consistency Assurance

1x Methodologist

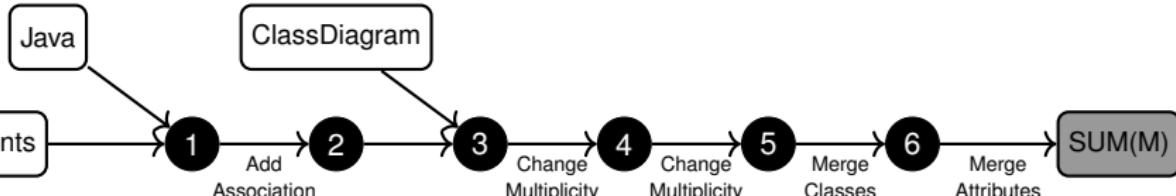


# Metamodel Integration

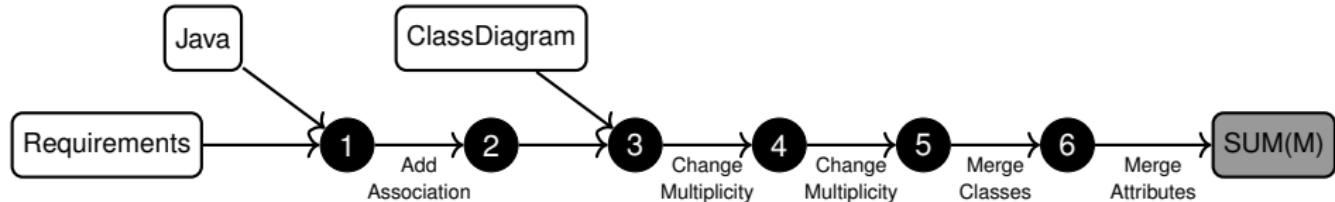
## Challenges

## Activities

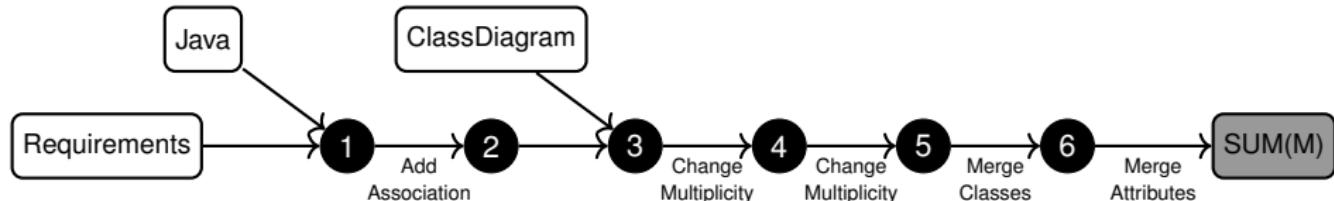
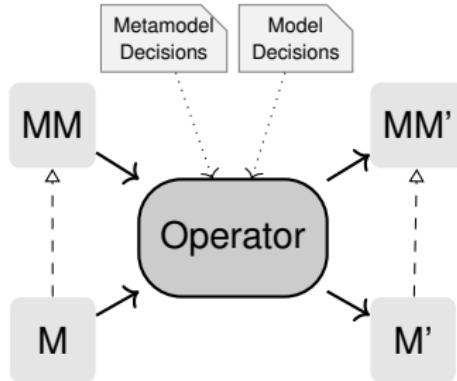
1. Formalize Consistency Rules
  2. Create explicit SUM(M)
  3. Support initial (Meta)Models:
    - a. Reuse initial Models
    - b. Fix initial Inconsistencies
    - c. Consistent initial Models
  4. Ensure Model Consistency
- SUMM
- nx User
1. Configuration of Operators
2. Initialization of SUM
3. Consistency Assurance



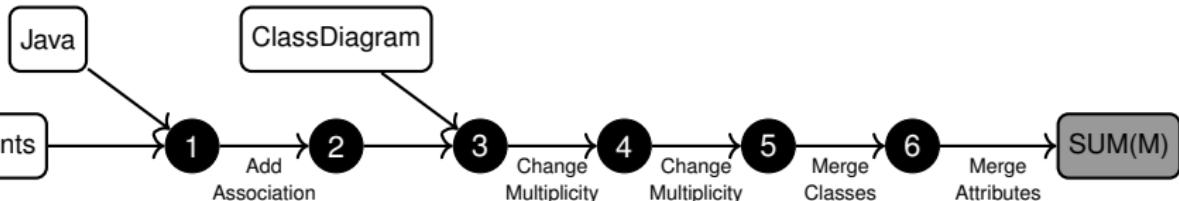
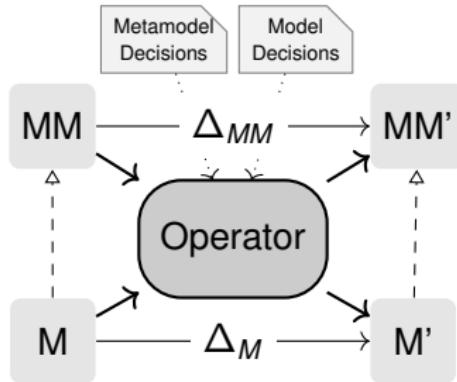
# 1. Configuration of Operators



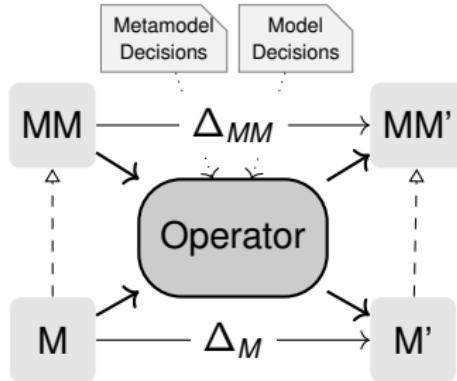
# 1. Configuration of Operators



# 1. Configuration of Operators

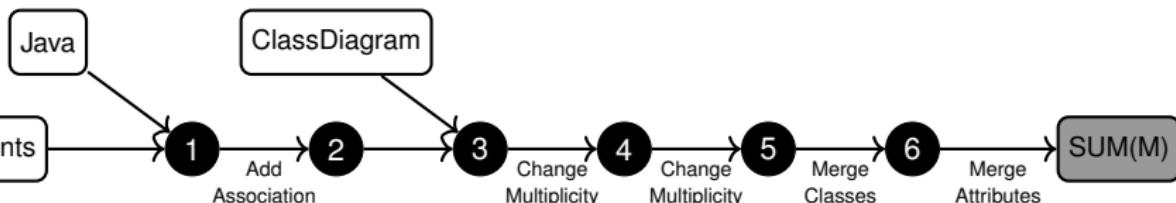


# 1. Configuration of Operators

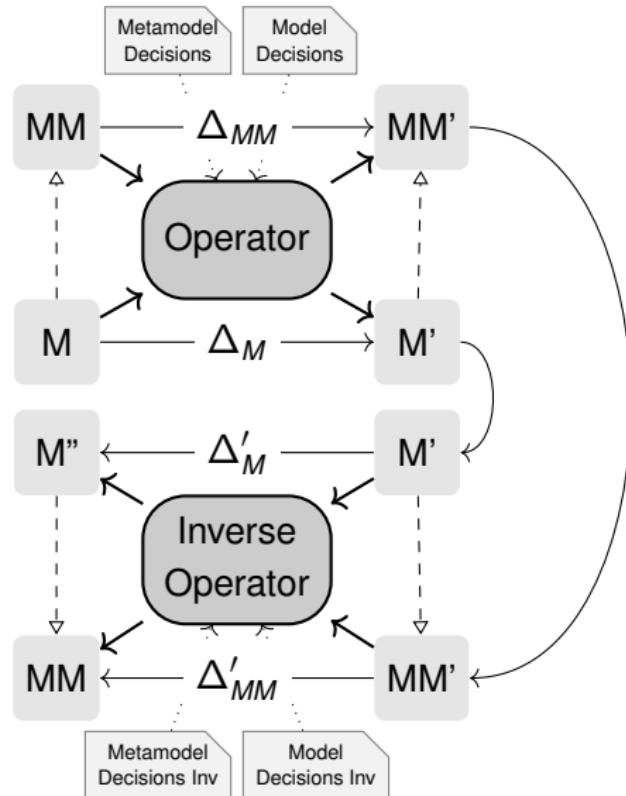


Chechik, Nejati, Sabetzadeh:  
**A Relationship-Based Approach to Model Integration (2012)**  
→ merge, composition, weaving

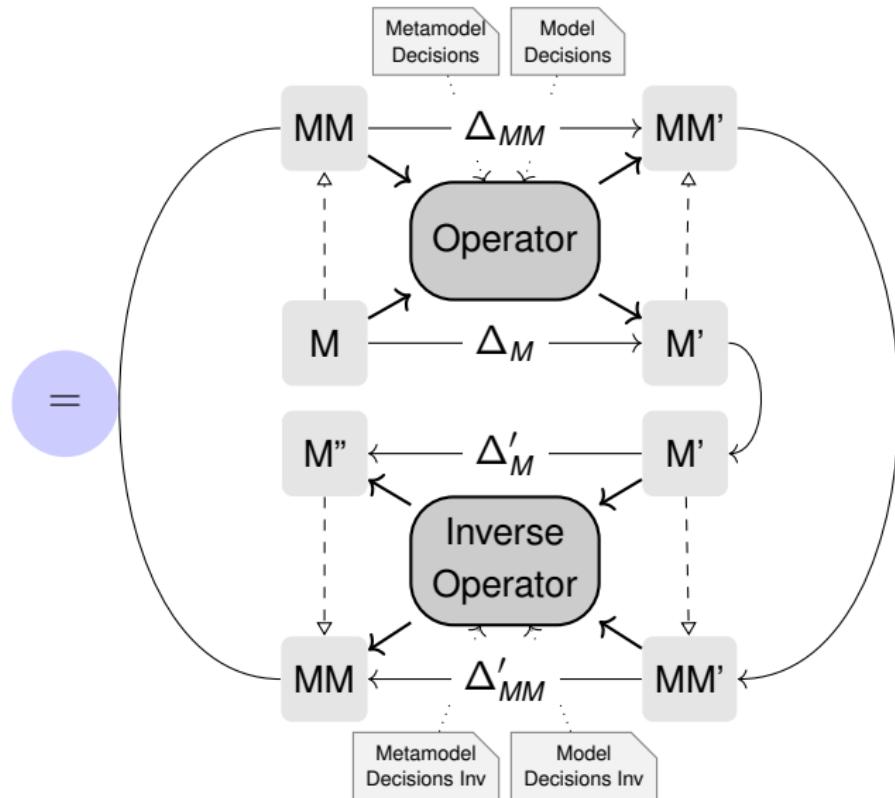
Herrmannsdoerfer et al.:  
**An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models (2011)**  
→ extended Coupled Operators



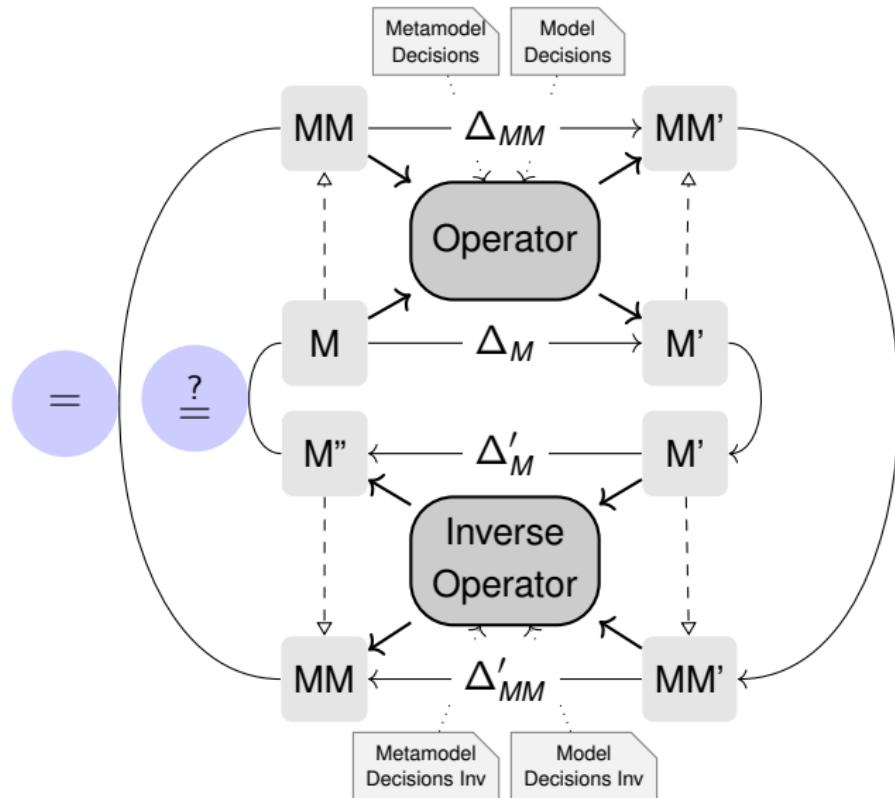
# 1. Configuration of Operators



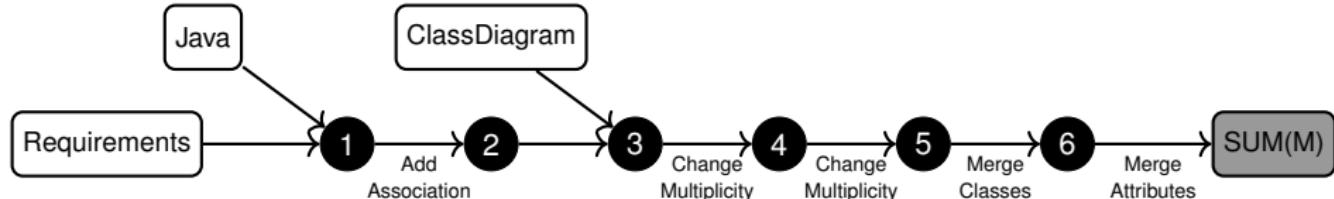
# 1. Configuration of Operators



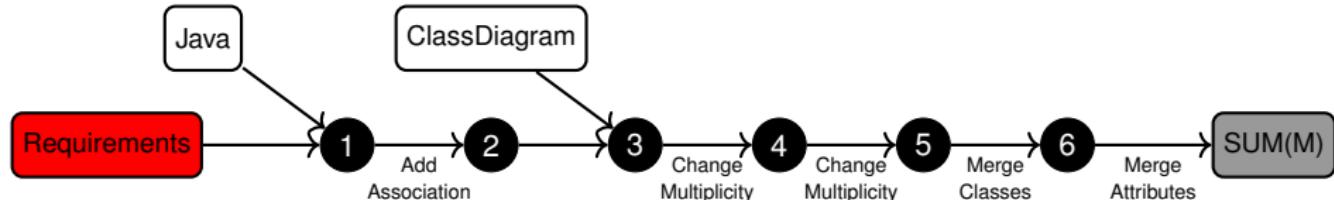
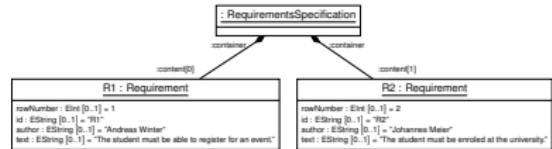
# 1. Configuration of Operators



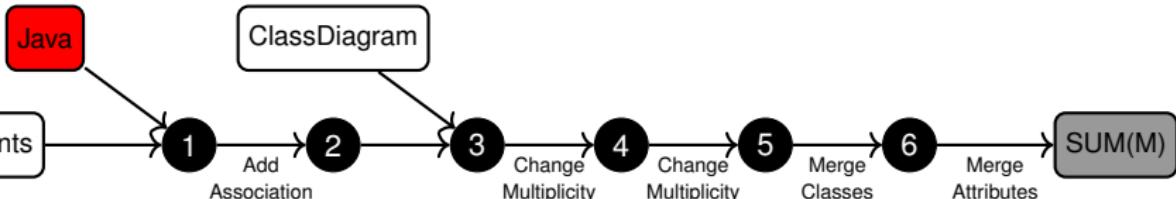
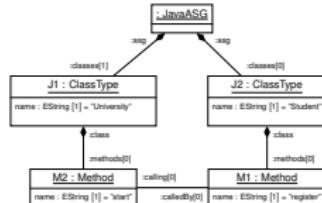
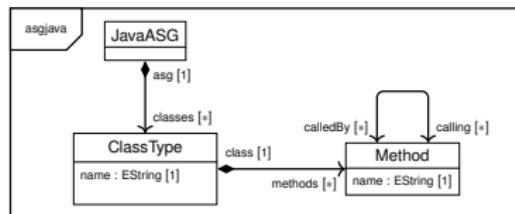
## 2. Initialization of SUM: Overview



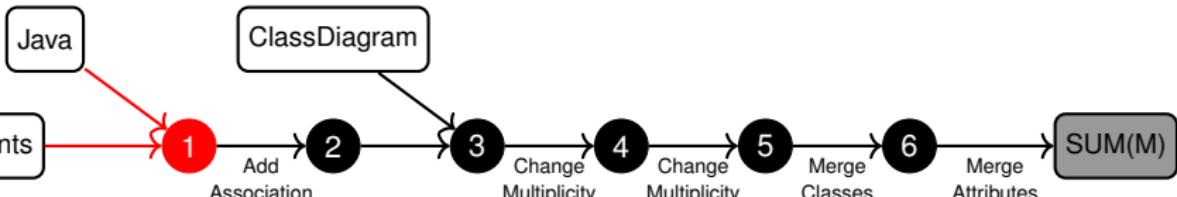
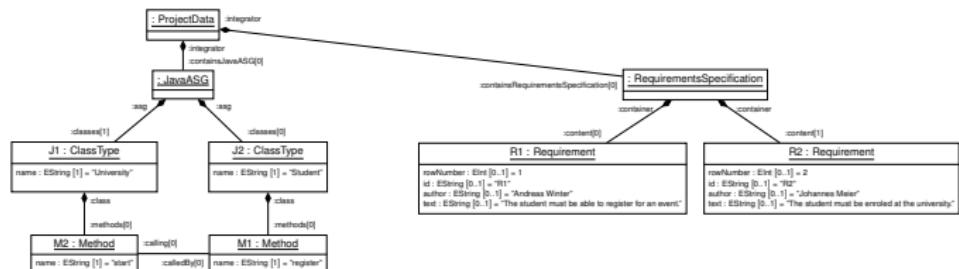
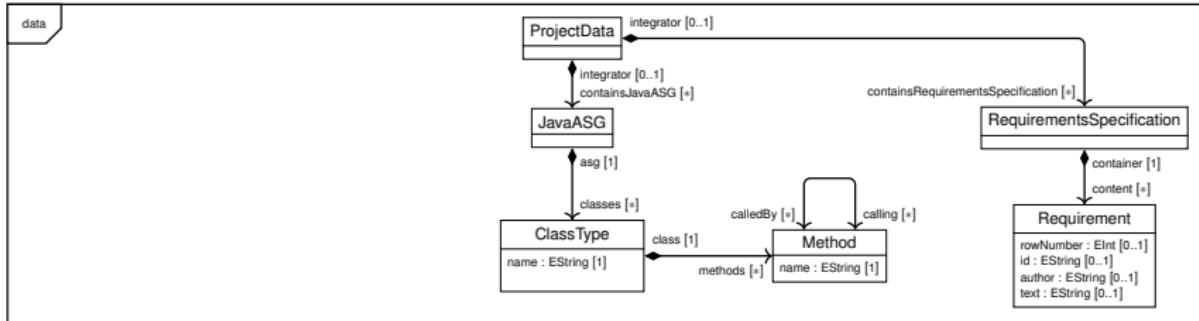
## 2. Initialization of SUM: Overview



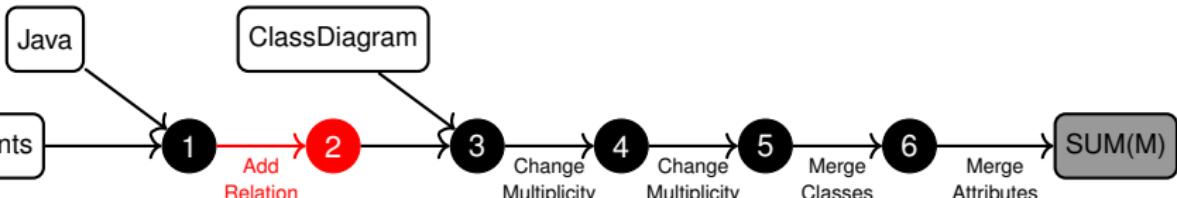
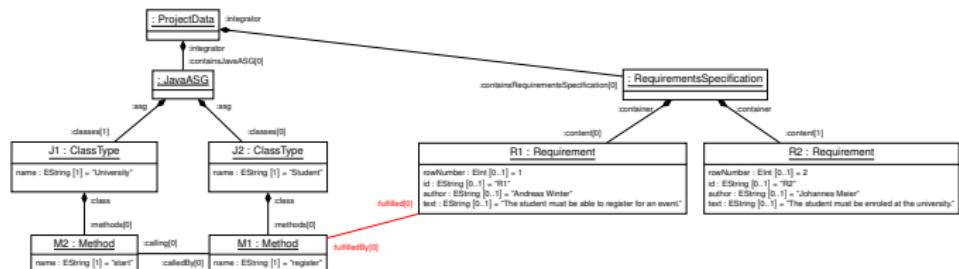
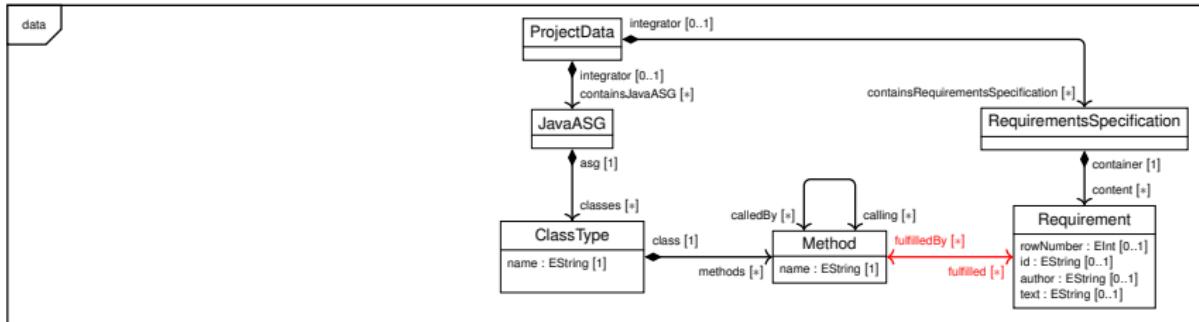
## 2. Initialization of SUM: Overview



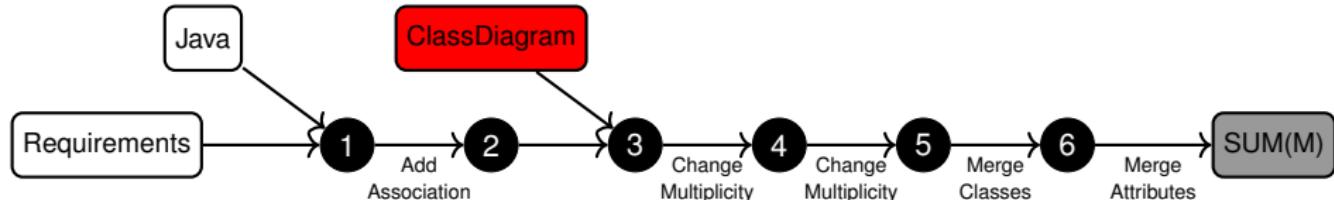
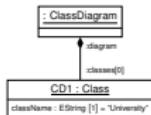
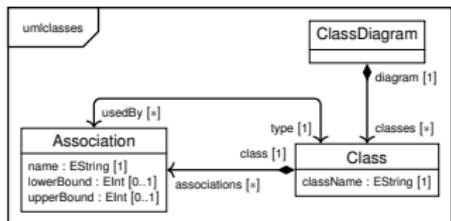
## 2. Initialization of SUM: Overview



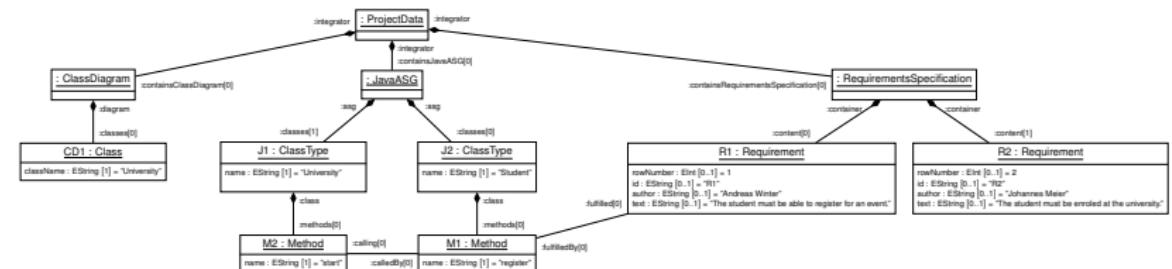
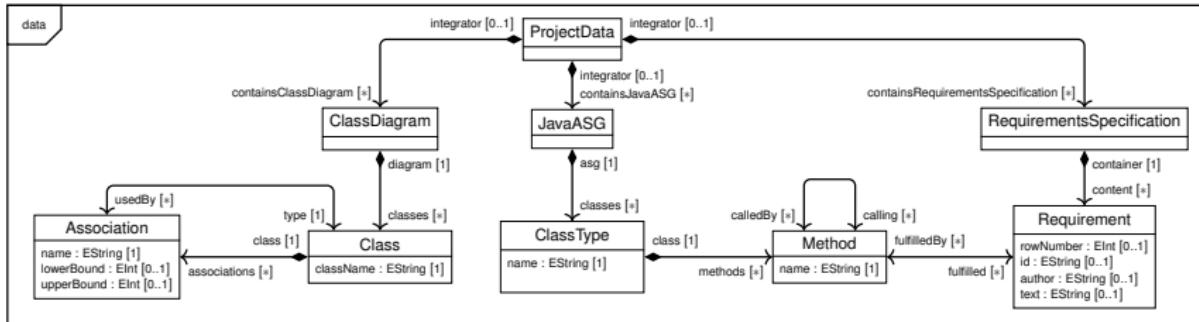
## 2. Initialization of SUM: Overview



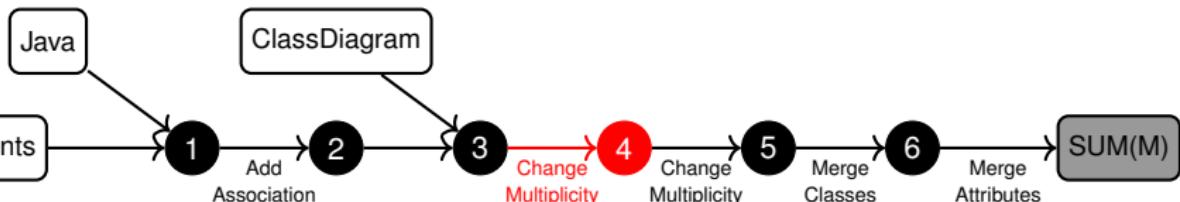
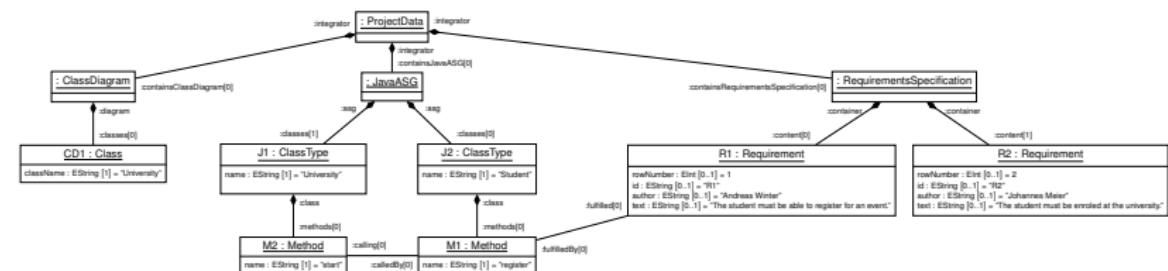
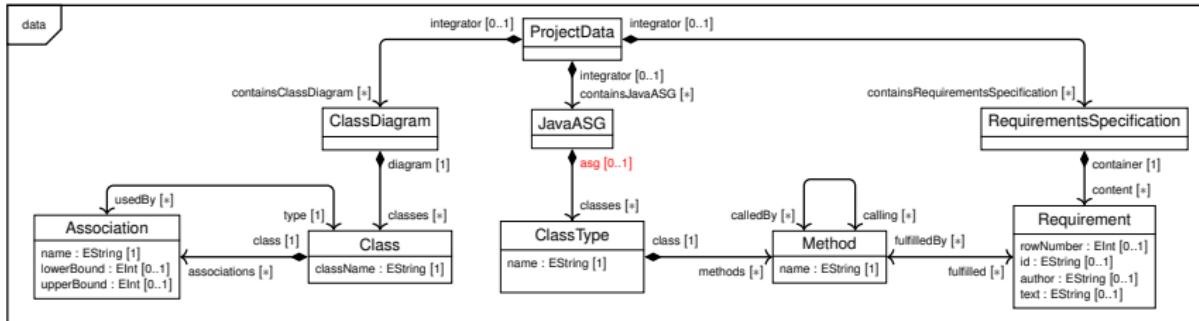
## 2. Initialization of SUM: Overview



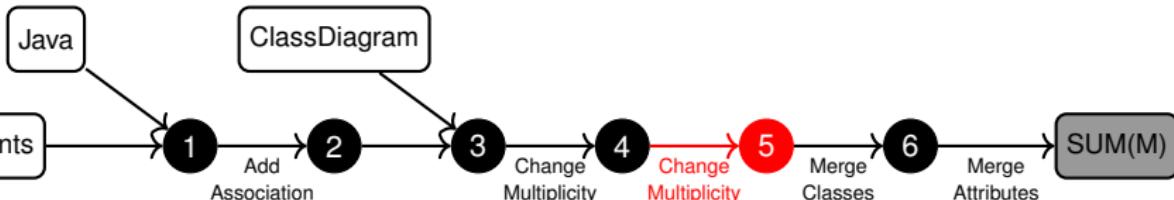
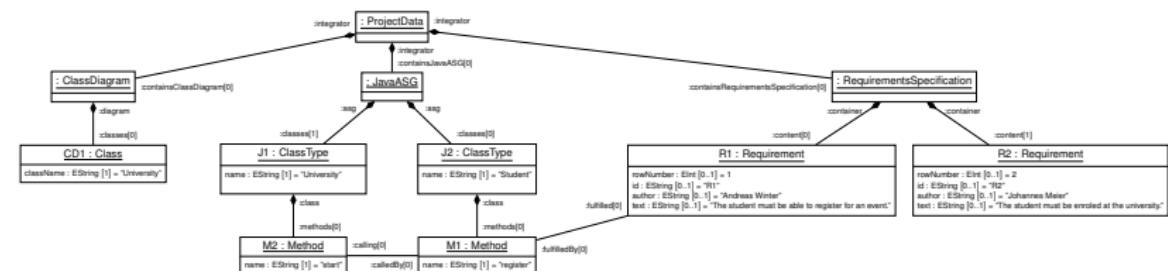
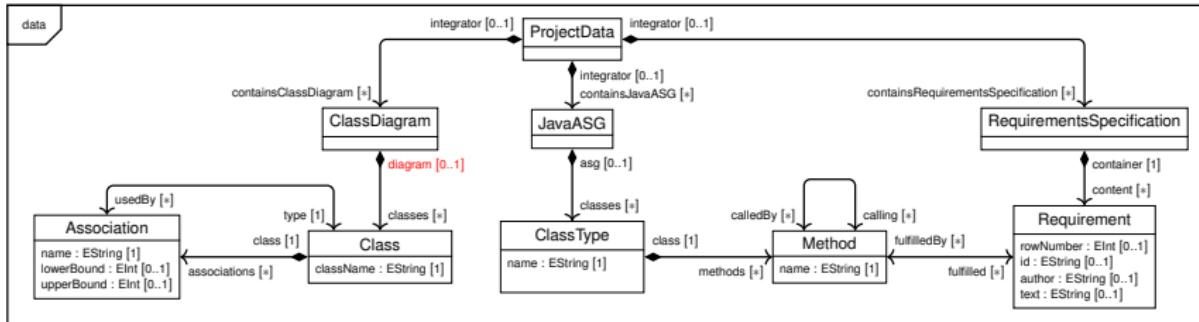
## 2. Initialization of SUM: Overview



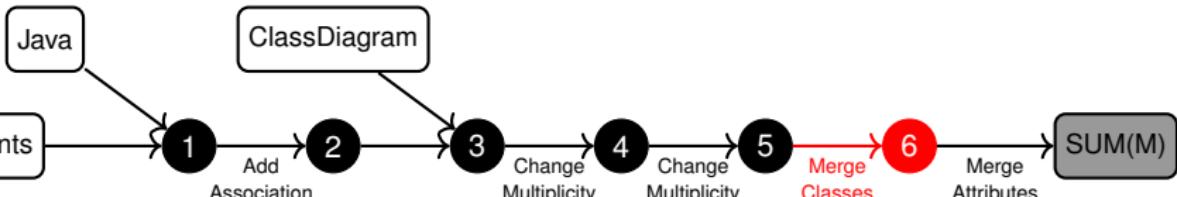
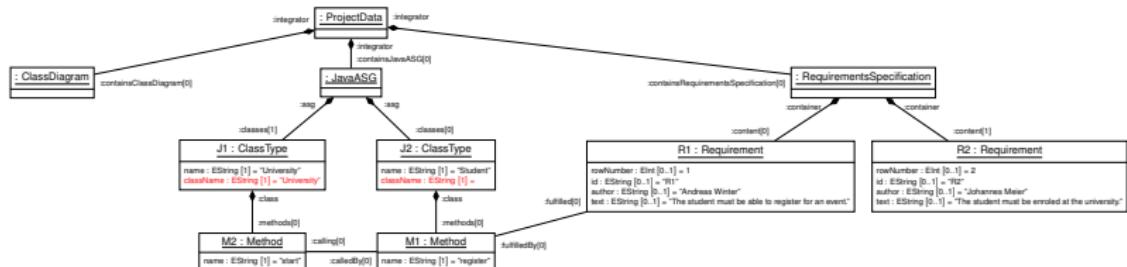
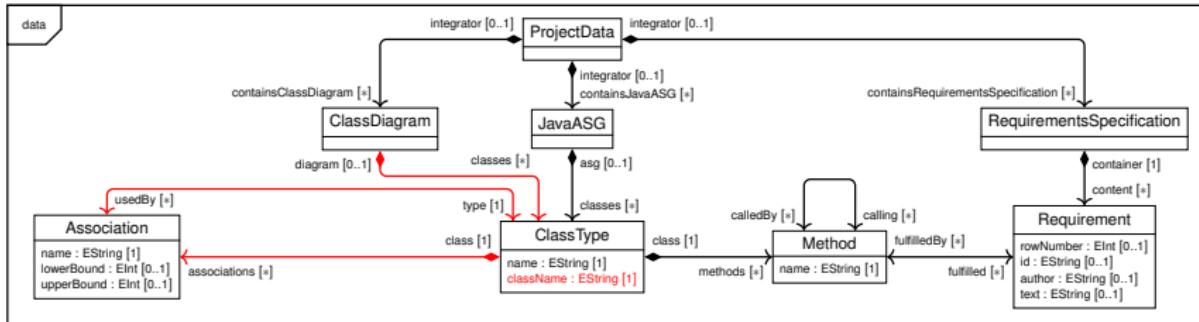
## 2. Initialization of SUM: Overview



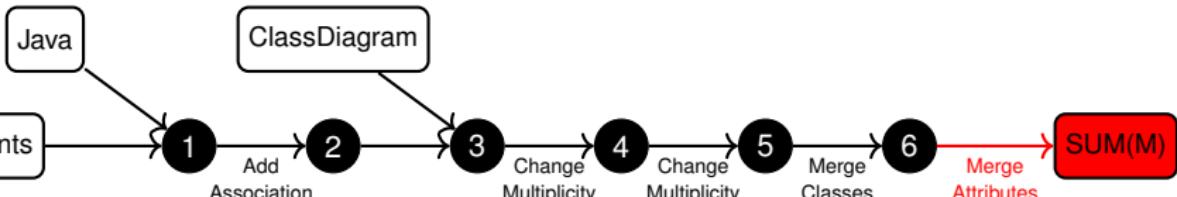
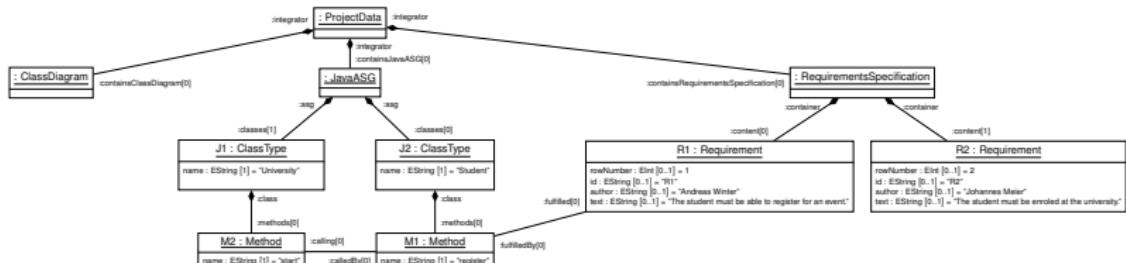
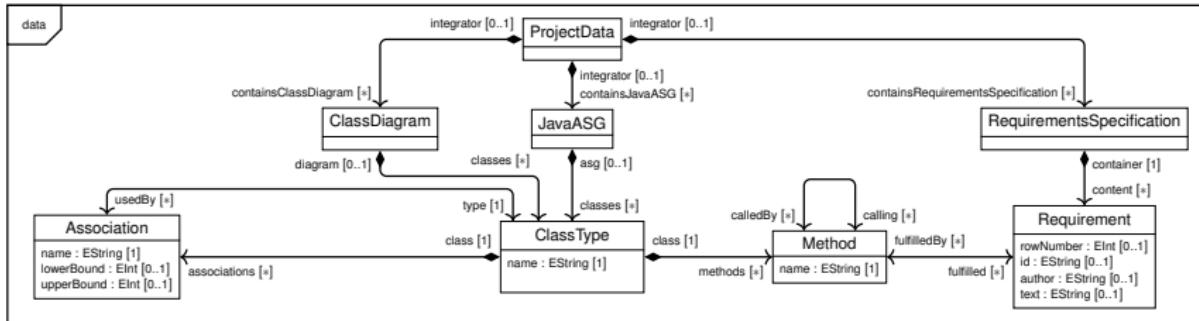
## 2. Initialization of SUM: Overview



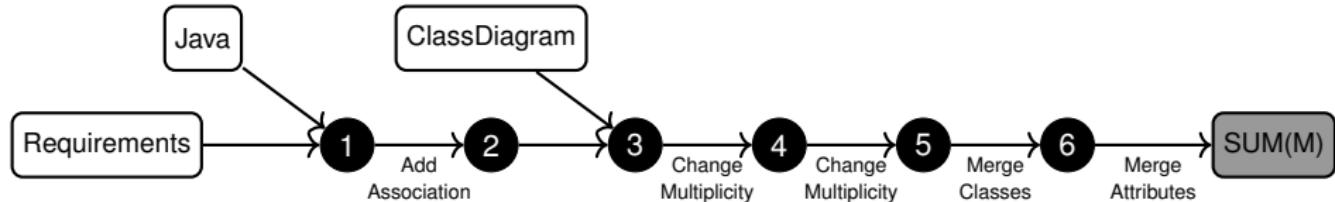
## 2. Initialization of SUM: Overview



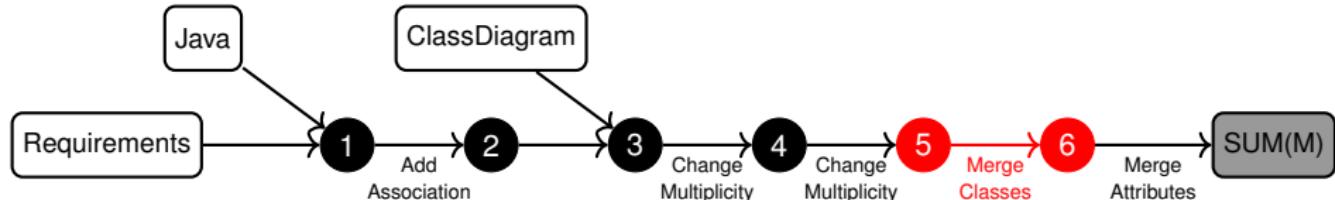
## 2. Initialization of SUM: Overview



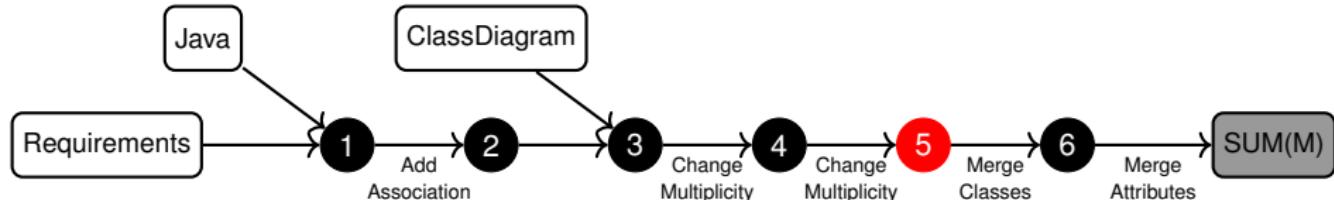
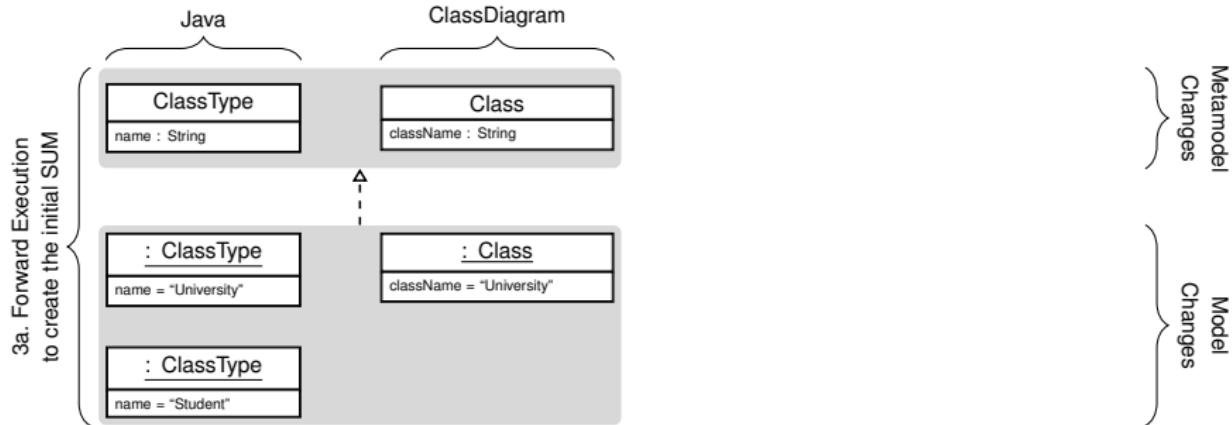
## 2. Initialization of SUM: Details



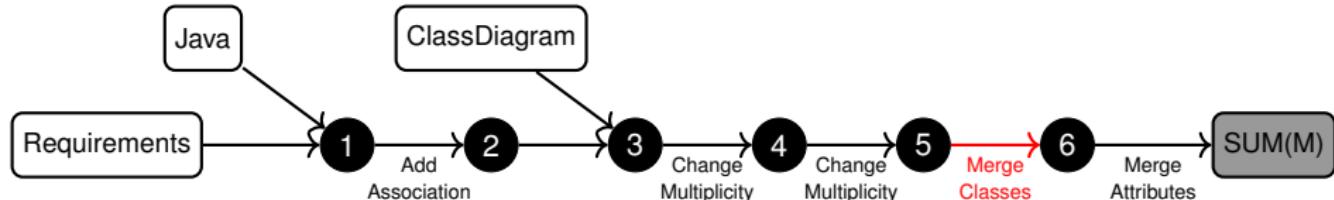
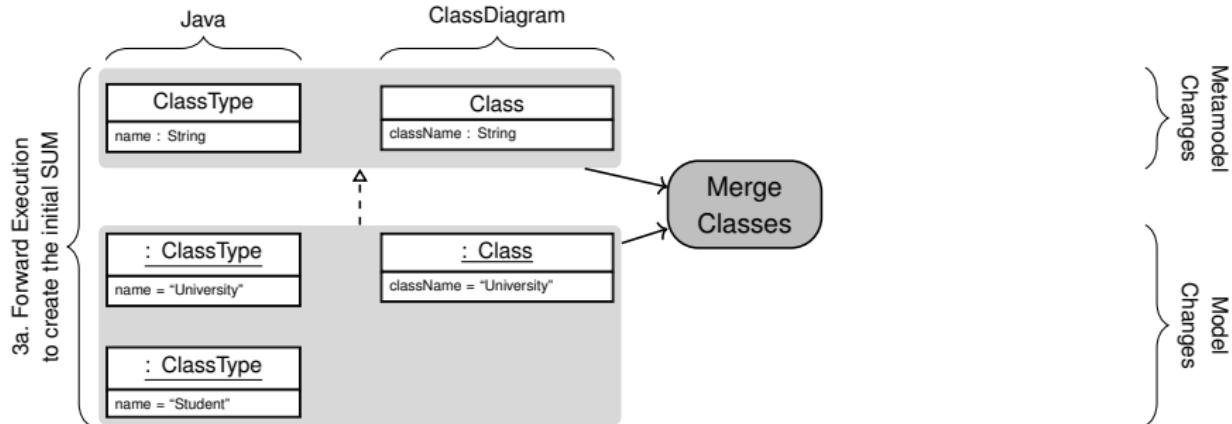
## 2. Initialization of SUM: Details



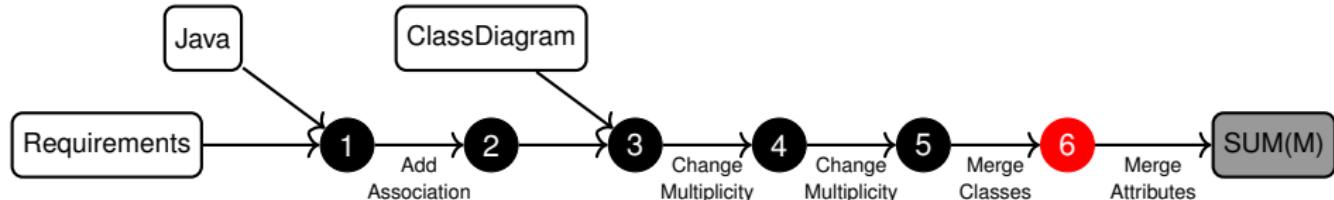
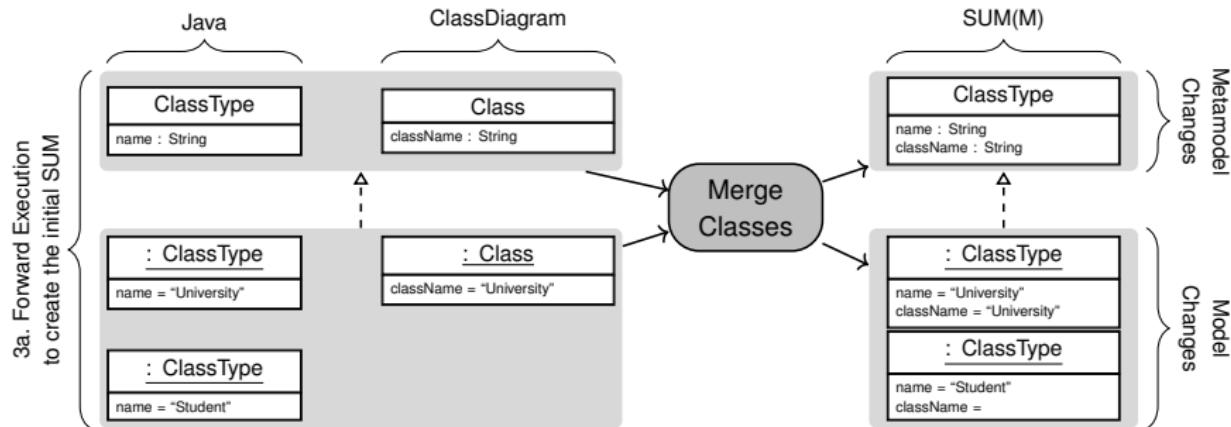
## 2. Initialization of SUM: Details



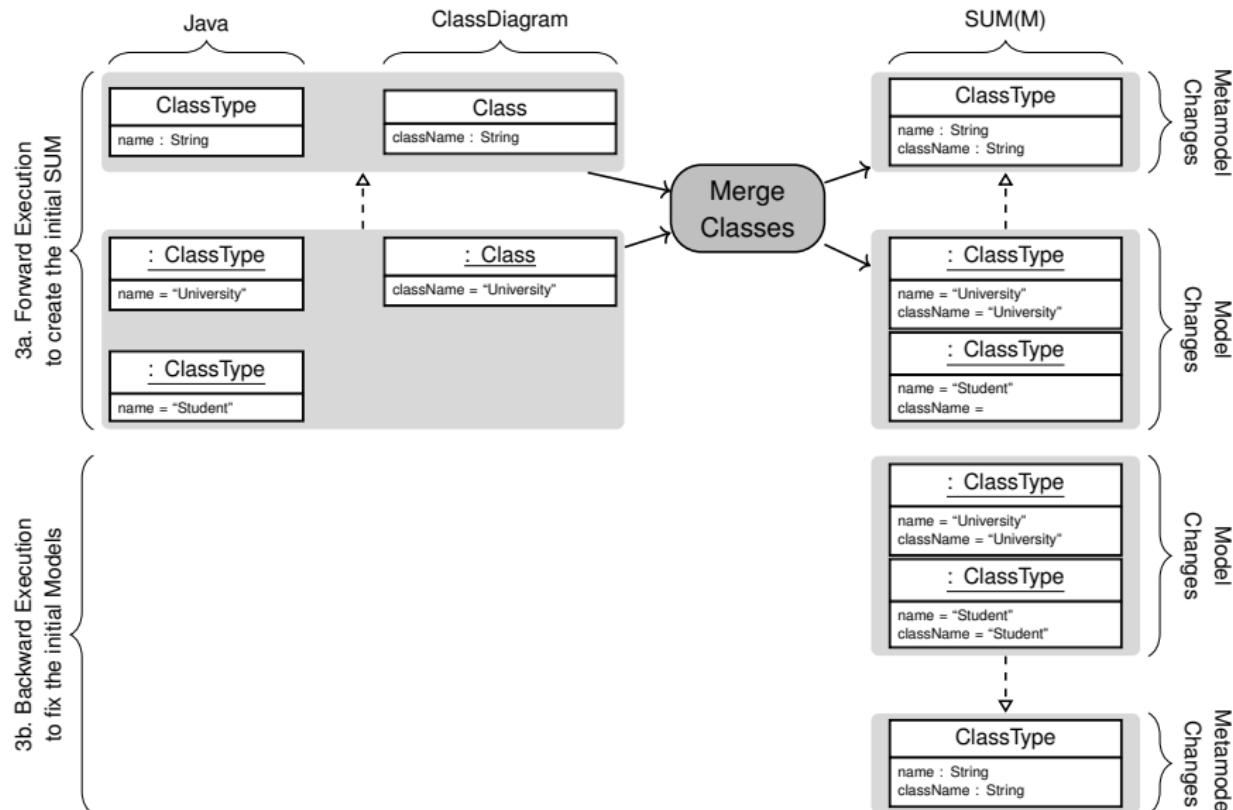
## 2. Initialization of SUM: Details



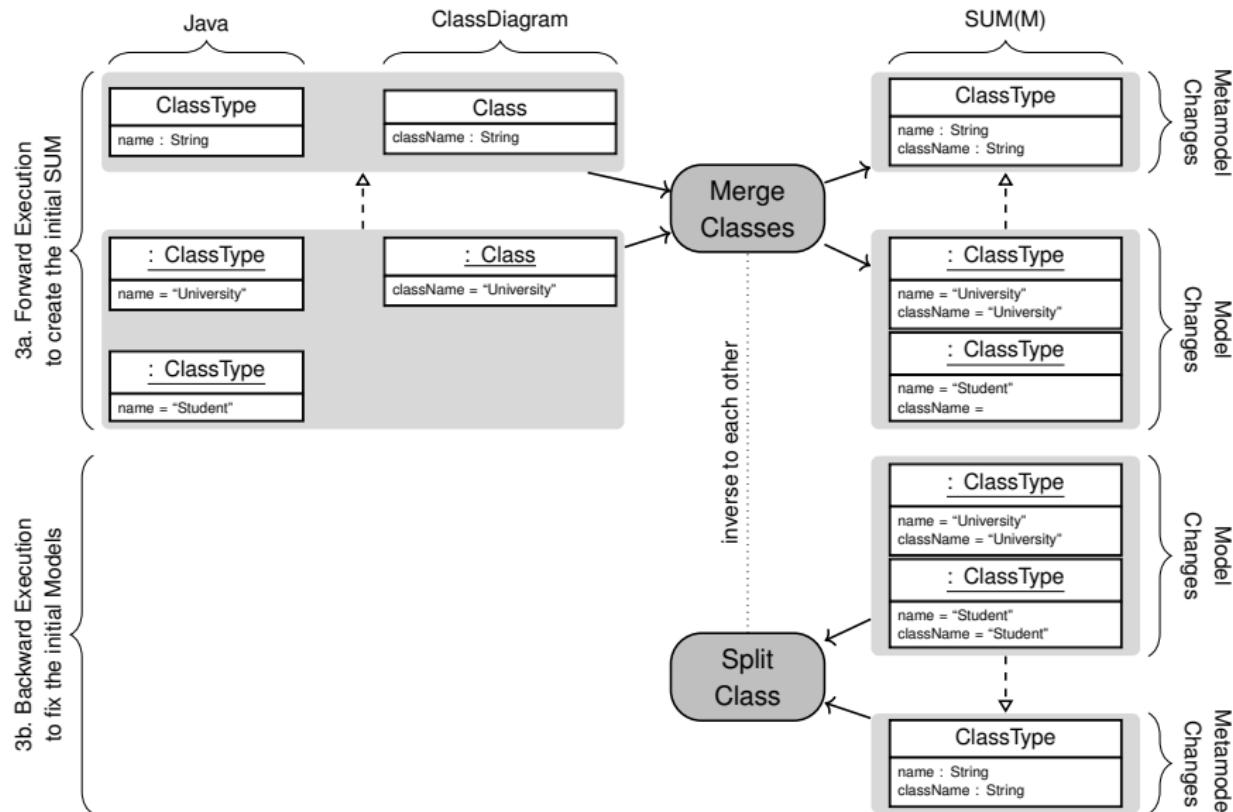
## 2. Initialization of SUM: Details



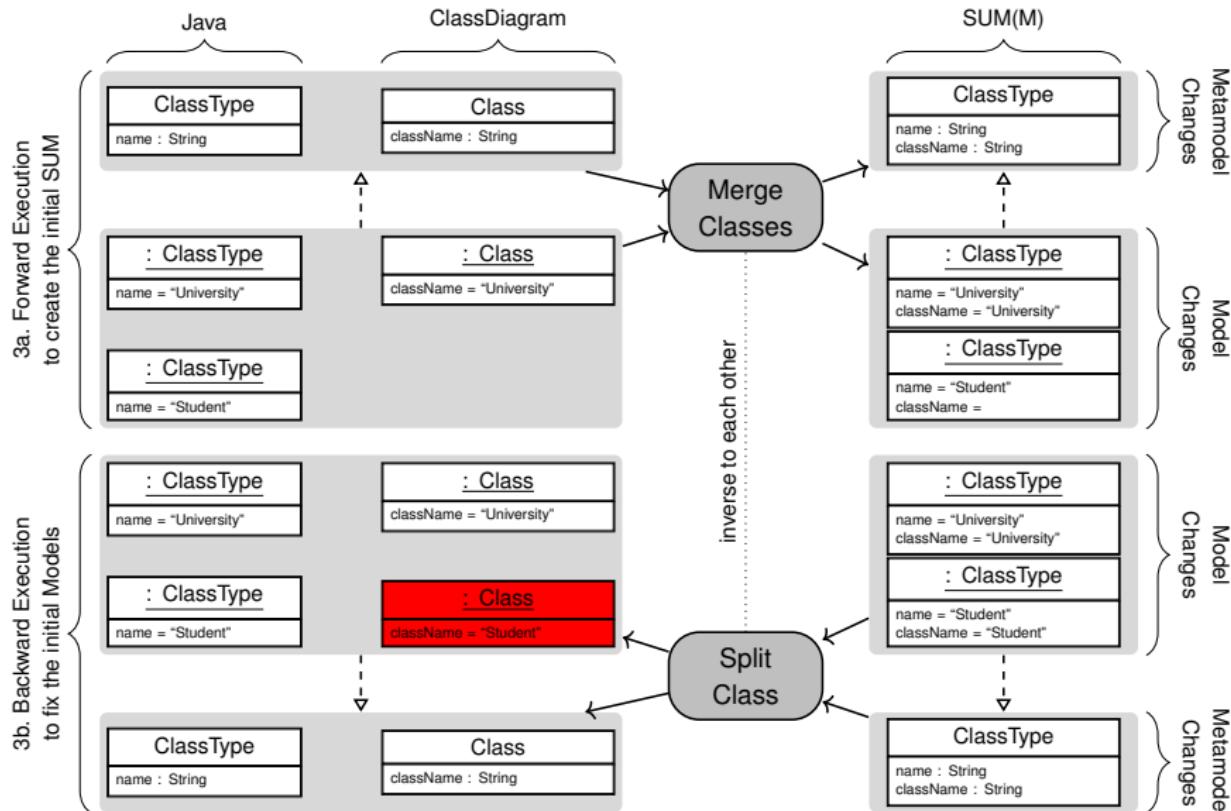
## 2. Initialization of SUM: Details



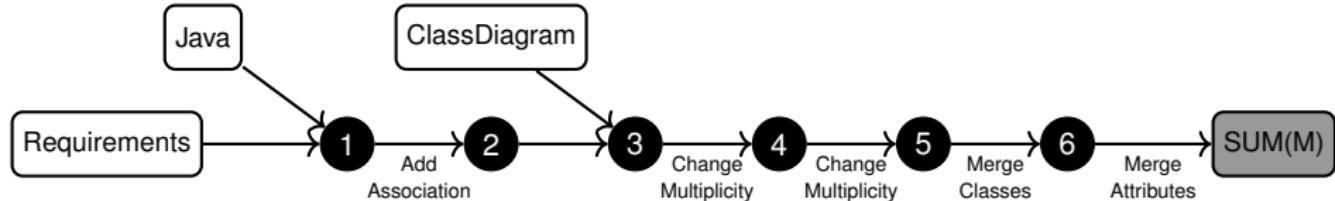
## 2. Initialization of SUM: Details



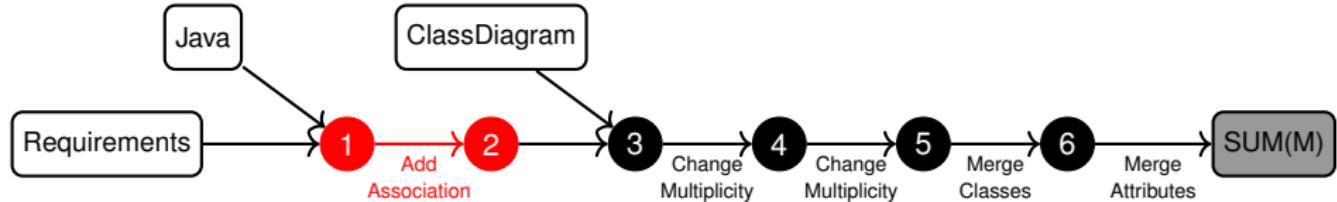
## 2. Initialization of SUM: Details



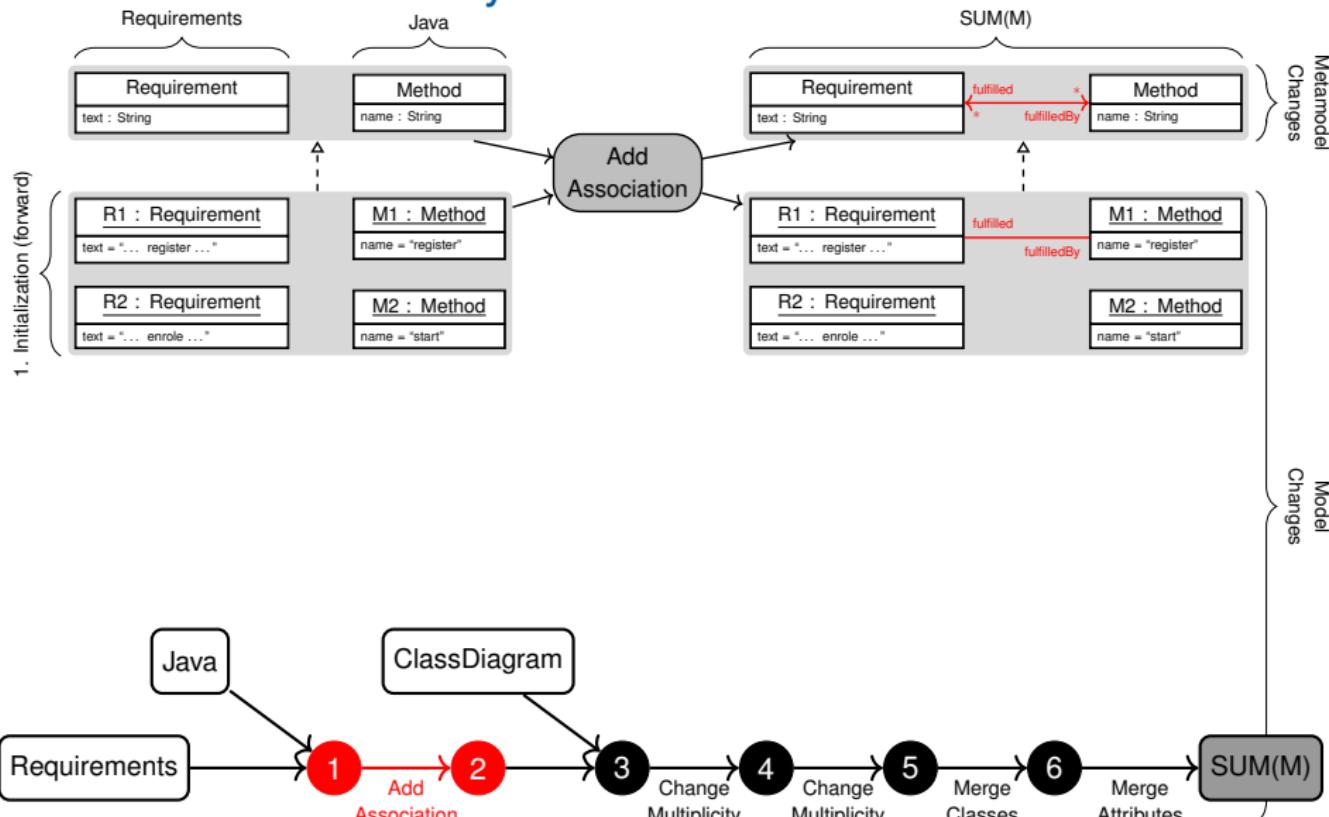
### 3. Consistency Assurance: Details



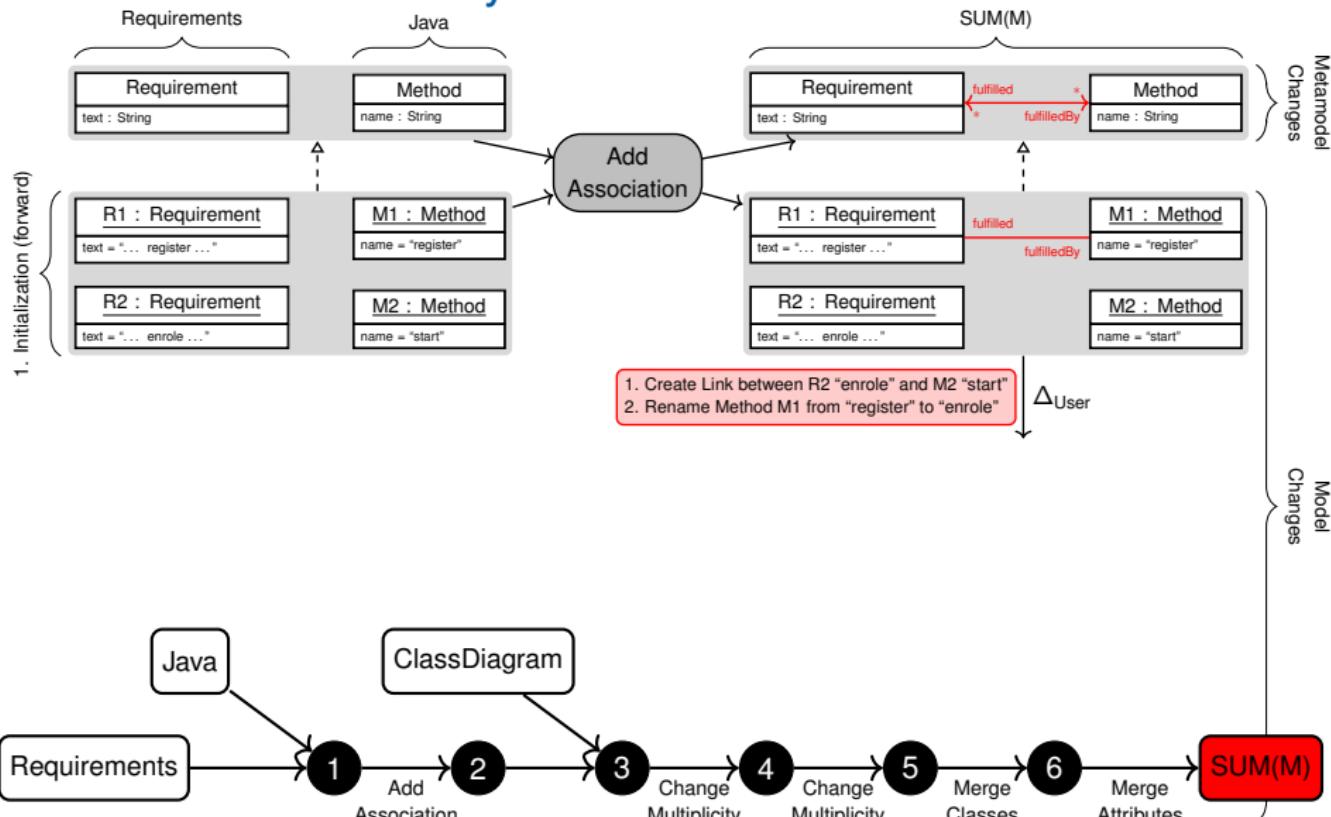
### 3. Consistency Assurance: Details



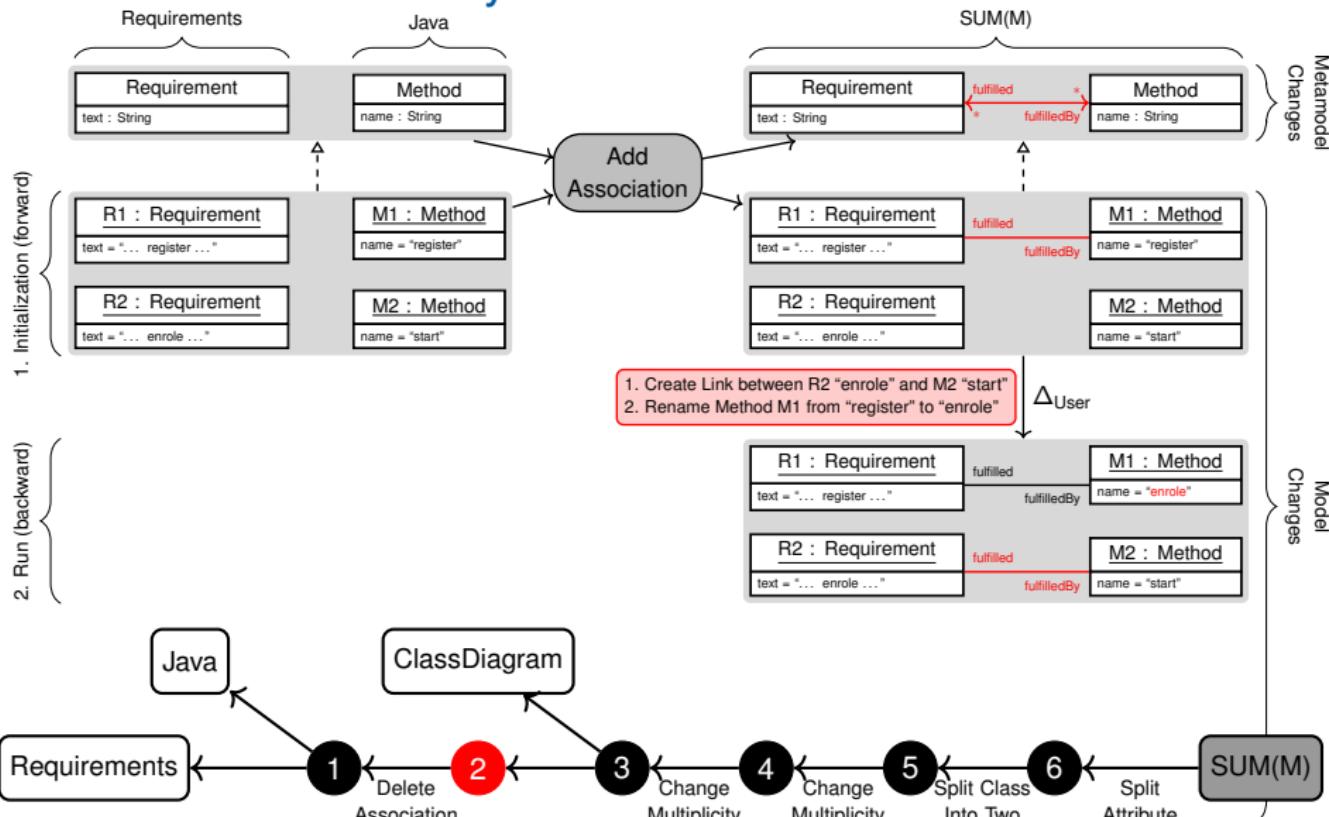
### 3. Consistency Assurance: Details



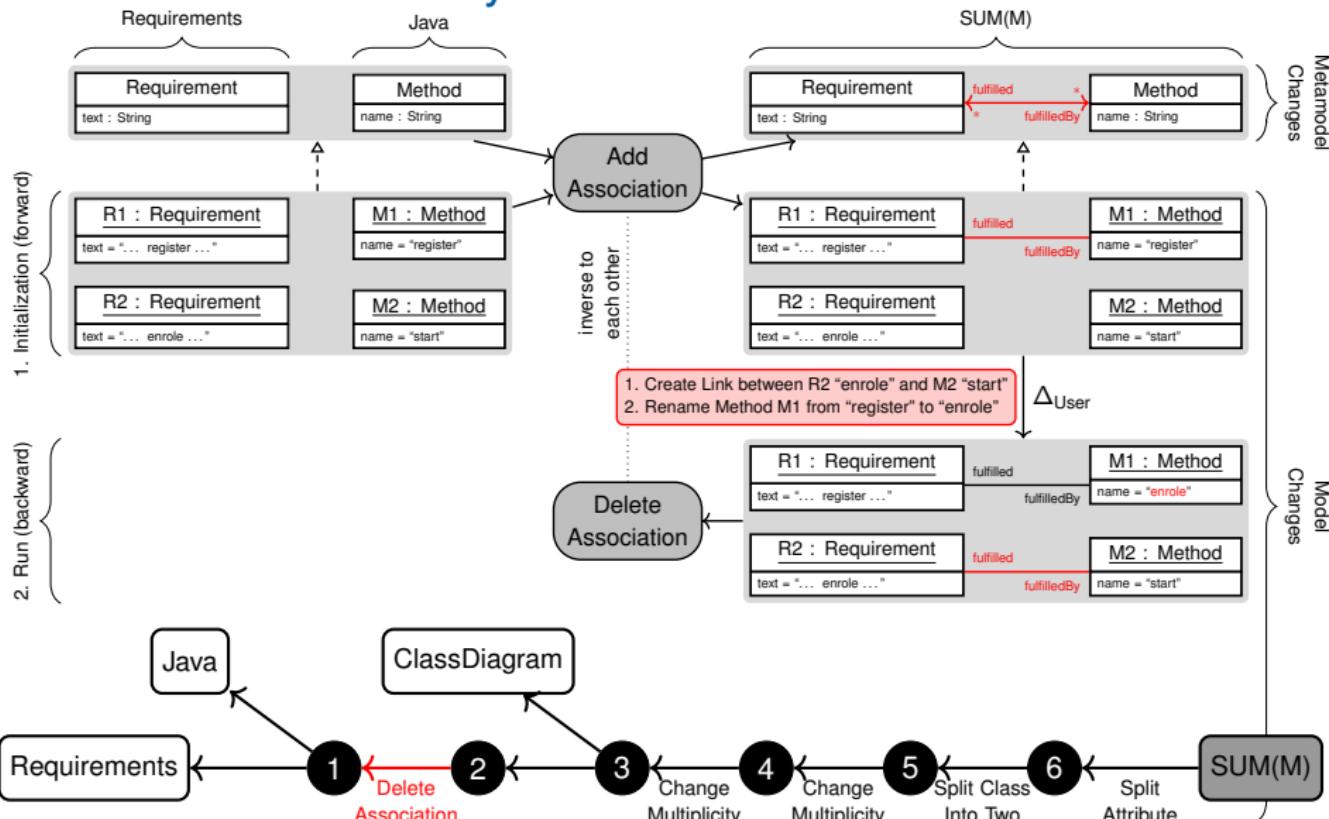
### 3. Consistency Assurance: Details



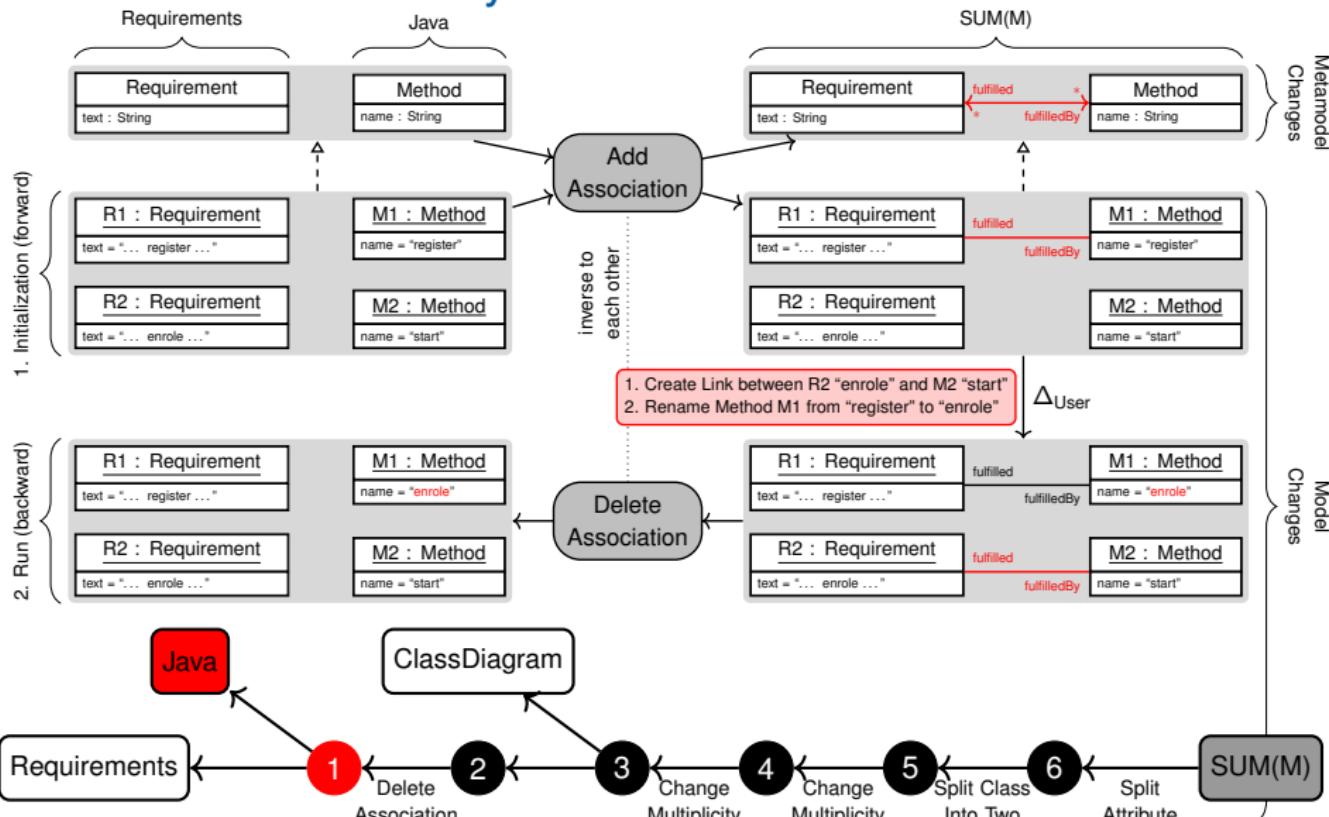
### 3. Consistency Assurance: Details



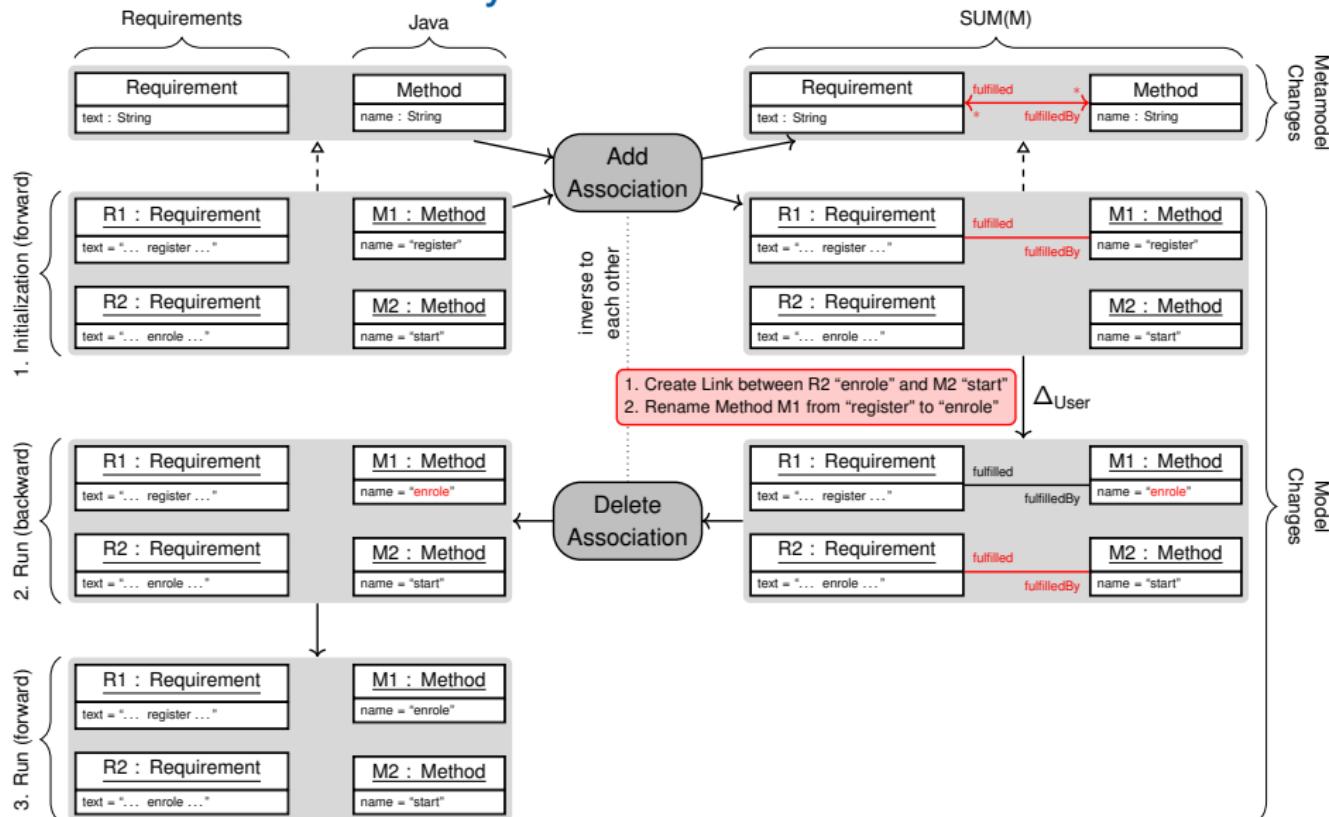
### 3. Consistency Assurance: Details



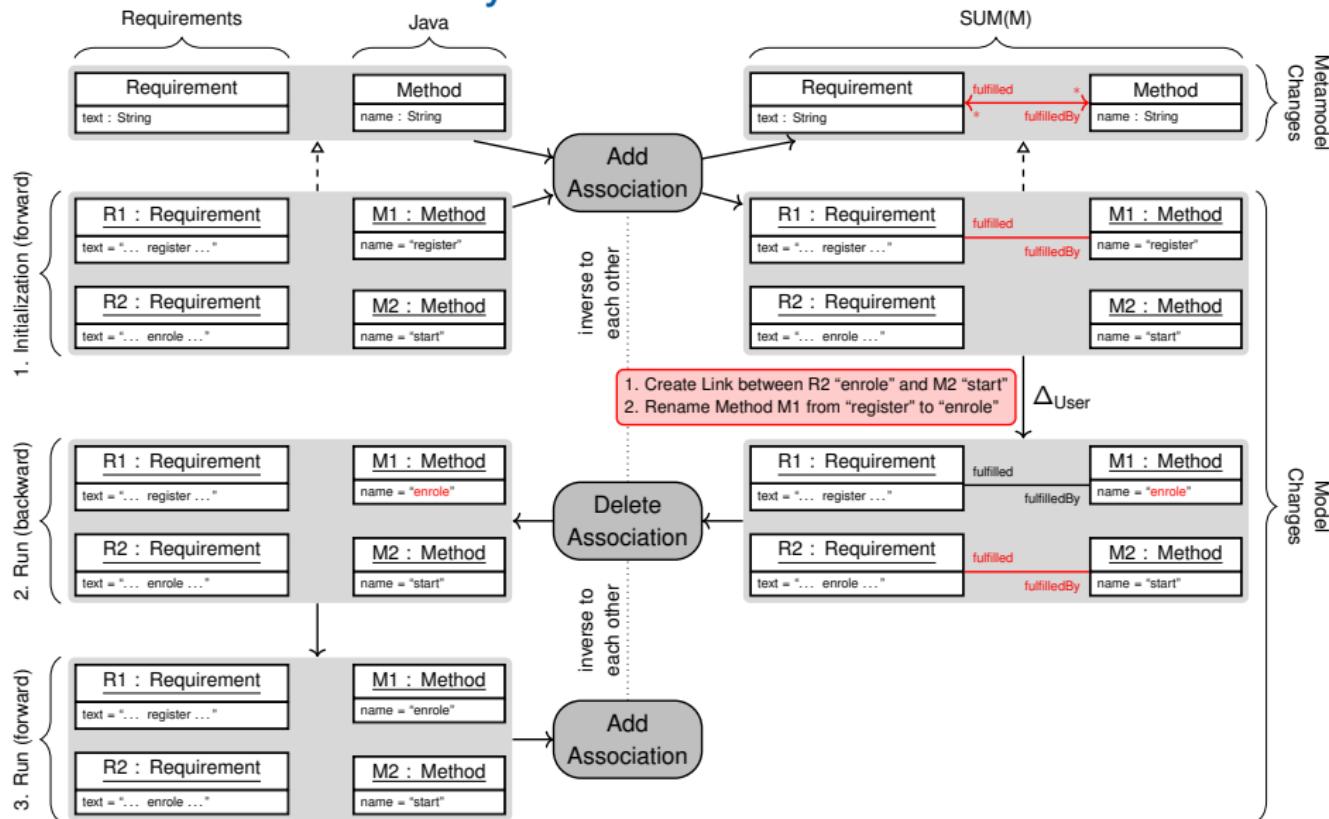
### 3. Consistency Assurance: Details



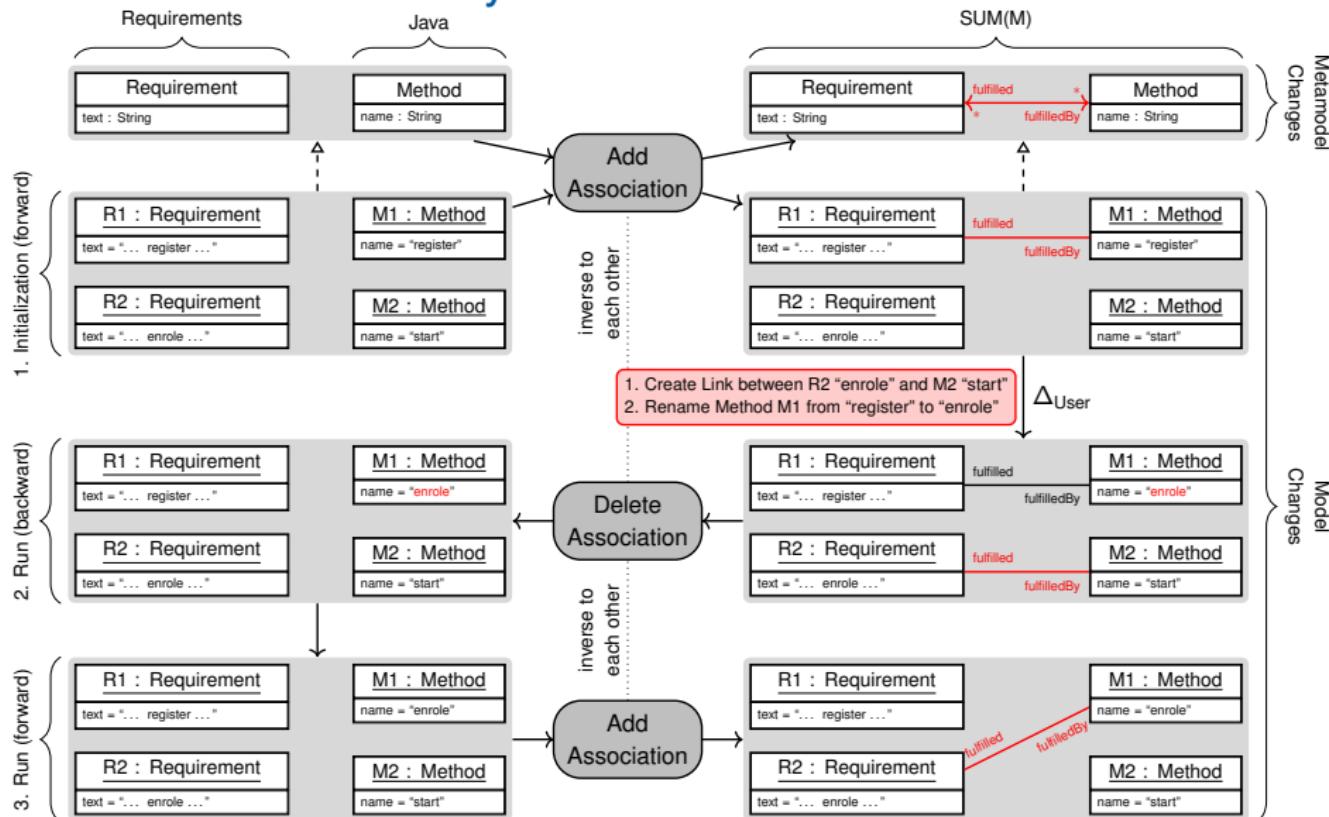
### 3. Consistency Assurance: Details



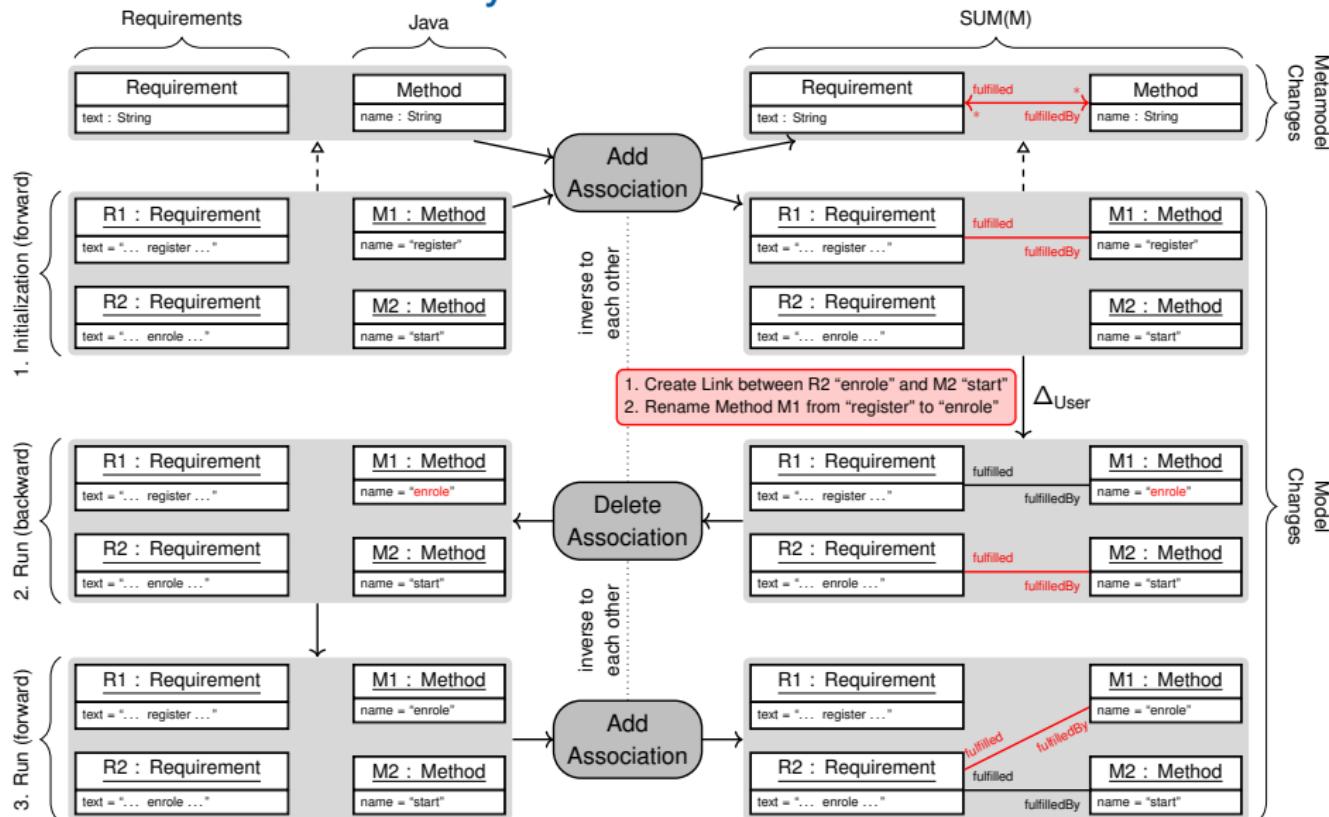
### 3. Consistency Assurance: Details



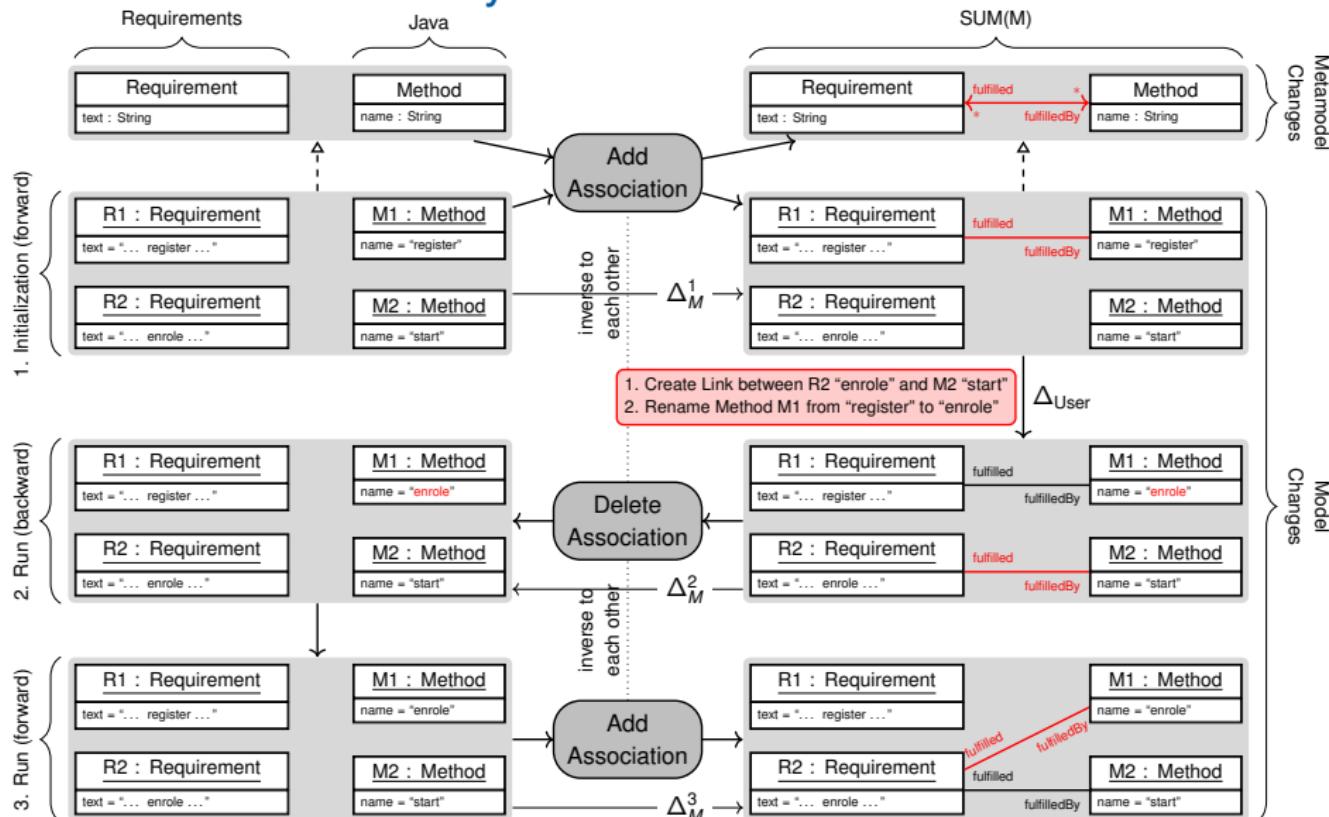
### 3. Consistency Assurance: Details



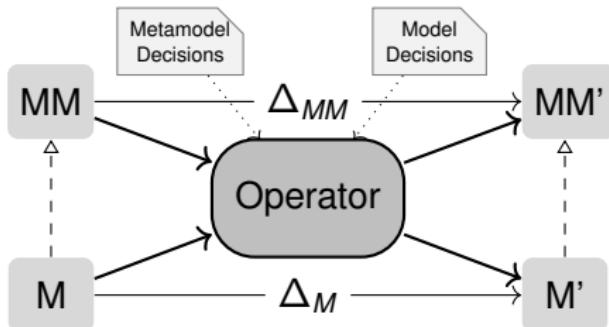
### 3. Consistency Assurance: Details



### 3. Consistency Assurance: Details



## Operators: Summary

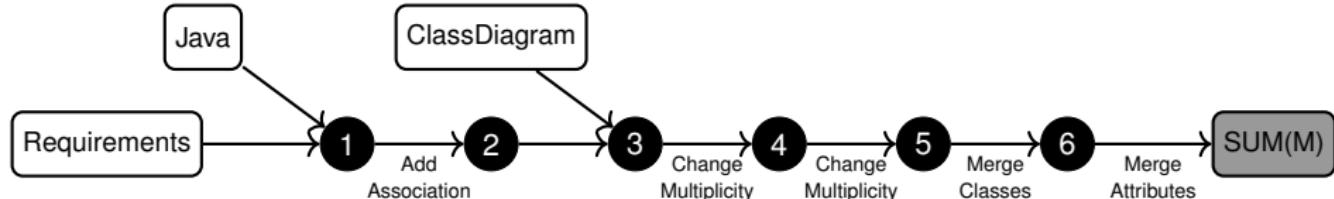


- **Metamodel Change  $\Delta_{MM}$ :**  
small Change in the Metamodel
- **Model Change  $\Delta_M$ :**  
handle Model-Co-Evolution (Coupled Operators [HVW11])
- Configurations by the Methodologist:
  - ▶ **Metamodel Decisions:** set Properties for wanted Metamodel Changes
  - ▶ **Model Decisions:** describe Model Changes for Consistency Rules
- **Bi-Directionality only for MM:** combine with inverse Operator
- currently 20 Operators implemented

## Summary

Operator-based bottom-up SUM-Approach for Model Consistency:

- Methodologist configures arbitrary, but stable Chain of configured Operators (once)
- User applies Changes and Model Consistency is ensured automatically by executing the Operator Chain
- separated Models are migrated to projectional Views on the SUM



## Literature I

- [ASB09] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. Supporting View-Based Development through Orthographic Software Modeling. Evaluation of Novel Approaches to Software Engineering (ENASE), pages 71–86, 2009.
- [BHK<sup>+</sup>14] Erik Burger, Jörg Henss, Martin Küster, Steffen Kruse, and Lucia Happe. View-based model-driven software development with ModelJoin. Software & Systems Modeling, 2014.
- [CNS12] Marsha Chechik, Shiva Nejati, and Mehrdad Sabetzadeh. A Relationship-Based Approach to Model Integration. Innovations Syst Softw Eng, 8(123):3–18, 2012.
- [EEC<sup>+</sup>14] Mahmoud El Hamlaoui, Sophie Ebersold, Bernard Coulette, Mahmoud Nassar, and Adil Anwar. Heterogeneous models matching for consistency management. In 2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS), pages 1–12. IEEE, may 2014.
- [HWW11] Markus Herrmannsdoerfer, Sander D. Vermolen, and Guido Wachsmuth. An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. Software Language Engineering, LNCS 6563:163–182, 2011.

## Literature II

- [IEE11] IEEE. ISO/IEC/IEEE 42010:2011 - Systems and software engineering - Architecture description. 2011(March):1–46, 2011.
- [KBL13] Max E Kramer, Erik Burger, and Michael Langhammer. View-centric engineering with synchronized heterogeneous models. Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling - VAO '13, pages 1–6, 2013.
- [LDC18] Manuel Leduc, Thomas Degueule, and Benoit Combemale. Modular Language Composition for the Masses. SLE 2018 - 11th ACM SIGPLAN International Conference on Software Language Engineering, 2018.
- [MW18] Johannes Meier and Andreas Winter. Towards Evolution Scenarios of Integrated Software Artifacts. Softwaretechnik-Trends, 38(2):63–64, 2018.
- [RJV09] J. R. Romero, Juan Ignacio Jaén, and Antonio Vallejillo. Realizing correspondences in multi-viewpoint specifications. Proceedings - 13th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2009, pages 163–172, 2009.
- [SK08] Andy Schürr and Felix Klar. 15 Years of triple graph grammars: Research challenges, new contributions, open problems. Lecture Notes in Computer Science, 5214 LNCS:411–425, 2008.