Towards Synchronizing Relations Between Artifacts in the Java Technological Space

Author: William Bombardelli da Silva
Advisor: Dr.-Ing. Frank Trollmann

Technische Universität Berlin
Fakultät IV Elektrotechnik und Informatik
Bachelorstudiengang Informatik

wbombardellis@win.tu-berlin.de

16.03.2016

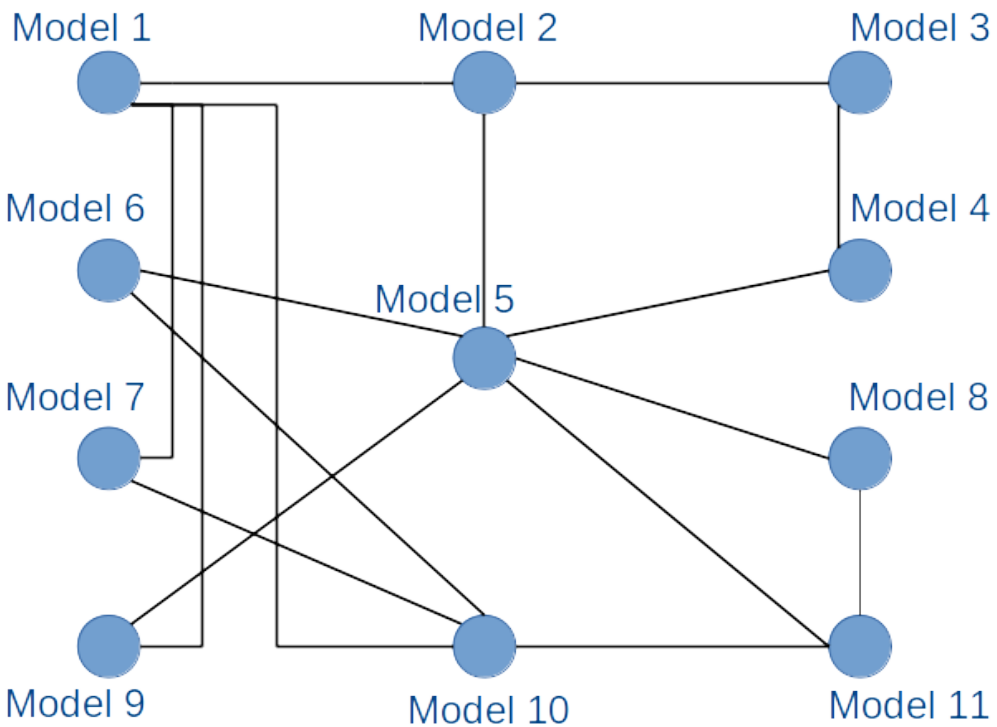
Organization

Background

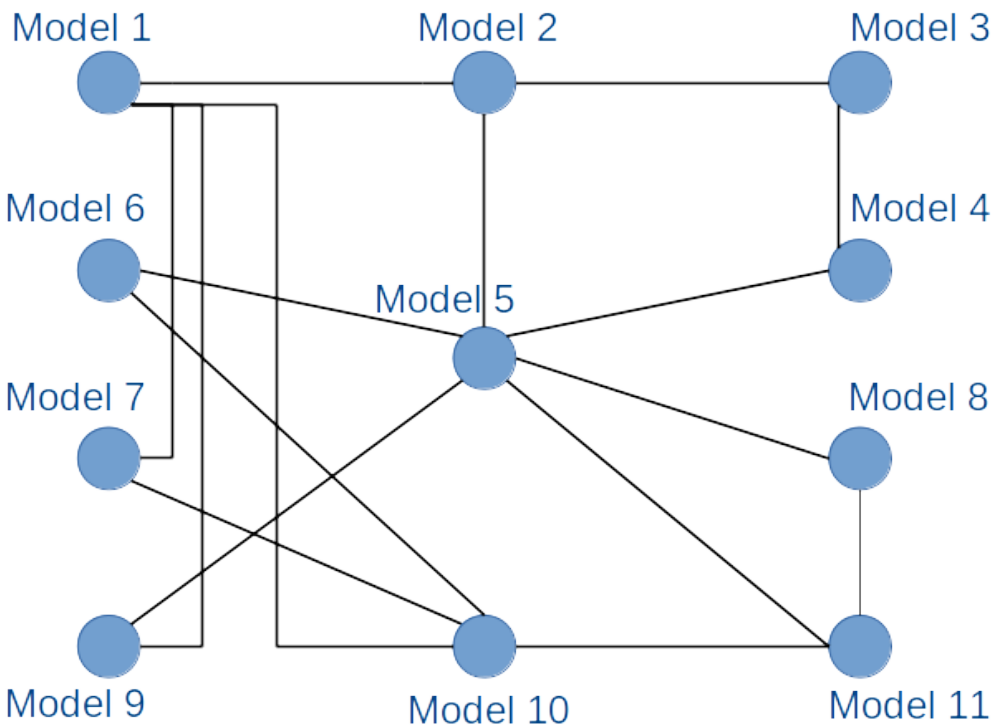
Models are used in Software Engineering

- New needs in industry and academy thrill new methods and paradigms.
- **Model-driven Engineering (MDE):** Software processes are oriented to models.
- One software may have several different models.

Models have to be kept consistent



Models have to be kept consistent



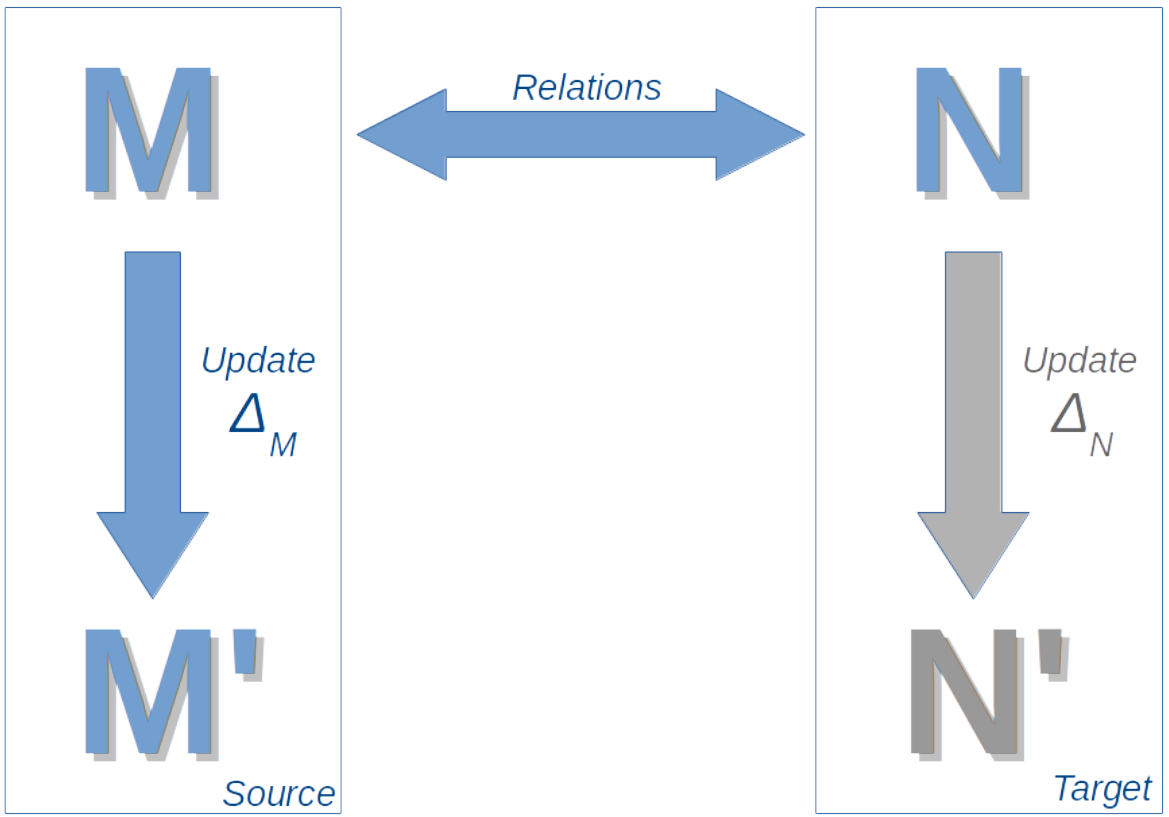
- Models are to be maintained consistent as they evolve.
- This means **models synchronization**.

Model Synchronization in the Network of Models

- For each edge of the network there is a synchronization task.

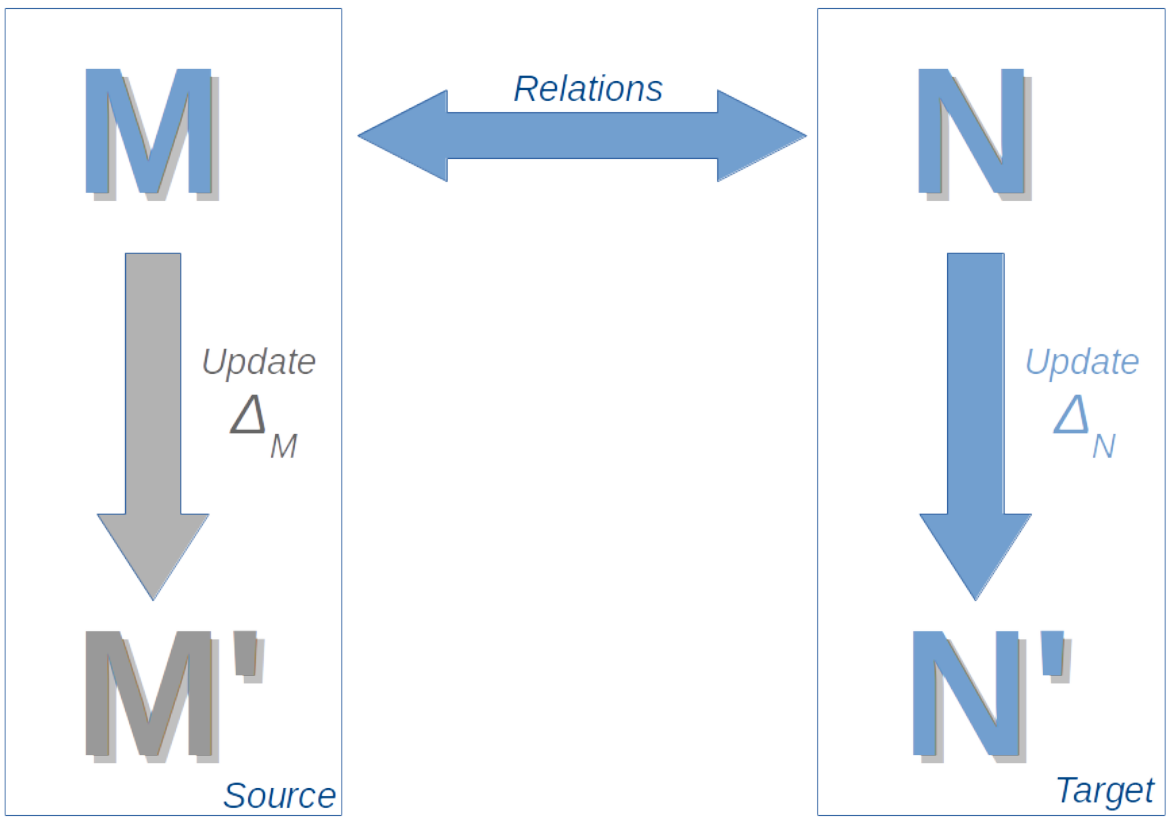
Model Synchronization in the Network of Models

- For each edge of the network there is a synchronization task.

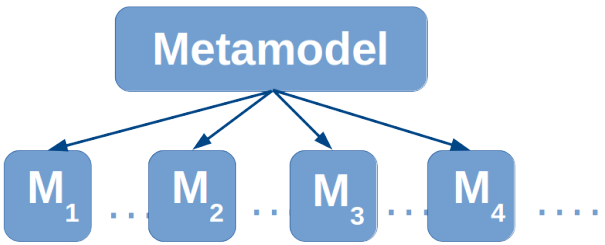


Model Synchronization in the Network of Models

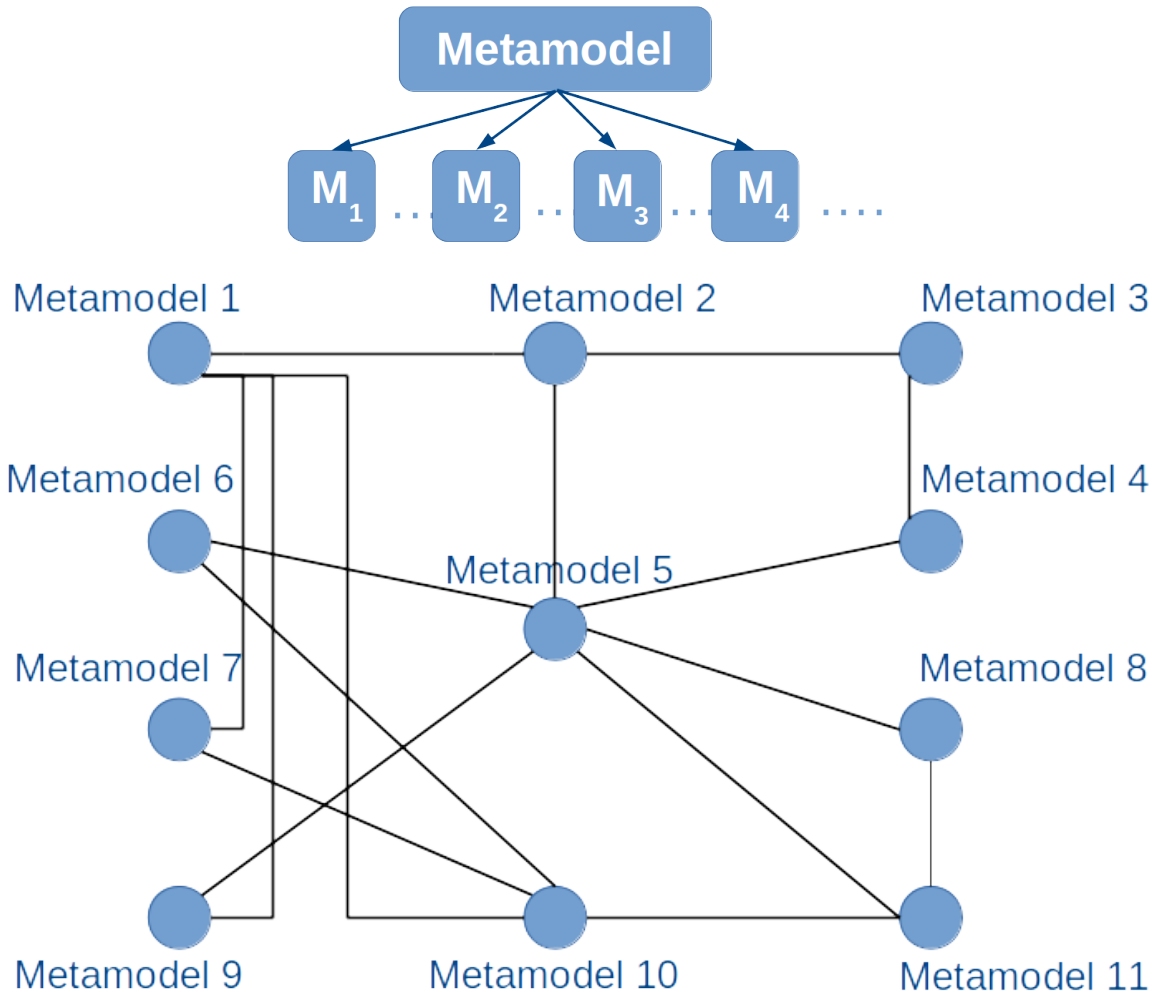
- For each edge of the network there is a synchronization task.



Relations are written between metamodels



Relations are written between metamodels



The problems of the current state-of-the-art

- Definitions of the **metamodels** in literature

The problems of the current state-of-the-art

- Definitions of the **metamodels** in literature
- Definitions of the **relations** in literature

The problems of the current state-of-the-art

- Definitions of the **metamodels** in literature
- Definitions of the **relations** in literature
- Approach able to treat synchronization of **complex technological spaces** with a large number of **tangled** metamodels.

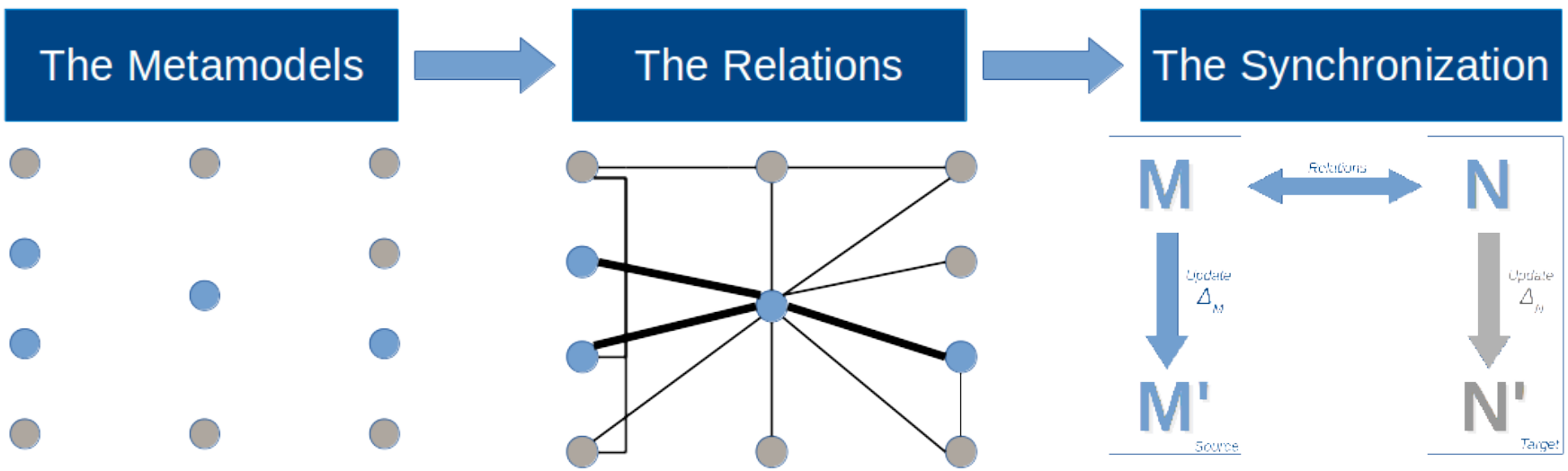
Objective

Three Steps

- Focus on the Java technological space.

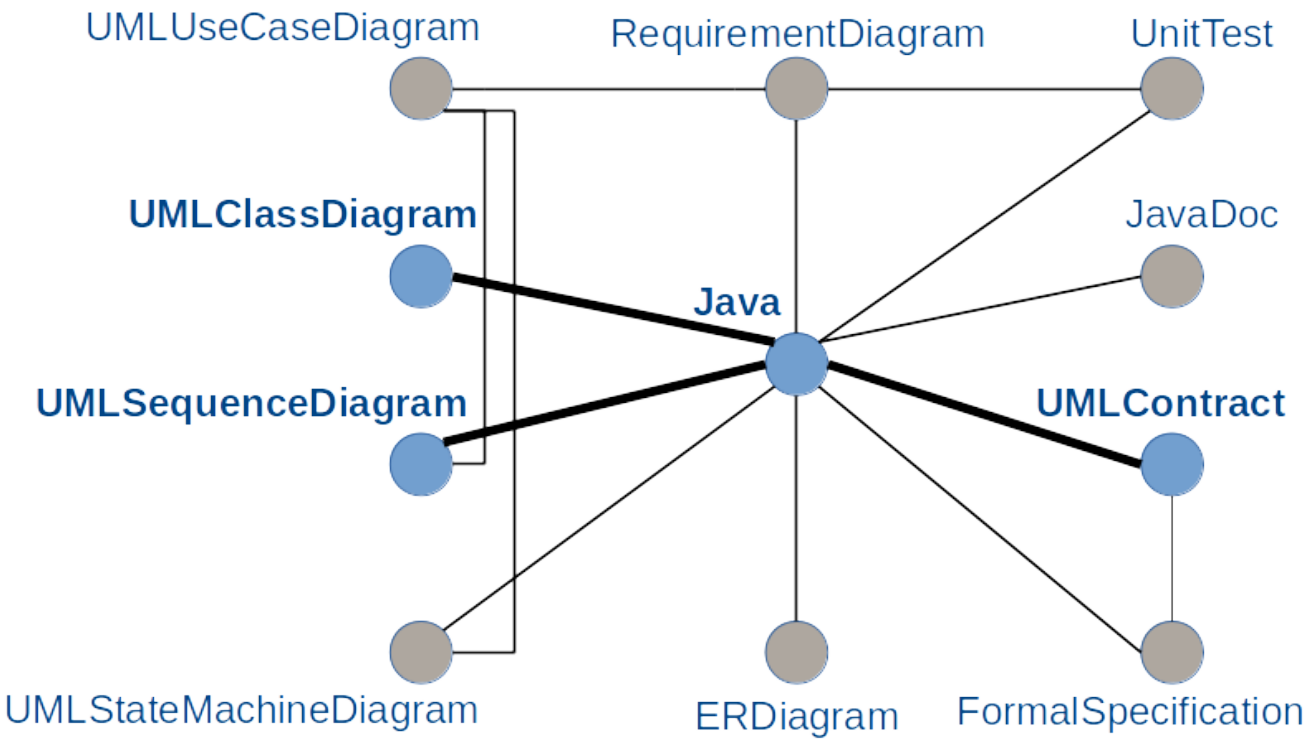
Three Steps

- Focus on the Java technological space.

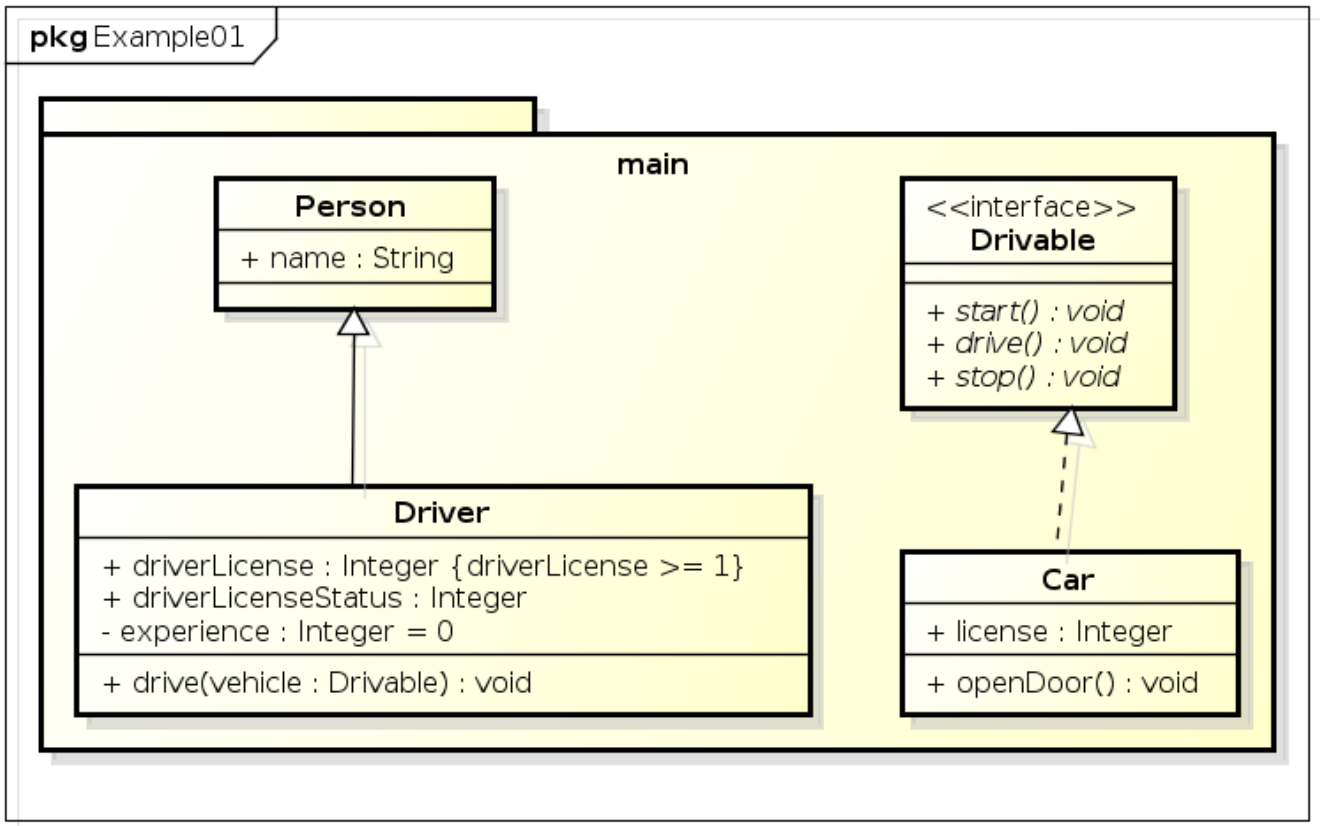


The Metamodels

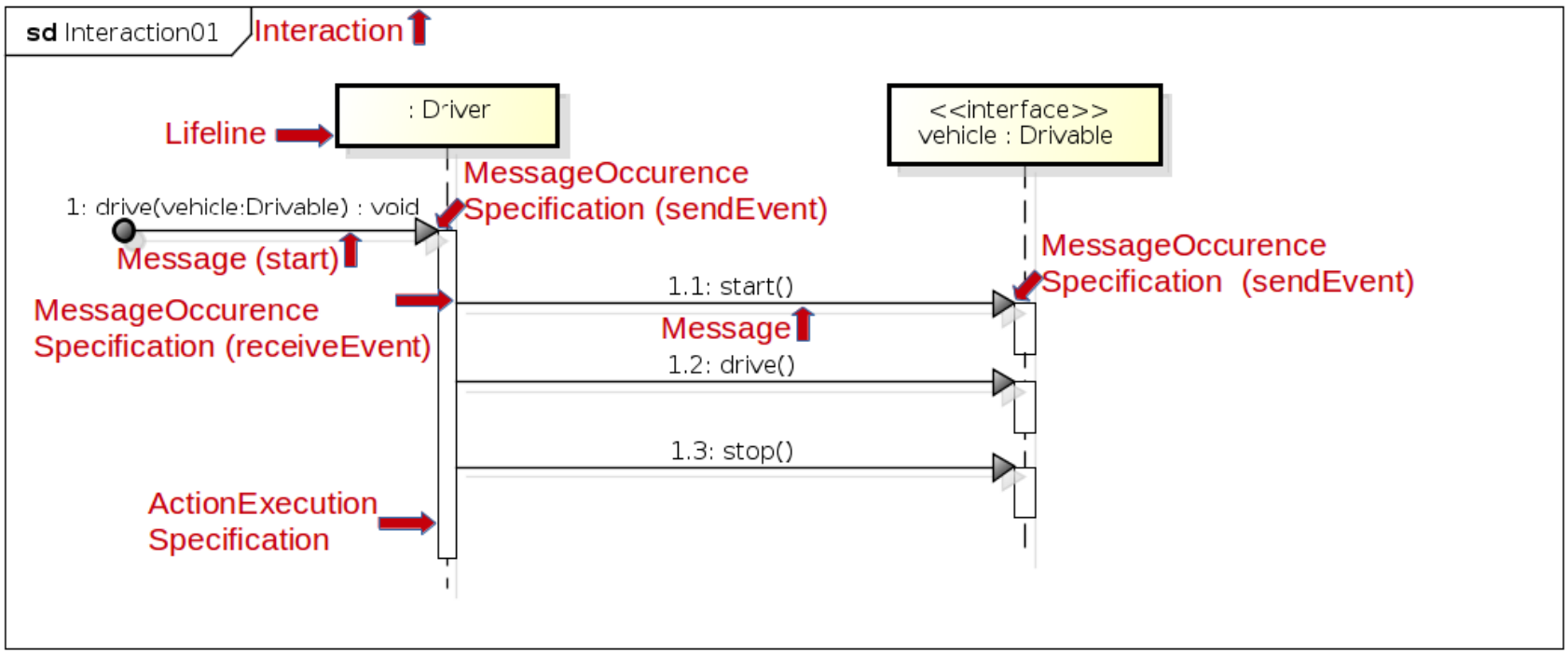
Some Metamodels of the Java Technological Space



UMLClassDiagram Concrete Syntax Example



UMLSequenceDiagram Concrete Syntax Example



UMLContract

- No concrete syntax defined
- Constraints (pre or postcondition or invariant) related to Properties or Operations
  - Opaque Expression (textual definition)
  - Interval

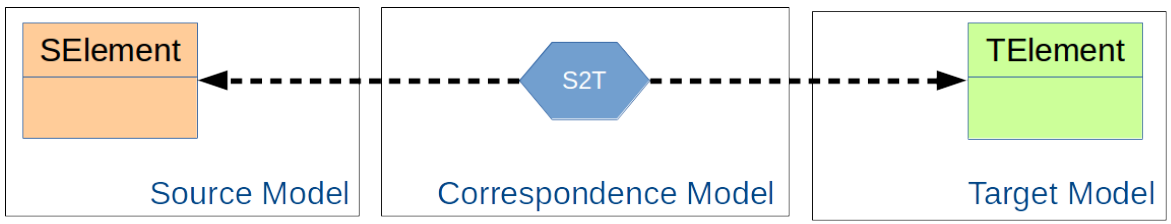
Java Concrete Syntax Example

```java
package main;

import de.silvawb.utils.*;

public class Driver extends Person {
    /*
     * Fields
     */
    @Inv(constraint = "driverLicense >= 1")
    public Integer driverLicense;
    public Integer driverLicenseStatus;
    private Integer experience = 0;

    /*
     * Methods
     */
    public void checkRep(){
        assert driverLicense >= 1;
    }
    public void driveCheckInvConstraint(Drivable vehicle){
        assert vehicle != null;
    }
    public void driveCheckPreConstraint(Drivable vehicle){
        assert driverLicenseStatus >= 1;
    }
    public void driveCheckPosConstraint(Drivable vehicle){
    }

    @Inv(constraint = "vehicle <> null")
    @Pre(constraint = "driverLicenseStatus >= 1")
    @Pos(constraint = "experience > experience@pre")
    @Interaction(interactionSequence = {
            "start","drive","stop",
    })
    public void drive(Drivable vehicle){
        checkRep();
        driveCheckInvConstraint(vehicle);
        driveCheckPreConstraint(vehicle);

        vehicle.start();
        vehicle.drive();
        vehicle.stop();

        checkRep();
        driveCheckInvConstraint(vehicle);
        driveCheckPosConstraint(vehicle);
    }
}
```

The Relations

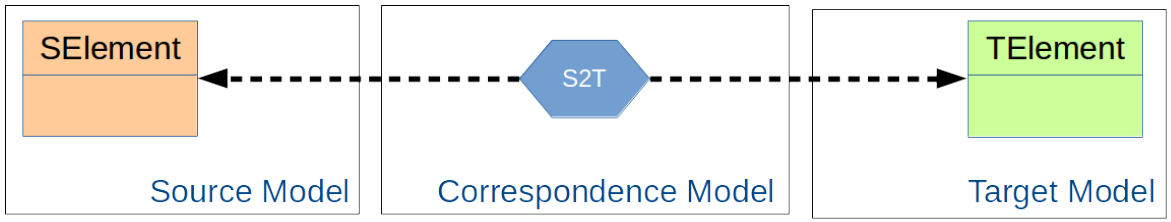
Triple Graph Grammar

- Relations coded by triple graphs

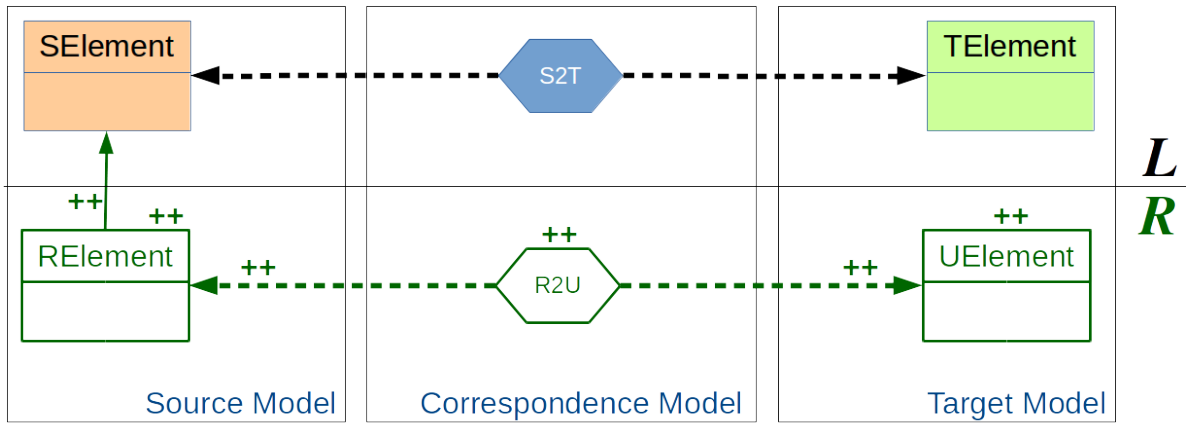


Triple Graph Grammar

- Relations coded by triple graphs

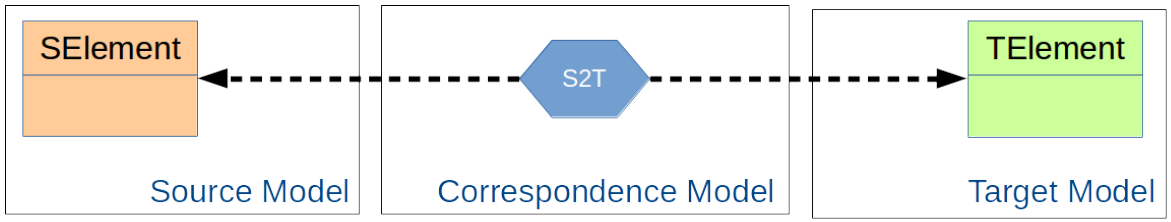


- Triple graphs are organized in triple rules $L \rightarrow R$.

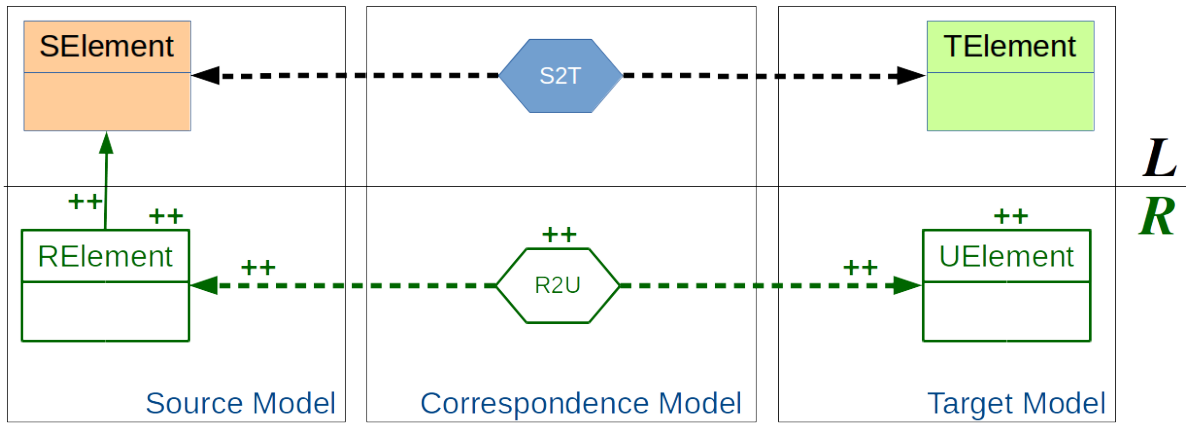


Triple Graph Grammar

- Relations coded by triple graphs

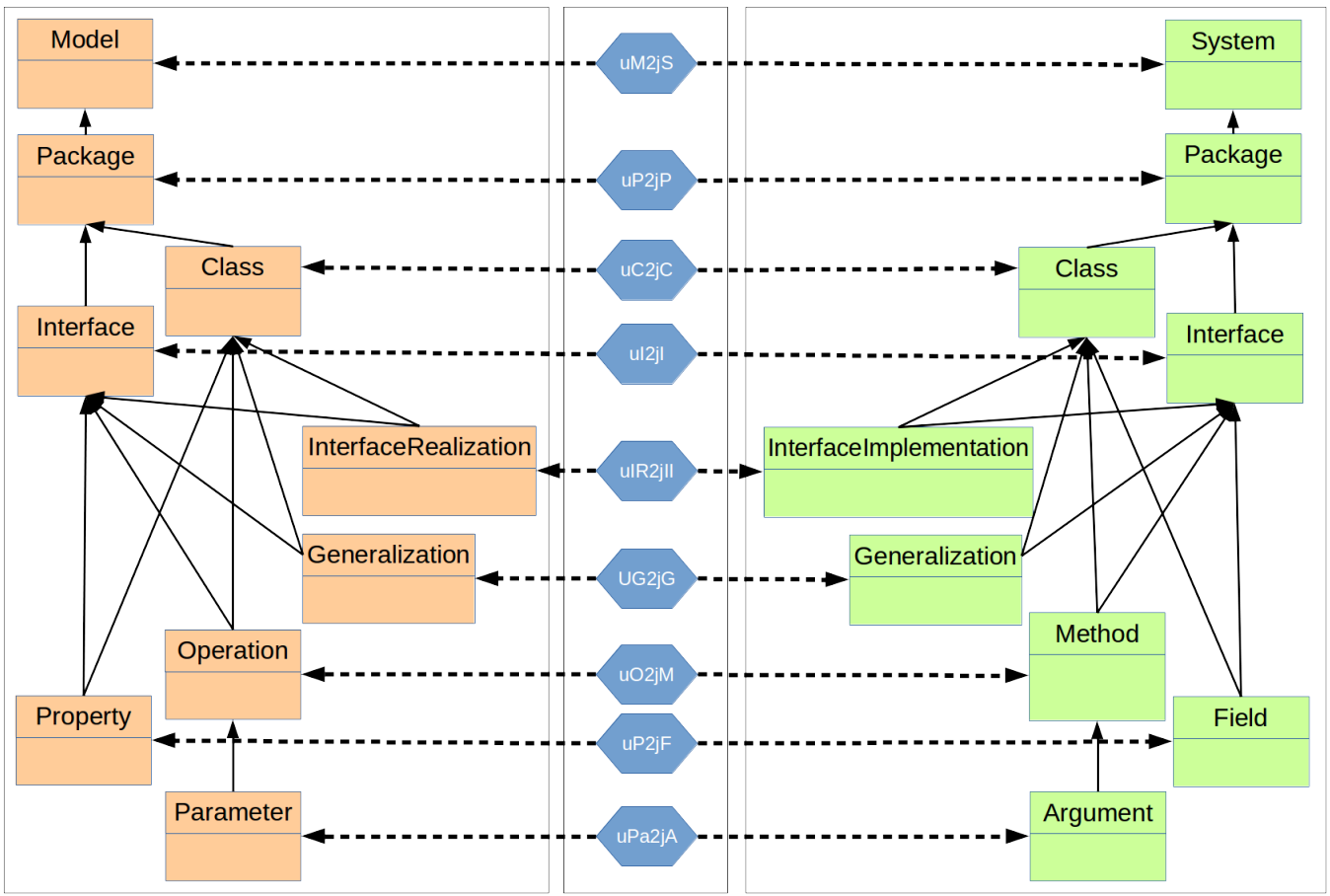


- Triple graphs are organized in triple rules $L \rightarrow R$.

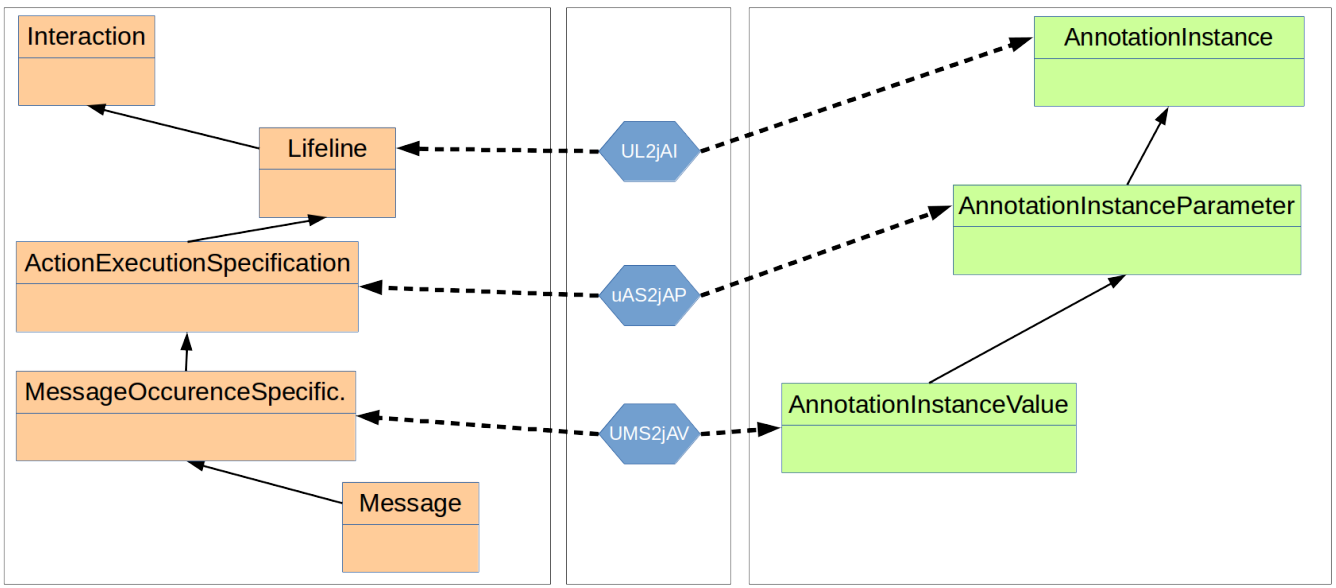


- These triple rules form a triple graph grammar (TGG) for each edge.

UMLClassDiagram2java



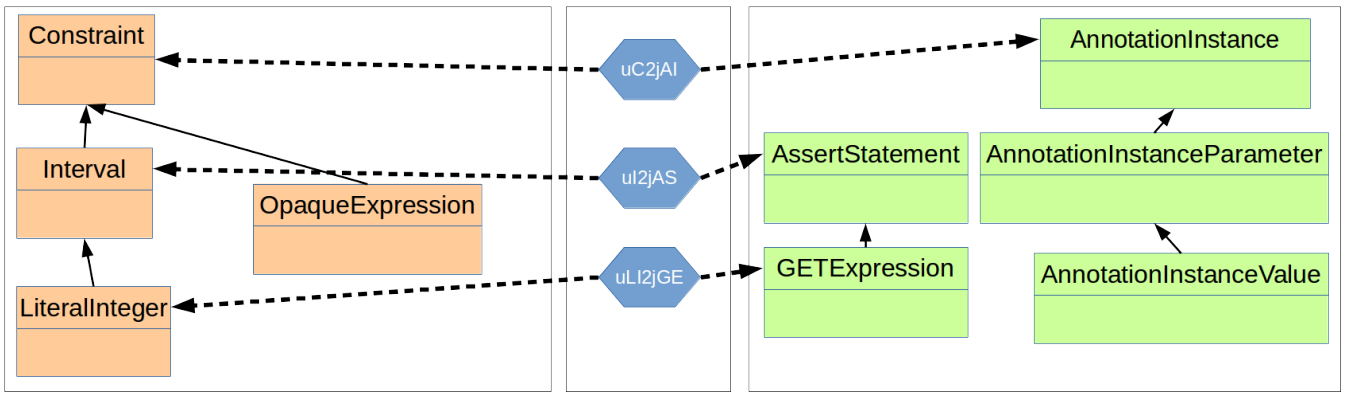
UMLSequenceDiagram2java



UMLSequenceDiagram2java Example

```java
package main;

import de.silvawb.utils.*;

public class Driver extends Person {
    /*
     * Fields
     */
    @Inv(constraint = "driverLicense >= 1")
    public Integer driverLicense;
    public Integer driverLicenseStatus;
    private Integer experience = 0;

    /*
     * Methods
     */
    public void checkRep(){
        assert driverLicense >= 1;
    }
    public void driveCheckInvConstraint(Drivable vehicle){
        assert vehicle != null;
    }
    public void driveCheckPreConstraint(Drivable vehicle){
        assert driverLicenseStatus >= 1;
    }
    public void driveCheckPosConstraint(Drivable vehicle){
    }

    @Inv(constraint = "vehicle <> null")
    @Pre(constraint = "driverLicenseStatus >= 1")
    @Pos(constraint = "experience > experience@pre")
    @Interaction(interactionSequence = {
            "start","drive","stop",
    })
    public void drive(Drivable vehicle){
        checkRep();
        driveCheckInvConstraint(vehicle);
        driveCheckPreConstraint(vehicle);

        vehicle.start();
        vehicle.drive();
        vehicle.stop();

        checkRep();
        driveCheckInvConstraint(vehicle);
        driveCheckPosConstraint(vehicle);
    }
}
```

UMLContract2java



UMLContract2java Example

```java
package main;

import de.silvawb.utils.*;

public class Driver extends Person {
    /*
     * Fields
     */
    @Inv(constraint = "driverLicense >= 1")
    public Integer driverLicense;
    public Integer driverLicenseStatus;
    private Integer experience = 0;

    /*
     * Methods
     */
    public void checkRep(){
        assert driverLicense >= 1;
    }
    public void driveCheckInvConstraint(Drivable vehicle){
        assert vehicle != null;
    }
    public void driveCheckPreConstraint(Drivable vehicle){
        assert driverLicenseStatus >= 1;
    }
    public void driveCheckPosConstraint(Drivable vehicle){
    }

    @Inv(constraint = "vehicle <> null")
    @Pre(constraint = "driverLicenseStatus >= 1")
    @Pos(constraint = "experience > experience@pre")
    @Interaction(interactionSequence = {
            "start","drive","stop",
    })
    public void drive(Drivable vehicle){
        checkRep();
        driveCheckInvConstraint(vehicle);
        driveCheckPreConstraint(vehicle);

        vehicle.start();
        vehicle.drive();
        vehicle.stop();

        checkRep();
        driveCheckInvConstraint(vehicle);
        driveCheckPosConstraint(vehicle);
    }
}
```

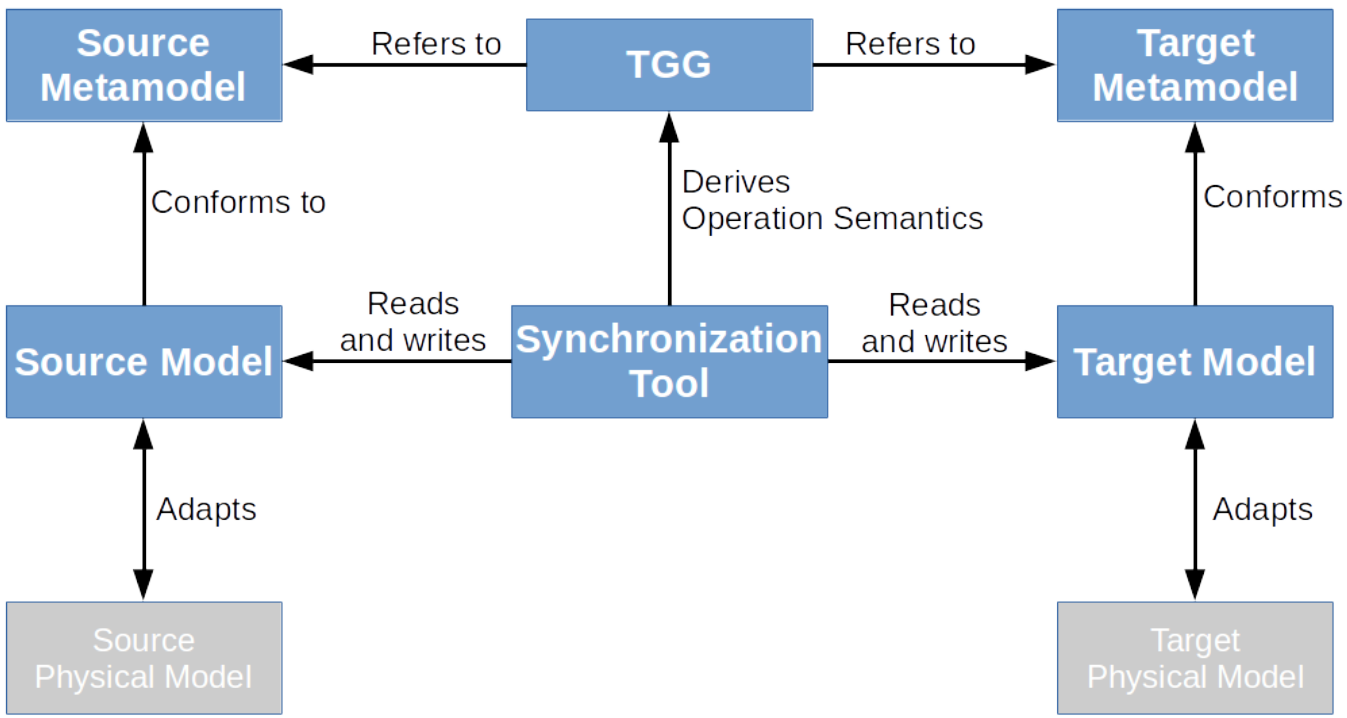
The Synchronization

Synchronization Scheme for Each TGG

- Following scheme for every edge of the network of metamodels

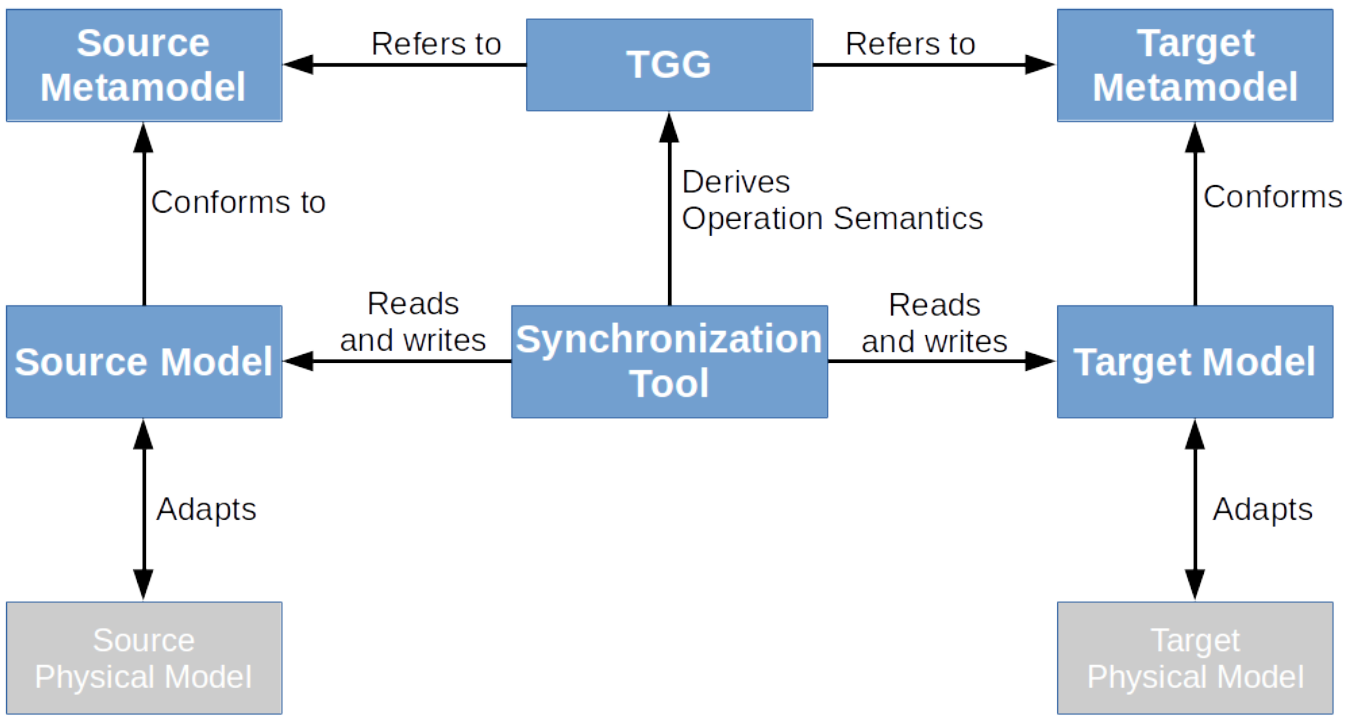
Synchronization Scheme for Each TGG

- Following scheme for every edge of the network of metamodels



Synchronization Scheme for Each TGG

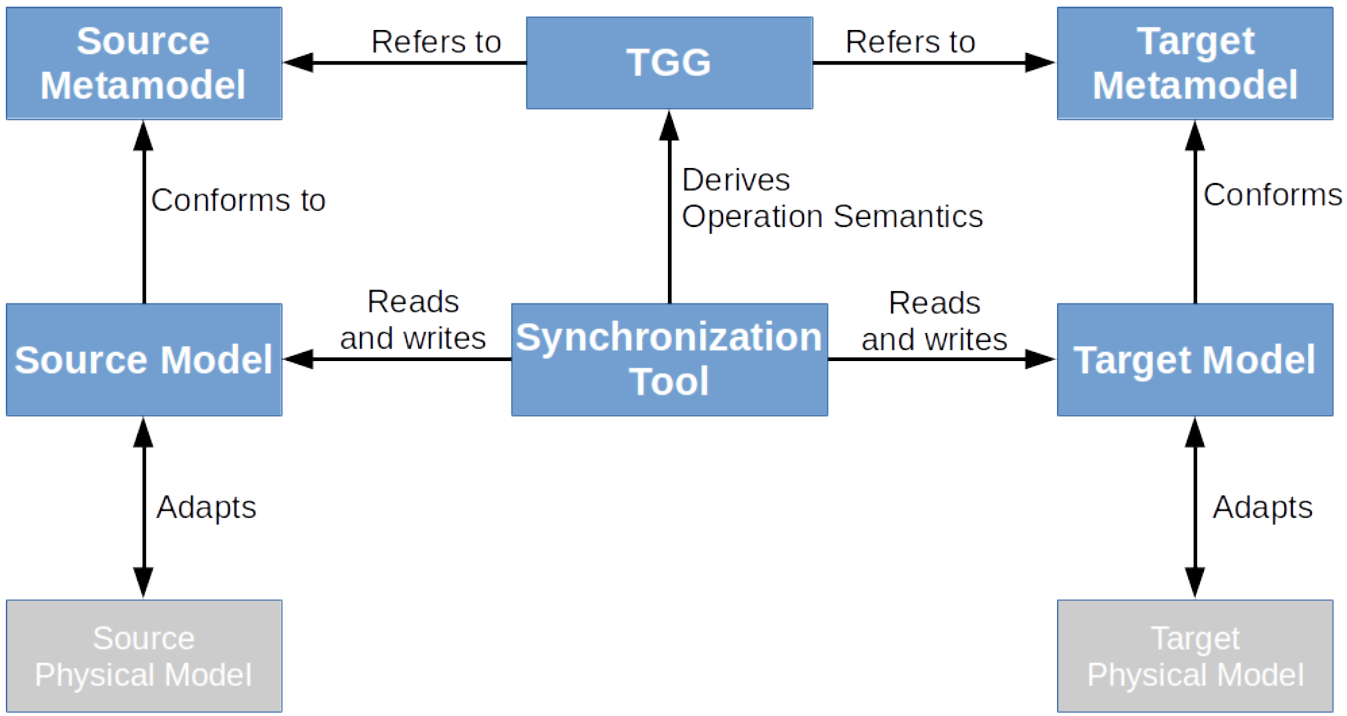
- Following scheme for every edge of the network of metamodels



- Treated separately by state-of-the-art approaches

Synchronization Scheme for Each TGG

- Following scheme for every edge of the network of metamodels



- Treated separately by state-of-the-art approaches
- How to treat the whole network of metamodels?

Synchronization Algorithm for the Network

function NETWORK SYNCHRONIZATION(G, v, v_{new}, δ_v)
    Update v to v_{new} in G
    for all $n_i = N(v)$ **do**
        Synchronize n_i according to v, v_{new} and δ_v
        if n_i was modified **then**
            Network Synchronization (G, n_i, $n_{i_{new}}$, δ_n)
        end if
    end for
    return G
end function

Synchronization Algorithm for the Network

function NETWORK SYNCHRONIZATION(G, v, v_{new}, δ_v)
    Update v to v_{new} in G
    for all $n_i = N(v)$ **do**
        Synchronize n_i according to v, v_{new} and δ_v
        if n_i was modified **then**
            Network Synchronization (G, n_i, $n_{i_{new}}$, δ_n)
        end if
    end for
    return G
end function

- Supposing only one modification at a time and unidirectional modifications.
- The algorithm always terminates (for G finite without cycles).
- The algorithm is deterministic (for deterministic synchronization).

Conclusion

Achieved goals

1. Metamodel definitions of artifacts from the Java Technological Space
   - Metamodels can be used in future works.
   - Metamodels are not complete.

Achieved goals

1. Metamodel definitions of artifacts from the Java Technological Space
   - Metamodels can be used in future works.
   - Metamodels are not complete.
2. Creation of a network of metamodels including the relations' formalizations
   - Exploration of TGGs for defining the relations
   - Evaluation of the definitions through forward transformation
   - Relations are not complete.

Achieved goals

1. Metamodel definitions of artifacts from the Java Technological Space
   - Metamodels can be used in future works.
   - Metamodels are not complete.
2. Creation of a network of metamodels including the relations' formalizations
   - Exploration of TGGs for defining the relations
   - Evaluation of the definitions through forward transformation
   - Relations are not complete.
3. The proposal of an algorithm for network synchronization
   - Novel view of the model synchronization problem
   - Algorithm has very limiting assumptions.

Thank you

Thank you for your attention

References

Krzysztof Czarnecki and Simon Helsen.

Feature-based survey of model transformation approaches.
IBM Systems Journal, 45(3):621–645, 2006.

Zinovy Diskin.

Model synchronization: Mappings, tiles, and categories.
In *Generative and Transformational Techniques in Software Engineering III*, pages 92–165. Springer, 2011.

Holger Giese, Stephan Hildebrandt, and Stefan Neumann.

Model synchronization at work: keeping sysml and autosar models consistent.
In *Graph transformations and model-driven engineering*, pages 555–579. Springer, 2010.

Florian Heidenreich, Jendrik Johannes, Mirko Seifert, and Christian Wende.

Jamopp: The java model parser and printer.
Technical report, Fakultät Informatik, Technological University of Dresden, Germany, 2009.

Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, and Yingfei Xiong.

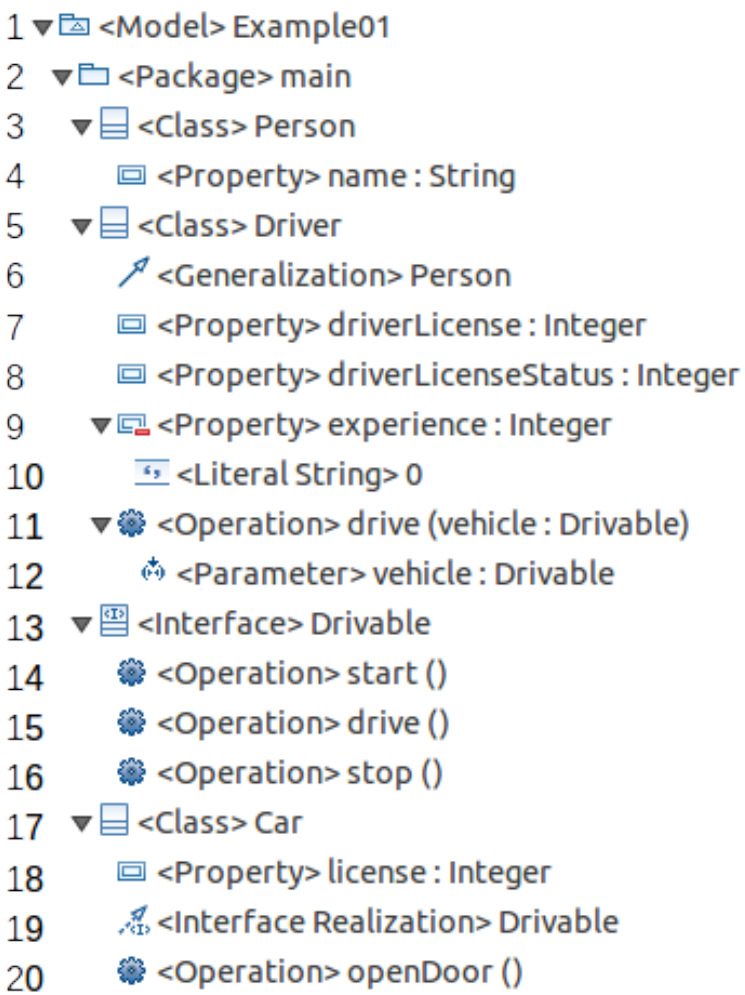
Correctness of model synchronization based on triple graph grammars.
In *Model Driven Engineering Languages and Systems*, pages 668–682. Springer, 2011.

OMG OMG.

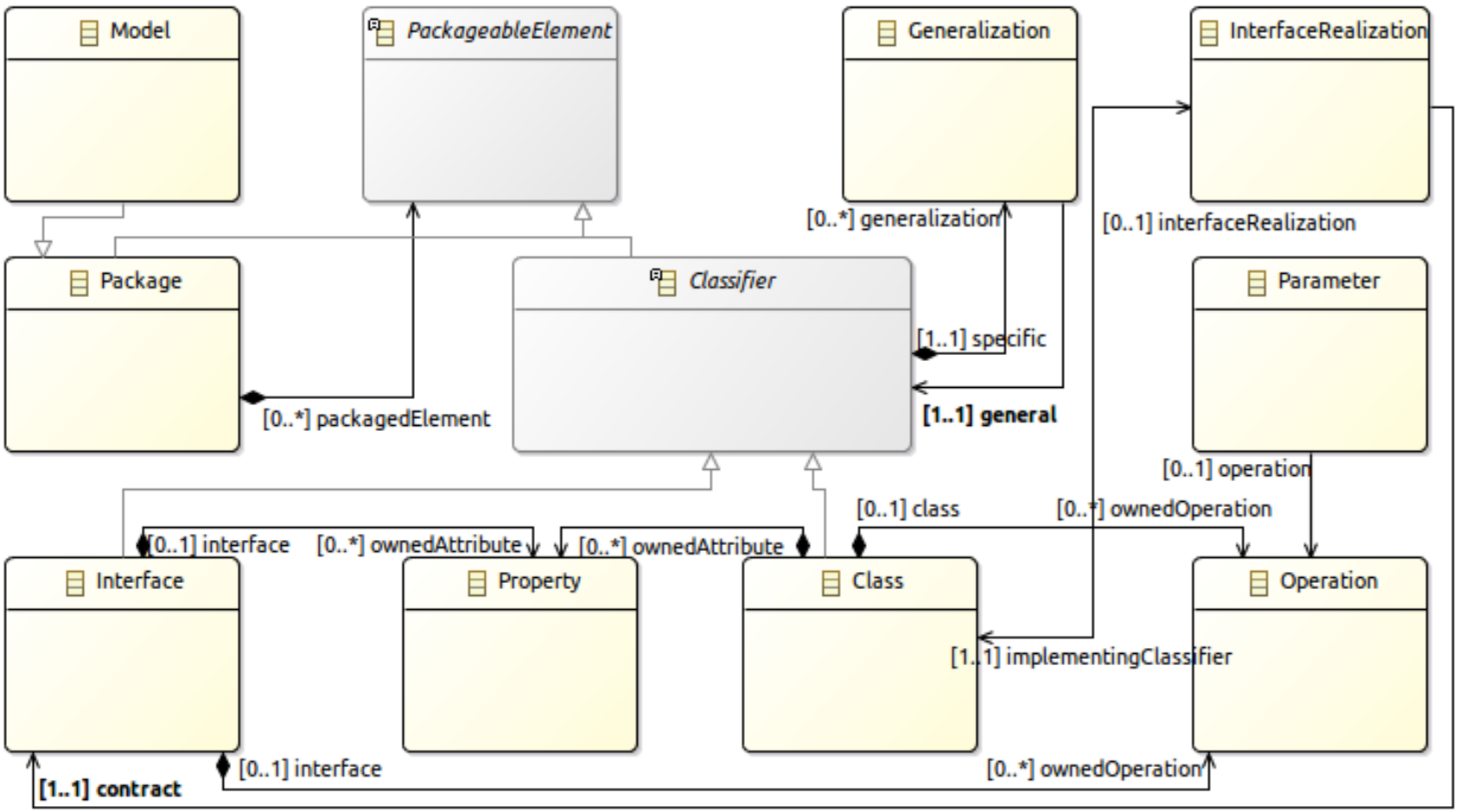
Unified modeling language (omg uml).
Superstructure, 2007.

Appendix

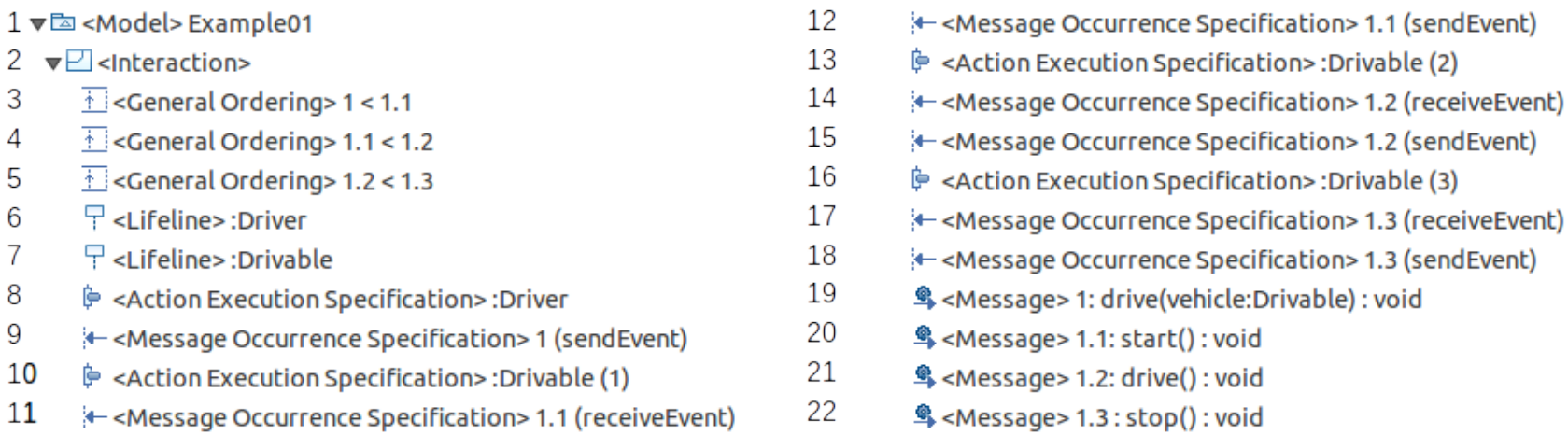
UMLClassDiagram Abstract Syntax Example



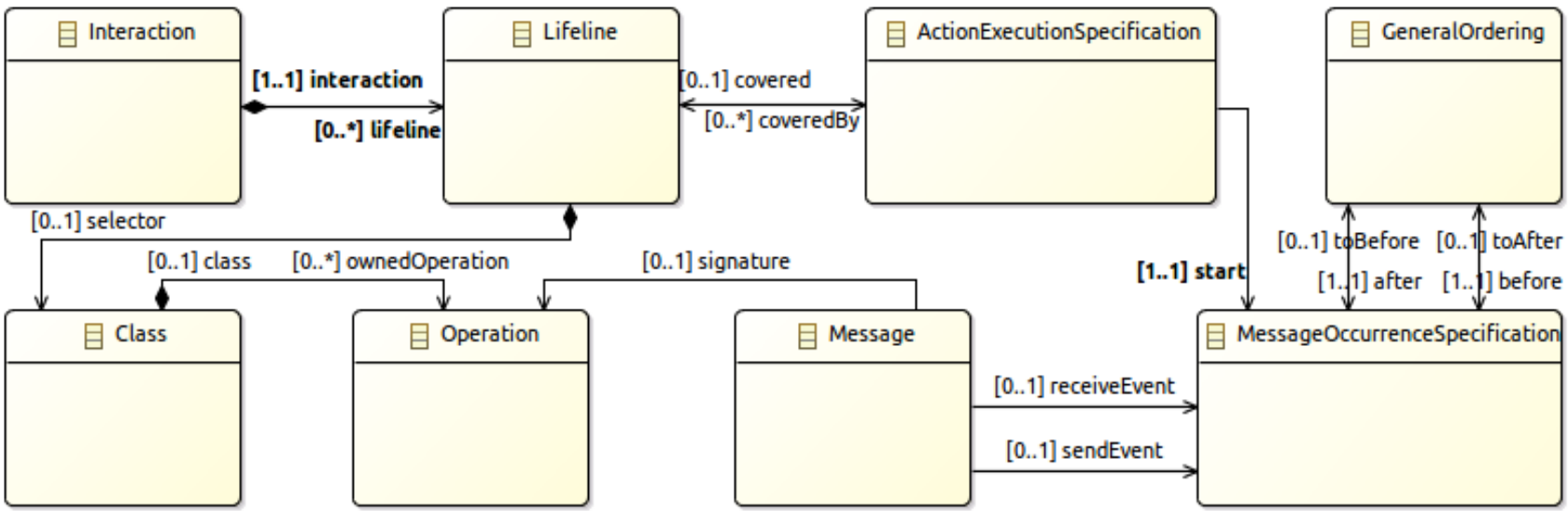
UMLClassDiagram Metamodel



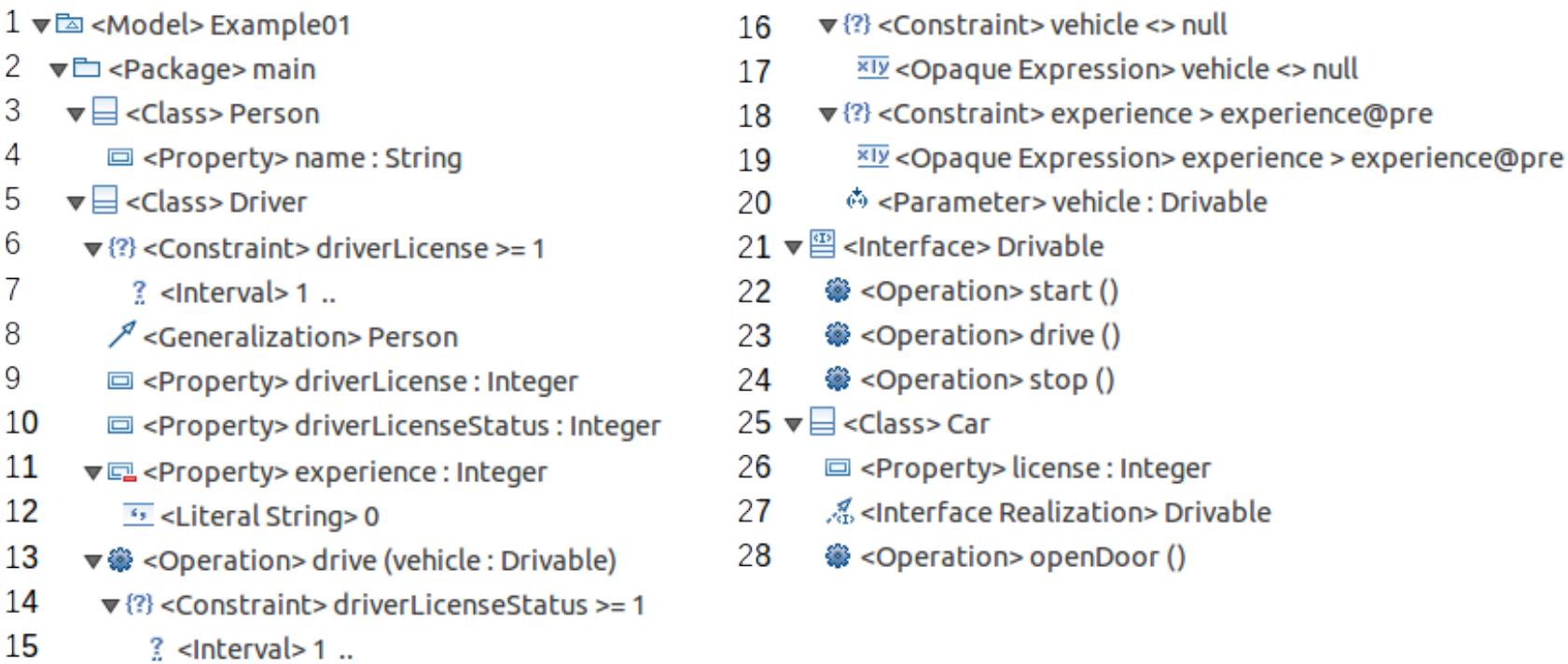
UMLSequenceDiagram Abstract Syntax Example



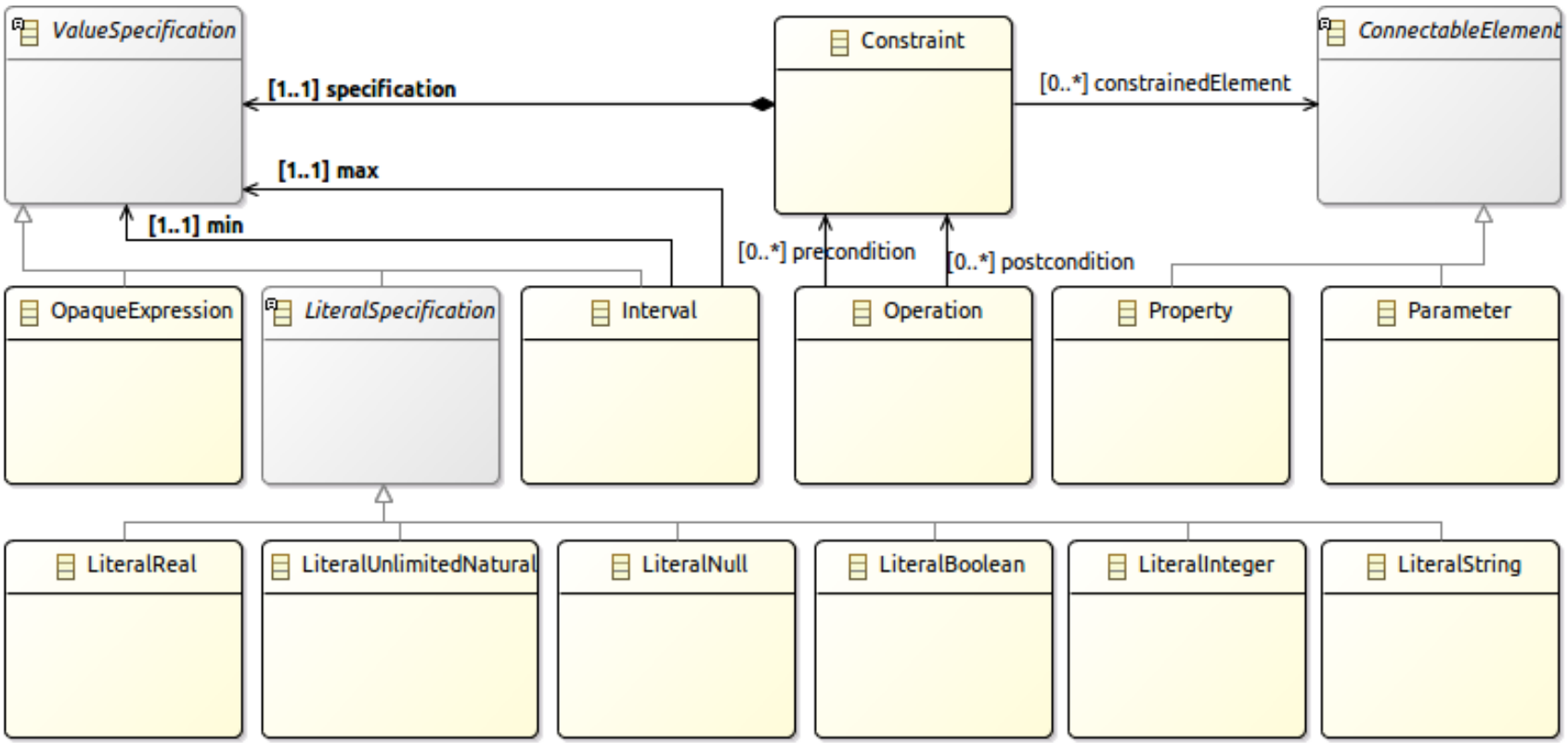
UMLSequenceDiagram Metamodel



UMLContract Abstract Syntax Example

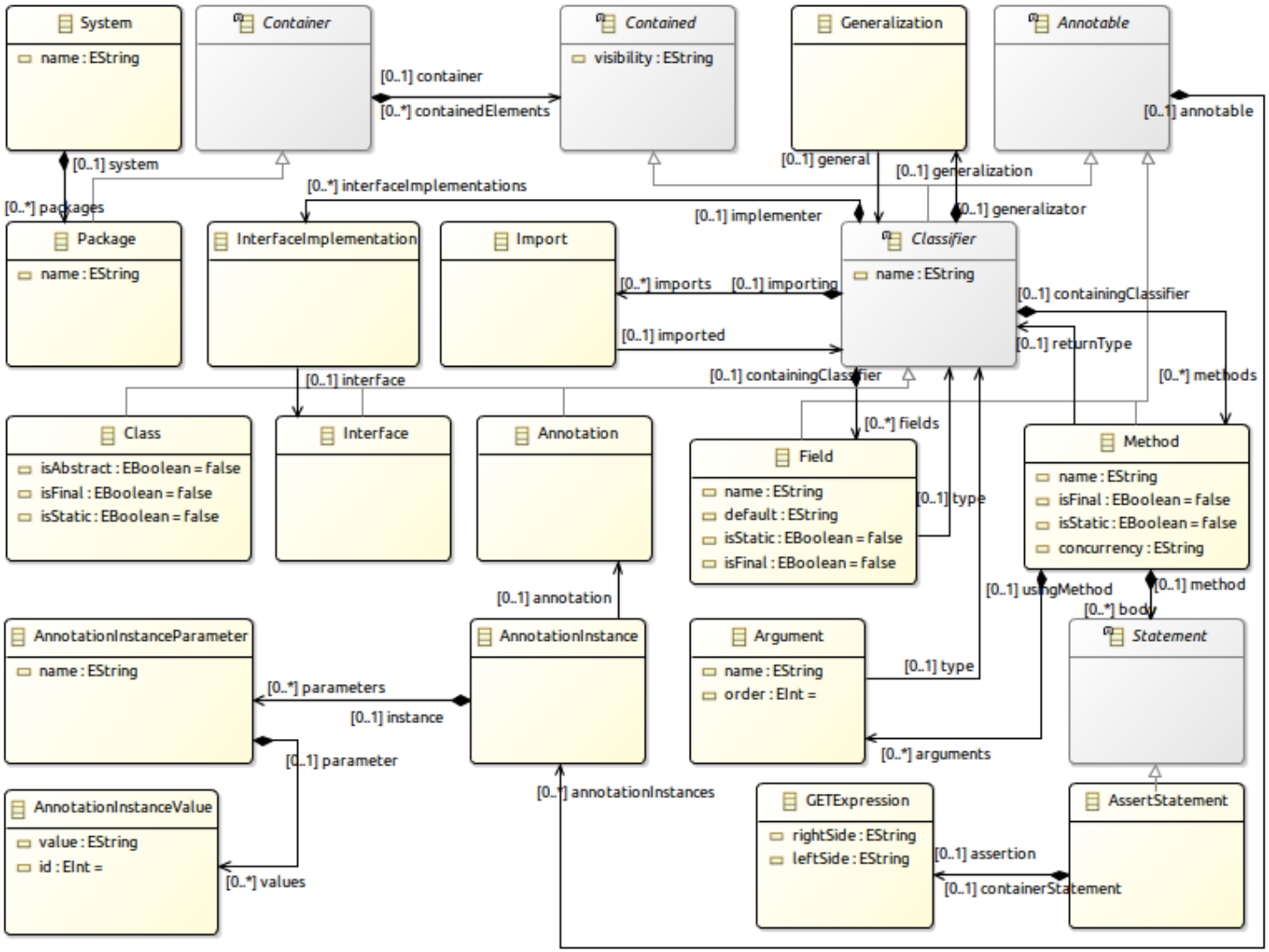


UMLContract Metamodel

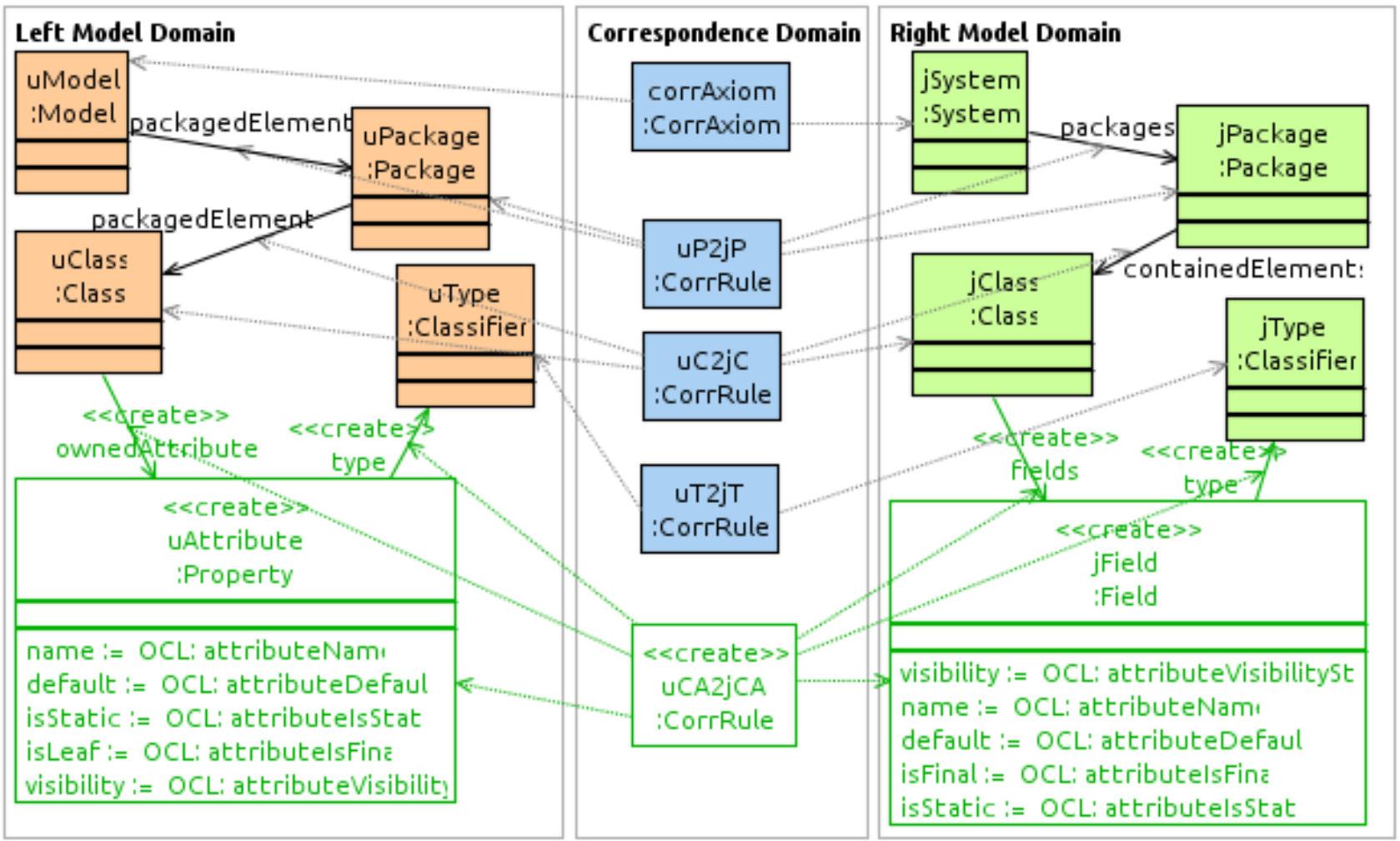


Java Abstract Syntax Example

Java Metamodel



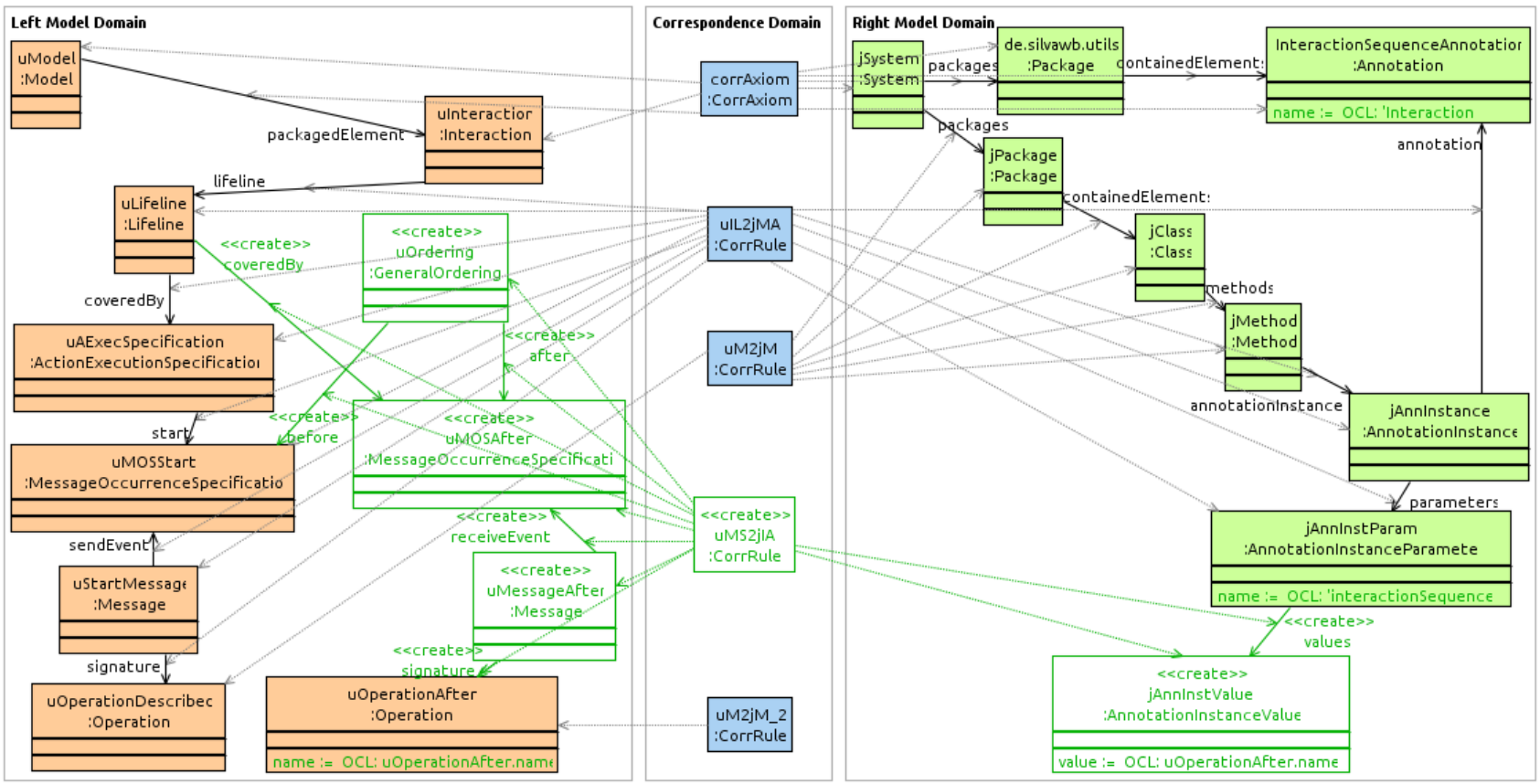
One triple rule for UMLClassDiagram2java



Result of the implementation for UMLClassDiagram2java

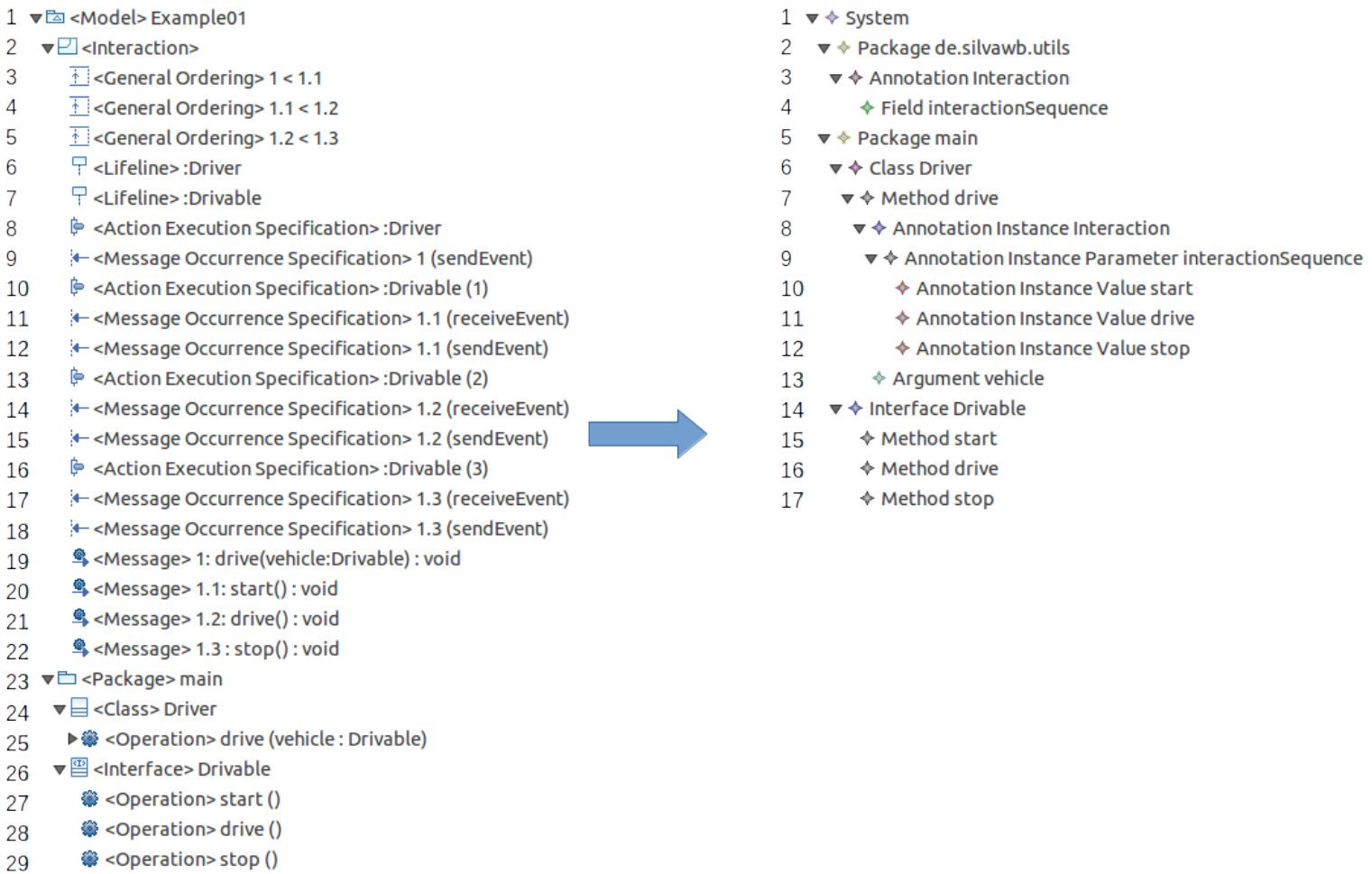
- Forward transformation was applied.

One triple rule for UMLSequenceDiagram2java

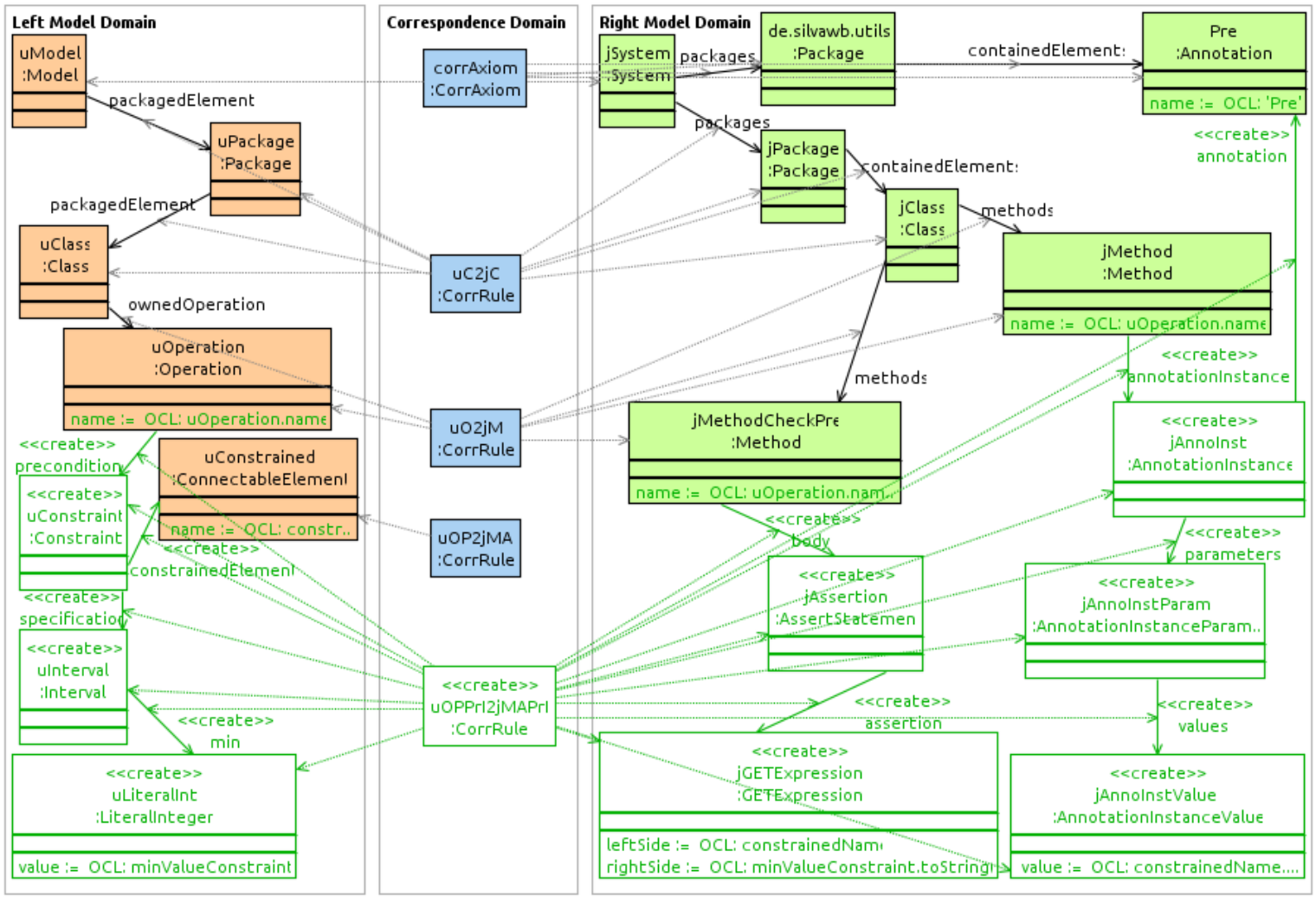


Implementation for UMLSequenceDiagram2java

- Forward transformation was applied.



One triple rule for UMLContract2java



Result of the Implementation for UMLContract2java

- Forward transformation was applied.