

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

1 Introduction

- Background
- Objective

2 Development

- The Metamodels
- The Relations
- The Synchronization

3 Conclusion

4 References

Background

1 Introduction

- Background
- Objective

2 Development

- The Metamodels
- The Relations
- The Synchronization

3 Conclusion

4 References

Models are used in Software Engineering

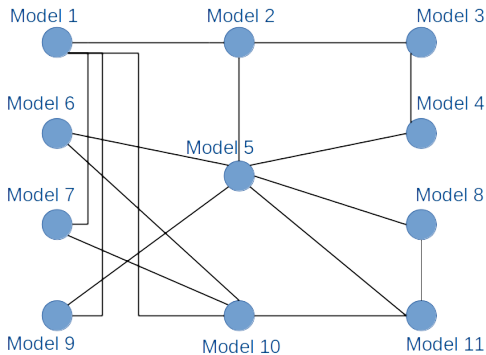


Models are used in Software Engineering

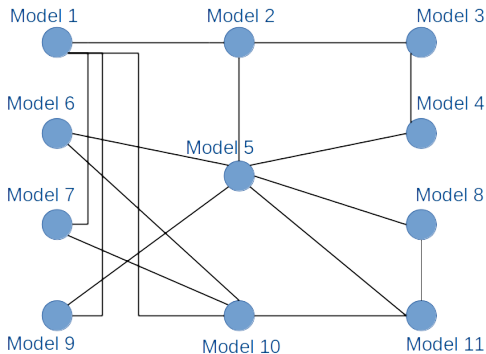


- New needs in industry thrill new methods and paradigms.
- **Model-driven Engineering (MDE):** Software processes are oriented to models.
- One software may have several different models.

Models have to be kept consistent



Models have to be kept consistent



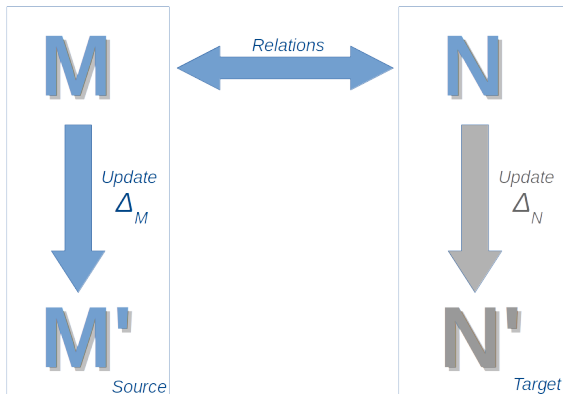
- Models are to be maintained consistent as they evolve.
- This means **models synchronization**.

Model Synchronization in the Network of Models

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

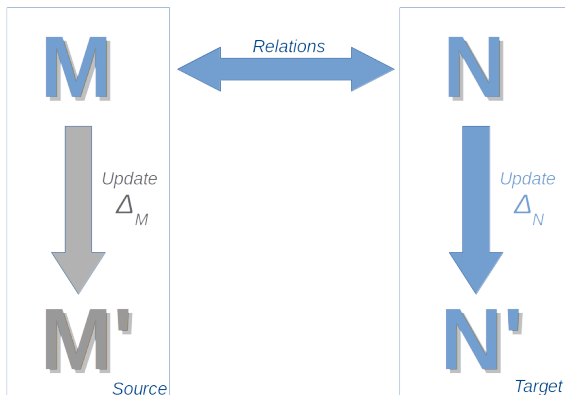
Model Synchronization in the Network of Models

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

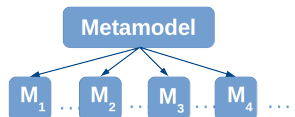


Model Synchronization in the Network of Models

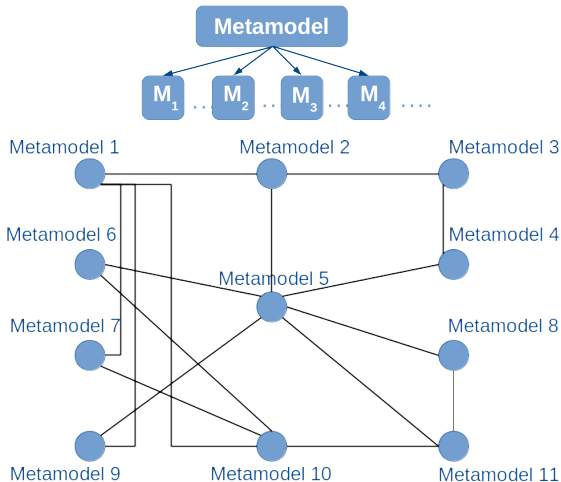
- For each edge of the network there is a synchronization task.



We work in the metamodel level



We work in the metamodel level



Objective

1 Introduction

■ Background

■ Objective

2 Development

■ The Metamodels

■ The Relations

■ The Synchronization

3 Conclusion

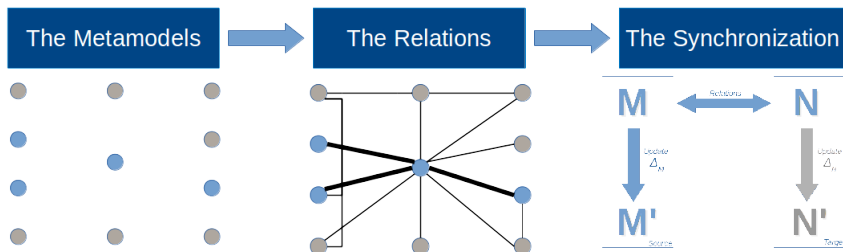
4 References

Three Steps

- Focus on the Java technological space.

Three Steps

- Focus on the Java technological space.



The Metamodels

1 Introduction

- Background
- Objective

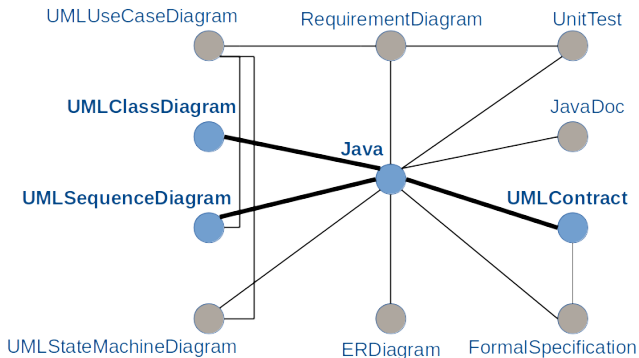
2 Development

- **The Metamodels**
- The Relations
- The Synchronization

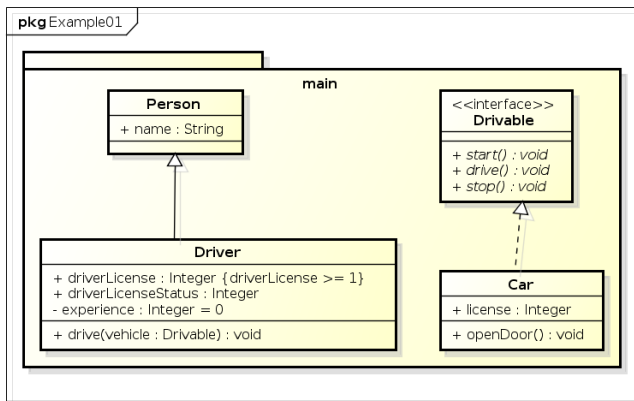
3 Conclusion

4 References

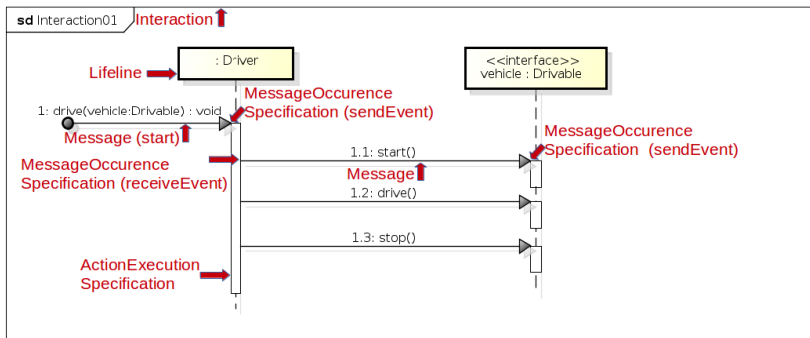
Some Metamodels of the Java Technological Space



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

```

1  package main;
2
3  import de.silvawb.utils.*;
4
5  public class Driver extends Person {
6      /*
7       * Fields
8       */
9      @Inv(constraint = "driverLicense >= 1")
10     public Integer driverLicense;
11     public Integer driverLicenseStatus;
12     private Integer experience = 0;
13
14     /*
15     * Methods
16     */
17     public void checkRep(){
18         assert driverLicense >= 1;
19     }
20     public void driveCheckInvConstraint(Drivable vehicle){
21         assert vehicle != null;
22     }
23     public void driveCheckPreConstraint(Drivable vehicle){
24         assert driverLicenseStatus >= 1;
25     }
26     public void driveCheckPosConstraint(Drivable vehicle){
27     }

```

```

28
29     @Inv(constraint = "vehicle <> null")
30     @Pre(constraint = "driverLicenseStatus >= 1")
31     @Pos(constraint = "experience > experience@pre")
32     @Interaction(interactionSequence = {
33         "start", "drive", "stop",
34     })
35     public void drive(Drivable vehicle){
36         checkRep();
37         driveCheckInvConstraint(vehicle);
38         driveCheckPreConstraint(vehicle);
39
40         vehicle.start();
41         vehicle.drive();
42         vehicle.stop();
43
44         checkRep();
45         driveCheckInvConstraint(vehicle);
46         driveCheckPosConstraint(vehicle);
47     }
48 }

```

The Relations

- 1 Introduction
 - Background
 - Objective
- 2 Development
 - The Metamodels
 - **The Relations**
 - The Synchronization
- 3 Conclusion
- 4 References

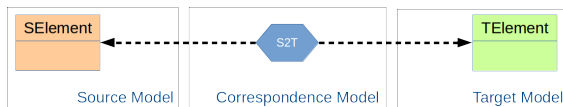
Triple Graphs

■ Relations coded by triple graphs



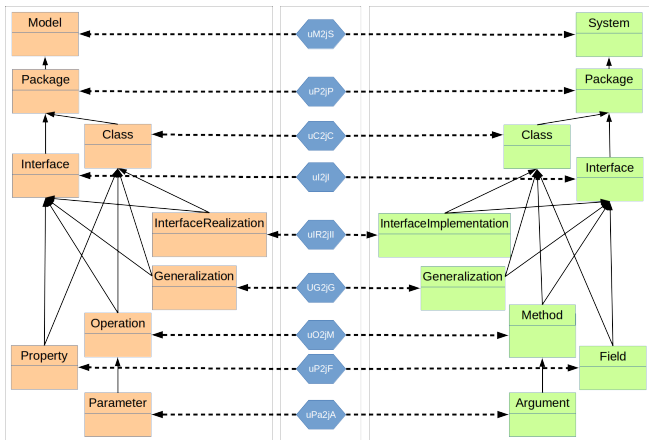
Triple Graphs

- Relations coded by triple graphs

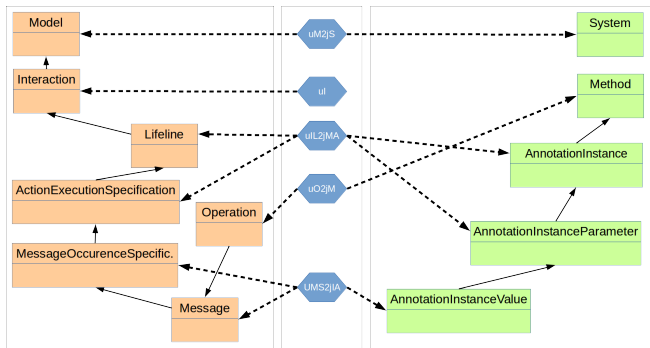


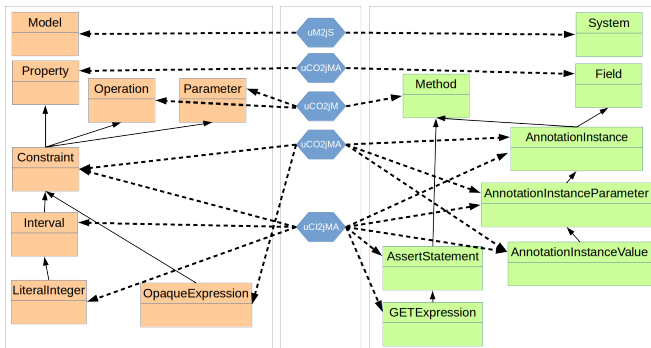
- Each edge of the network has a triple graphs grammar (TGG).
- A TGG is basically a grammar constructed upon triple graphs.

UMLClassDiagram2java



UMLSequenceDiagram2java





The Synchronization

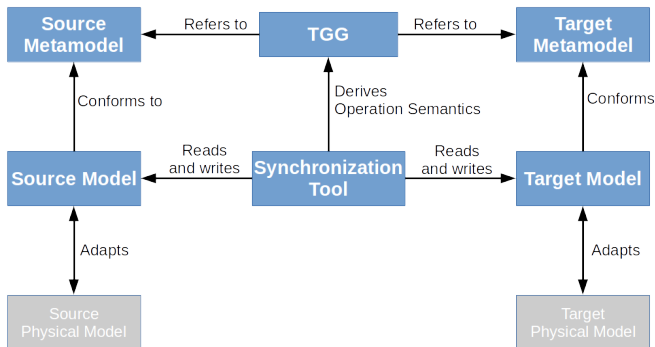
- 1 Introduction
 - Background
 - Objective
- 2 Development
 - The Metamodels
 - The Relations
 - The Synchronization
- 3 Conclusion
- 4 References

Synchronization Scheme for Each TGG

- Following scheme for every edge of the network of metamodels

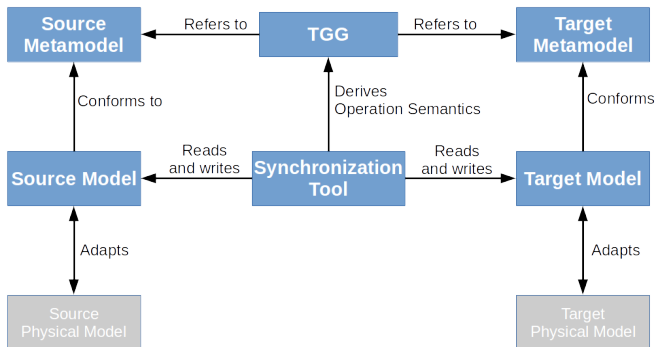
Synchronization Scheme for Each TGG

- Following scheme for every edge of the network of metamodels



Synchronization Scheme for Each TGG

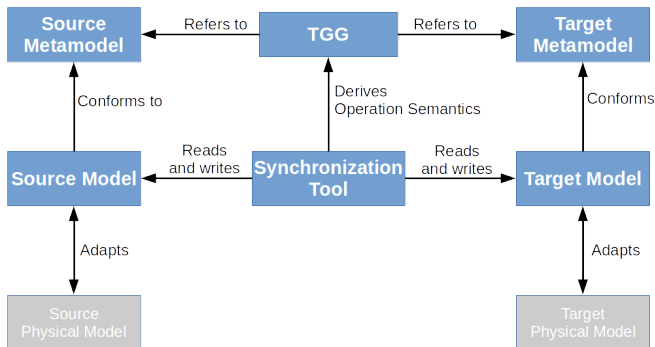
- Following scheme for every edge of the network of metamodels



- 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( $G, v, v_{new}, \delta_v$ )
    Update  $v$  to  $v_{new}$  in  $G$ 
    for all  $n_i = N(v)$  do
        Synchronize  $n_i$  according to  $v, v_{new}$  and  $\delta_v$ 
        if  $n_i$  was modified then
            Network Synchronization ( $G, n_i, n_{i_{new}}, \delta_n$ )
        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 ($G, n_i, n_{i_{new}}, \delta_n$)

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

1 Introduction

- Background
- Objective

2 Development

- The Metamodels
- The Relations
- The Synchronization

3 Conclusion

4 References

Achieved goals

- 1 Metamodel definitions of artifacts from the Java Technological Space
 - Contributive for the use with TGG
 - Simplifications and incompleteness

Achieved goals

- 1 Metamodel definitions of artifacts from the Java Technological Space
 - Contributive for the use with TGG
 - Simplifications and incompleteness
- 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
 - Simplifications

Achieved goals

- 1 Metamodel definitions of artifacts from the Java Technological Space
 - Contributive for the use with TGG
 - Simplifications and incompleteness
- 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
 - Simplifications
- 3 The proposal of an algorithm for network synchronization
 - Novel view of the model synchronization problem
 - 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.

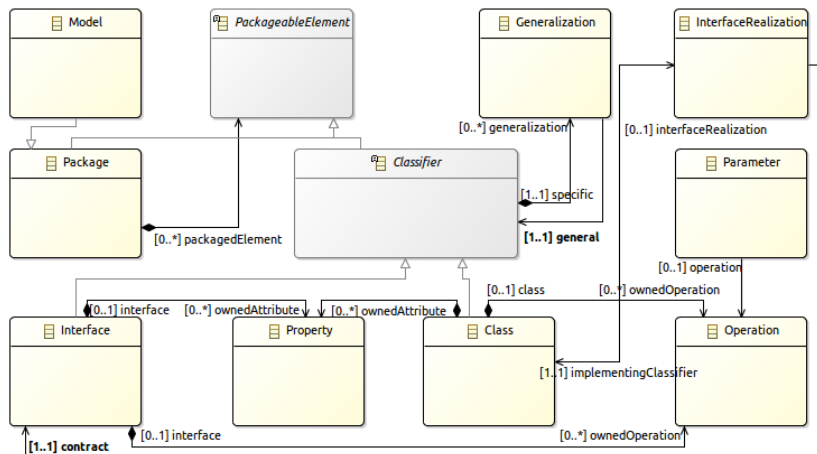
Unified modeling language (omg uml).
Superstructure, 2007.

Appendix

UMLClassDiagram Abstract Syntax Example

- 1 ▼  <Model> Example01
- 2 ▼  <Package> main
- 3 ▼  <Class> Person
- 4  <Property> name : String
- 5 ▼  <Class> Driver
- 6  <Generalization> Person
- 7  <Property> driverLicense : Integer
- 8  <Property> driverLicenseStatus : Integer
- 9 ▼  <Property> experience : Integer
- 10  <Literal String> 0
- 11 ▼  <Operation> drive (vehicle : Drivable)
- 12  <Parameter> vehicle : Drivable
- 13 ▼  <Interface> Drivable
- 14  <Operation> start ()
- 15  <Operation> drive ()
- 16  <Operation> stop ()
- 17 ▼  <Class> Car
- 18  <Property> license : Integer
- 19  <Interface Realization> Drivable
- 20  <Operation> openDoor ()

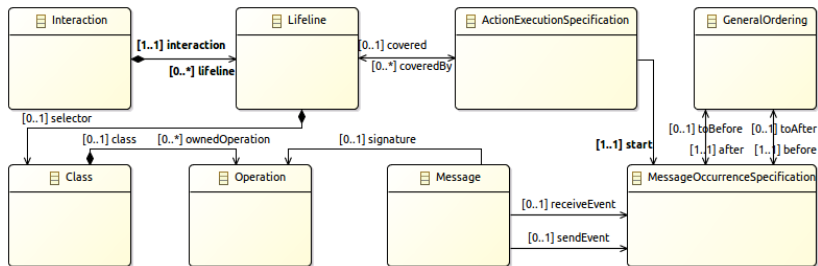
UMLClassDiagram Metamodel



UMLSequenceDiagram Abstract Syntax Example

1	▼ [Model] <Model> Example01	12	⚡ <Message Occurrence Specification> 1.1 (sendEvent)
2	▼ [Interaction] <Interaction>	13	🔗 <Action Execution Specification> :Drivable (2)
3	📄 <General Ordering> 1 < 1.1	14	⚡ <Message Occurrence Specification> 1.2 (receiveEvent)
4	📄 <General Ordering> 1.1 < 1.2	15	⚡ <Message Occurrence Specification> 1.2 (sendEvent)
5	📄 <General Ordering> 1.2 < 1.3	16	🔗 <Action Execution Specification> :Drivable (3)
6	👤 <Lifeline> :Driver	17	⚡ <Message Occurrence Specification> 1.3 (receiveEvent)
7	👤 <Lifeline> :Drivable	18	⚡ <Message Occurrence Specification> 1.3 (sendEvent)
8	🔗 <Action Execution Specification> :Driver	19	🗨️ <Message> 1: drive(vehicle:Drivable) : void
9	⚡ <Message Occurrence Specification> 1 (sendEvent)	20	🗨️ <Message> 1.1: start() : void
10	🔗 <Action Execution Specification> :Drivable (1)	21	🗨️ <Message> 1.2: drive() : void
11	⚡ <Message Occurrence Specification> 1.1 (receiveEvent)	22	🗨️ <Message> 1.3: stop() : void

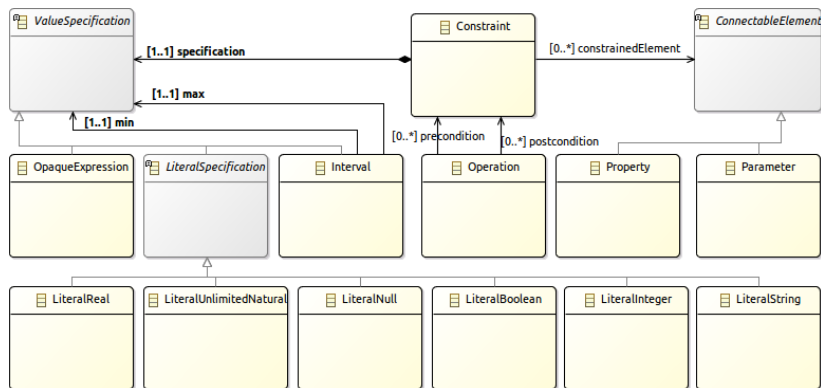
UMLSequenceDiagram Metamodel



UMLContract Abstract Syntax Example

```
1 ▼ [Model] <Model> Example01
2 ▼ [Package] <Package> main
3 ▼ [Class] <Class> Person
4   [Property] <Property> name : String
5 ▼ [Class] <Class> Driver
6   ▼ [Constraint] <Constraint> driverLicense >= 1
7     [Interval] ? <Interval> 1 ..
8   [Generalization] [Generalization] <Generalization> Person
9   [Property] [Property] <Property> driverLicense : Integer
10  [Property] [Property] <Property> driverLicenseStatus : Integer
11  ▼ [Property] <Property> experience : Integer
12    [Literal String] [Literal String] <Literal String> 0
13  ▼ [Operation] <Operation> drive (vehicle : Drivable)
14    ▼ [Constraint] <Constraint> driverLicenseStatus >= 1
15      [Interval] ? <Interval> 1 ..
16  ▼ [Constraint] <Constraint> vehicle <> null
17    [Opaque Expression] [Opaque Expression] <Opaque Expression> vehicle <> null
18  ▼ [Constraint] <Constraint> experience > experience@pre
19    [Opaque Expression] [Opaque Expression] <Opaque Expression> experience > experience@pre
20    [Parameter] [Parameter] <Parameter> vehicle : Drivable
21  ▼ [Interface] <Interface> Drivable
22    [Operation] [Operation] <Operation> start ()
23    [Operation] [Operation] <Operation> drive ()
24    [Operation] [Operation] <Operation> stop ()
25  ▼ [Class] <Class> Car
26    [Property] [Property] <Property> license : Integer
27    [Interface Realization] [Interface Realization] <Interface Realization> Drivable
28    [Operation] [Operation] <Operation> openDoor ()
```

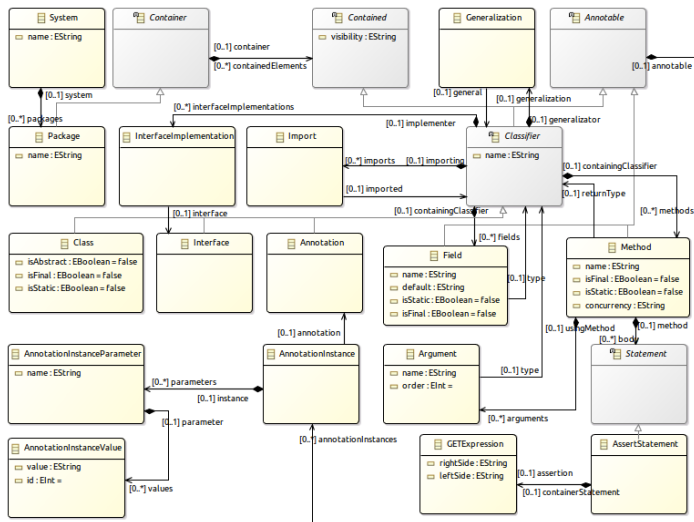
UMLContract Metamodel



Java Abstract Syntax Example

```
1 ▼ ◆ System Example01
2 ▼ ◆ Package main
3 ▼ ◆ Class Person
4   ◆ Field name
5 ▼ ◆ Class Driver
6   ▼ ◆ Field driverLicense
7     ▼ ◆ Annotation Instance Inv
8       ▼ ◆ Annotation Instance Parameter constraint
9         ◆ Annotation Instance Value driverLicense >= 1
10    ◆ Field driverLicenseStatus
11    ◆ Field experience
12 ▼ ◆ Method drive
13   ▼ ◆ Annotation Instance Inv
14     ▼ ◆ Annotation Instance Parameter constraint
15       ◆ Annotation Instance Value vehicle <= null
16   ▼ ◆ Annotation Instance Pre
17     ▼ ◆ Annotation Instance Parameter constraint
18       ◆ Annotation Instance Value driverLicenseStatus >= 1
19 ▼ ◆ Annotation Instance Pos
20   ▼ ◆ Annotation Instance Parameter constraint
21     ◆ Annotation Instance Value experience > experience@pre
22 ▼ ◆ Annotation Instance Interaction
23   ▼ ◆ Annotation Instance Parameter interactionSequence
24     ◆ Annotation Instance Value start
25     ◆ Annotation Instance Value drive
26     ◆ Annotation Instance Value stop
27   ◆ Argument vehicle
28 ▼ ◆ Method checkRep
29   ▼ ◆ Assert Statement driverLicense >= 1
30     ◆ GET Expression 1
31 ▼ ◆ Method driveCheckInvConstraint
32   ◆ Argument vehicle
33 ▼ ◆ Method driveCheckPreConstraint
34   ◆ Argument vehicle
35   ▼ ◆ Assert Statement driverLicenseStatus >= 1
36     ◆ GET Expression 1
37 ▼ ◆ Method driveCheckPosConstraint
38   ◆ Argument vehicle
39   ◆ Generalization Person
40   ◆ Import de.silvawb.utils.Inv
41   ◆ Import de.silvawb.utils.Pre
42   ◆ Import de.silvawb.utils.Pos
43   ◆ Import de.silvawb.utils.Interaction
44 ▼ ◆ Interface Drivable
45   ◆ Method start
46   ◆ Method drive
47   ◆ Method stop
48 ▼ ◆ Class Car
49   ◆ Field license
50   ◆ Method openDoor
51   ◆ Method start
52   ◆ Method drive
53   ◆ Method stop
54   ◆ Interface Implementation Drivable
```


Java Metamodel



Result of the implementation for UMLClassDiagram2java

- Forward transformation was applied.


```
1 ▼ <Model> Example01
2 ▼ <Package> main
3 ▼ <Interface> Drivable
4   <Operation> start ()
5   <Operation> drive ()
6   <Operation> stop ()
7 ▼ <Class> Car
8   <Property> license : Integer
9   <Interface Realization> Drivable
10  <Operation> openDoor ()
```



```
1 ▼ System Example01
2 ▼ Package main
3 ▼ Interface Drivable
4   Method start
5   Method drive
6   Method stop
7 ▼ Class Car
8   Field license
9   Method openDoor
10  Interface Implementation Drivable
```

Implementation for UMLSequenceDiagram2java

■ Forward transformation was applied.



```
1 ▼ <Model> Example01
2 ▼ <Interaction>
3   <General Ordering> 1 < 1.1
4   <General Ordering> 1.1 < 1.2
5   <General Ordering> 1.2 < 1.3
6   <Lifeline> :Driver
7   <Lifeline> :Driver
8   <Action Execution Specification> :Driver
9   <Message Occurrence Specification> 1 (sendEvent)
10  <Action Execution Specification> :Drivable (1)
11  <Message Occurrence Specification> 1.1 (receiveEvent)
12  <Message Occurrence Specification> 1.1 (sendEvent)
13  <Action Execution Specification> :Drivable (2)
14  <Message Occurrence Specification> 1.2 (receiveEvent)
15  <Message Occurrence Specification> 1.2 (sendEvent)
16  <Action Execution Specification> :Drivable (3)
17  <Message Occurrence Specification> 1.3 (receiveEvent)
18  <Message Occurrence Specification> 1.3 (sendEvent)
19  <Message> 1: drive(vehicle:Drivable) : void
20  <Message> 1.1: start() : void
21  <Message> 1.2: drive() : void
22  <Message> 1.3: stop() : void
23 ▼ <Package> main
24 ▼ <Class> Driver
25   <Operation> drive (vehicle : Drivable)
26 ▼ <Interface> Drivable
27   <Operation> start ()
28   <Operation> drive ()
29   <Operation> stop ()
```

```
1 ▼ ♦ System
2 ▼ ♦ Package de.silvawb.utils
3   ▼ ♦ Annotation Interaction
4     ♦ Field interactionSequence
5 ▼ ♦ Package main
6   ▼ ♦ Class Driver
7     ▼ ♦ Method drive
8       ▼ ♦ Annotation Instance Interaction
9         ▼ ♦ Annotation Instance Parameter InteractionSequence
10          ♦ Annotation Instance Value start
11          ♦ Annotation Instance Value drive
12          ♦ Annotation Instance Value stop
13        ♦ Argument vehicle
14 ▼ ♦ Interface Drivable
15   ♦ Method start
16   ♦ Method drive
17   ♦ Method stop
```

Result of the Implementation for UMLContract2java

■ Forward transformation was applied.

```
1 ▼ [Model] Example01
2 ▼ [Package] main
3 ▼ [Class] Driver
4   ▼ [Constraint] driverLicense >= 1
5     [Interval] 1 ..
6   [Property] driverLicense : Integer
7   [Property] driverLicenseStatus : Integer
8   [Property] experience : Integer
9   [Operation] drive (vehicle : Drivable)
10  ▼ [Constraint] driverLicenseStatus >= 1
11    [Interval] 1 ..
12  ▼ [Constraint] vehicle <= null
13    [Opaque Expression] vehicle <= null
14  ▼ [Constraint] experience > experience@pre
15    [Opaque Expression] experience > experience@pre
16  [Parameter] vehicle : Drivable
17 ▼ [Interface] Drivable
18   [Operation] start ()
19   [Operation] drive ()
20   [Operation] stop ()
```



```
1 ▼ System
2 ▼ Package de.silvawb.utils
3   Annotation Inv
4   Annotation Pos
5   Annotation Pre
6 ▼ Package main
7   Class Drive
8   Field driverLicense
9   Annotation Instance Inv
10  Annotation Instance Parameter
11    Annotation Instance Value driverLicense >= 1
12  Field driverLicenseStatus
13  Field experience
14  Method checkRep
15  Assert Statement driverLicense >= 1
16    GET Expression 1
17  Method driveCheckPreConstraint
18    Argument vehicle
```

```
19  Assert Statement driverLicenseStatus >= 1
20    GET Expression 1
21  Method driveCheckPosConstraint
22    Argument vehicle
23  Method drive
24  Annotation Instance Pos
25  Annotation Instance Parameter
26    Annotation Instance Value experience > experience@pre
27  Annotation Instance Pre
28  Annotation Instance Parameter
29    Annotation Instance Value driverLicenseStatus >= 1
30  Annotation Instance Inv
31  Annotation Instance Parameter
32    Annotation Instance Value vehicle <= null
33  Argument vehicle
34  Method driveCheckInvConstraint
35    Argument vehicle
```