

D4.2 Synchronization of blended notations

BUMBLE

Blended Modelling for Enhanced Software and Systems Engineering



Project Acronyms

<ACR>	<Acronyms>
BUMBLE	Blended Modelling for Enhanced Software and Systems Engineering
DSML	Domain-Specific Modeling Language
UML	Unified Modeling Language
EMF	Eclipse Modeling Framework
UML-RT	UML for Real-time
EBNF	Extended Backus-Naur Form
XML	eXtensible Markup Language
Alf	Action language for foundational UML
ETL	Epsilon Transformation Language
MML	Mapping modeling Language
GMF	Graphical Modeling Framework
PSS	Portable test and Stimulus Standard
JAVACC	Java Compiler Compiler
HOT	Higher order transformation
DMA	Direct Memory Access
SM	State Machine
MOF	Meta-Object Facility
Papyrus-RT	Papyrus for Real-Time
QVT	Query/View/Transformation
QVTo	QVT operational

Table of contents

1.	Introduction	5
2.	State of the art on synchronization and HOTS	5
3.	Synchronization solutions	6
3.1.	Synchronization for Portable test and Stimulus Standard (UC1)	6
	Input artefacts	7
	Generation of graphical and textual syntaxes	8
	Mapping and Synchronization	9
3.2.	Synchronization for UML-RT State Machines (UC1, UC2)	13
3.3.	Synchronization between system and software artefacts	15
3.4.	Synchronization with DClare	17
4.	HOTS for bidirectional synchronization	18
5.	Next steps	20
	References	21
	Appendix 1 - UML-RT metamodels	22

1. Introduction

In this deliverable we describe the core activities and results of task T4.3 in Work Package 4 (WP4). More specifically, the theories defined in T4.1 and T4.2 in terms of mapping between DSMLs are exploited to drive higher order transformations (HOTs), which are designed and implemented in T4.3, to satisfy the core requirements BC1, BC4, BC9 (as described in D2.2).

HOTs are powerful transformations defined at the metamodeling level [TIS09]. Given two DSMLs and a mapping model between them, HOTs are able to automatically generate bidirectional model transformations for synchronizing models conforming to the two DSMLs. Specific HOTs will realize the mapping rules defined in the previous tasks of the BUMBLE project (see D4.1).

The usage of HOTs entails at least three scenarios:

- Synchronization across different languages (either same or different notations)
- Synchronization across different notations of the same language
- Co-evolution mechanisms for models conforming to an evolving language

Except for specific synchronization with the transformation tool DClare (see Section 3.4) that also applies to the MPS environment, the rest of this deliverable concerns synchronization in the context of the Eclipse Modeling Framework (EMF), since the projectional nature of MPS requires a different kind of synchronization mechanisms.

The remainder of this deliverable is structured as follows. In Section 2 we provide an overview of the state of the art in synchronization including HOTs. In Section 3 we describe the actions and results in relation to the solutions achieved in this WP for synchronization across notations as well as across different modeling artefacts. All solutions have successfully been implemented (or are being implemented) in industrial use-cases in BUMBLE. In Section 4 we introduce the usage scenarios and the intended solutions for HOTs together with the selected technologies and the motivation behind those choices. We conclude the deliverable by outlining the next steps in Section 5.

2. State of the art on synchronization and HOTs

Very little has been done in combined means for synchronised editing in both textual and graphical syntaxes and customisation of the concrete syntaxes for MOF-based languages (including Ecore and UML).

Synchronised editing in multiple (customisable) concrete syntaxes for non-MOF DSMLs can also be realised with Qt (<https://www.qt.io/>) by using a model-view architecture where the 'model' reflects the DSML concepts and any 'view' is actually an editor for some concrete syntax. However, Qt has no support for DSML-based automation (e.g., autocompletion, validation), which means that a DSML engineer needs to program most of it manually (although well supported by IDE tools).

Several research efforts have been directed to mixing textual and graphical modeling. A textual editor for the Action Language for Foundational UML (Alf) has been developed based on Xtext [LAZ11].

In [AND07], the authors provide an approach for defining combined textual and graphical DSMLs based on the AToM3 tool. Starting from a metamodel definition, different diagram types can be assigned to different parts of the metamodel. A graphical concrete syntax is assigned by default, while a textual one can be given by providing triple graph grammar rules to map it to its corresponding portion of the common metamodel.

Charfi et al. [CHA09] explore the possibilities to define a single concrete syntax supporting both graphical and textual notations. Synchronization is provided only for a very small subset of UML while we focus on generation of synchronization mechanisms from any Ecore-based DSML.

In [SCH08], the authors provide the needed steps for embedding generated EMF-based textual model editors into graphical editors defined in terms of the Eclipse Graphical Modeling Framework (GMF). That approach provides pop-up boxes to textually edit elements of graphical models rather than allowing seamless editing of the entire model using a chosen syntax (i.e., blending). The focus of their solution is on the integration of editors based on EMF, while the aim in BUMBLE, and specifically T4.3, is to generate automatically synchronization mechanisms across existing Ecore-based textual and graphical notations. Moreover, the change propagation mechanisms proposed by the authors are on-demand triggered by the modeller's commit, while in BUMBLE we focus on on-the-fly change propagation across the modeling views also.

Related to the switching between graphical and textual syntaxes, two approaches are proposed to ease transformations of models containing both graphical and textual elements. The first is Grammarware [WIM05], by which a model is exported as text. The second is Modelware [WIM05], by which a model containing graphical and textual content is transformed into a fully graphical model. Transformation from mixed models to either text or graphics is on demand rather than on-the-fly and the approach does not allow concurrent editing. Mixed textual and graphical modeling can also be realised with Qt, where the approach is to use a graphical environment with embedded textual editors. However, DSML engineers would need to realise most of this manually. Mixed notations and the possibility to switch between them are supported in JetBrains MPS for non-MOF DSMLs and rely on the principle of projectional editing.

There exist approaches for synchronising graphical and textual models via semi-automated mechanisms in the form of synchronisation model transformations. These model transformations are, in some approaches, also generated, thanks to HOTS [MAR15], which are though specific to the DSML at hand. The aim of T4.3, and the work described in this deliverable, is to provide cross-DSML HOTS for any Ecore-based DSML.

3. Synchronization solutions

In this section we describe the solutions achieved in this WP for synchronization across notations as well as across different modeling artefacts. Note that the transformations described in this section are not generated by HOTS, but they were manually written to gather a variegated set of blueprints for the HOTS. All solutions have successfully been implemented (or are being implemented) in industrial use-cases in BUMBLE (UC1, UC2, UC6, described in D2.1).

3.1. Synchronization for Portable test and Stimulus Standard (UC1)

In this section we describe a generic solution to support seamless runtime synchronization between graphical and textual syntaxes for multiple DSMLs. The high-level architecture of this solution is shown in Figure 1. The framework takes in input DSMLs in the form of: (i) metamodels defined in Ecor, (ii) textual domain-specific languages (DSLs), or (iii) grammar portions. From them, it generates graphical and textual notations automatically through the BUMBLE-M2T-metamodel and BUMBLE-Language-Analyzer components, respectively. The generated graphical and textual notations are saved in an XML format and serve as an input for mapping and synchronization activities.

The BUMBLE-Mapping-Editor component is developed to define explicit mappings between graphical and textual syntaxes. Mappings are also saved in an XML format and serve as an input for the synchronization mechanisms. For runtime synchronization, mapping rules are materialized as an EBNF grammar through the BUMBLE-EBNF-Generator component.

In the following we show the usage of the proposed framework for blended modeling and more importantly synchronization of graphical and textual notations for the Portable test and Stimulus Standard (PSS) use case.

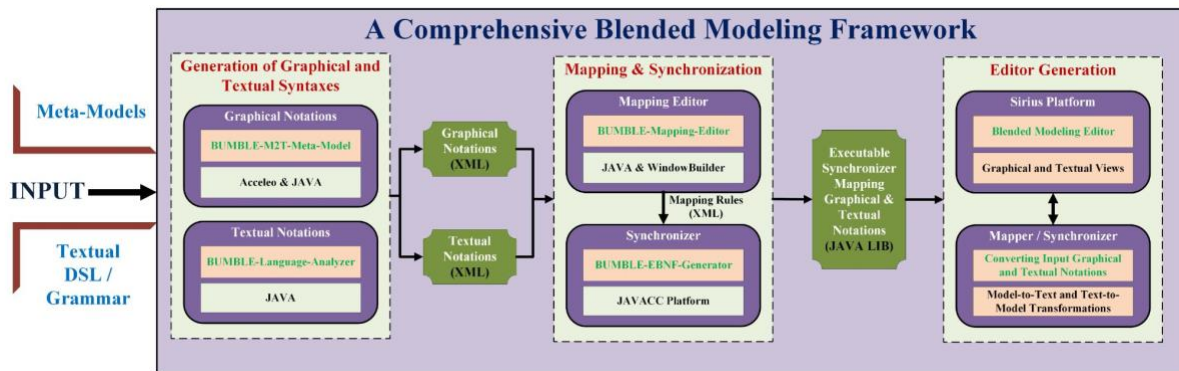


Figure 1 - The high level architecture of proposed framework

Input artefacts

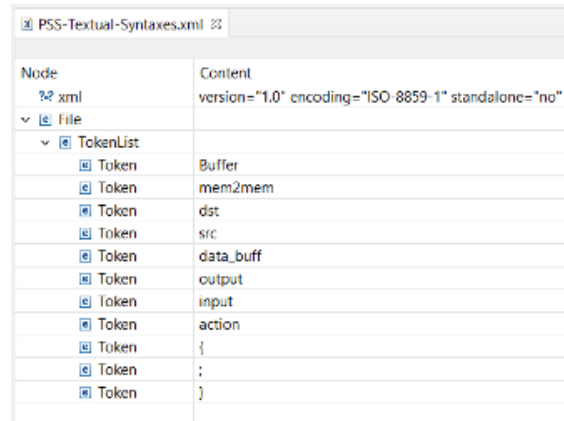
The proposed framework requires a textual DSL/grammar (plain text) and a metamodel (in Ecore) for the generation of textual and graphical notations respectively. In our use case, PSS carries along a C++-like DSL for the specification of test intents. The samples of PSS DSL are available in the PSS specification¹ and can be given as an input to our framework for the generation of textual notations for PSS. On the other hand, a canonical metamodel for PSS was not available, to the best of our knowledge. Therefore, we proposed such a PSS metamodel and implemented it using Ecore in EMF, as shown in Figure 2. The main PSS concepts like actions, objects, resources etc. and their semantics are included in the metamodel. This metamodel is given as an input to our framework for the generation of graphical notations.

¹<https://www.accellera.org/downloads/standards/portable-stimulus#:~:text=The%20Portable%20Test%20and%20Stimulus,of%20integration%20under%20different%20configurations>

For the generation of graphical notations, the BUMBLE-Language-Analyzer (Figure 4-a) is developed via model-to-text transformations in Acceleo and Java. Particularly, it takes a metamodel as an input and generates the corresponding graphical notation and semantic information in XML format. For demonstration, the PSS metamodel is given as an input and the generated XML file is shown in Figure 4-b. This XML file is further utilized in the mapping process as described in the remainder of this section.



Figure 4-a - BUMBLE-Language-Analyzer



Node	Content
?xml	version="1.0" encoding="ISO-8859-1" standalone="no"
file	
TokenList	
Token	Buffer
Token	mem2mem
Token	dst
Token	src
Token	data_buff
Token	output
Token	input
Token	action
Token	{
Token	:
Token	}

Figure 4-b - PSS textual syntaxes in XML

Mapping and Synchronization

The domain expert's feedback is crucial in the mapping process. For this purpose, a mapping editor is developed as shown in Figure 5. In terms of reusability and portability, BUMBLE-Mapping-Editor is flexible and can be used for any pair of graphical and textual notations. Importantly, all the interface components are generated dynamically through the XML files. Particularly, it displays the graphical and textual notations from the related XML files that are generated by BUMBLE-Language-Analyzer. Furthermore, it displays the repository of symbols to be associated with the graphical notation through mappings (containing addresses and IDs of symbols) and, therefore, other symbols specific to the domain can be added to the mapping editor with simplicity. In addition, it provides AND/OR operators to define complex mappings between graphical and textual elements (e.g., one graphical element may correspond to the combination of several textual elements and vice versa). This mapping file is utilized to generate the corresponding EBNF grammar for synchronization purposes.

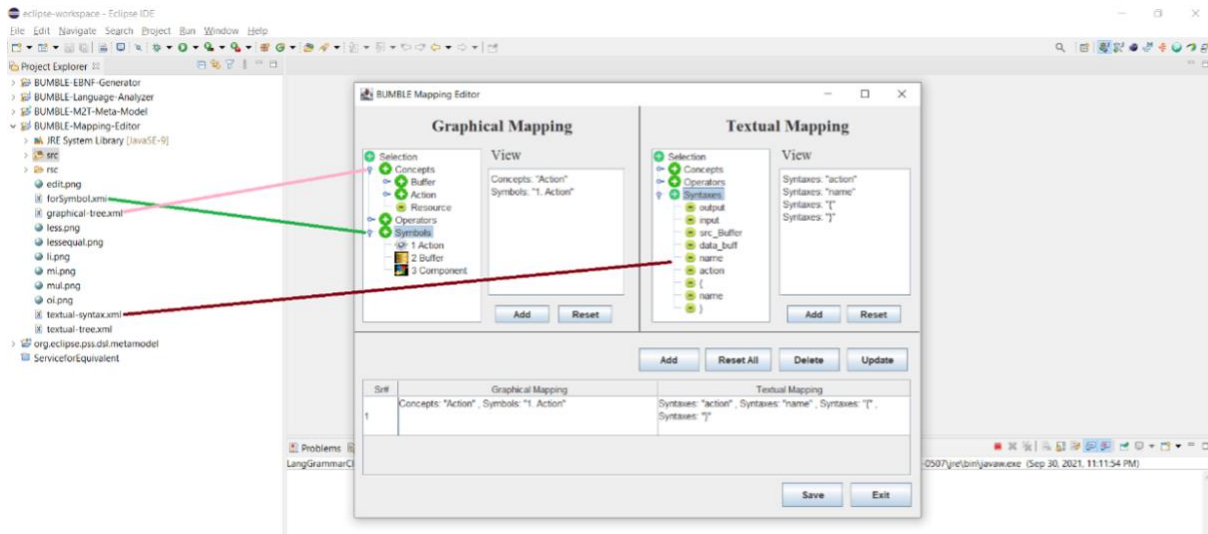


Figure 5 - BUMBLE Mapping Editor with PSS example

In Figure 6 we show the mapping between the PSS graphical and textual notations, as well as the association of graphical symbols to the PSS concepts, as furtherly described in deliverable D4.1.

The BUMBLE-EBNF-Generator component is responsible for generating an EBNF grammar by utilizing the mapping XML. This step is essential for seamless synchronization and switching between graphical and textual notations. The JAVACC² platform is used to implement this component. Few EBNF mapping grammar rules for the PSS action and buffer concepts are given here for demonstration purposes:

Rule 1: <Action> ::= <Graphical-Action> | <Textual-Action>

Rule 2: <Graphical-Action> ::= <Name><Symbol> | <Name><Symbol><Relationship><Graphical-Data Buffer>*

Rule 3: <Textual-Action> ::= action <Name> { } | action <Name> { (<Textual-Data Buffer>)* }

Rule 4: <Data Buffer> ::= <Graphical-Data Buffer> | <Textual-Data Buffer>

Rule 5: <Graphical-Data Buffer> ::= <Type><Name><Symbol>

Rule 6: <Textual-Data Buffer> ::= <Type> data_buff <Name>;

Rule 7: <Relationship> ::= Containment <Symbol>

Rule 8: <Name> ::= ([a-z][A-Z][0-9])*

Rule 9: <Symbol> ::= ([a-z][\][A-Z][0-9])*

Rule 10: <Type> ::= input | output

For better understanding, consider a simple PSS DSL example where an action having two buffers is defined as:

```
action mem2mem {
    input data_buff src Buffer;
    output data_buff dst Buffer;
}
```

Based on the aforementioned EBNF rules, a parsing tree of the given example is shown in Figure 6 and is used to achieve seamless synchronization and switching between graphical and textual notations. Please note that terminal symbols are displayed in orange boxes.

² <https://javacc.github.io/javacc/>

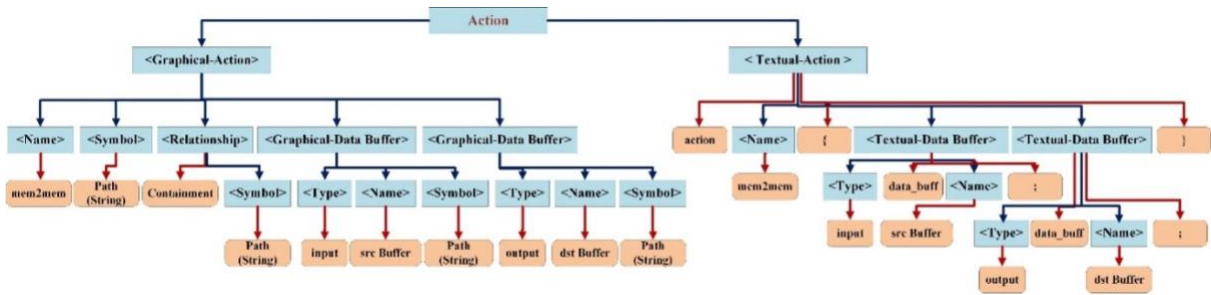


Figure 6 - Parsing tree of EBNF rules for PSS action concept

The implementation of EBNF is accomplished through the JAVACC platform as shown in Figure 7-a. More specifically, all EBNF rules are implemented in a JAVACC template file (i.e. representing a grammar). It is pertinent to mention that the implementation of EBNF is done as a service, so that it can be easily imported in other tools for editor generation like Eclipse Sirius. To verify the EBNF service in JAVACC, a test client application is also developed and shown in Figure 7-b. It provides an interface to input textual or graphical syntaxes and subsequently calls the EBNF service to receive the equivalent textual or graphical syntaxes accordingly as shown in Figure 7-b.

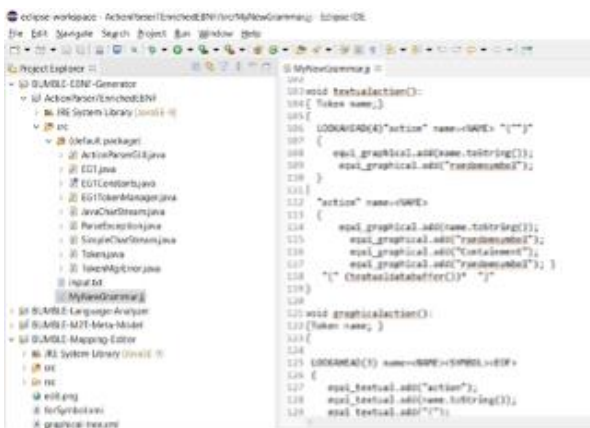


Figure 7-a - EBNF implementation in JAVACC

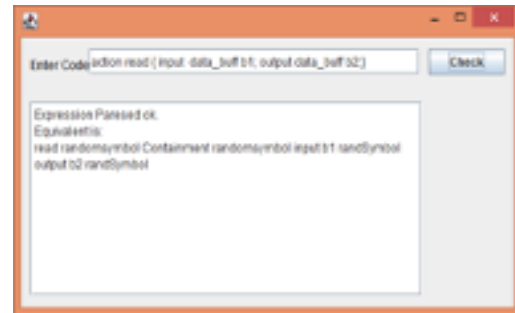


Figure 7-b - Test client for EBNF Service

Finally, Eclipse Sirius is used to generate the blended modeling editor. The architecture of the editor generation component is shown in Figure 8. In the first step, a graphical editor is generated in Sirius by utilizing the PSS metamodel. In the second step, a textual view is incorporated in the graphical editor for the specification of textual PSS. The blended modeling editor supports seamless synchronization and switching between graphical and textual notations using Java services, model-to-text and text-to-model transformations, and the Synchronizer component. Particularly, Java services are used to communicate with the Synchronizer, while model-to-text and text-to-model transformations are intermediate components to perform suitable propagation of model changes between graphical and textual notations and display them in the blended modeling editor accordingly for seamless switching.

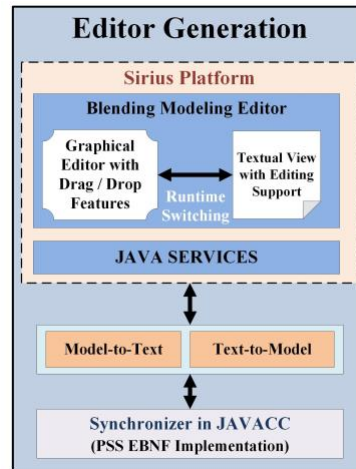


Figure 8 - Architecture of editor generation component

The generated PSS blended modeling editor is shown in Figure 9. Particularly, it offers drag/drop functionality for different PSS graphical elements (provided in a palette) with suitable symbols that were chosen during the mapping process. The “Update Views” button is provided to synchronize the two notations on-demand. The outcome of the synchronization process (i.e. Successful, Done with Errors and Failed) is displayed too. Here, a simple PSS Direct Memory Access (DMA) example is considered where an action (mem2mem) and two buffers (input source and output destination) are modeled graphically, as shown in Figure 9. Subsequently, a seamless switching is performed via the “Update Views” button to generate the corresponding textual syntax. After successful synchronization, a component (dma_c) is defined textually and synchronization is performed to represent the dma_c component graphically as shown in Figure 9.

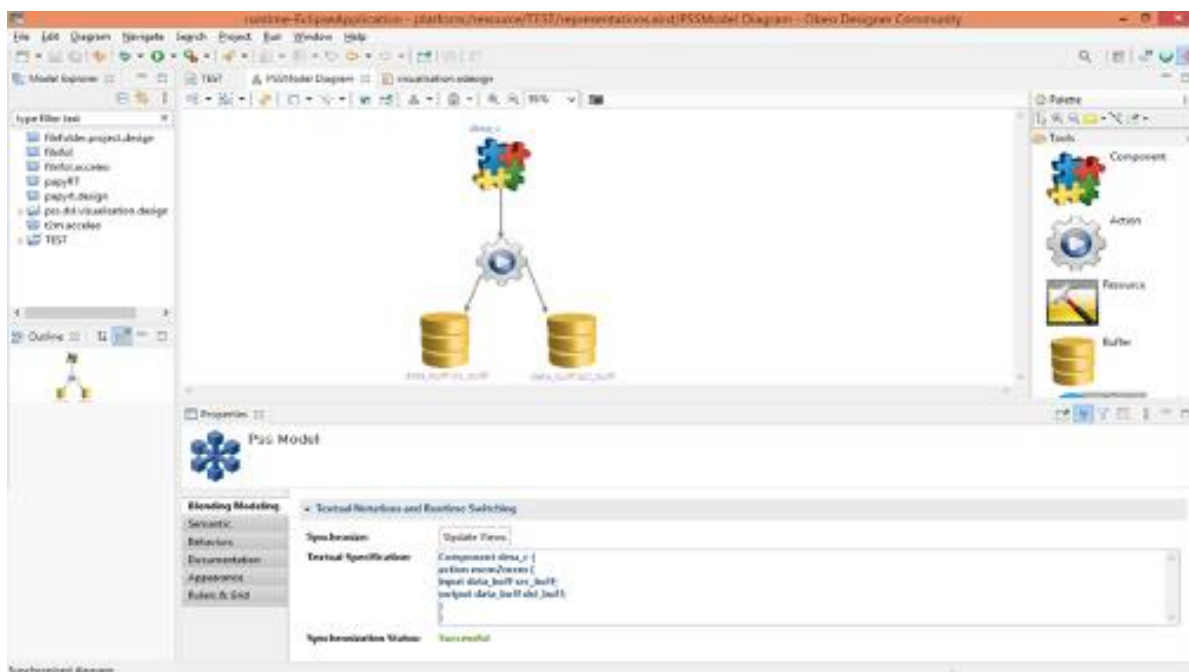


Figure 9 - PSS Blended Modeling Editor

3.2. Synchronization for UML-RT State Machines (UC1, UC2)

UML-RT is a real-time profile for UML that aims to simplify the ever-increasing complex software architecture specification for real-time embedded systems and it relies on state machines for behavior modeling. Most off-the-shelf UML modeling tools focus on graphical editing features and do not allow seamless graphical--textual editing. To overcome this challenge, we define a textual notation for UML-RT state-machines (SMs) and synchronize it with the graphical notations available in Papyrus-RT (open source) and RTist (commercial tool from one of the BUMBLE's partners, HCL). Note that in this section we describe the solution for Papyrus-RT (UC1), which is orthogonal to the one in RTist (UC2).

Figure 10 provides a high-level architecture that includes all the components required to synchronize between graphical and textual notations for UML-RT SMs and how they are connected together. In the following, we provide the description for each of the components.

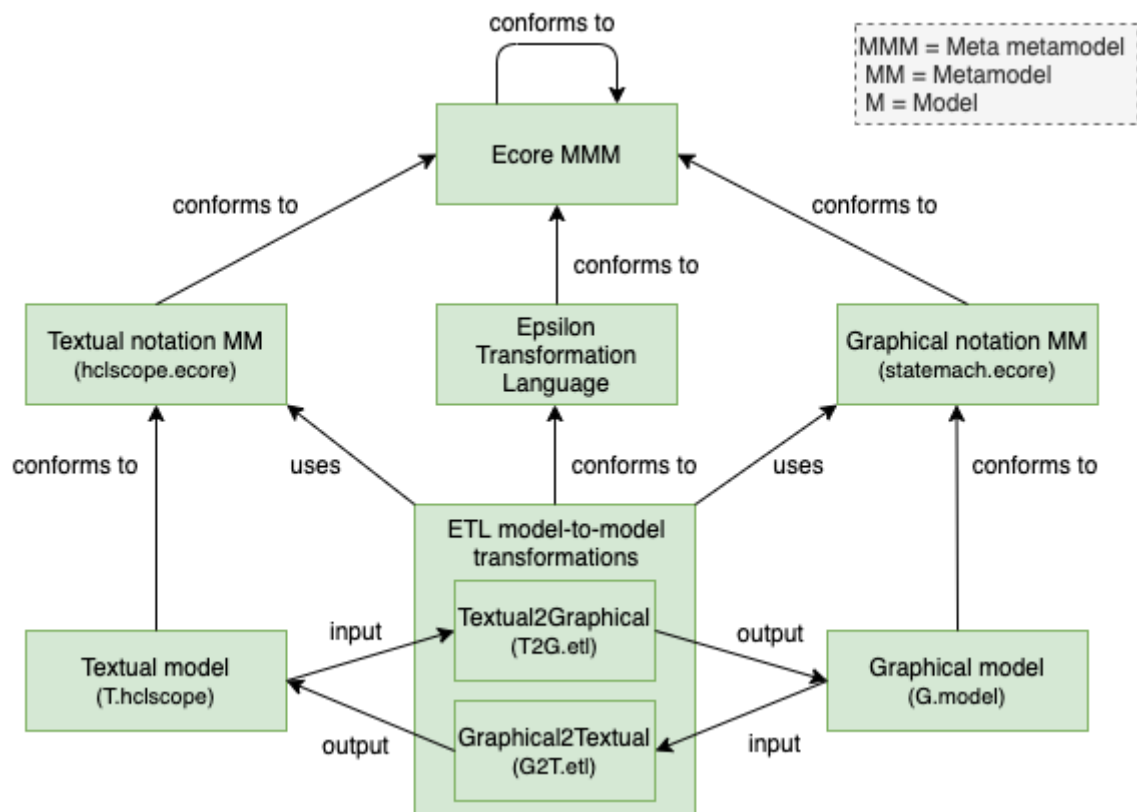


Figure 10: High-level architecture for the synchronization between the graphical and textual notations for UML-RT state machines.

Ecore MMM: The Ecore meta-metamodel is a language that conforms to itself and is used to define Ecore metamodels.

Metamodels: Metamodels (defined in Ecore) conceive the concepts, the relationships and well-formedness rules that formalize how the meta-concepts can be legally combined for valid UML-RT SM models. The UML-RT SMs use case consists of two metamodels that are used to define the graphical and textual notations.

- **Graphical notation MM:** The metamodel used to describe the graphical model for UML-RT SMs is the underlying metamodel used in Papyrus-RT and it is available as open-source (the metamodel in Ecore is depicted in Figure 15 in Appendix 1).
- **Textual notation MM:** The metamodel used to describe the textual model for UML-RT SMs is defined manually using the Xtext language workbench (the metamodel in Ecore is depicted in Figure 16 in Appendix 1). It takes into consideration i) the feedback from UML-RT's customers and architects in RTist, and ii) the UML-RT metamodel portion describing state-machines, as blueprint. Moreover, the scope provider by Xtext is customized to only allow cross-references for elements declared in the same model file, support the inheritance mechanism, and restrict transitions to only cross-reference pseudo states and states that are on the same level of nesting as the transition, or their immediate entry and exit points. Consequently, this customization contributes to the enforcement of the UML-RT's modularity.

Models: In order to instantiate the textual and graphical notation metamodels, we create two models (i.e., textual model and graphical model) respectively. These models will be used both as source and target models, depending on the direction of the transformation.

Epsilon Transformation Language (ETL): ETL is a hybrid model transformation language that provides both declarative execution schemes for simple transformations and imperative features for supporting complex transformations. ETL Transformations are organized in modules that contain a number of transformation rules that specify one source and target parameters.

ETL Transformations: In order to provide the synchronization mechanisms for UML-RT SMs, we define two model-to-model unidirectional transformations. The transformations are exogenous and horizontal, as the models are expressed in different modeling languages and reside in the same abstraction level. The first step consists of identifying the mapping in order to determine which elements in the source model are actually mapped into a corresponding element in the target model. The majority of elements have a one-to-one mapping, as the modeling languages are conceptually similar. In such circumstances, the transformation rules are rather straightforward and contain one source and one target parameter (see Listing 1). On the occasions where one element in the source model is mapped to more than one element in the target model, the transformation rules contain one source and multiple target parameters (see Listing 2). Lastly, in the event of multiple elements in the source model mapping to one single element in the target model, the transformation rules need to be defined separately as it is not possible to define transformation rules with multiple source parameters (e.g., transformation rules for transitions defined in the Textual2Graphical transformation).

Textual2Graphical: This transformation takes as input the textual model and produces as output the graphical model. Listing 1 is an example of the transformation rule written in ETL for transforming a Junction from the source metamodel (i.e., textual notation MM) into a JunctionPoint in the target metamodel (i.e., graphical notation MM).

```
rule Junction2Junction
  transform s: Source!Junction
  to t: Target!JunctionPoint
  {
    t.name=s.name;
  }
```

Listing 1 - Textual2Graphical ETL rule

and conversely, system engineers with more insight into the implementation. The automated evaluation of these guidelines provides faster feedback loops between the engineers.

We propose a bridge between the C++ IDE (JetBrains CLion) and the modeling tool (IBM Rational Rhapsody). The code and model notations will be blended in the following two ways: 1) when editing code in CLion, a software engineer will be able to see the related portion of the model she is editing. For example, when editing a class, she will see the state machine in which that class is a state. 2) When editing the model in Rhapsody, a system engineer will be able to see the related portion of the implementation she is editing. For example, when editing a state machine, the related code showing the class definitions corresponding to the states in that state machine will be shown. In both cases, the additional information is aimed to be shown inside the IDE by means of plug-ins for CLion and Rhapsody, thus blending the two notations. Figure 11 gives an overview of the proposed bridge and its constituent components.

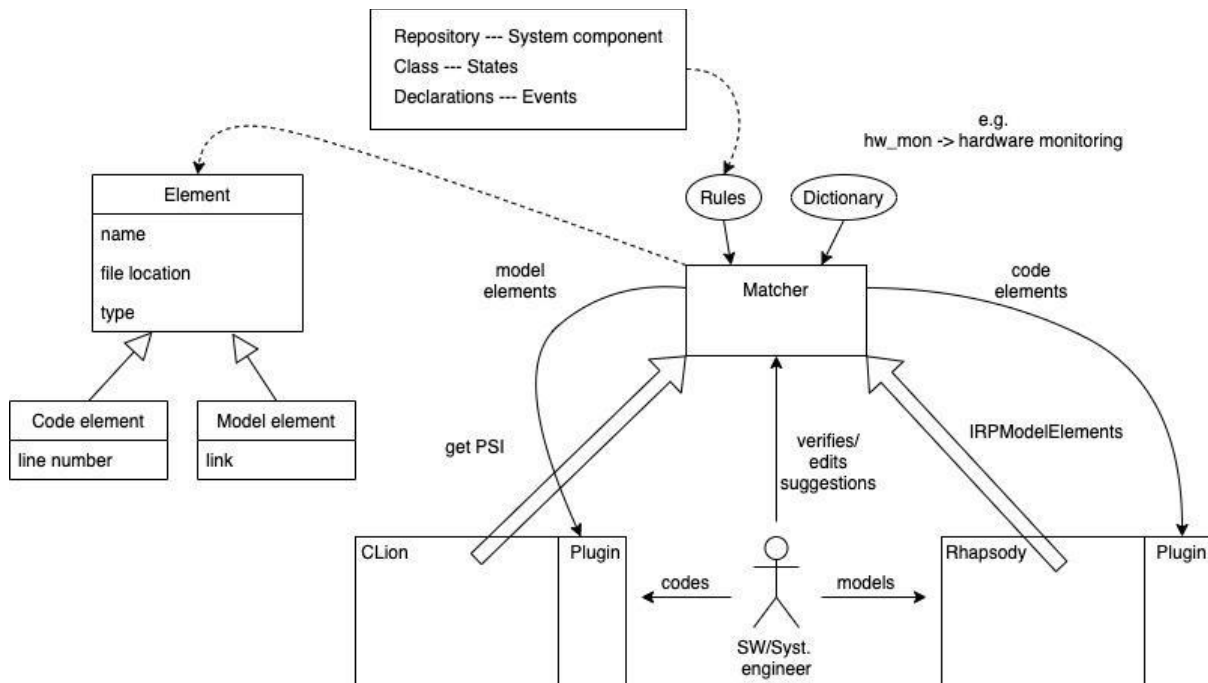


Figure 11 - System and software artefacts synchronization

There, the two Matcher items match specific code and model elements, as defined by the rules, and store the mined elements in a generic common format as elements. In the current phase, we have identified three initial architectural guidelines that must be checked. Correspondingly, we match code repositories to system components in the model, classes in the code to states in the model, and specific declarations in the code to events (signals) in the model. The dictionary is used to assist the matcher in finding equivalent elements that are named differently in model and code (for example because there can be no spaces in variable names in the code). Blending of the notations is shown in the figure as the plugins for the two development tools, that will show the elements of the model/code.

As part of the study, we have performed a validation workshop (prior to starting implementation) with engineers at Saab. Initial feedback of the engineers indicates a positive view of the proposed solution. For example, a software engineer remarked that blending the notations would assist in improving understanding of the model. "This will help me to learn to know the model and understand the model

better. Currently, I don't look at the model." Another remark: "As a software engineer, this will help me to see what has changed in the model and will give some help to see how complete the implementation is."

Based on the outcome of the validation workshop, currently a prototype implementation is being developed, focusing initially on creating the mapping rules (see D4.1) required for mapping implementation to model and vice versa. An important outcome of the validation workshop was that the mapping rules should be made in such a way that they do not become another maintenance burden in the development process ("As a system engineer I would be unhappy with maintaining rules if it means to change a term in multiple places.").

3.4. Synchronization with DClare

DClare realises blended modeling by transforming the abstract syntaxes of different modeling languages. This is especially valuable in MPS where the synchronization of multiple concrete syntaxes for a given abstract syntax is already supported by MPS itself. However, in MPS the abstract syntax is very much driven by its concrete appearance in the editors. This implies that blending multiple concrete syntaxes in MPS requires the transformation of multiple abstract syntaxes. This is exactly what DClare supports.

There is also a need for transformations between models of different Ecore-based models in EMF. That is why we plan to realise model-transformations within EMF too. The engine is the same for EMF and MPS. There is already an integration with MPS, a connector for EMF is still to be developed.

Figure 12 shows the basic architecture of DClare-based transformation specifications and executions. The small dark blue rectangle is the connector that connects MPS models to the DClare engine. The engine reacts to any change in the models and changes the opposite models according to the transformation specification. The generator of MPS is extended to also generate the transformation rules that are executed by the DClare engine.

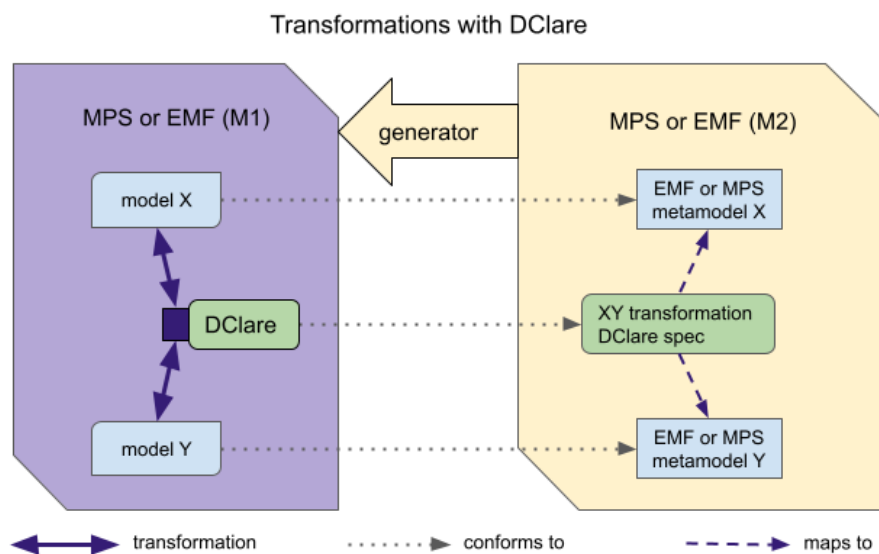


Figure 12 - Transformations with DClare

The specification of the transformation is based on attribute and rule specifications. Within MPS we realised a so-called 'language aspect' for defining attributes and rules in MPS. The (meta)language for this is an extension of the base-language of MPS itself. This guarantees easy adaptation by the MPS community.

Figure 13 shows how transformations are defined in MPS using DClareForMPS rulesets.

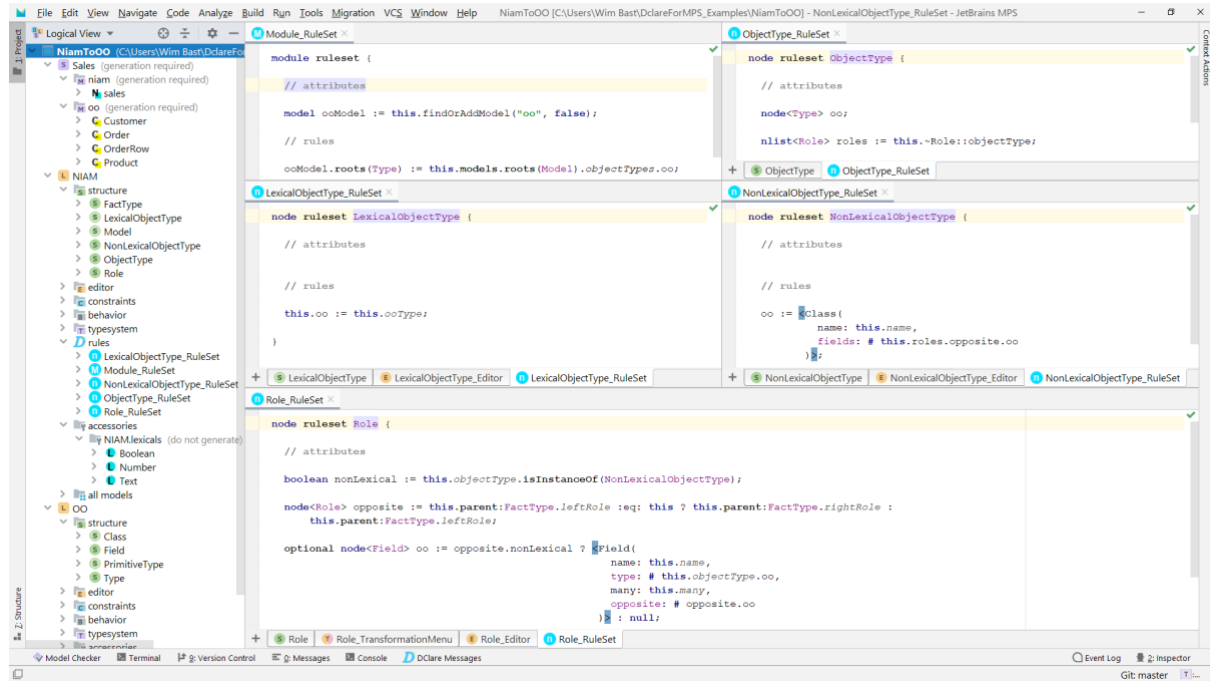


Figure 13 - DClareForMPS in MPS

4. HOTS for bidirectional synchronization

The main objective of WP4 is to provide a set of HOTS to semi-automatically generate the synchronization infrastructure, in terms of bidirectional model transformations, across concrete syntaxes. Being able to automatically generate synchronization mechanisms through HOTS brings the following advantages: (i) the solution is generic and not tailored to a specific DSML; (ii) the solution is flexible and allows co-evolution in response to DSML evolution (since synchronization mechanisms are not fixed but rather generated from the DSML metamodel, whenever the metamodel evolves, the mechanisms can be incrementally re-generated from the evolved DSML); (iii) the definition of HOTS is a one-time effort, which can then be transparently leveraged by developers to generate their own synchronization infrastructure for a given DSML.

On the one hand, the use cases refined in WP2, the architecture and methodology defined in WP3, and the visualization specific DSMLs and mapping models defined in WP3-WP4 will be exploited to define the solutions in WP4 as well as to evaluate the resulting implementation. On the other hand, the capabilities of the solutions defined in WP4 will influence the overall architecture and methodology defined in WP3. We envision that results from WP4 (HOTS for the generation of synchronization mechanisms from DSMLs) will be also exploited by WP5 for collaborative modeling.

In Figure 14 we provide an architecture for the generation and usage of model transformations generated via HOTS in BUMBLE. Given two DSMLs defined in terms of Ecore (in EMF), a mapping model, conforming to the mapping metamodel defined in D4.1, conceives the mapping rules for synchronizing models conforming to the two DSMLs. Interestingly, this architecture and the transformations in it entail multiple usage scenarios, as follows:

- In case the DSMLs are two entirely disjoint (but somehow connected/dependent) languages, the generated transformations provide **synchronization across different languages** (either same or different notations).
- In case the DSMLs represent two notations of the same language, the generated transformations provide **synchronization across different notations** of the language.
- In addition, in case DSML₂ represents an evolution of DSML₁, the generated transformations provide **co-evolution** mechanisms for models conforming to DSML₁.

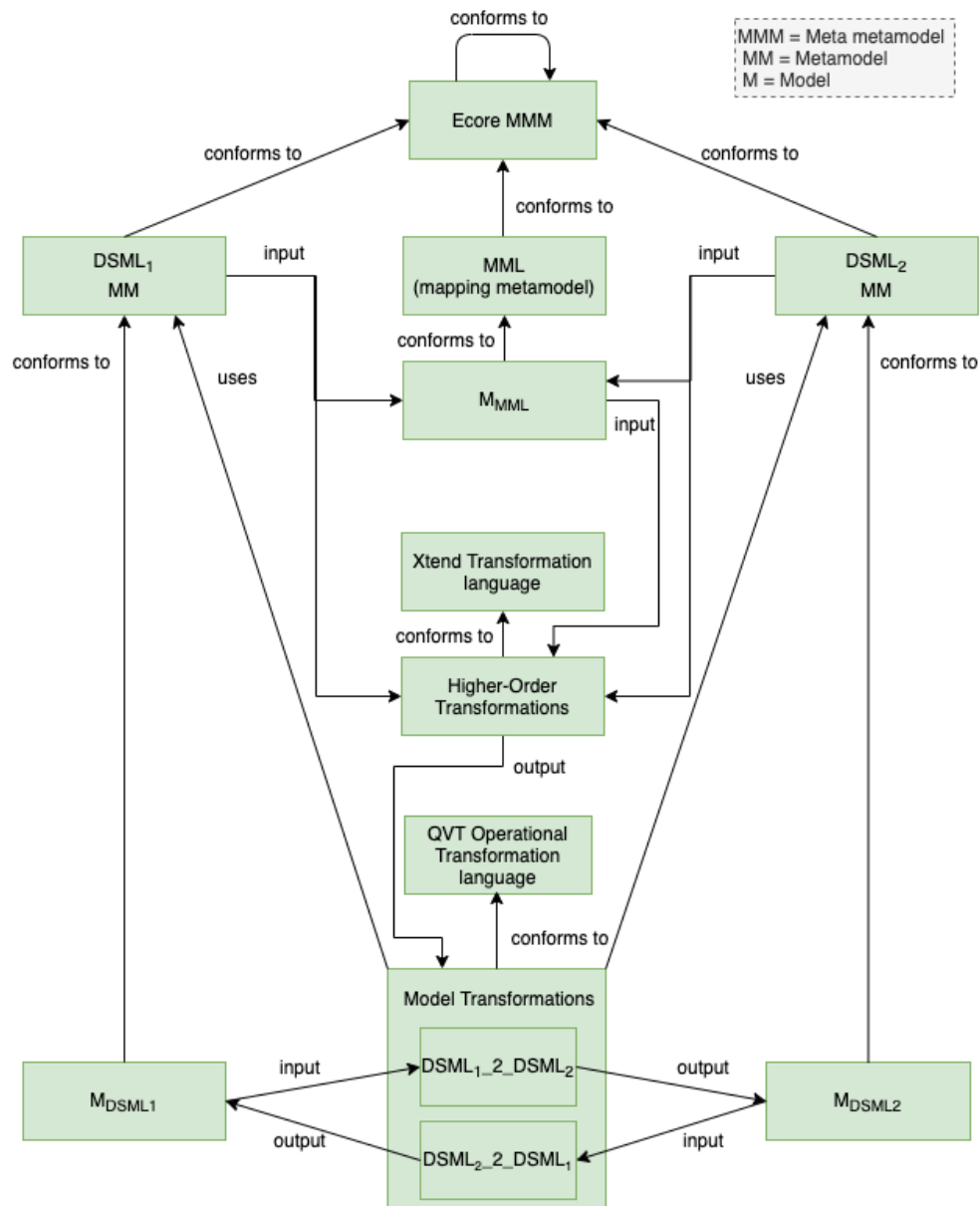


Figure 14 - Architecture for HOTS and bidirectional synchronization

In Figure 14 we can see the mapping modeling language (MML) described in D4.1, which is used to formalize the mapping rules between $DSML_1$ and $DSML_2$ in a mapping model M_{MML} . This model together with the two DSMLs are given in input to our set of HOTs defined in Xtend. The output of the HOTs are synchronization model transformations defined in QVT Operational. These model transformations are then in charge of automated synchronization between two instances of $DSML_1$ and $DSML_2$, that is to say M_{DSML_1} and M_{DSML_2} . The type of transformation (i.e., endogenous, exogenous, out-of-place, in-place) depends on the nature of the two DSMLs, as explained in the scenarios above.

Xtend was chosen as the language for implementing the HOTs for multiple reasons. First of all, it is a flexible and expressive dialect of Java, and it compiles into readable Java 8 compatible source code. In addition, any existing Java library can be used seamlessly. The compiled output is readable and pretty-printed, and runs usually at least as fast as the equivalent handwritten Java code. Xtend provides powerful macros, lambdas, operator overloading and several other modern language features. Finally, Xtend is included in the Eclipse release train and follows all Eclipse releases.

For model transformations generated via our HOTs in Xtend we chose QVT Operational (QVTo), which is an implementation of the Operational Mappings Language defined by MOF 2.0 Query/View/Transformation (QVT). The reasons behind this choice were first of all the fact that QVT is a MOF standard, and since our focus is on MOF languages, a transformation language also based on MOF is preferred. In addition, QVTo brings together benefits from both the declarative and imperative QVT and it is very well-suited for both exogenous and endogenous transformations, also in-place.

Currently, we are implementing the HOTs described in this section. More specifically, we are isolating the transformation concepts that will be embedded in the HOTs from the mapping information that will be encoded as mapping rules in the mapping models leveraged by the HOTs.

5. Next steps

We will continue working on the synchronization of system and software artefacts since it requires specific solutions based on consistency management. We will exploit results from other activities in all tasks of WP4 for this. We are working on DClare for EMF to offer the powerful bidirectional transformation means to Ecore-based metamodels too.

Regarding the core activity of the WP, namely T4.3, after implementing the HOTs, we will validate them by exploiting the use-cases mentioned in this deliverable. Moreover, we will exercise them for the three scenarios (i.e., sync across different notations, sync across languages, co-evolution) in order to fine tune them.

References

- [AND07] Andrés, F. P., De Lara, J., & Guerra, E. (2007, October). Domain specific languages with graphical and textual views. In *International Symposium on Applications of Graph Transformations with Industrial Relevance* (pp. 82-97). Springer, Berlin, Heidelberg.
- [LAZ11] Lazăr, C. L. (2011). Integrating Alf editor into UML editors. *Studia Universitatis Babes-Bolyai, Informatica*, 56(3).
- [AND07] Domain specific languages with graphical and textual views
- [CHA09] Charfi, A., Schmidt, A., & Priestersbach, A. (2009, June). A hybrid graphical and textual notation and editor for UML actions. In *European Conference on Model Driven Architecture-Foundations and Applications* (pp. 237-252). Springer, Berlin, Heidelberg.
- [SCH08] Scheidgen, M. (2008, June). Textual modeling embedded into graphical modeling. In *European Conference on Model Driven Architecture-Foundations and Applications* (pp. 153-168). Springer, Berlin, Heidelberg.
- [WIM05] Wimmer, M., & Kramler, G. (2005, October). Bridging grammarware and modelware. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 159-168). Springer, Berlin, Heidelberg.
- [MAR15] Maro, S., Steghöfer, J. P., Anjorin, A., Tichy, M., & Gelin, L. (2015, October). On integrating graphical and textual editors for a UML profile based domain specific language: an industrial experience. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering* (pp. 1-12).
- [TIS09] Tisi, M., Jouault, F., Fraternali, P., Ceri, S., & Bézivin, J. (2009, June). On the use of higher-order model transformations. In *European Conference on Model Driven Architecture-Foundations and Applications* (pp. 18-33). Springer, Berlin, Heidelberg.

Appendix 1 – Excerpt of UML-RT metamodels

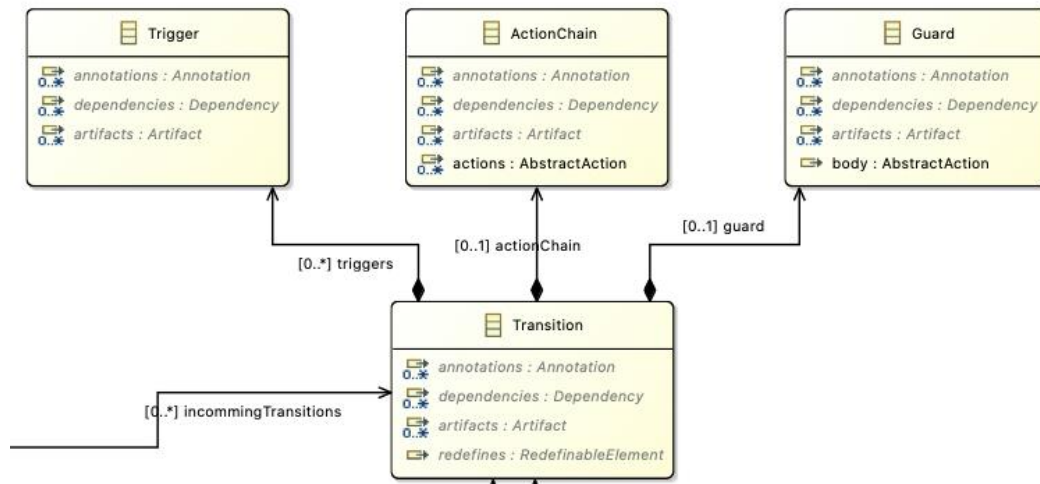


Figure 15 – Excerpt of the UML-RT metamodel in Ecore for graphical modelling in Papyrus-RT

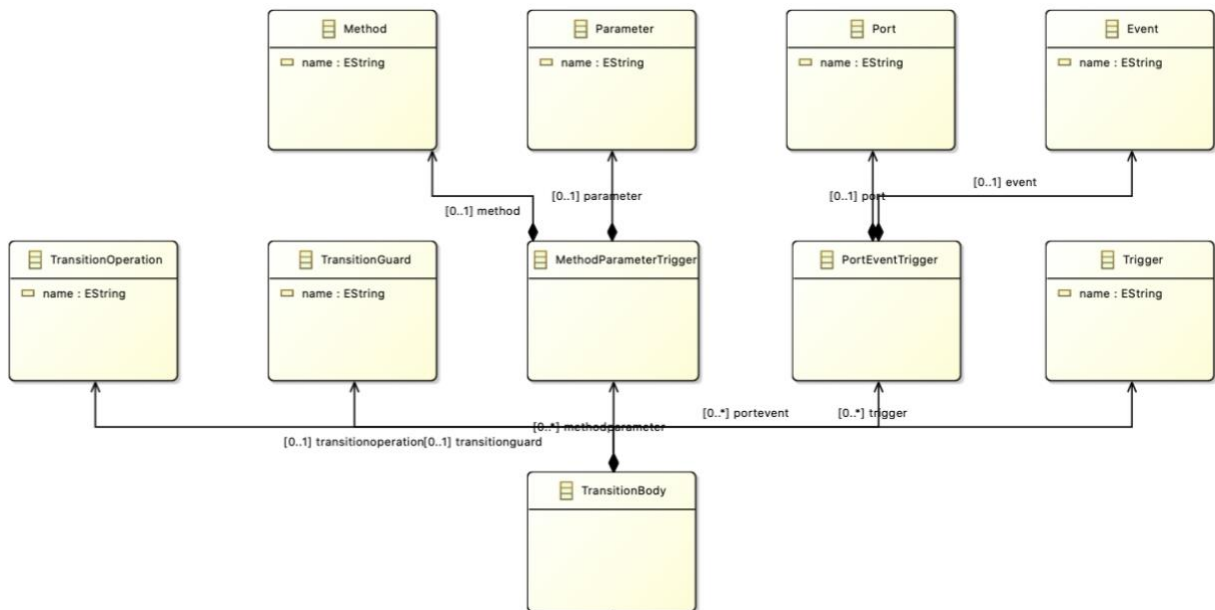


Figure 16 – Excerpt of the UML-RT metamodel in Ecore for textual modelin in Xtext