ITEA Office
High Tech Campus 69 - 3
5656 AG Eindhoven
The Netherlands

T + 31 88 003 6136
E info@itea3.org
W www.itea3.org

ITEA 3 is a EUREKA strategic ICT cluster programme

BUMBLE Deliverable D4.1

Mapping rules for blended notations generation, bidirectional synchronization, and co-evolution

Project Acronyms

<ACR>	<Acronyms>
BUMBLE	Blended modeling 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
XLST	eXtensible Stylesheet Language Transformations
ETL	Epsilon Transformation Language
MML	Mapping modeling Language
ATL	Atlas Transformation Language
PSS	Portable test and Stimulus Standard
AMW	Atlas Model Weaver
HOT	Higher order transformation
MEO	Mapping Ecore-OWL
RDF	Resource Description Framework
DIML	Diagram Interchange Mapping Language
MOF	Meta-Object Facility

Versions

Release	Date	Reason of change	Status	Distribution
V0.1	20/09/2021	Defined structure	Draft	WP4 partners
V0.2	01/11/2021	Complete draft of the first version of the deliverable	Draft	BUMBLE consortium
V1.0	11/11/2021	Updated with comments on V0.2, to be submitted to ITEA portal	Final	Uploaded to ITEA portal
V1.1	08/01/2023	Draft of the updates for the second version of the deliverable	Draft	WP4 partners
V1.2	08/02/2023	Complete draft of the second version of the deliverable	Draft	BUMBLE consortium
V2.0	20/02/2023	Updated with comments on V1.2, to be submitted to ITEA portal	Final	Uploaded to ITEA portal

Executive Abstract

The focus of this deliverable is on the definition and modeling of mapping rules across DSMLs for model transformation generation purposes. Mapping rules are modeled via a mapping language described in this deliverable. The mapping language is a structured and formalised means for precisely describing mapping rules between two or more DSMLs.

The definition of a mapping language is pivotal for multiple activities in BUMBLE. Explicit mapping rules enable us to link in a deterministic manner multiple DSMLs for synchronization, migration, and reconciliation activities.

In this deliverable, we describe the theory and process followed to achieve the mapping rules as well as to define and validate the mapping language.

Table of contents

1. Introduction

In this deliverable, we report on the activities carried out as part of tasks T4.1 and T4.2 in WP4, in particular for what concerns the definition, implementation, and validation via application to industrial use-cases of a *mapping language* in the context of the Eclipse Modeling Framework (EMF).

In this scope, a mapping language is a structured and formalised means for precisely describing mapping rules between two or more domain-specific modeling languages (DSMLs). In this context, a modeling language is intended to be defined in terms of a metamodel.

The definition of a mapping language is pivotal for multiple activities in BUMBLE. Explicit mapping rules enable us to link in a deterministic manner multiple DSMLs; in BUMBLE, the rules are a fundamental input to

1. correctly generate editors from a DSML definition. In this case, the DSML is mapped to one or more notation-specific DSMLs (see Section 3).
2. correctly synchronize models between two different DSMLs. In this case, the DSMLs, which may represent two different notations (in terms of abstract and concrete syntaxes), are mapped to one another (see Section 4).
3. provide co-evolution mechanisms in terms of model transformations for migration and reconciling blended models in case of (WP4-5).

Once the mapping language is defined and implemented, its instances (mapping models) will be used as input for the generation and synchronization of transformations. More specifically, a mapping model represents the guiding principle driving the transformation to properly generate correct information from one model or to properly propagate changes across models. If defined at the meta-metamodel level across multiple DSMLs, mapping models could even be used for driving co-evolution across DSMLs (and thereby notations).

The work and solutions described in this deliverable contribute to the following BUMBLE Technology Bricks and requirements:

Technology bricks	Description of main contributions	Main requirements
Editor Generators	The mapping rules and language defined and formalised in this deliverable are pivotal for the correct generation of editors from DSML definitions	BC1, BC2, BT1, BT2, BT3, BT4
Blended Model Access	The mapping rules and language defined and formalised in this deliverable are pivotal for the correct generation of model transformations for synchronization by higher-order transformations	BC3, BC4, BC10, BC11, BC12, BC13, BT7, BT8, BT9, BT10, BT11, BT12, BT13, BT19, BT21, BT25
Meta-(model) co-evolution	The mapping rules and language defined and formalised in this deliverable are pivotal for the correct generation of model transformations	BC9, BT22, BT24

	for co-evolution by higher-order transformations	

The remainder of the deliverable is structured as follows. In Section 2, we provide an overview of the state of the art in mapping modeling/description. In Section 3, we describe the actions and results in relation to the explicit and implicit mapping of blended notations, with direct application to multiple industrial use cases. In Section 4, we introduce our modeling language for flexible mapping and conclude in Section 5.

2. State of the art on mapping modeling/description

Various mapping languages have been proposed in the literature to support different model management operations (e.g., model transformation, model migration, model integration). In the following, we present these contributions and highlight their advantages and disadvantages with respect to our approach.

In [5], authors propose a mapping metamodel based on the Eclipse Modeling Framework (EMF), that supports mapping specifications between two metamodels. Moreover, they contribute with the Atlas Model Weaver (AMW) tool, which simplifies mapping visualization, and enables the generation of transformation models conforming to Atlas Transformation Language (ATL) from the mapping model. In addition, AMW allows for the generation of a textual representation of the mapping model and validates the conformity of the latter to its metamodel. However, the mapping metamodel only provides one-to-one, one-to-many, and many-to-one relationships, and restricts the specification of more complex mappings (e.g., target metamodel contains elements that do not have a correspondence with any of the elements of the source metamodel, thus, need to be created). Moreover, the AMW tool generates a read-only textual representation of the mapping model and is restricted to the generation of ATL model transformations. Ecore2Ecore1 is a plugin, distributed with EMF, that was originally implemented with the goal of supporting metamodel evolution and is widely used for such purpose. Nevertheless, being that it allows the definition of mappings between two metamodels, it can be used to define mapping models that could serve as input to higher-order transformations (HOTs) UI and generate language-specific model transformations. However, just like AMW, it does not provide a way for the user to specify more complex mappings, and it does not restrict correspondences that are not valid bindings.

In [3], authors propose a textual mapping language called MEO (Mapping Ecore-OWL) that aims to enable the use of RDF resources as EMF objects and the serialization of EMF objects in RDF resources. The approach is based on EMF, and it defines correspondences between the domain model (conforming to Ecore), and the OWL ontology model. Moreover, it supports the generation of paired ATL transformations from HOTs, which automates the process of defining a bridge between EMF objects and RDF resources. However, the mapping metamodel is specific to OWL/RDF Resources.

In [4], authors propose a solution to enable the exchange of models between meta-modeling tools, thus, supporting interoperability, and avoiding vendor lock-in. This approach uses bridges at the meta-meta level to export metamodels from different environments into an intermediate one and uses binding components to create tree structures of the metamodels. The main contribution of this approach is the graphical mapping language that is used to map between elements of the trees. Moreover, the mapping language is used as input to the code generator that outputs Epsilon Transformation Language (ETL) transformations. However, this mapping-based approach is focused on enabling the exchange of models between different meta-modeling tools, and as such, it specifies mapping correspondences between elements of meta-metamodels, and generates model transformations for metamodels, while our approach aims to specify mapping correspondences between elements of metamodels and generate model transformations for models.

In [2], authors propose Malan, a MApping LANguage that supports the definition of a schema mapping, between a source and target data schema. The mappings can be defined both textually and graphically (not simultaneously) using Papyrus. The graphical mapping is supported by the definition of a UML profile that contains a stereotype that defines the mapping concept for UML. In addition, these mappings are used as input to the Malan processor that generates a transformation. However, this approach manifests a few limitations. To begin with, the source and target schema should be expressed as UML class diagrams. Now even though UML is a widely used modeling language, and allows for the definition of the mapping concept using UML profiles, this restricts the use of Ecore metamodels. Moreover, the transformation program only generates XSLT stylesheets that convert XML documents into XML, HTML, or plain text documents.

In [6], authors propose a solution for the integration of heterogeneous modeling languages that incorporates both the definition of a mapping language and a rule definition language. Even though their objectives differ from ours, being that the mapping language is defined independently from the rule integration language, it can support other model manipulations (e.g., model transformations). However, all metamodels (i.e., source, target, mapping, and integration) conform to the ADONIS meta-metamodel, in order to avoid conflicts among metamodels. Unless we define mappings from the ADONIS meta-metamodel to Ecore meta-metamodel, we cannot use Ecore models and metamodels.

In [1], authors propose DIML, a Diagram Interchange Mapping Language, that aims to define mappings between elements of MOF-based modeling languages (e.g., UML), and Diagram Interchange (DI) languages. DI is not restricted to UML, therefore, in a broad context, DIML can be used to create and transform visual diagrams for various MOF-based DSMLs. However, DIML is still a specific-purpose mapping language with a limited purpose of defining the concrete syntax of MOF-based modeling languages, and cannot be applied to more generic examples.

3. Explicit and implicit mapping of blended notations

In this section, we describe the actions and results in relation to the explicit and implicit mapping of blended notations, with direct application to multiple industrial use cases (UC1, UC2, UC6, described in D2.1).

3.1. Explicit mappings

As part of a prototype for the generation and synchronization of blended editors in the EMF, we designed and implemented a mapping editor between input graphical and textual notations using Java and the WindowBuilder library. The inputs to the mapping editor are represented by:

- A DSML defined in terms of Ecore (in EMF)
- A library of symbols for mapping to the specific graphical notation

- A set of textual concepts, extracted from a given EBNF grammar, for mapping to the specific textual notation

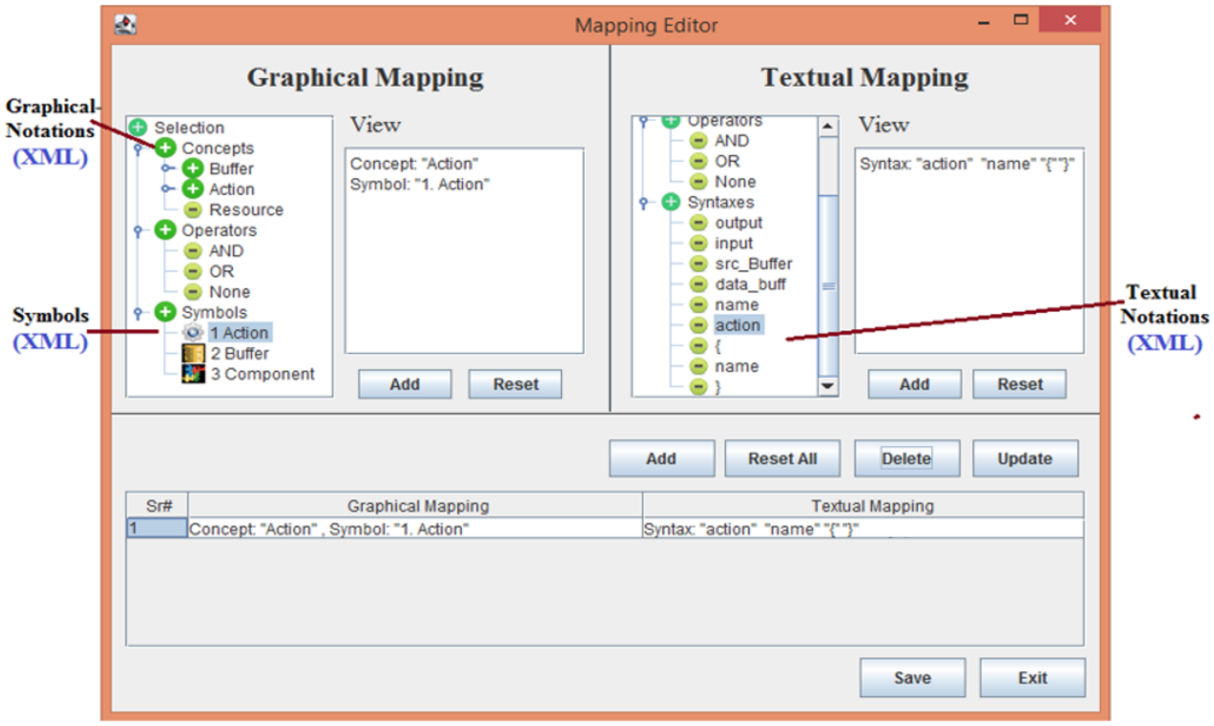
Note that all these elements can be customized and replaced. More specifically, any DSML defined in Ecore can be given as input to the editor. The library of symbols can be customized by removing and adding symbols, and the set of textual concepts can be any as long as it obeys an EBNF grammar.

The mappings are saved in an ad-hoc XML format, and it is used by another component of the prototype as input for the implementation of synchronization mechanisms between graphical and textual notations.

In terms of reusability and portability, the 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 XML files. Figure 1 displays the graphical and textual notations from the related XML files that are generated by the first component. Furthermore, it displays the repository of symbols to be associated with the graphical notation through the XML mapping file (containing addresses and IDs of symbols), and, therefore, other symbols specific to the domain can be added to the mapping editor with simplicity. Furthermore, 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 implement a corresponding EBNF grammar used for the synchronization process.

In Figure 1, we show the mapping between the Portable test and Stimulus Standard[1] (PSS) graphical and textual notations, as well as the association of graphical symbols to the PSS concepts. More specifically, the symbol with *Id =1* and *Name = Action* is associated with the graphical action concept. The assigned symbol will be available in the blended modeling editor for the modeling of the graphical action. On the other side, the textual syntax for action is specified as: *action name {}*. Subsequently, this mapping between graphical and textual action can be added to the queue (*grid*). Similarly, the mapping between graphical and textual notations for other PSS concepts, like buffer, objects, etc. is performed and saved in an XML file as well. Further details about explicit mapping features of the editor can be found at [10].

[1]The Portable Test and Stimulus Standard (PSS) defines a specification to create a single representation of stimulus and test scenarios usable by a variety of users across many levels of integration under different configurations. This representation facilitates the generation of diverse implementations of a scenario that run on a variety of execution platforms, including, but not necessarily limited to, simulation, emulation, FPGA prototyping, and post-silicon. With this standard, users can specify a set of behaviors once and observe consistent behavior across multiple implementations.



Concept: "Action"
Symbol: "1. Action"

Syntax: "action" "name" "{" "}"

Sr#	Graphical Mapping	Textual Mapping
1	Concept: "Action" , Symbol: "1. Action"	Syntax: "action" "name" "{" "}"

Figure 1 - Mapping editor

3.2. Implicit mappings

A different approach to the explicit mapping editor approach in Section 3.1 is represented by the implicit mapping rules that we encoded in a dedicated prototype for the synchronization of blended editors for UML-RT state machines. In this case, since both notations for UML-RT are defined a-priori and not intended to be customized by the user, the mapping rules are embedded in the model transformations in charge of the synchronization between graphical and textual notations.

In Listing 1, we depict an excerpt of the transformation in charge of propagating changes from graphical to textual notation. Note that these transformations were implemented using ETL in EMF. Alternatively, in [9] we describe a solution where the transformations are implemented using the Query/ View/Transformation Operational (QVTo) language.

```
rule Trigger2Trigger
   transform s: Source!Trigger
   to t: Target!Trigger, mpt:Target!MethodParameterTrigger,m:Target!Method, pa:
Target!Parameter, pet: Target!PortEventTrigger,
p:Target!Port , e:Target!Event {
         if (s.name.matches(".*\\..*")){
           p.name = s.name.split("\\.").first();
           e.name = s.name.split("\\.").second();
           pet.port = p;
           pet.event = e;
         }
        else if (s.name.matches(".*\\(.*")){
           m.name = s.name.split("\\(").first();
           pa.name = s.name.split("\\(").second();
          pa.name = s.name.split("\\)").first();
          mpt.method = m;
          mpt.parameter = pa;
        }
        else {
        t.name = s.name;
        }
    }
```

Listing 1 - Mappings implicitly defined in an ETL model transformation

As we can see in this specific rule, from an element of type *Trigger* in graphical UML-RT state-machine, the transformation generates a textual element of type *Trigger* and a set of other elements composing it. Although this is a transformation rule, it actually materializes a precise mapping rule between *Trigger* in the graphical notation and *Trigger* in the textual notation. While this solution may be preferable in the specific case where notations are not supposed to change or when the user is not intended to customize mappings, it is not flexible enough for our final purpose being a flexible mapping modeling solution that can be used for: generating blended editors, co-evolving them, and generate synchronization transformations. Nevertheless, both explicit and implicit mappings shown in this section laid the ground for the mapping modeling language described in the next section.

4. A flexible mapping modeling language

Given the experiences with mapping described in the previous section and in conjunction with the project requirements (core requirements BC1, BC4, BC9, as described in D2.2), in [7,8], we created a *mapping modeling language (MML)* defined as a metamodel, i.e., the most suitable form for our purposes. We investigated different possible technological choices, more specifically, Xtext and JetBrains MPS for a textual mapping modeling language and Ecore for a tree-based mapping modeling language. Since the core usages of MML would be to (i) support the definition of explicit mapping rules between DSMLs in a user-friendly manner and (ii) provide a transformation-friendly input to the generation of editors and synchronization mechanisms, we opted for an implementation in Ecore. The additional advantage is that a textual notation for it could be defined in Xtext exploiting the very same BUMBLE features. In Figure 2, we depict the MML defined and implemented in Ecore, and in the following, we detail the metaconcepts of the MML.

The **MappingModel** serves as the root of the metamodel and is a tuple `<name, Rules*, SourceMetamodels*, TargetMetamodels*, MainSourceMetamodel>`, where `name` is a unique name for `MappingModel`, `Rules*` is a possibly empty set of elements of type `MappingRule`, `SourceMetamodels*` and `TargetMetamodels*` are sets of elements of types `SourceMetamodel` and `TargetMetamodel` respectively, with at least one element each. `MainSourceMetamodel` is a single element of type `SourceMetamodel` that in the case of multiple `SourceMetamodels` is required to indicate the `SourceMetamodel` to be used at the entry point of the transformation to be generated.

The **MappingRule** is a tuple `<name, operator, condition, comment, source, helperLiteral, target, ChildRules*, ChildHelpers*>`. For `MappingRules` contained in `MappingModel`, we refer to them as *immediate mapping rules*, while for `MappingRules` contained in other `MappingRules` or `HelperStatements`, we refer to them as *child mapping rules.* name is a unique name for `MappingRule`, and `operator` represents the type of operator between mappings (i.e., `assignment, addition`). This is required when it comes to Collections to determine whether the user intends to append an element to the Collection or to reinitialize the Collection by deleting all previous elements and adding the new one. `condition` supports the definition of a condition that can be interpreted in different ways depending on the type of `source` and `target` elements of the mapping rule (i.e., mapping guard for EClasses and OCL filter for EReferences and EAttributes). `comment` supports the definition of comments to the `MappingRule,` which can help the user keep track of the piece of generated code with the corresponding `MappingRule.` `source` and `target` are optional elements of type EObject that represent the source and target elements of the `MappingRule`. For *immediate mapping rules*, both `source` and `target` must be defined, while for *child mapping rules* there exist three different possible scenarios.

SC1: `source != null` **and** `target != null` - (a non-empty set of input elements in the source model are transformed into a non-empty set of output elements in the target model)
SC2: `source == null` **and** `target != null` - (a non-empty set of output elements are added to the target model)
SC3: `source != null` **and** `target == null` - (a non-empty set of input elements in the source model facilitates the navigation of model elements)

`helperLiteral` is used for EEnumLiterals and is included since EcoreQualifiedNameProvider does not support EEnumLiterals, thus they are not indexed. To surpass this limitation, we need two references; one to the EEnum and the other to the EEnumLiteral. Thus, source or target will be used to reference EEnum and helperLiteral to reference EEnumLiteral. `ChildRules*` is a possibly empty set of elements of type `MappingRule`, while `ChildHelpers*` is a possibly empty set of elements of type `HelperStatement`.

SourceMetamodel and **TargetMetamodel** represent the DSMLs that will be involved in the mapping and inherit all members of `Metamodel`. A **Metamodel** is a tuple `<name, model>`, where name is a unique model name and `model` is the EPackage representing the root element of a particular metamodel involved in the mapping.

A **HelperStatement** is a tuple `<statement, ChildRules*, ChildHelpers*>` where statement is a unique element that allows the user to define statements; for the moment, we support OCL and QVTo statements. `ChildRules*` is a possibly empty set of elements of type `MappingRule`, while `ChildHelpers*` is a possibly empty set of elements of type `HelperStatement`.

Operator is an enumeration with two mutually exclusive possible values, being: `assignment`, used when a single input element in the source model is mapped to a single output element in the target model, or when a non-empty set of input elements in the source model are mapped to a non-empty set of output elements in the target model by re-initializing the set of output elements, and `addition`, used when a non-empty set of input elements in the source model are transformed into a non-empty set of output elements in the target model by adding to the set of output elements.

After defining the metaconcepts of MML, we leverage the features provided by Xtext in combination with EMF to automatically generate textual and tree-based editors. Afterwards, we customize them to provide a more user-friendly and precise scoping as well as more intuitive labeling of the mapped model elements. More specifically, we specialize the MappingRuleItemProvider class, to limit the scope for elements source, target, and EEnumLiteral. Limiting the scope, especially for the source and target, plays a significant role in reducing the likelihood of errors on the part of the user. For instance, the customization of scoping limits the user to defining child mapping rules (i.e., mapping rules that link EReferences, EAttributes, and EEnums) only if there exists a navigation path from the source and target element (i.e., EClass) of the main mapping rule to the source and target of the child mapping rule. Moreover, we specialize the ItemLabelProvider class, to provide intuitive labeling, similar to qualified names. Moreover, we specialize the Formatter class to customize indentation, line breaks, white spaces, etc., to improve the readability of MML textual models.

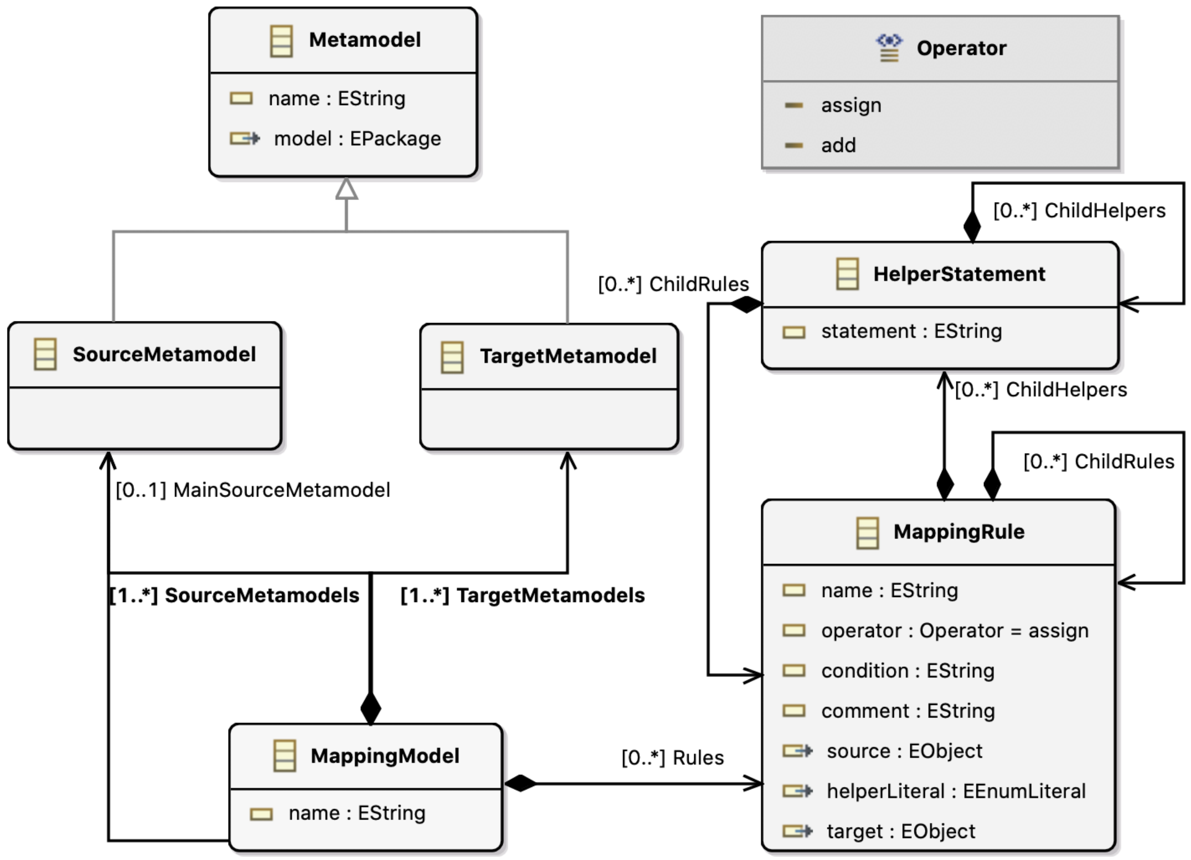


Figure 2 - Mapping metamodel in Ecore

5. Use of MML and its validation

The MML has been exploited to design the higher-order transformations in charge of generating synchronisation transformations (see D4.2) and is employed in most automation aspects between notations in EMF. We have validated MML by applying it to two use cases. The corresponding DSMLs and the respective mapping models can be found in our GitHub repository[2]. The first use case refers

[2] https://github.com/MLJworkspace/BlendedModellingSolution

to the UML-RT language, more specifically, the subset for modelling state machines, where the DSMLs (in the following, we will refer to them as DSML_A and DSML_B) represent the graphical and textual notation of the UML-RT language. The second use case concerns two disjoint DSMLs, one for describing and manipulating calendars, while the other for describing and manipulating organisational structures.

5.1 UML-RT use case

UML-RT is a real-time profile that aims to simplify the ever-increasing complexity of the software architecture specification for real-time embedded systems. UML-RT enables both structure modeling and behavior modeling of real-time systems. This use case focuses on the behavioral part, which is represented using state-machine diagrams. Considering that both DSML_A and DSML_B represent two different notations of the UML-RT language, they contain similar concepts.

For this use case, we have defined two mapping models: **Textual2Graphical** and **Graphical2Textual**. The Textual2Graphical mapping contains a total of 71 mapping rules, of which 66 (93 %) of them fall under SC1, one under SC2 (1.4 %), and four under SC3 (5.6 %). Eight mapping rules contain conditions, of which seven are in the form of guards, as they are applied to mapping rules that link two EClasses, while one is in the form of an OCL filter. The Graphical2Textual mapping contains a total of 61 mapping rules, of which 56 (91.8 %) fall under SC1, five under SC2 (8.2 %), and no mapping rule falls under SC3. 14 mapping rules contain conditions, of which seven are in the form of guards, as they are applied to mapping rules that link two EClasses, while the others are in the form of OCL filters.

Making a comparison between the two mapping models, we notice that the most significant differences are with regard to SC2 and SC3. While in the Textual2Graphical mapping model, only 1.4 % of the mapping rules fall under SC2 (i.e., are used for adding a non-empty set of elements in the output model), in the Graphical2Textual mapping model, 8.2 % of the mapping rules fall under SC2. This is a consequence of the fact that the DSML representing the textual notation contains more concepts that are either not present in the DSML representing the graphical notation (e.g., TransitionBody) or are more specialized (e.g., InitialTransition). The high number of mapping rules that contain conditions in the Graphical2Textual mapping model compared to the Textual2Graphical one is another indicator of the specialization of concepts. With regard to SC3, we notice that while the Graphical2Textual mapping model has no mapping rules falling under this category, in the Textual2Graphical mapping model, 5.6 % of the mapping rules are used to facilitate the navigation of elements in the textual model that cannot be directly accessed.

5.2 Calendar and Organization use case

The second use case relates to two disjoint DSMLs where one is used to describing a meeting calendar for an organization, while the other is used to describe the organization. The **Calendar2Organization** mapping model contains a total of 50 mapping rules, of which 45 (90 %) fall under SC1, one under SC2 (2 %), and four under SC3 (8 %). Eight mapping rules contain conditions, and they are all in the form of OCL filters. Furthermore, this mapping model introduces the use of HelperStatements in the form of for loops and if conditional statements. The **Organization2Calendar** mapping model contains a total of 40 mapping rules, of which 36 (90 %) fall under SC1, two under SC2 (5 %), and two under SC3 (5 %). Ten mapping rules contain conditions, of which seven are in the form of guards as they are applied to mapping rules that link two EClasses, while three are in the form of OCL filters. Furthermore, this mapping model introduces the use of HelperStatements in the form of if-conditional statements.

5.3 Use case comparison

Comparing the distribution of the mapping rules between the three scenarios, in the first use case, the number of mapping rules that fall under SC2 and SC3 is mainly due to the specialisation of concepts, while in the second use case, it is due to semantic and syntactic differences. Despite the fact that there is no significant difference between the number of mapping rules falling under SC1 for the first and the second use case, we still argue that the second use case is more complex than the first, since while in the UML-RT use case, there is a string similarity between the mapped elements of the involved DSMLs and similarity in the structure of the DSMLs, in the second use case such similarities cannot be found. Furthermore, while the first use case covers only a subset of the concepts of the MML, the second use case covers all concepts of the MML, including the HelperStatement and helperLiteral, which we could not validate in the first use case. What adds to the complexity of the second use case is that, while the mapping models for the UML-RT use case exhibit a flatter hierarchy (a maximum of two-level deep-nested hierarchies), the mapping models of the second use case exhibit a deeper hierarchy, reaching a maximum of five-level deep-nested hierarchy. This is the case in the Calendar2Organization mapping model, where the Division2Department mapping rule is made up of a mix of two consecutive HelperStatements and three mapping rules that cover the three scenarios.

6. Conclusion

As part of this deliverable, we described the definition, implementation, and validation of a mapping modeling language, in the context of EMF, through industrial use cases. The MML is developed with a blended modeling approach to support both textual and tree-based notations, which exhibit complementary usability features and encapsulate the minimum set of concepts necessary for defining deterministic mappings, keeping the language concise, and avoiding unneeded verbosity. In addition, the MML has been validated using two use cases which, combined, utilize all of the concepts contained within the MML. In D4.2, the defined mapping models for these two use cases are used as input to higher-order transformations that generate synchronization, migration, and reconciliation transformations.

References

[1] Marcus Alanen, Torbjörn Lundkvist, and Ivan Porres. A mapping language from models to di diagrams. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, Model Driven Engineering Languages and Systems, pages 454–468, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] Arnaud Blouin, Olivier Beaudoux, and Stephane Loiseau. Malan: A mapping language for the data manipulation. In Proceedings of the eighth ACM Symposium on Document Engineering, pages 66–75, 2008.

[3] Guillaume Hillairet, Frederic Bertrand, Jean Yves Lafaye, et al. Bridging emf applications and rdf data sources. In Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering, SWESE, 2008.

[4] Heiko Kern, Vladimir Dimitrieski, Fred Stefan, and Milan Celikovic. Mapping-based exchange of models between meta-modeling tools. 10 2014.

[5] Denivaldo Lopes, Slimane Hammoudi, Jean Bezivin, and Frederic Jouault. Mapping specification in MDA: From theory to practice. In Interoperability of enterprise software and applications, pages 253–264. Springer, 2006.

[6] Srdjan Zivkovic, H Kuhn, and Dimitris Karagiannis. Facilitate modeling using method integration: An approach using mappings and integration rules. 2007.

[7] Latifaj, M., Ciccozzi, F., & Mohlin, M. Higher-Order Transformations for the Generation of Synchronization Infrastructures in Blended Modeling. Frontiers in Computer Science, 4, 166, 2023.

[8] Latifaj, M. (2022, October). The path towards the automatic provision of blended modeling environments. In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (pp. 213-216).

[9] Latifaj, M., Ciccozzi, F., Anwar, M. W., & Mohlin, M. (2022, August). Blended Graphical and Textual Modelling of UML-RT State-Machines: An Industrial Experience. In Software Architecture: 15th European Conference, ECSA 2021 Tracks and Workshops; Växjö, Sweden, September 13–17, 2021, Revised Selected Papers (pp. 22-44). Cham: Springer International Publishing.

[10] Anwar, M.W., Latifaj, M., Ciccozzi, F. (2022). Blended Modeling Applied to the Portable Test and Stimulus Standard. In: Latifi, S. (eds) ITNG 2022 19th International Conference on Information Technology-New Generations. Advances in Intelligent Systems and Computing, vol 1421. Springer, Cham.