frontiers | Frontiers in Computer Science

TYPE Original Research
PUBLISHED 04 January 2023
DOI 10.3389/fcomp.2022.1008062

OPEN ACCESS

EDITED BY
Bernhard Thalheim,
University of Kiel, Germany

REVIEWED BY
Neeraj Kumar Singh,
École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications (ENSEEIHT), France
Giuseppe Destefanis,
Brunel University London, United Kingdom
Mohamed Wiem Mkaouer,
Rochester Institute of Technology, United States

*CORRESPONDENCE
Malvina Latifaj
malvina.latifaj@mdu.se

SPECIALTY SECTION
This article was submitted to Software, a section of the journal Frontiers in Computer Science

RECEIVED 31 July 2022
ACCEPTED 15 December 2022
PUBLISHED 04 January 2023

CITATION
Latifaj M, Ciccozzi F and Mohlin M (2023) Higher-order transformations for the generation of synchronization infrastructures in blended modeling. *Front. Comput. Sci.* 4:1008062. doi: 10.3389/fcomp.2022.1008062

Higher-order transformations for the generation of synchronization infrastructures in blended modeling

Malvina Latifaj[1]*, Federico Ciccozzi[1] and Mattias Mohlin[2]

[1]School of Innovation, Design and Engineering, Mälardalen University, Vasteras, Sweden, [2]HCL Technologies, Malmo, Sweden

Introduction: Blended modeling aims at boosting the development of complex multi-domain systems by enabling seamless multi-notation modeling. The synchronization mechanisms between notations are embodied in model transformations. Manually defining model transformations requires specific knowledge of transformation languages, and it is a time-consuming and error-prone task. Moreover, whenever any of the synchronized languages or notations evolves, those transformations become obsolete.

Methods: In this paper, we propose an automated solution for generating synchronization transformations in an industrial setting.

Results: The approach entails i) the specification of mapping rules between two arbitrary domain-specific modeling languages leveraging a mapping modeling language, appositely defined for this purpose, and ii) the automatic generation of synchronization model transformations driven by the mapping rules.

Discussion: We validated the proposed approach in two use cases. Although our main goal was to provide a solution for synchronization between graphical and textual notations of UML-RT state machines, the proposed approach is language- and notation-agnostic.

KEYWORDS

blended modeling, multi-notation, automatic generation, model transformations, higher-order transformations, mapping modeling language

1. Introduction

Demands on software functionality and quality increase at a very fast pace, and the interconnected nature of software-intensive systems makes complexity of software grow exponentially. A rather direct consequence is that the time and costs for software development increase notably. Model-Driven Engineering (MDE) has been largely adopted in industry as a powerful means to effectively tame the complexity of software-intensive systems and their development, as shown by empirical studies (Hutchinson et al., 2011), by using domain-specific abstractions formalized in domain-specific modeling languages (DSML). DSMLs allow domain experts, who may not be software experts, to describe complex functions in a more domain-focused and human-centric way than if using traditional programming languages. DSMLs formalize the *communication* language of engineers at the level of domain-specific concepts such as an

engine and wheels for a car. These concepts may not exist in other domains. Moreover, DSMLs support more efficient integration of software with designs and implementations of other disciplines. Domain-specific modeling demands a high level of customization of modeling tools, typically involving combinations and extensions of DSMLs and tailoring of the modeling tools for their respective development domains and contexts. Furthermore, tools are expected to provide multiple modeling means, e.g., textual and graphical, to satisfy the requirements set by different development phases, stakeholder roles, and application domains.

However, domain-specific modeling tools, especially those based on the Unified Modeling Language (UML) and its profiles (as in the industrial tool addressed in this paper), traditionally focus on one specific notation, which is most often graphical or textual. This limits human communication, especially across engineering disciplines. A notation that is well understood by one engineering discipline may not be as easily understood by engineers from another discipline. Moreover, engineers from the same or different disciplines may have different notation preferences; not supporting multiple notations negatively affects the throughput of engineers. Besides the limits to communication, choosing one particular notation also limits the pool of available tools to develop and manipulate models that may be needed. For instance, choosing a graphical notation limits the usability of text manipulation tools such as text-based diff/merge, which is essential for team collaboration. This mutual exclusion suffices the needs of developing small-scale applications with only few stakeholder roles.

For larger systems, with heterogeneous components and entailing different domain-specific aspects and different types of stakeholders, mutual exclusion of notations is too restrictive and voids many of the benefits that MDE can bring about. When applying MDE in large-scale industrial projects, efficient team support is crucial. Therefore, modeling tools need to allow different stakeholders to work on overlapping parts of models using different concrete syntaxes or simply notations. In addition, the diversity of stakeholders leads to the need for domain-specific editing facilities, which can be graphical, table-based, form-based, and for many domains also textual (e.g., formal verification Lilius and Paltor, 1999).

1.1. Blended modeling

We have defined the notion of blended modeling in a previous work (Ciccozzi et al., 2019) as follows:

> *Blended modeling is the activity of interacting seamlessly with a single model (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes), allowing a certain degree of temporary inconsistencies.*

Blended modeling is expected to aid in keeping the cognitive flow of modeling effective and efficient, offering stakeholders a proper set of intertwined formalisms, notations, and supporting computer-aided mechanisms. This is important in the design of modern systems, as their complexity has been increasing exponentially over the past years (Persson et al., 2013).

At first sight, the notion of blended modeling may seem similar to or overlapping with multi-view modeling (Cicchetti et al., 2019) (and even multi-paradigm modeling) that is based on the paradigm of viewpoint/ view/ model as formalized in the ISO/IEC 42010 standard.[1] Multi-view modeling is commonly based on viewpoints (i.e. "conventions for the construction, interpretation, and use of architecture views to frame specific system concerns" Emery and Hilliard, 2009) that are materialized through views, which are composed of one or more models. In blended modeling, the focus is *not* on identifying viewpoints and related views, but rather on providing multiple blended editing and visualizing notations to interact with a set of concepts.

The blended modeling paradigm focuses on the provision of multiple concrete syntaxes, or simply *notations*, for a non-empty set of abstract syntactical concepts. As such, it aims to accommodate different notations, each designed for particular modeling needs. The main implication of this definition of blended modeling is that it assumes a single abstract syntax supported by multiple concrete syntaxes. However, although this definition is theoretically correct, in reality, language-specific modeling frameworks can benefit from multiple abstract syntaxes for the following reasons.

To begin with, the definition and management of a concrete syntax may be simpler if directly related to a dedicated abstract syntax. Relating this to our industrial case, where we started from a graphical concrete syntax, the first step involved the definition of a textual concrete syntax by first defining a dedicated abstract syntax. This was needed since the two abstract syntaxes were not fully matching, and it generally allows for high syntax-specific customizations. Furthermore, since notations are usually meant to be highly customizable to user needs, and each notation serves a different purpose, often at a different level of detail, it may not be practical to adapt an existing abstract syntax supporting one notation to another. In addition, depending on the needs of two different users, there might exist two "different" notations of the same type (e.g., two different textual notations focusing on different aspects of the same modeling concepts tailored to two different user types). Having a dedicated abstract syntax per notation alleviates possible syntactical "pollution" caused by reusing and adapting an existing abstract syntax, which was not envisioned for that particular notation. Moreover, there exist scenarios where different notations are represented by different existing notation-specific DSMLs, formalizing the same underlying language with a significant overlap, to serve the

1 https://www.iso.org/standard/50508.html

needs of different communities, stakeholders, and/or purposes. Therefore, in practice, blended modeling is not always limited to seamless interaction with a single abstract syntax through multiple notations, but it rather entails more complex cases.

The representative scenario of our work is a single underlying language (set of concepts) formalized through multiple abstract syntaxes. We define this as *extended blended modeling*. In this case, the abstract syntaxes may represent either two partly overlapping formalizations of the same DSMLs or even two entirely different DSMLs, provided that they are in some way related to each other.

1.2. Our contribution

Our overall contribution is the means to automate the definition of synchronization mechanisms across multiple notations, regardless of whether the underlying abstract syntaxes associated with the notations represent the same or disjoint languages. The mechanisms described in this paper were conceived for automating the engineering of synchronization transformations across multiple notations of the same language (UML-RT), but they have a broader applicability since they can produce synchronization transformations across notations of different languages as well as means for co-evolution across different versions of a language. Technically, we provide a solution for modeling environments based on the Eclipse Modeling Framework (EMF) (Steinberg et al., 2008) and DSMLs described using EMF's meta-metamodel, Ecore.

1.3. Paper organization

The remainder of the paper is organized as follows. The industrial setting and supporting arguments on the motivation behind this work are described in Section 2. We discuss work related to the problem domain in Section 3. The developed approach and implementation are described in detail in Section 4. Experiences from validating the approach are included in Section 5. A discussion is provided in Section 6 and the paper is concluded in Section 7.

2. Industrial setting, core problem, and expected benefits

The research work described in this paper was carried out in cooperation with HCL Technologies, which offers an industrial Eclipse-based modeling tool, RTist,[2] for the development of complex, event-driven, and real-time software. The tool is designed to support UML and its real-time profile (UML-RT). More specifically, the tool provides support for specifying UML-RT architectures and applications by means of graphical composite structure diagrams, for modeling structural information, and state-machine diagrams, for modeling behavior. Furthermore, the tool provides specific features to complement models with fine-grained algorithmic behaviors by embedding C/C++ action code in state-machines.

The long-term goal of HCL behind this research effort is to improve the process of engineering software applications by enabling developers, which will also be referred to as users in the remainder of the paper, to work on overlapping parts of a model using different modeling notations (i.e., graphical and textual) seamlessly in the same modeling environment. Although the ultimate goal of this effort is to provide a blended modeling environment for the entire UML-RT language, in this paper we focus on the most complex part, namely the provision of a flexible solution for blended modeling of UML-RT state-machines. Starting from the canonical graphical concrete syntax for UML-RT state machines, the provision of a blended modeling environment with seamless synchronization can be broken down into two main steps which we have successfully carried out in Latifaj et al. (2021, 2022). More specifically, in Latifaj et al. (2021) we describe the effort of designing, implementing and integrating a textual notation for UML-RT state machines in RTist. In Latifaj et al. (2022) we contribute with the customization of the textual editor for UML-RT state-machines with advanced formatting features including systematic support for hidden regions which group hidden tokens (e.g., comments, whitespaces) between two semantic tokens and the provision of synchronization mechanism between textual and graphical notations. However, the synchronization mechanisms were manually defined in terms of model transformations between two specific DSMLs describing textual and graphical notations. If any of the two concrete syntaxes underwent a change, the mechanisms became obsolete.

The specific industrial aim of this work was instead to provide an automated solution for generating synchronization infrastructures between potentially evolving concrete syntaxes of UML-RT. To allow for evolution of the entailed DSMLs and the co-evolution of the synchronization mechanisms, in this work we contributed with the design and implementation of a mechanism for the automatic generation of the infrastructure required for seamless synchronization, i.e., model transformations, between virtually any pair of Ecore-based DSMLs (not only graphical and textual UML-RT state-machines), that may or may not represent two different concrete syntaxes of the same language. The provision of automatic means for generating model synchronization transformations from two given DSMLs that may or not represent two versions of the same language simplifies remarkably the life of modeling tool developers, but also allows domain experts without particular knowledge in model transformations to practically

2 https://www.hcltech.com/software/rtist

put in place the infrastructure needed for synchronization purposes. In our specific industrial setting, our approach brings the following benefits.

- It provides the means for seamless synchronization of the standard graphical and the newly introduced textual concrete syntaxes for UML-RT state-machines. Being a standard language, UML-RT is unlikely to evolve frequently. However, customers require viewing and editing UML-RT models using various specialized notations, each described by a specific DSML. These DSMLs are tailored to customer needs and, unlike the underlying UML-RT based language, they may be subject to frequent changes. Thanks to our solution, as soon as any of the notations undergoes a change, the synchronization mechanisms can be regenerated with a minimal mapping effort from the developers. Without our approach, the actual model synchronization transformations would need to be manually edited by the developers, which is clearly a risky, error-prone, and time-consuming task. Our solution gives architects and developers the possibility to experiment when extending or evolving either concrete syntax.
- Similarly, in case the UML-RT language itself would evolve, alongside its concrete syntaxes, without a solution like this based on automatic generation of synchronization, all transformations would need to either be co-evolved manually, which is again, an error-prone and time-consuming effort, or re-written from scratch in case of deep changes to the language and/or the related concrete syntaxes. Our solution eases this process and provides the means for a more flexible and "fast prototyping" kind of modeling language and tool engineering process. Engineers and developers can sketch changes to either of the concrete syntaxes and try them out with automated generated synchronization, too.
- If any other language would be included in the tool ecosystem for, e.g., modeling multi-domain systems, alongside UML-RT, our solution aids in establishing synchronization infrastructures between them, pairwise. Automation gives flexibility but also the possibility to "try out" alternatives without having to spend much effort and time writing and validating synchronization transformations, and instead focus on the languages, their concrete syntaxes, and how they are supposed to interact.

3. Related work

Prior to describing the literature related to our work and comparing other approaches with ours, we want to emphasize that not all solutions dedicated to the automatic generation of model transformations relate to our research. For instance, we limit our focus to the automatic generation of horizontal model transformations and do not analyze approaches toward the automatic generation of vertical model transformations as described in Ráth et al. (2010), or generation of model transformations for the exchange of models between meta-modeling tools such as Kern et al. (2014), since the mapping correspondences are defined between elements of M3 level models, while we target the specification of mapping correspondences between elements of M2 level models.

3.1. Blending graphical and textual editors

With respect to the proposed solutions dedicated to the blending of textual and graphical editors, a large portion of tools that offer graphical and textual notations such as Charfi et al. (2009), Umple by Lethbridge et al. (2021), Excalibur by Ries et al. (2018), Light UML,[3] MetaUML,[4] PlantUML,[5] or FXDiagram,[6] provide a limited set of features as one of the notations is read-only and is only used for visualization purposes; thus editing the model *via* multiple notations is not possible and that violates the base notion of blended modeling that allows interacting (i.e., write and read) with the model *via* multiple notations.

For another category of tools, the notations are predefined and cannot be customized, and the solution is language-specific. Alternatively, our approach is language-agnostic, meaning that the synchronization mechanisms can be generated for arbitrary DSMLs. For instance, Maro et al. (2015) provide a solution for the semi-automatic generation of textual editors from UML profile-based DSMLs and the implementation of synchronization mechanisms with the existing graphical editor; however, the developed transformations are specific to the considered UML profile. Addazi and Ciccozzi (2021) propose a blended modeling framework, but the solution is specific to UML-based DSMLs. Lazăr (2011) develops a textual editor for the Action Language for Foundational UML (Alf), but the editing capabilities are restricted only to some parts of a UML model, thus they do not cover the complete model. Scheidgen (2008) proposes embedded textual editors for existing graphical models, but the solution only provides pop-up boxes to textually edit elements of graphical models rather than allowing seamless editing of the entire model.

On another note, while the majority of tools intermixing between graphical and textual editors do so in a parser-based fashion, tools such as JetBrains MPS[7] and MelanEE (Atkinson

3 http://lightuml.sourceforge.net

4 https://github.com/ogheorghies/MetaUML

5 https://plantuml.com

6 https://jankoehnlein.github.io/FXDiagram/

7 https://www.jetbrains.com/mps/concepts/

and Gerbig, 2016) follow a projectional approach where the abstract syntax tree (AST) is modified directly upon every change, and the changes are automatically reflected in the different concrete syntaxes that are visualized as projections. This bypasses the stages of parser-based approaches where the parser must first check the correctness of the syntactic aspects and then construct the AST from the character sequence that users input through text editors. In terms of textual notations, tools that follow a projectional approach only imitate the behavior of parser-based textual editors, and are actually limited to a fixed format. Lastly, the interested reader can find a more extensive systematic review of solutions dedicated to the blending of multiple notations in David et al. (2022). These solutions, however, are based on the concept of only one abstract syntax, whereas our focus is on multiple abstract syntaxes that may represent two partially overlapping formalizations of the same DSML or even two completely different DSMLs as long as they are related in some way.

3.2. Model weaving

Model weaving allows the definition of relationships and correspondences between metamodel elements in a weaving model, and allows the execution of operations based on them (e.g., a model transformation can be automatically generated based on a weaving model) (Bézivin, 2005). Several publications in the literature have been devoted to efforts to automate the generation of model transformations by means of weaving models. Lopes et al. (2006) propose a mapping metamodel based on the Eclipse Modeling Framework (EMF), and contribute with tools that enable the generation of model transformations conforming to the Atlas Transformation Language (ATL) from a mapping model. Didonet Del Fabro and Valduriez (2009) build on that work and propose a solution to use matching transformations for the creation of weaving models that can automate the production of executable model transformations. These approaches focus on the semi-automatic creation of weaving models and their manual adjustment for semantically and syntactically similar languages, and the manual creation of weaving models for semantically and syntactically different languages for several purposes, including model transformations. However, relying only on metamodel data to create weaving models (i.e., mapping models) without considering the developer's intentions does not guarantee the accuracy of the mapping model with respect to the requirements. The use of inaccurate and ambiguous weaving models may result in incorrect model transformations that do not meet the initial requirements. Manual verification and adjustment of an extensive weaving model can be as challenging as finding a needle in the haystack. This might lead to the creation of mapping models being simpler than manually fine-tuning automatically generated ones. In addition, while one may argue that the weaving approaches provide a flexible and automated way to derive mapping models, they may not be able to properly deal with semantic differences among the mapped languages (i.e., semantics often needs human understanding to be correctly managed). In our setting, the mapping modeling language is not the main focus, but rather a key enabler for the overall goal being the definition of powerful higher-order transformations (HOTs) for generating synchronization transformations and addressing challenging cases, such as synchronization between different languages.

In a nutshell, by allowing for more complex unambiguous mappings, we can cater to a wider range of languages, and provide powerful mechanisms to support the translation of these mappings to model transformations, thus the generation of the synchronization infrastructure between languages and notations. By doing so, we also increase the generalizability of our approach. Lastly, from a usability point of view, these solutions tend to provide a tree-based editor only, while providing an additional textual editor can prove useful thanks to its syntax-agnostic editing features.

Ecore2Ecore[8] is a plugin, distributed with EMF, that was originally implemented with the goal of supporting metamodel evolution. As it is possible to define mappings between two metamodels, we presume that it could be used to define mapping models that, in turn, can be used to generate language-specific model transformations. However, the solution does not provide mechanisms for the generation of model transformations. On another note, Diskin et al. (2017) propose a theoretical framework where traceability *mappings are regarded as a core aspect of transformations definition and management* and our approach can be considered a materialization of this message. Blouin et al. (2008) propose Malan, a MApping LANguage that supports mutually exclusive graphical and textual definitions of schema mappings in Papyrus. The source and target schemas are expressed as UML class diagrams, and the solution only generates XSLT stylesheets that convert XML documents into other formats, such as HTML or plain text. Hillairet et al. (2008) propose Mapping Ecore-OWL, a textual mapping language that defines correspondences between EMF objects and RDF resources. The approach generates ATL transformations that enable the use of RDF resources as EMF objects and the serialization of EMF objects in RDF resources. While the last two solutions provide mapping languages and semi-automatic approaches for the generation of model transformations, in contrast to our work, they do not provide support for Ecore-based DSMLs.

8 https://eclipse.googlesource.com/emf/org.eclipse.emf/+/R2_8_3/plugins/org.eclipse.emf.mapping.ecore2ecore/

4. Proposed solution

Consider the model of our solution depicted in Figure 1. Depending on what the involved pair of DSMLs represents, we focus on two different scenarios:

1. $DSML_A$ and $DSML_B$ represent two notations of the same language (e.g., graphical and textual UML-RT state-machines), then the generated M2M transformations provide synchronization across different notations of the same language.
2. $DSML_A$ and $DSML_B$ are disjoint, then the generated M2M transformations provide synchronization across different notations of different languages.

Our approach was designed and implemented with open-source technologies in the Eclipse Modeling Framework (EMF)[9] ecosystem and can thereby be leveraged by any EMF-based tool, as for the RTist case. More specifically, we provide a semi-automatic approach where developers are relieved from writing model synchronization transformations, and, instead, focus their efforts in describing how they want concepts across DSMLs to be mapped using a specifically defined mapping modeling language. There are two main contributions to our approach, since to generate synchronization transformations, the user first needs to be given the means to define the relationships between concepts from both notations, and second the user needs to be given the means to generate synchronization transformations based on the defined relationships. Therefore, given a pair of DSMLs, $DSML_A$ and $DSML_B$, defined in terms of Ecore, our first contribution (C1), is an Ecore-based Mapping Modeling Language (MML), which gives the user the ability to simply model mapping rules between the two DSMLs. Mapping models constitute the only manual input required for the approach to generate synchronization transformations; thereby, it is crucial that the information in these models is correctly captured and unambiguous. For our second contribution (C2), Higher-Order Transformations (HOTs) implemented using Xtend[10] take as input the instantiated mapping models that capture the mapping rules and use $DSML_A$ and $DSML_B$ to resolve the references of the mapped elements and generate synchronization mechanisms between the two DSMLs in terms of two unidirectional model transformations conforming to the QVT operational (QVTo) language.[11] In the remainder of this section, we provide details on the definition of MML and HOTs together with a rationale behind the choices made in the process and details on how we implemented them. The complete implementation can be found in our GitHub repository[12]).

9 https://www.eclipse.org/modeling/emf/

10 https://www.eclipse.org/xtend/

11 https://wiki.eclipse.org/QVTo

12 https://github.com/MLJworkspace/BlendedModellingSolution

4.1. Mapping modeling language

We refer to a mapping language as a structured and formalized means for the specification of mapping rules between two or more DSMLs. The definition of the mapping language is given in terms of a metamodel; thus, it can also be defined as the correspondence of elements between arbitrary metamodels (Lopes et al., 2006). A mapping language provides a fundamental input to correctly synchronize models conforming to different DSMLs, as explicit mapping rules link multiple DSMLs deterministically. In our specific case, mapping rules in those models drive the HOTs to properly generate model transformations conforming to QVTo. The mapped DSMLs shall conform to the Ecore meta-metamodel and may represent two different notations of a same language, as in our UML-RT state-machines use-case.

Although more intuitive and easier to interact with than complex model transformations, MML is still intended for users with meta-modeling knowledge. Understanding of the meta-modeling concepts and structure is essential to properly describe how concepts between DSMLs are intended to be mapped. Definition of mapping rules instead of manually writing model transformations is particularly useful for domain experts with no specific knowledge of model transformation languages, but also for developers who can benefit from a semi-automatic, more accurate, and less cumbersome approach for establishing synchronization mechanisms. Practically, domain knowledge is the only required input. Also maintenance of the generated model transformations in response to evolving DSMLs or requirements can be performed by domain experts, since adjustments only need to be made at the level of the mapping models, while HOTs would use them to regenerate model transformations. That is to say that developers are not expected to manually edit generated model transformations at any point in the process.

As part of our effort in defining MML, we conducted an analysis to identify the input required to correctly generate model transformations. Considering that one of the main goals of this study was to minimize the amount of manual input required from the user, we first identified the maximum set of information that could be automatically retrieved from DSMLs ($DSML_A$ and $DSML_B$). For instance, the mapping rule type (i.e., abstract or non-abstract) is calculated based on whether the `source` element is abstract or not. Moreover, disjunctive/disjuncted mapping rules and inheriting/inherited mapping rules are automatically generated by examining the hierarchical structure of the involved metamodels. The user is also relieved from invoking the corresponding mapping rules as they can also be retrieved automatically. The assignment operator is automatically generated in the case of mono-valued attributes or properties (i.e., :=), while in the case of multi-valued elements the user must manually define it based on whether the goal is to add elements to the collection (i.e., +=) or to reinitialize

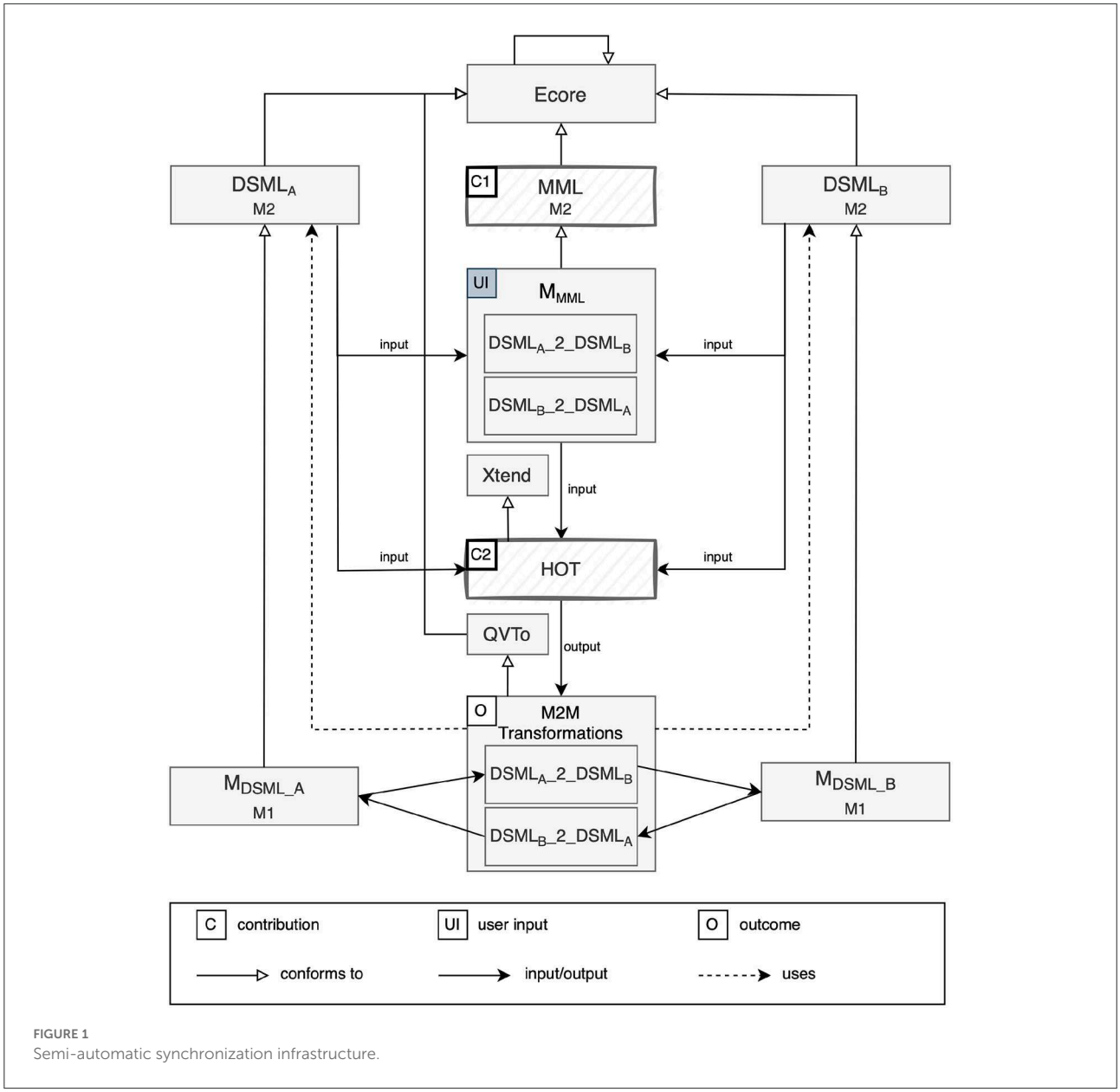

FIGURE 1
Semi-automatic synchronization infrastructure.

the collection with the element, dropping all previous elements (i.e., :=). Navigation operators (. and ->) are also automatically generated by assessing whether the source is a single element or a collection of elements. In addition, navigation paths are also automatically generated when they involve containment references or when there is a single non-containment reference between two EClasses. The remaining information would have to be manually provided by the user. Based on this, we defined the concepts to be included in the MML metamodel.

Once identified the concepts that MML should include, the last step was to implement it, focusing on abstracting away the implementation—specific details (e.g., syntax and semantics) of model transformation languages, and allowing the user to focus exclusively on the specification of mapping rules. To comply to our overall settings, we implemented the MML in a blended modeling fashion, allowing the user to interact with MML *via* both textual and tree-based editors. Blending for MML is useful, since it combines the strength of text for syntax-agnostic editing operations, such as copy and paste, search and replace, auto-complete features, and a good integration with widely used versioning and configuration tools, with clear structural overview features typical of tree-based editors.

We assume that MML users have experience with at least one object-oriented programming (OOP) language; therefore, several syntactical features of MML are similar to those of OOP languages. In addition to that, users have an advantage if

they have some knowledge of the Object Constraint Language[13] (OCL), since in certain cases mapping rules require the specification of conditions for the correct navigation of elements and for the expression of the so-called "guards" in the generated transformations.

MML is developed using the Xtext language workbench.[14] Xtext relies on EMF and it generates an Ecore model that represents the abstract syntax tree (AST), lexer, parser, and the corresponding Java code. In Figure 2 we describe the MML metamodel defined as an Ecore model, and in the following we detail the metaconcepts of 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 `SourceMetamodel`s 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*>` where `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 mapping rule which can help the user keep track of the piece of generated code with the corresponding mapping rule. `source` and `target` are optional elements of type `EObject` that represent the source and target elements of the MappingRule. `helperLiteral` is used for `EEnumLiteral`s and is included since `EcoreQualifiedNameProvider` does not support `EEnumLiteral`s, 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 transformation 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 into 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. This is particularly useful in the tree-based editor for distinguishing between different metaelements that may have the same name. Moreover, we specialize the `Formatter` class to customize indentation, line breaks, white spaces, etc., to improve the readability of MML textual models.

4.2. Higher-order transformation

The automatic generation of model transformations for synchronization purposes is achieved by means of HOTs. According to their definition (Tisi et al., 2009), HOTs are

13 https://wiki.eclipse.org/OCL

14 https://www.eclipse.org/Xtext/

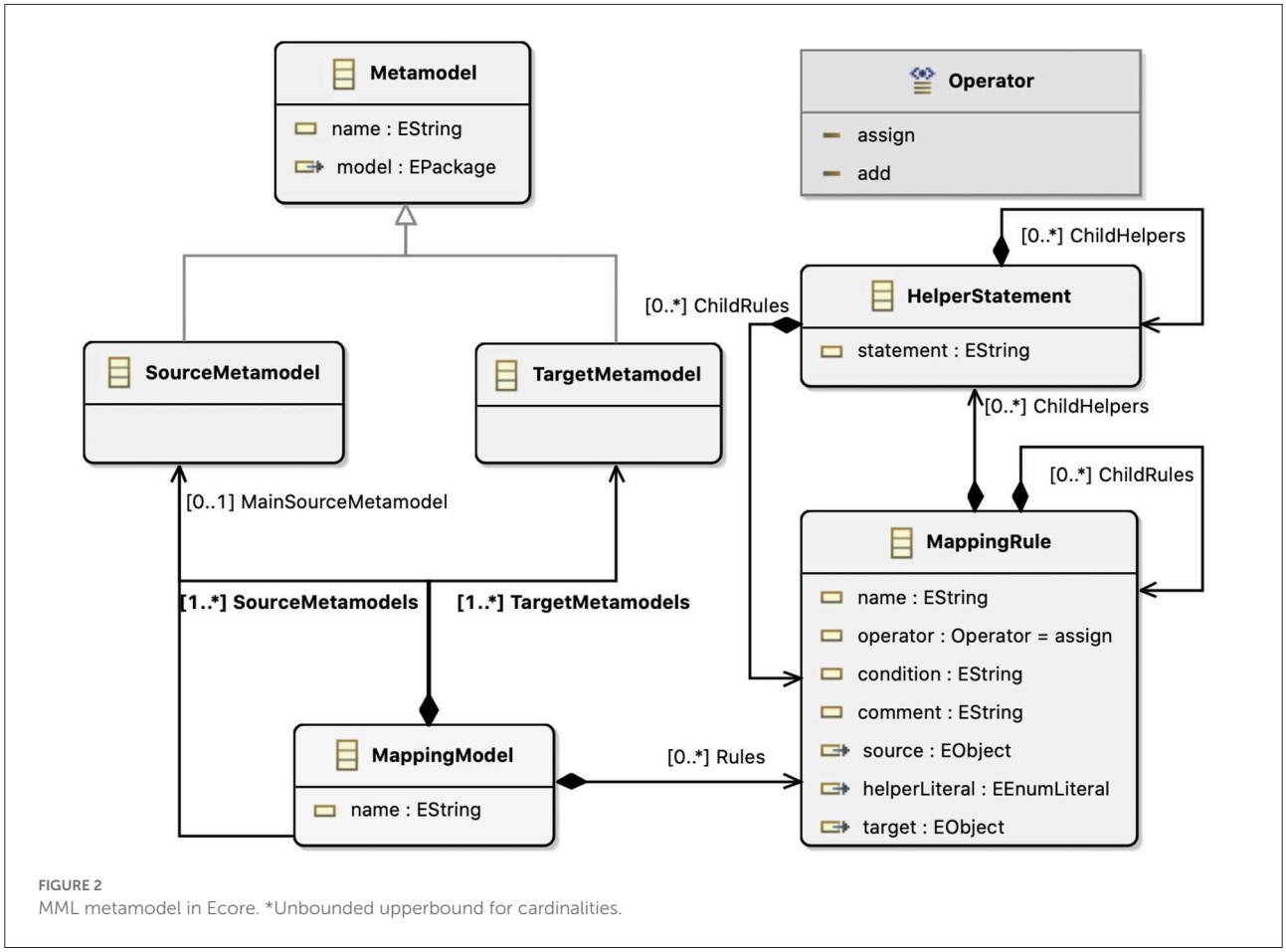

FIGURE 2
MML metamodel in Ecore. *Unbounded upperbound for cardinalities.

particular model transformations that generate, in turn, model transformations. In our case, HOTs are defined at meta-metalevel using the Xtend language and automatically generate unidirectional model transformations in QVTo for synchronization purposes. The synchronization infrastructure generated by the HOTs consists of two unidirectional model transformations. As depicted in Figure 1, starting from two DSMLs, $DSML_A$ and $DSML_B$, and two sets of high-level mapping rules between them defined in two mapping models, one per direction (i.e., $DSML_A\_2\_DSML_B$ and $DSML_B\_2\_DSML_A$, the HOTs generate two unidirectional QVTo transformations that, when executed, take a model instance of one DSML, $DSML_A$ and $DSML_B$ respectively, and transform it into a model instance of the other DSML, $DSML_B$ and $DSML_A$ respectively. Each mapping rule defined in the mapping models is transformed into one or more mapping operations in the generated QVTo transformations. The choice of QVTo as target transformation language was due to its suitability for both in-place and out-of-place, as well as endogenous and exogenous transformations, and for its imperative programming fashion, which is particularly suitable for automatic generation of complex algorithms. Moreover, we opted for multiple unidirectional transformations rather than bidirectional transformations to simplify the maintainability of the generated transformations and their manageability in the target modeling tool ecosystem.

The HOTs are implemented in Xtend, a flexible dialect of Java, which compiles into readable Java 8 compatible source code and is particularly suitable for the generation of pretty-printed textual artifacts. The remainder of this section is structured in paragraphs corresponding to the different metaconcepts of MML to maximize readability. There, we describe how the HOTs combine the input specified by the user in the mapping models with the information automatically extracted from the mapped DSMLs in order to generate the model transformations for synchronization.

Mapping Model: `MappingModel` is the root element of the mapping language and represents the starting point for traversing a mapping model. The user assigns a name to it, which is then used to generate the name of the model transformation. If there is more than one `SourceMetamodel`, the user must select the `MainSourceMetamodel`, which represents the metamodel that is used as the entry point of the model transformation. Alternatively, in the case of only

one `SourceMetamodel`, the latter is automatically selected as `MainSourceMetamodel`.

Metamodels: The information extracted from `SourceMetamodel` and `TargetMetamodel` is used to generate the `modeltype`, and `transformation signature` of the model transformations. The user loads the DSMLs and selects the EPackages to be mapped that will be used as the source and target of the model transformation. The HOTs automatically retrieve the name and nsURI of the EPackages to generate the `modeltype`, identify the direction of the model transformation, as specified in the mapping model, and generate parts of the transformation signature according to the following pattern:

```
in «SourcePackageName»Model :
«SourcePackageName»,
out «TargetPackageName»Model :
«TargetPackageName»
```

Mapping Rule: The mapping rules can be grouped based on the EObject that contains them as follows.

1. `MappingRule`s are contained in `MappingModel` and we refer to them as *immediate mapping rules*. `source` and `target` of these mappings are objects of type EClass.
2. `MappingRule`s are contained in other `MappingRule`s or `HelperStatement`s and we refer to them as *child mapping rules*. `source` and `target` of these mappings are objects of type EReference, EAttribute, or EEnum.

The mapping rules in the first category are used to generate the mapping declaration, whereas those in the second are utilized to generate the body of the mapping operations. In the following, we detail the implementation of the features that apply to each category.

4.2.1. Immediate mapping rules

Mapping operation name: The name of the mapping operation is automatically generated as: `«sourceElementName»2«targetElementName»`. This not only reduces the amount of manual effort from the user, but it also increases readability, as the naming follows a specific standard pattern and is rather intuitive. Moreover, to minimize the risk of errors when mapping elements with the same name, source and target elements are printed using fully qualified names (i.e., `modelName::elementName`), thanks to our customized model editors.

Mapping operation type: The generated mapping operations can be abstract or non-abstract. Abstract mapping operations are used when the target of the mapping operation is abstract. This information is automatically extracted from the target DSML; hence, it does not require user input. Before printing a mapping rule, the HOTs determine whether the target element of the mapping rule is an abstract or non-abstract EClass. In the case of an abstract EClass, the mapping operation is printed as abstract according to the following pattern: `Abstract«sourceElementName»2-«targetElementName»`.

Conditions: For immediate mapping operations, the source and target elements are EClasses, therefore conditions that are manually defined by the user are automatically generated as `when` clauses that are evaluated to determine in which circumstances the mapping operation should be executed.

Inheritance: The concept of inheritance allows reuse of mapping operations under the condition that the signature of the inherited mapping conforms to the one of the inheriting mapping. The source and target of any potential inherited mapping rule must be supertypes of, or identical to, the source or target of the inheriting mapping rule. The HOTs iterate through each of the immediate mapping rules in the mapping model and determine whether the mapping operation under analysis inherits from any of the iterated mapping rules. If it does, after the transformation signature and the **inherits** keyword, the names of the inherited mapping rules are printed (in the case of multiple inherited mapping rules, they are separated by a comma).

Disjunction: Invocation of a disjunct mapping operation results in an assessment of disjunct candidate mapping operations. To determine whether a mapping operation is disjunctive and, if so, to identify the disjunct candidates, the HOTs iterate through all the immediate mapping rules of the mapping model and identify those where the source and target are identical or subtypes of the source and target of the potentially disjunctive mapping rule. If these mapping rules exist, the analyzed mapping rule is considered disjunctive and is named according to the following pattern: `«sourceElementName»2«targetElementName» Disjunct`.

After printing the signature of the mapping operation and the disjuncts keyword, the HOTs print the identified disjunct candidates. A mapping can be both abstract and disjunctive. The user needs to define the mapping rule only once in the mapping model and the HOTs will generate two rules: one abstract and one disjunctive, since QVTo does not allow to combine them into one.

4.2.2. Child mapping rules

There are three different possible scenarios for child mapping rules, depending on the values of the source and target attributes.

SC1: `source!=null and target!=null`
SC2: `source==null and target!=null`
SC3: `source!=null and target==null`

In SC1 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. In SC2 a non-empty set of output elements are added to the target model. In SC3 a non-empty set of input elements in the source model facilitates the navigation of model elements in the generated transformations. This is used for complex and possibly ambiguous navigation cases, such as the one depicted in Figure 3, where on the left-side is illustrated an excerpt from the source metamodel and on the right side an excerpt from the target metamodel.

Consider that the user defines an immediate mapping rule `Organization2Company` as detailed in Figure 4. The user then wants to map the value of the `name` attribute of `Person` in the source metamodel, to the `managerName` attribute of `Company` in the target metamodel, by defining the mapping rule `name2managerName`. The correct navigation path in this case would be `self.department.ma-nager.name`, where `self = Organization`. To navigate to the `name` attribute, starting from `Organization`, there is, in fact, also another path; `self.department.secretary.name`. However, this navigation path is not considered correct, as it would map the name of a `Person` that is a secretary in `Department` to `managerName` in `Company`. Therefore, the information to navigate to `name` attribute *via* `manager` reference is to be decided by the user, as the HOTs cannot automatically determine which path to take. Therefore, the user needs to define an additional mapping rule `manager2null` that will guide the HOTs in generating the expected model transformation. Being that `manager` is not an immediate reference of `Organization` HOTs must automatically generate the navigation path from `Organization` to `manager`. For that reason, we implement a recursive Depth-First Search (DFS) algorithm, which starts at the root node (i.e., `Organization`) and explores as far as possible along each EClass, before backtracking (unless it finds the target).

Invoking rule: In QVTo, mapping operations are run with an explicit rule-invocation style, which initiates execution from an entry mapping operation generally found in the main function, and invokes the other mapping operations in a nested manner. The entry mapping operation that should be invoked is automatically determined.

OCL expressions: For sub-mapping operations, derived from child mapping rules, where the source and target elements are EReferences or EAttributes, `condition` attributes of the mapping rules are used to specify OCL expressions.

Navigation operators: The HOTs determine the navigation operator based on whether the source of the mapping rule is a single object or a collection of objects (i.e., by checking the `upperBound`). A single object is navigated using the *dot* (.) operator, whereas collections of objects are navigated using the *arrow* (->) operator.

HelperStatement: It is intended to facilitate the definition of complex mappings requiring the use of `for` loops, `while` loops, or `if/else` conditional statements. `HelperStatements` are contained in `MappingRules` (i.e., immediate mapping rules) similarly to how they are defined in mapping operations in QVTo transformations. Moreover, they may contain other `HelperStatements` and `MappingRules` that are generated within the loop or statement defined by `HelperStatement`.

5. Validation and use cases

The proposed approach and reference implementation have been validated by means of two use cases and model-to-text testing. During this process, the implementation of our solution was validated in multiple testing phases and the results of each phase were analyzed and used, when needed, to tune the implementation. In Section 5.1 we provide details on the two use cases in isolation and then conduct a comparison between the two, while in Section 5.2 we provide the details of the model-to-text test cases. Lastly, in Section 5.3 we provide an example of our industrial use case.

5.1. Use cases

The first use case refers to the UML-RT language, more specifically the subset for modeling state-machines, where $DSML_A$ 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 organizational structures. Both use cases encompass scenarios entailed by our solution.

5.1.1. 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. As a result of textual concrete syntax requirements aimed at maximizing usability and reducing learning curves, the different DSMLs for different notations are required. In fact, the DSML associated with the textual notation has evolved to fit the needs of various customers, so we have been able to also support the co-evolution of model transformation in response to DSML changes. In the following, we provide more details on the mapping models and generated QVTo transformations.

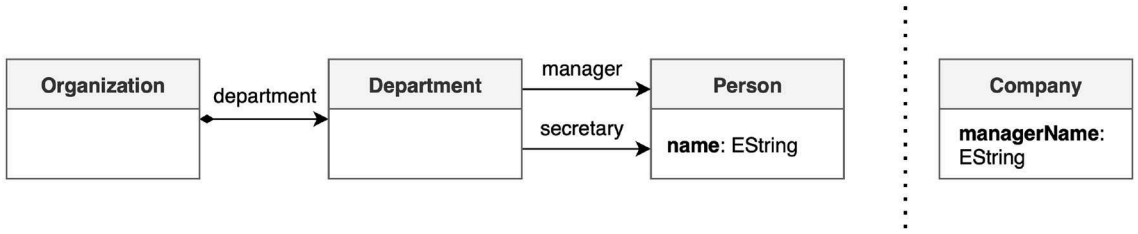

FIGURE 3
Ambiguous navigation.

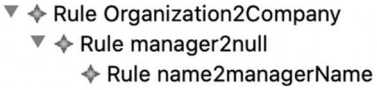

FIGURE 4
Ambiguous navigation mapping rules.

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. This mapping model generates a total of 29 main mapping operations in the output QVTo transformation from a total of 25 mapping rules that link EClasses. The four additional mapping operations are the result of the `abstract` and `disjunctive` mappings that are automatically calculated from the HOTs.

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 other are in the form of OCL filters. This mapping model generates a total of 26 mapping operations in the output QVTo transformation from a total of 22 mapping rules that link EClasses. The same reasoning as in the case of `Textual2Graphical` mapping applies for the four additional mapping operations.

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.

5.1.2. Calendar and organization use case

The second use case relates to two disjoint DSMLs where one is used to describe a meeting calendar for an organization, while the other is used to describe the organization. An organization consists of personnel that can have different availability (e.g., available or on vacation) and is divided into multiple departments. Each department is responsible for multiple projects, which in turn consist of multiple work packages and external partners. Each work package has a status (e.g., active or non-active) and consists of multiple tasks for which external partners and/or organization personnel are in charge. A calendar can be split into multiple divisions, where each division consists of active meetings, non-active meetings, and personnel that is not participating in any meeting. Active and non-active meetings consist of a group of participants composed of internal, external, and non-available participants, and an agenda composed of multiple tasks. These are two semantically and syntactically disjoint DSMLs, thus the definition of mapping links might not be as intuitive as for the first use case. They are related to one another, as depending on the status of work packages in the model representing the organization and people in charge, meetings are automatically created in the calendar. There is a similar relation for the reverse transformation. In the following, we provide more details on the mapping models and generated QVTo transformations.

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. This mapping model generates a total of 12 mapping operations in the output QVTo transformation from a total of 11 mapping rules that link EClasses.

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. This mapping model generates a total of 12 mapping operations in the output QVTo transformation from a total of 11 mapping rules that link EClasses.

Making a comparison between the two mapping models, we notice that the number of mapping rules that fall under SC1 is equal in both. It is important to note that the two DSMLs contain an approximately equal number of elements (i.e., Organization contains 35 elements, while Calendar contains 39 elements) and an equal number of EClasses; thus, they are of relatively similar sizes, which deeply affects the distribution of mapping rules with regard to the three scenarios. Assuming that there is no loss of information (typically occurs when not all elements of the involved DSMLs are linked by mapping rules), after the execution of the forward and backward transformations, in the case of two DSMLs of significantly different sizes, we believe that it is likely to have a higher number of mapping rules associated with SC2 and SC3.

5.1.3. 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 specialization of concepts, while in the second use case it is due to semantic and syntactical 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. As a consequence, the generation of QVTo transformations for the second use case demonstrating more complex mappings is a stronger indication of the powerful HOTs.

5.2. Model-to-text testing

Validation was performed based on model-to-text tests classified by Tiso et al. (2013) as i) conformance tests, ii) semantic tests, and iii) textual tests. Taking into account the different types of model-to-text tests, for each mapping model, we evaluated whether the generated QVTo transformations matched the expected QVTo transformations. In detail, we defined transformation test cases `<MappingModel_file, Exp_QVTo_file>`, where `MappingModel_file` represents the mapping model used to determine the links between elements of the source and target metamodels, and `Exp_QVTo_file` represents the expected QVTo transformations that we manually defined. For a test case to pass, the output of the HOTs (generated QVTo transformations) must match `Exp_QVTo_file`. In the following, we provide more details on the different types of model-to-text tests.

5.2.1. Conformance tests

They were used to verify whether the generated transformations were structured textual artifacts that conformed to the QVTo language. QVTo has specific rules that specify how statements can be written, and the set of these rules constitutes the syntax of the language. Failed conformance tests typically occur due to possible syntactical mistakes, such as missing or unbalanced parentheses, missing or unbalanced quotes, missing colons or semicolons, misspelled variables, and so on. When a QVTo file for which conformance tests have failed is opened, the syntax errors are flagged in the file, together with a message error. The most common message errors in these cases are: *missing "x" to complete scope, "x" expected instead of "y," "x" expected after "y," unrecognized variable(x)* and so on. For instance, in the UML-RT use case, the majority of mapping rules have as source and target elements with the same name. However, when the source and target of a mapping operation have the same name, only the element belonging to the metamodel that is first defined is taken into account. This can lead to unrecognized variables in the body or condition of the mapping operations. An example of a failed conformance

test is illustrated in Figure 5. While in the mapping model the user has defined the source as `hclScope/StateMachine` and target as `statemach/StateMachine`, the first defined metamodel is `statemach` in Line 1, thus if we hover over `StateMachine` elements in Line 8 we notice that they are both `statemach: StateMachine`. When trying to access the `states` EReference of the `StateMachine` element from the second metamodel (i.e., `hclScope`) the variable is unrecognized. Therefore, to avoid such failed conformance tests, we print the fully qualified name of the source and target elements as shown in Line 9 in Figure 10. Failed conformance tests can also be the result of incomplete or incorrect mapping models defined by the users. For instance the `State2CompositeState` mapping rule in Line 11 is invoked automatically, meaning that the user does not need to define the mapping rule that should be invoked when assigning the value of the source element to the target element. The only requirement is that the `State2CompositeState` mapping rule should be created by the user in the mapping model (a rule that has not been defined in the mapping model cannot be invoked).The absence of this rule in the mapping model before executing the HOTs would result in an error, and the user would have to revisit the mapping model and define the `State2CompositeState` mapping rule.

5.2.2. Semantic tests

They are used to verify whether the generated transformations adhere to the semantics of the QVTo language. Failed semantic tests are often due to missing mapping operations, incorrect hierarchical structure, incorrect type of mapping operations (abstract/non-abstract), missing/incorrect inheritance and disjunct candidates, and so on. For instance, our HOTs are expected to automatically identify whether a mapping operation inherits another. Mistakes in the implementation could lead to the HOTs failing to identify inheriting mapping operations. This would not trigger any syntactical error in the file and the generated transformations would be executed. However, the resulting models of the generated transformations would not be semantically correct. The lack of error messages makes these types of errors not easy to locate. Another interesting example would be the one illustrated in Figure 3. While the user expects the transformation to navigate from `Organization` to `name` *via* `self.department.manager.name`, the HOTs generate the path `self.department.secretary.name`. The generated QVTo transformation would be executable and syntactically correct but semantically wrong. Instead of mapping the `name` of a person who holds the role of a manager in the department to the `managerName` in a company, it would, in fact, map the `name` of a person who holds the role of a secretary in the department to the `managerName` in a company. For that reason we have introduced mapping rules as per SC3 where `source!=null and target==null`. These rules aim to assist the user specify the correct navigation path for accessing a particular element of the source metamodel, thus avoiding failed semantic tests. An example of such rule is illustrated in Figure 4, where the `manager2null` mapping rule ensures that the correct navigation path is generated to access the name of the manager instead of the name of the secretary. It is common to encounter such a scenario when the navigation path from one EClass to another includes multiple non-containment references. For instance, in Figure 3, navigation between EClass `Department` and EClass `Person` can be accomplished through `manager` or `secretary` references.

5.2.3. Textual tests

They are used to verify whether the textual elements of the generated model transformations have the required format. Errors discovered through these tests are not identified through conformance testing. Examples of textual testing involve checking whether the name of the transformation is the one inputted by the user or whether the names of the mapping operations are defined following the defined template. Furthermore, these tests verify whether the generated transformations adhere to the QVTo formatter (e.g., new lines, indents, white spaces).

On a side note, to increase the reliability of our approach, we complemented the aforementioned tests by manually defining target models that represent the expected outputs of the execution of the generated QVTo transformations. We compared the XML representation of the manually defined target models and the generated target models using the XML compare tool.[15] As part of this process, it is important to leverage extensive input models, which conform either to $DSML_A$ or $DSML_B$, depending on the direction of the transformation), that cover as many variations and combinations of concepts as possible, thus minimizing the likelihood of untested scenarios. While the generated target models were identical to the manually defined target models (with the exception of the line order), it is still a valid concern whether the target models generated are correct and meet the requirements. In order to generate correct target models, two conditions must be met; i) the user must select "correct" mappings that illustrate the requirements, and ii) the HOTs should generate the expected QVTo transformations based on the input data (i.e., involved DSMLs and mapping model). While we have executed model-to-text test cases to validate the correctness of the HOTs, there is no guarantee that the user will select the "correct" mappings that conform to the requirements. Since mappings reflect the user intentions, we cannot provide guarantees on their appropriateness, meaning their reflection of the user's intentions. However, such a risk is apparent also on the traditional approach of manually

15 https://extendsclass.com/xml-diff.html

```
1  modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';
2  modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
3
4  transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
5  main() {
6  hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachineDisjunct();
7  }
8  mapping StateMachine :: StateMachine2StateMachine0() : StateMachine
9  when {self.states -> size() = 1}
10 {
11     result.top := self.states -> first().map State2CompositeState();
12     result.name := self.name;
13 }
```

FIGURE 5
Example of failed conformance test.

writing model transformations. Furthermore, in addition to the limitations built in the MML by customizing the scope provider, the execution of the HOTs in cases where the mapping model is not correct (e.g., the user is mapping an `EClass` to an `EReference`) will generate an error message. In summary, in light of the validation results, we can argue with certain confidence that MML contains all those concepts needed to specify deterministic unidirectional mappings between two Ecore-based DSMLs, and mapping models can then be effectively used to generate well-formed model transformations.

5.3. Example

In Figure 6 we provide an example of a mapping model between the excerpts of $DSML_A$ and $DSML_B$ illustrated in Figures 7, 8. In Figure 9 we illustrate the properties view for three mapping rules defined in Figure 6, to provide the reader with a more concrete example of the manual input that is required from the user in different cases. The requirements for transforming from $DSML_A$ to $DSML_B$ are as follows:

- `StateMachine` element in M_{DSML_A} is transformed to StateMachine element in M_{DSML_B}. Moreover, being that a StateMachine element in M_{DSML_B} must contain only one direct `CompositeState` element, if the `StateMachine` element in M_{DSML_A} contains only one direct `State`, the latter is transformed to a `CompositeState`; alternatively if the `StateMachine` element in M_{DSML_A} contains more than one direct `State` a new `CompositeState` element is created in M_{DSML_B}.
- A `State` element in M_{DSML_A} is transformed to a `SimpleState` element in M_{DSML_B}, if the `State` element in M_{DSML_A} does not contain any other elements.
- A `State` element in M_{DSML_A} is transformed to a `CompositeState` element in M_{DSML_B} if the `State` element in M_{DSML_A} contains at least one element.

To begin with, the user would instantiate a new mapping model and give it a name (e.g., Textual2Graphical). Upon loading the metamodels, the user would select the respective EPackages to define the source and target metamodels. `MainSourceMetamodel` is not required in this particular instance, since there is only one `SourceMetamodel` (i.e., $DSML_A$) that is automatically assigned as `MainSourceMetamodel`. Following this, the user would begin defining mapping rules in accordance with the requirements. First, the user would map the root elements of both metamodels, which in our case are the `StateMachine` elements. Based on the first requirements, there are two ways that the user can define the mapping rule. The first option consists of defining one single `StateMachine2StateMachine` and then using two `HelperStatements` to specify the conditional statements. The second option consists of the user defining two `StateMachine2StateMachine` mapping rules and specifying the condition, by using the `condition` property of the each mapping rule. While the first option is more similar to OOP and can be easier for modeling tool developers, the second option can be more intuitive for domain experts. Therefore, in this example we detail the second option where we define two `StateMachine2StateMachine` mapping rules. The details of the first mapping rule `StateMachine2StateMachine` can be seen in Figure 9. The user would have to define the `source`, `target`, and `condition`. The `name` is automatically generated, while the `operator` is set to `assign` by default. The user would then define a new child mapping rule (i.e., `states2top`) and define the `source`, `target`, and `condition`. Both of these mapping rules fall under SC1. An interesting mapping rule falling under SC2, is the `null2top`, defined as a child mapping rule for the second `StateMachine2StateMachine` mapping rule. The `null2top` mapping rule creates a new `CompositeState` element in M_{DSML_B}, for which there is no match in M_{DSML_A}.

In Figure 10 we present a more extensive excerpt of the `Textual2Graphical` mapping model for UML-RT state

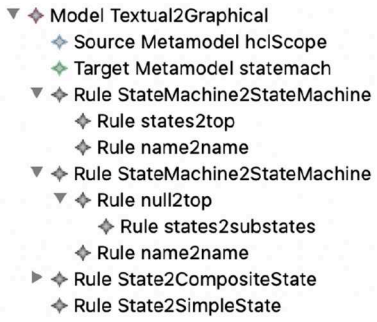

FIGURE 6
Mapping model.

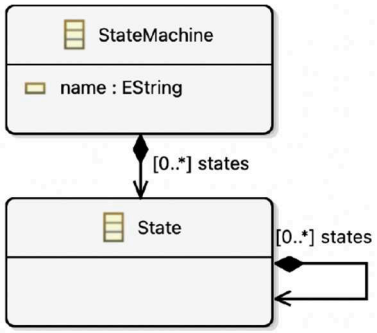

FIGURE 7
$DSML_A$. *Unbounded upperbound for cardinalities.

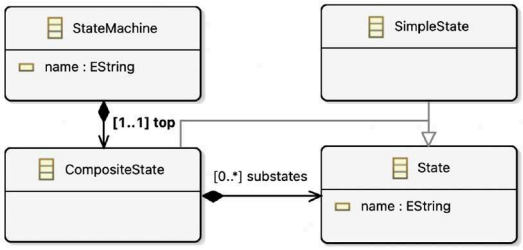

FIGURE 8
$DSML_B$. *Unbounded upperbound for cardinalities.

machines on the left-hand side and an excerpt of the generated QVTo transformation on the right-hand side. The generated QVTo transformation is the output of the execution of the HOTs that take as input *DSML*, $DSML_B$, and `Textual2Graphical` mapping model. There are a few peculiarities to highlight here.

The first `StateMachine2StateMachine` mapping rule generates Lines 12–13. However, being that the mapping model contains two mapping rules named `StateMachine2StateMachine` where `sources` and `targets` are identical, the HOTs generate three mapping operations, where one of them (Line 9) is a disjunctive mapping operation that disjuncts the other two (Lines 12 and 19). The disjunctive mapping operation is invoked on Line 6 and the first matching candidate (i.e., `StateMachine2StateMachine0` or `StateMachine2StateMachine1`) is selected. HOTs determine the order in which the disjunctive candidates are printed based on whether they have a mapping condition. As can be seen, `StateMachine2StateMachine0` on Line 12 has a mapping condition; thus, it is printed as the first disjuncted mapping operation on Line 10. To define the model transformations manually, the users would have to possess a strong understanding of these details. More specifically, users would need to understand the syntax of the model transformation language (i.e., QVTo), the concept of disjunction (e.g., the order in which the disjunct candidates appear), and which rules to invoke in particular situations (e.g., Line 6). Instead, with our solution, the user is only required to define two mapping rules, specifying the `source`, `target`, and `condition` attributes. By doing so, the HOTs would be able to automatically generate model transformations that conform to the QVTo syntax and include concepts that the user is not expected to understand. Consequently, this reduces the amount of effort and expertise required.

Furthermore, Lines 21–25 detail the generation of mappings from child mapping rules according to SC2 described in Section 4.2, where `source == null` and `target == top`. The referred `EClass` of `EReference top`, is `CompositeState`, thus the latter is added to the target model. Alternatively, Line 16 details the generation of a mapping from a child mapping rule according to SC1, where `source == name` and `target == name`.

On another note, the `State2State` mapping rule in the mapping model has been generated as two mapping operations: `AbstractState2State` in Line 47, because the `State` class in the target metamodel is abstract, and `State2StateDisjunct` in Line 44, as in the mapping model there are two rules (`State2SimpleState` and `State2CompositeState`) that fulfill the conditions to be disjuncted mapping operations (sources are identical, while targets are subtypes). Moreover, since `SimpleState` and `CompositeState` extend `State` in the target metamodel, the mapping operations in Lines 28 and 38 inherit the abstract mapping operation in Line 47, having identical sources. The example above further illustrates the reduced effort and expertise needed, as it eliminates the need for the user to comprehend the concepts of abstract mappings, inheritance, and disjunction. Additionally, HOTs reduce the likelihood of human-errors by automatically analyzing the involved DSMLs and mapping models.

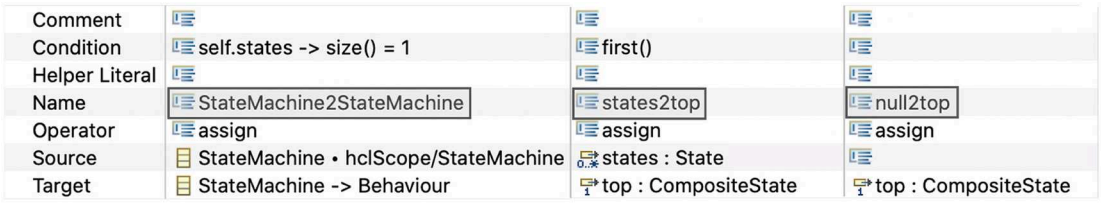

FIGURE 9
Properties view for the mapping rules defined in Figure 6.

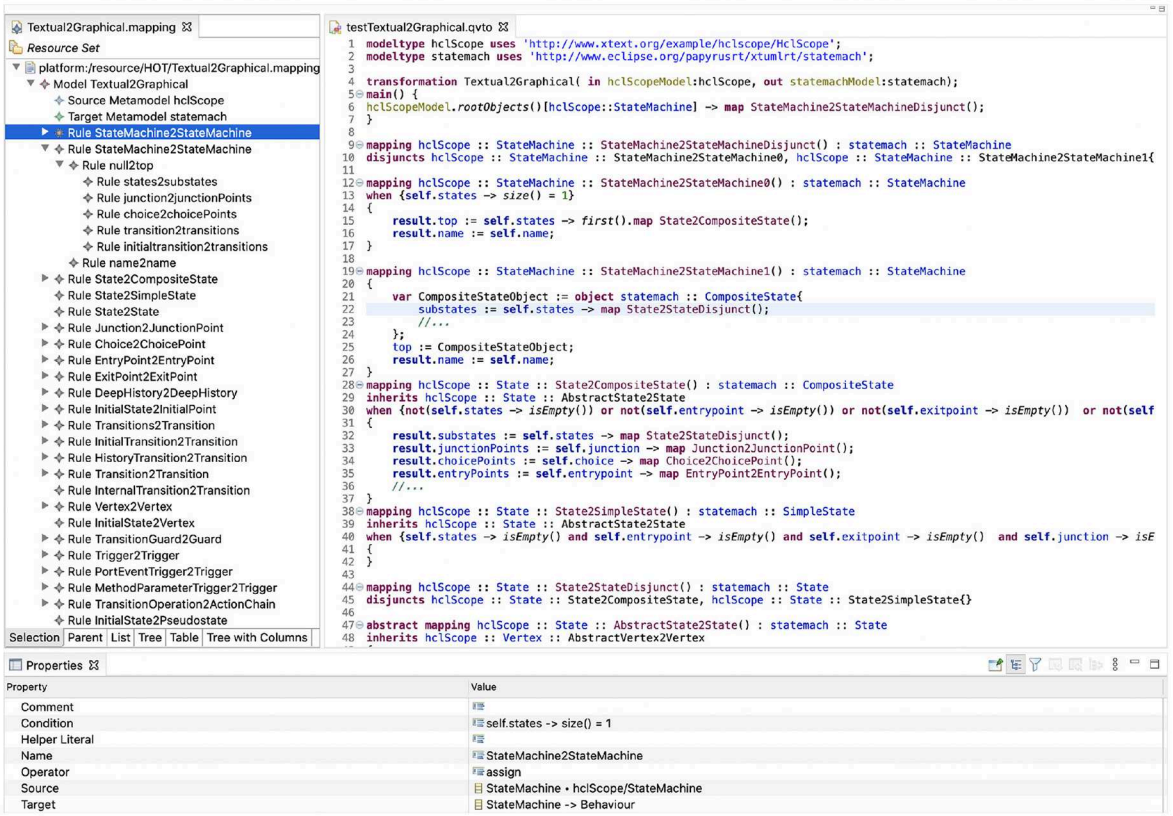

FIGURE 10
Textual to graphical mapping model and generated QVTo transformations for UML-RT state machines.

6. Discussion

In this section we reflect on several aspects of our approach. We present the design principles that guided our solution, followed by an analysis of the benefits and an identification of the limitations and possible solutions.

6.1. Design principles

Three main principles have guided the design of the proposed solution, as outlined below.

1. Separation of concerns: A strict separation between domain logic and implementation-specific details reduces complexity and allows for increased reusability, maintanability and extensibility of the solution. Moreover, the definition of domain logic in a separate model (i.e., mapping model), using a language that provides a higher level of abstraction than model transformation languages, facilitates the understanding and solving of problems and ensures that software developers are not exposed to unnecessary information.
2. Consideration of user's intentions: A complete and carefully written specification of how the DSMLs are mapped to one

another forms the basis for producing complete and accurate model transformations. As a result, it is essential to provide the developer with the ability to capture his/her intentions in the form of unambiguous mapping links between elements of two DSMLs as semantics often needs human understanding to be correctly managed.

3. Tooling that seamlessly integrates the target audience's current tool ecosystem: An essential aspect of successful software is the ability to seamlessly integrate with the environment that the target audience already uses. Our target modeling environment is the well-established Eclipse Modeling Framework, thereby our approach focuses on Ecore-based DSMLs and we opted for technologies (Xtend, QVTo, Xtext, etc.) that integrate seamlessly with the Eclipse environment.

6.2. Analysis of the benefits

The adoption of a novel approach for the synchronization infrastructure between multiple notations for blended modeling can be challenging due to the customers' uncertainty of whether the benefits outweight the costs. Our proposed approach is therefore subjected to a cost-benefit analysis while simultaneously being compared with traditional approaches.

Firstly, we will discuss the *transition costs* involved. The transition costs consist of i) implementation costs and ii) training costs. *Implementation costs* are concerned with the adaptation of an organization's existing systems to integrate the proposed approach. Since our tool seamlessly integrates with EMF tools, we do not incur any costs in this regard. There is, however, a concern that companies with existing synchronization infrastructures would have to create the mapping models from scratch although they already have the synchronization infrastructure in place. Nevertheless, the benefits of doing so outweigh the costs in a significant manner due to the following reasons. First, as metamodels evolve, the co-evolution of the model transformations can be facilitated, as the respective changes can be made at a higher level of abstraction. Moreover, in case $DSML_A$ and $DSML_B$ would represent two versions of the same language DSML (e.g., $DSML_B$ is an evolution of $DSML_A$), the generated transformations would instead provide model co-evolution. Further, it enables faster prototyping, allowing for user feedback prior to releasing a new version of the modeling tool, and verifying that the requirements of the users are understood and met. As part of our strategy to further reduce costs, we intend to use reverse transformations that build mapping models from model transformations. *Training costs* are instead concerned with the time and resources required to learn to utilize the proposed approach. Many industries are cautious to adopt new technologies that require a considerable amount of training and practice before they can be effectively implemented. Nonetheless, when applications are implemented with a focus on user experience, training is less complex, faster, and more effective. Since users are interacting with the MML, we have minimized the training costs by designing the MML to be as simple and intuitive as possible. MML exhibits the following characteristics:

- Encapsulates the minimum set of concepts necessary for defining deterministic mappings, keeping the language concise, and avoiding unneeded verbosity.
- Developed with a blended modeling approach to support textual and tree-based notations, which exhibit complementary usability features.
- Syntactically similar to object-oriented programming languages, which pushes down the learning curve for the average software developer.
- Raises the level of abstraction by allowing the user to focus on the domain's logic instead of dealing with lower-level model transformations.

While users must become familiar with MML and while at first glance it may appear to be an additional overhead, it is in fact a one-time effort which proves beneficial in the long run. Compared to model transformations, mapping models require significantly less input, resulting in lowers effort on the part of the developer. MML also enables domain experts without model transformation knowledge to be involved in the definition of the mapping models, since domain logic is presented in a format that is easily understood by all stakeholders rather than embedded in boilerplate model transformations. This could even reduce the time of development and number of errors caused by misunderstandings or miscommunications between domain experts and developers. The approach has also demonstrated acceptance and practical applicability in an industrial setting among HCL developers, who used it i) to define mapping models for generating the synchronization infrastructure between graphical and textual DSMLs, and ii) to co-evolve the mapping models and consequently the model transformation in response to changes in the textual DSML until the latter was refined to its present form. Moreover, HCL is currently applying this approach for model co-evolution/migration purposes. To further decrease the learning curve and consequently, the training costs, we have contributed with a tutorial to an established MDE community (ICSA conference), and we plan on delivering a tutorial at a premier conference for practitioners and researchers interested in software architecture. Finally, we plan to contribute more examples and step-by-step tutorials to the online repository. As we presented a summary of potential costs and benefits associated with our approach, we would like to emphasize that the settings in which an organization operates is an instrumental factor in this analysis. The size of the involved DSMLs and the frequency of their evolution, for example, can greatly impact the

decision on whether the adoption of the approach is appropriate for the specific organization. Our first recommendation is for interested industrial parties to conduct their own cost-benefit analysis using this example as a guide. In addition, we recommend a gradual and step-wise adoption of the approach through the establishment of a multi-functional team staffed with both domain experts and software developers to investigate the integration and usability of the approach in their particular settings through our prototype.

6.3. Limitations and possible solutions

As a result of this research, we have identified a number of limitations and potential solutions associated with the automatic generation of the synchronization infrastructure for blended modeling.

Bi-directionality. Our study focused on the use of unidirectional mappings (and generated transformations) instead of bidirectional ones. While our synchronization approach has the same goal of bidirectional transformations, there are multiple reasons for which we made this decision. Unidirectionality facilitates the management and maintenance of the synchronization infrastructure. Although a bidirectional approach would have been a theoretically more elegant solution, we had to adapt to the existing tool ecosystem and the knowledge base of the tool engineers. We chose a pragmatic approach, trying to provide engineers with a "tool" (i.e., mapping language) as close to their metamodeling and object-oriented knowledge as possible. Moreover, since the involved DSMLs could be non-bijective, which is most likely in the case of two disjoint DSMLs, there is a higher risk of significant differences between the forward and backward transformations, leading to transformations being non-invertible (Stevens, 2010). While there could be an opportunity to incorporate the definition of bidirectional mappings, a proper balance must be found also with respect to the usability of the tool and the correctness of the generated model transformation with respect to the requirements.

Interoperability. Another interesting point relates to the use of MML as a core artifact for interoperability between model transformation languages (Jouault and Kurtev, 2007). In fact, MML is designed with a focus on generalizability; in our context, this is defined as the ability to use the same mapping model to drive the generation of model transformations in multiple model transformation languages. Generalizability is achieved by ensuring the separation of the *domain-logic* from *implementation-specific details*. The domain logic is included in the MML, whereas implementation-specific details are specified in the HOTs, which are specific to a transformation language. By providing an automatic generation of mapping models from existing model transformations, the generated mapping models could be used as input to other HOTs for generating model transformations conforming to other transformation languages. Although we cannot exclude that MML could need to be extended to support specific transformation languages, we are confident that the eventual changes would be relatively minor and only related to language-specific details that would require user input to generate correct transformations.

In-place transformations. In this work we assume that there exist mapping links between all elements of $DSML_A$ and $DSML_B$. Nevertheless, one of the involved DSMLs (e.g., $DSML_A$) can have higher expressiveness than its counterpart (e.g., $DSML_B$). In this regard, it should be noted that the existence of mapping links for all elements cannot be guaranteed, therefore, when executing the transformation from $DSML_A$ to $DSML_B$ and then executing the reverse transformation, there is a risk of information loss. We assume that scenarios with disjoint DSMLs will generally be more affected by this phenomenon compared to simpler scenarios. This can be mitigated by leveraging in-place transformations that can propagate changes to the target model (which can be the same as the source model), without reconstructing it from scratch, thereby preventing information loss.

7. Conclusions

In this paper, we presented our effort toward an approach for the automatic generation of synchronization model transformations to support blended modeling in an industrial setting. The resulting solution entails the provision of automatic means for the generation of model synchronization transformations across multiple notations of the same or different languages. This was accomplished by first designing and implementing a mapping modeling language (MML) in terms of an Ecore model. MML is instantiated in terms of mapping models, which define mapping relations between elements of two arbitrary Ecore-based DSMLs. Given the two DSMLs in terms of Ecore models and two mapping models (one per direction), we provide a set of higher-order transformations (HOTs) that generate synchronization model transformations in QVT Operational between the two DSMLs. The HOTs were implemented using Xtend.

Depending on what the two DSMLs represent, the generated transformations support two types of synchronization:

1. The DSMLs represent two notations of the same language (e.g., graphical and textual UML-RT state-machines), then the generated transformations provide synchronization across different notations of the same language.
2. The DSMLs are disjoint, then the generated transformations provide synchronization across different notations of different languages.

Validation of the solution was performed by leveraging two use cases that represented the two aforementioned scenarios: synchronization across different notations of one language (UML-RT state-machines), and synchronization across different languages (calendar and organizational structure use case). In addition to multiple testing phases, the solution applied to UML-RT was deemed very promising by the engineers and tool architects at HCL.

Several directions for future increments of this research and engineering effort have been identified. One direction is to tune the HOTs and leverage the proposed MML for the generation of co-evolution transformations to provide automated support for model co-evolution in response to metamodel evolution. Another interesting direction is related to the use of MML for interoperability between model transformation languages. In fact, we plan to provide automatic generation of mapping models from existing synchronization transformations. The generated mapping models can then be used as input to other HOTs for generating synchronization transformations conforming to other transformation languages than QVTo.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://github.com/MLJworkspace/BlendedModellingSolution.

Author contributions

All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

Funding

This work was supported by Vinnova through the ITEA BUMBLE project (rn. 18006).

Conflict of interest

MM was employed by the company HCL Technologies.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Addazi, L., and Ciccozzi, F. (2021). Blended graphical and textual modelling for uml profiles: a proof-of-concept implementation and experiment. *J. Syst. Software* 175, 110912. doi: 10.1016/j.jss.2021.110912

Atkinson, C., and Gerbig, R. (2016). *Flexible deep modeling with MelanEE*. Modellierung 2016-Workshopband.

Bézivin, J. (2005). On the unification power of models. *Software Syst. Model.* 4, 171–188. doi: 10.1007/s10270-005-0079-0

Blouin, A., Beaudoux, O., and Loiseau, S. (2008). "Malan: a mapping language for the data manipulation," in *Proceedings of the Eighth ACM Symposium on Document Engineering* (São Paulo), 66–75.

Charfi, A., Schmidt, A., and Spriestersbach, A. (2009). "A hybrid graphical and textual notation and editor for uml actions," in *European Conference on Model Driven Architecture-Foundations and Applications* (Eschede: Springer),237–252.

Cicchetti, A., Ciccozzi, F., and Pierantonio, A. (2019). Multi-view approaches for software and system modelling: a systematic literature review. *Software Syst. Model.* 18, 3207–3233. doi: 10.1007/s10270-018-00713-w

Ciccozzi, F., Tichy, M., Vangheluwe, H., and Weyns, D. (2019). "Blended modelling-what, why and how," in *MPM4CPS Workshop* (Munich).

David, I., Latifaj, M., Pietron, J., Zhang, W., Ciccozzi, F., Malavolta, I., et al. (2022). Blended modeling in commercial and open-source model-driven software engineering tools: a systematic study. *Software Syst. Model. Appear.* doi: 10.1007/s10270-022-01010-3

Didonet Del Fabro, M., and Valduriez, P. (2009). Towards the efficient development of model transformations using model weaving and matching transformations. *Software Syst. Model.* 8, 305–324. doi: 10.1007/s10270-008-0094-z

Diskin, Z., Gómez, A., and Cabot, J. (2017). "Traceability mappings as a fundamental instrument in model transformations," in *International Conference on Fundamental Approaches to Software Engineering* (Uppsala: Springer), 247–263.

Emery, D., and Hilliard, R. (2009). "Every architecture description needs a framework: expressing architecture frameworks using ISO/IEC 42010," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture* (Cambridge: IEEE), 31–40.

Hillairet, G., Bertrand, F., and Lafaye, J. Y. (2008). "Bridging emf applications and rdf data sources," in *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering SWESE* (Karlsruhe).

Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011). "Empirical assessment of mde in industry," in *Proceedings of the 33rd International Conference on Software Engineering* (Waikiki, HI), 471–480.

Jouault, F., and Kurtev, I. (2007). On the interoperability of model-to-model transformation languages. *Sci. Comput. Program.* 68, 114–137. doi: 10.1016/j.scico.2007.05.005

Kern, H., Stefan, F., Dimitrieski, V., and Čeliković, M. (2014). "Mapping-based exchange of models between meta-modeling tools," in *Proceedings of the 14th Workshop on Domain-Specific Modeling* (Portland, OR), 29–34.

Latifaj, M., Ciccozzi, F., Anwar, M. W., and Mohlin, M. (2022). "Blended graphical and textual modelling of uml-rt state-machines: an industrial experience," in *Accepted in ECSA Post-Proceedings* (Berlin: Springer).

Latifaj, M., Ciccozzi, F., Mohlin, M., and Posse, E. (2021). "Towards automated support for blended modelling of uml-rt embedded software architectures," in *15th European Conference on Software Architecture ECSA 2021, 13 Sep 2021*, Virtual (originally Växjö), Sweden.

Lazăr, C.-L. (2011). Integrating alf editor with eclipse uml editors. *Studia Univers. Babes Bolyai Inform.* 56, 27–32. doi: 10.5038/1937-8602.56.2.1

Lethbridge, T. C., Forward, A., Badreddin, O., Brestovansky, D., Garzon, M., Aljamaan, H., et al. (2021). Umple: model-driven development for open source and education. *Sci. Comput. Program.* 208, 102665. doi: 10.1016/j.scico.2021.102665

Lilius, J., and Paltor, I. P. (1999). "vUML: a tool for verifying uml models," in *14th IEEE International Conference on Automated Software Engineering* (Cocoa Beach, FL: IEEE), 255–258.

Lopes, D., Hammoudi, S., Bézivin, J., and Jouault, F. (2006). "Mapping specification in mda: from theory to practice," in *Interoperability of Enterprise Software and Applications* (Berlin: Springer), 253–264.

Maro, S., Steghöfer, J.-P., Anjorin, A., Tichy, M., and Gelin, L. (2015). "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* (Pittsburg, CA), 1–12.

Persson, M., Törngren, M., Qamar, A., Westman, J., Biehl, M., Tripakis, S., et al. (2013). "A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems," in *Proceedings of the International Conference on Embedded Software, EMSOFT 2013, Vol. 10* (Montreal, QC: IEEE), 1–10.

Ráth, I., Ökrös, A., and Varró, D. (2010). Synchronization of abstract and concrete syntax in domain-specific modeling languages. *Software Syst. Model.* 9, 453–471. doi: 10.1007/s10270-009-0122-7

Ries, B., Capozucca, A., and Guelfi, N. (2018). "Messir: a text-first DSL-based approach for UML requirements engineering (tool demo)," in *Proceedings of the 11th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2018* (Boston, MA: ACM), 103–107.

Scheidgen, M. (2008). "Textual modelling embedded into graphical modelling," in *European Conference on Model Driven Architecture-Foundations and Applications* (Berlin: Springer), 153–168.

Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: Eclipse Modeling Framework*. London: Pearson Education.

Stevens, P. (2010). Bidirectional model transformations in qvt: semantic issues and open questions. *Software Syst. Model.* 9, 7–20. doi: 10.1007/s10270-008-0109-9

Tisi, M., Jouault, F., Fraternali, P., Ceri, S., and Bézivin, J. (2009). "On the use of higher-order model transformations," in *European Conference on Model Driven Architecture-Foundations and Applications* (Eschede: Springer), 18–33.

Tiso, A., Reggio, G., and Leotta, M. (2013). "A method for testing model to text transformations," in *AMT©MoDELS* (Miami, FL).