To Séverine

for illuminating paths I never knew existed

Abstract

Collaborative model-driven software engineering addresses the complexities of developing software systems by prioritizing models as core artifacts and leveraging the collective expertise of diverse stakeholders. To effectively realize this approach, the employed modeling environments must be equipped with features that support and enhance collaboration. These environments should, among other capabilities, provide support for multiple notation types, enabling stakeholders to engage with models using their preferred notation or the notation most appropriate for their tasks. Additionally, they should offer multiple views and perspectives that allow stakeholders to interact with pertinent information only, and implement access control mechanisms to ensure information security. However, the adoption of these features can be challenging, partly because of their resource-intensive and tedious development nature, as well as the necessity for continuous updates to keep up with the evolution of modeling languages.

This doctoral thesis proposes a model-driven approach to address this challenge by facilitating the development of blended modeling environments featuring multiple views and ensuring modeled information security. The proposed framework leverages automation to reduce the manual effort and expertise traditionally required for i) the provision of synchronization mechanisms between graphical and textual notations for blended modeling, ii) the provision of synchronization mechanisms between view models and base model in multi-view modeling, and iii) the consistent definition and enforcement of access permissions. This research, therefore, lowers the barriers to adopting these collaborative features by facilitating their development and evolution in face of changes to underlying modeling languages.

Sammanfattning

Samarbetsbaserad modelldriven mjukvaruutveckling hanterar utmaningarna med att skapa komplexa mjukvarusystem genom att prioritera modeller som centrala artefakter och dra nytta av det samlade kunnandet hos utvecklingsteamet. För att detta ska fungera effektivt krävs att modelleringsmiljöerna är utrustade med funktioner som främjar och förbättrar samarbete. Dessa miljöer bör bland annat stödja olika typer av notationer så att olika användare kan arbeta med modeller på det sätt som passar dem bäst för den specifika uppgiften. Dessutom bör de erbjuda flera vyer som tillåter användarna att fokusera på relevant information och implementera åtkomstkontrollmekanismer för att säkerställa informationssäkerhet. Att införa dessa funktioner kan dock vara utmanande.

Denna doktorsavhandling presenterar ett modelldrivet ramverk som underlättar utvecklingen av samarbetsbaserade modelleringsmiljöer med flera vyer och mekanismer för att implementera och upprätthålla informationssäkerhet. Det föreslagna ramverket använder automatisering för att minska det manuella arbetet och den expertis som vanligtvis krävs för i) att synkronisera mellan grafiska och textuella notationer, ii) att synkronisera mellan vyer och den övergripande basmodellen vid flervymodellering, och iii) att konsekvent definiera och verkställa åtkomstbehörigheter. Genom denna forskning minskas hindren för att integrera dessa samarbetsfunktioner genom att underlätta deras utveckling och se till att de hålls uppdaterade med de senaste modelleringsspråken.

Acknowledgment

"It seems to me shallow and arrogant for any man in these times to claim he is completely self-made, that he owes all his success to his own unaided efforts. Many hands and hearts and minds generally contribute to anyone's notable achievements."

- *Walt Disney*

As I reflect on the journey that culminated in this doctoral thesis, I want to extend my gratitude towards the people whose hands, hearts, and minds contributed to this achievement. To begin with, I opt to address my supervisors as *you* instead of resorting to the impersonal *he*, since I genuinely seek to express my heartfelt gratitude directly to them, acknowledging their support and guidance, as they truly deserve every bit of recognition.

To Federico Ciccozzi! You have been a mentor, a guide, and an inspiration. Your belief in me, often expressed not in grand speeches but in quiet demonstrations of trust and encouragement, has been the foundation of my confidence and growth. Your influence extends beyond my academic achievements, shaping me into a better researcher, thinker, and above all, a better person. For all the lessons taught, both spoken and unspoken, and for the confidence you have placed in me, I am deeply grateful. To Antonio Cicchetti! Your insightful discussions have not only enriched my work but have also deepened my understanding and appreciation of the field. I am thankful for your humor and the relaxed, enjoyable moments we shared over lunches and coffee breaks, which have greatly enhanced my comfort and enjoyment throughout my doctoral journey.

I extend my sincere thanks to Cristina Seceleanu and Mikael Sjödin for their invaluable assistance in reviewing drafts during internal processes. Furthermore, I express my appreciation to Radu Dobrin for his understanding and support in administrative affairs, and for involving me in other interesting endeavors which provided me with the opportunity to engage with other aspects of academia beyond the confines of my doctoral studies.

I would like to thank my colleagues for their support during our shared rollercoaster journey. Their willingness to listen, offer advice, and share a moment of joy has been much appreciated. Special thanks go to Robbert Jongeling, whose roles as a friend, mentor, therapist, and supporter have been instrumental. His rational insights have brought clarity in times when emotions ran high, and I am profoundly thankful to have shared this journey with him.

Last but not least, I owe a deep debt of gratitude to my family for their immeasurable support and love throughout my doctoral journey. To my parents, who have been my rock and my refuge: you have instilled in me the courage to dream big and the fortitude to follow through, and I carry a piece of your strength and wisdom in all that I do. I am deeply grateful for every word of encouragement and every moment of comfort you have given me. This achievement belongs as much to you as it does to me, and I hope I have made you as proud of me as I am endlessly grateful for you!

To you and all my loved ones, this thesis symbolizes not just my academic endeavors but also the personal transformation I have undergone during this period. It is a reflection of the cumulative achievements and growth I have experienced, all of which were made possible by your constant support and encouragement. This accomplishment speaks as much to your influence in my life as to my own efforts! Blessed to have you!

Malvina Latifaj
Västerås, May, 2024

List of Publications

Publications included in this thesis[1]

Paper A: *Blended Modeling in Commercial and Open-source Model-Driven Software Engineering Tools: A Systematic Study.* Istvan David, Malvina Latifaj, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer, Regina Hebig. Software and Systems Modeling Journal (SoSyM), 2022.

Paper B: *Towards Automated Support for Blended Modeling of UML-RT Embedded Software Architectures.* Malvina Latifaj, Federico Ciccozzi, Mattias Mohlin, Ernesto Posse. 15th European Conference on Software Architecture (ECSA), 2021.

Paper C: *Blended Graphical and Textual Modeling of UML-RT State -Machines: An Industrial Experience.* Malvina Latifaj, Federico Ciccozzi, Muhammad Waseem Anwar, Mattias Mohlin. Post-Proceedings of the 15th European Conference on Software Architecture (ECSA), 2021.

Paper D: *Automatic Generation of Synchronization Mechanisms for Blended Modeling.* Malvina Latifaj, Federico Ciccozzi, Mattias Mohlin. Frontiers of Computer Science Journal (Frontiers), 2023.

[1]The included publications have been reformatted to comply with the thesis layout.

Paper E: *Role-Based Access Control for Collaborative Modeling Environments.* Malvina Latifaj, Federico Ciccozzi, Antonio Cicchetti. Journal of Object Technology (JOT), 2024. *Conditionally accepted at the time of writing.*

Related publications not included in this thesis

Paper T: *The Path Towards the Automatic Provision of Blended Modeling Environments.* Malvina Latifaj. ACM Student Research Competition at the 25th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2022.

Paper U: *Cross-Platform Blended Modeling with JetBrains MPS and Eclipse Modeling Framework.* Malvina Latifaj, Hilal Taha , Federico Ciccozzi, Antonio Cicchetti. 19th International Conference on Information Technology : New Generations (ITNG), 2022.

Paper V: *Blended Modeling Applied to the Portable Test and Stimulus Standard.* Muhammad Waseem Anwar, Malvina Latifaj, Federico Ciccozzi. 19th International Conference on Information Technology : New Generations (ITNG), 2022.

Paper W: *Metamodel Portioning for Flexible and Secure Architectural Views.* Malvina Latifaj, Federico Ciccozzi, Antonio Cicchetti. 3rd International Workshop on Model-Driven Engineering for Software Architecture (MDE4SA), 2023.

Paper X: *Blended Modeling for Software Architectures.* Malvina Latifaj, Federico Ciccozzi, Muhammad Waseem Anwar, Ivano Malavolta , Kousar Aslam. 20th IEEE International Conference on Software Architecture (ICSA), 2023.

Paper Y: *Cross-Platform Migration of Software Architectural UML-RT Models.* Malvina Latifaj, Federico Ciccozzi, Antonio Cicchetti, Mattias Mohlin. 17th European Conference on Software Architecture (ECSA), 2023.

Other publications

Paper Z: *Find the Way in the Jungle of Quality of Service in Industrial Cloud: A Systematic Mapping Study.* Malvina Latifaj, Federico Ciccozzi, Séverine Sentilles. 11th International Conference on Cloud Computing and Services Science (CLOSER), 2021.

Contents

I Thesis **1**

1 Introduction **3**

2 Background and related work **7**
- 2.1 Model-driven software engineering 7
  - 2.1.1 Modeling languages 9
  - 2.1.2 Model transformations 9
  - 2.1.3 Mapping modeling languages 15
- 2.2 Blended modeling 17
- 2.3 Multi-view modeling 20
- 2.4 Access control 21
- 2.5 UML real-time profile 23

3 Research overview **25**
- 3.1 Research problem and motivation 25
- 3.2 Research goals 26
- 3.3 Research methodology 28

4 Research results **33**
- 4.1 Thesis research contributions 33
- 4.2 Paper contributions 37

5 Discussion **43**
- 5.1 Research subgoals and contributions 43

5.2 Holistic overview of the proposed solution 48
5.3 Reflections and prospects 51
5.4 Beyond the initial scope 55

6 Conclusions 57

Bibliography 59

II Included Papers 67

7 Paper A:
Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study 69

7.1 Introduction 71
7.1.1 What is blended modeling? 71
7.1.2 What is *not* blended modeling? 72
7.1.3 Motivation and aim 73
7.1.4 Structure 75
7.2 Background 75
7.2.1 Multiple notations 75
7.2.2 Seamless interaction 77
7.2.3 Inconsistency management 80
7.2.4 Related secondary literature 82
7.3 Study design 87
7.3.1 Process 87
7.3.2 Research questions 91
7.3.3 Search and selection 92
7.3.4 Classification framework definition 101
7.3.5 Data extraction 101
7.3.6 Data validation 102
7.3.7 Data analysis 102
7.4 Results 103
7.4.1 Overview 104
7.4.2 User-oriented characteristics (RQ1) 108

7.4.3 Realization-oriented characteristics (RQ2) 114
7.5 Orthogonal findings 118
7.5.1 Number of notation types and overlap of notations . . 118
7.5.2 Seamless interaction 119
7.5.3 Flexibility and inconsistency management 120
7.5.4 Technological trends 121
7.6 Discussion 122
7.6.1 Takeaways 122
7.6.2 Challenges and opportunities 123
7.7 Threats to validity 127
7.7.1 External validity 128
7.7.2 Internal validity 128
7.7.3 Construct validity 128
7.7.4 Conclusion validity 129
7.8 Conclusions 129
Bibliography 130
Bibliography 135

8 Paper B: Towards automated support for blended modeling of UML-RT embedded software architectures 149
8.1 Introduction 151
8.1.1 Problem, motivation, and the RTist case 152
8.1.2 Paper contribution 153
8.2 Blended modeling: what and why 153
8.3 A textual notation for UML-RT state-machines 155
8.3.1 Textual language workbench 156
8.3.2 Definition of a textual notation 156
8.3.3 Scoping 158
8.3.4 Integration in RTist 159
8.4 Challenges 159
8.4.1 Incoming cross-references 159
8.4.2 Model information not represented in the textual notation 160
8.4.3 Model element unique identifiers 160

8.4.4 Code formatting and comments 160
8.5 Related work 161
8.6 Outlook 162
Bibliography 163

9 Paper C:
Blended graphical and textual modeling of UML-RT state-machines:
An industrial experience **167**
9.1 Introduction 169
9.1.1 Problem, motivation, and the RTist case 169
9.1.2 Paper contribution 170
9.1.3 Paper outline 171
9.2 Blended modelling: what and why 171
9.3 Design solution 173
9.3.1 Textual notation for UML-RT state-machines 174
9.3.2 Synchronization transformations 178
9.4 Implementation 182
9.4.1 Textual language and editor for UML-RT 182
9.4.2 Synchronization 185
9.5 Validation and discussion 190
9.6 Related work 194
9.7 Outlook 195
Bibliography 197

10 Paper D:
Automatic generation of synchronization mechanisms for blended
modeling **201**
10.1 Introduction 203
10.1.1 Blended modeling 204
10.1.2 Our contribution 206
10.1.3 Paper organization 206
10.2 Industrial setting, core problem, and expected benefits 206
10.3 Related work 209
10.3.1 Blending graphical and textual editors 209
10.3.2 Model weaving 211

10.4 Proposed solution . . . 212
10.4.1 Mapping modeling language . . . 214
10.4.2 Higher-order transformation . . . 219
10.5 Validation and use cases . . . 226
10.5.1 Use cases . . . 226
10.5.2 Model-to-text testing . . . 230
10.5.3 Example . . . 234
10.6 Discussion . . . 238
10.6.1 Design principles . . . 238
10.6.2 Analysis of the benefits . . . 239
10.6.3 Limitations and possible solutions . . . 241
10.7 Conclusions . . . 243
Bibliography . . . 245

**11 Paper E:
Role-based access control for collaborative modeling environments 249**
11.1 Introduction . . . 252
11.2 Background . . . 254
11.2.1 Eclipse modeling framework and Ecore . . . 254
11.2.2 Multi-view modeling . . . 254
11.2.3 Access control . . . 255
11.3 Running example . . . 257
11.3.1 User roles . . . 257
11.3.2 Views . . . 258
11.3.3 Access permissions . . . 259
11.4 Proposed approach . . . 259
11.4.1 Setting up the multi-view modeling environment . . . 261
11.4.2 Access permissions definition . . . 268
11.4.3 Access permissions enforcement . . . 274
11.5 Illustrative example . . . 278
11.6 Related work . . . 282
11.7 Discussion . . . 284
11.8 Conclusions and future work . . . 285
Bibliography . . . 287

I

Thesis

Chapter 1

Introduction

Collaborative Model-Driven Software Engineering (MDSE) emerges from the understanding that the development of complex software systems, centered around models as fundamental development artifacts, transcends the capabilities of individual contributors [1, 2]. It acknowledges that a single individual cannot encompass the breadth of expertise needed; instead, it thrives on the combined efforts of individual stakeholders, each providing specialized skills and perspectives [3]. Modeling environments serve as shared workspaces that facilitate the collaborative efforts of the involved stakeholders. As such, they need to integrate a set of features that embrace and foster diversity – spanning stakeholders' expertise, perspectives, roles, and preferences – and enhance collaboration [4].

This diversity is particularly evident in the various notations (e.g., graphical and textual) employed by stakeholders to interact with the model, a reflection of personal preferences, expertise, or the notation's suitability for specific tasks. To accommodate these differences, there is a need for blended modeling environments [5] that support multiple notations, allowing every stakeholder to contribute effectively [6]. Moreover, stakeholders require tailored perspectives and views on the model to focus on pertinent information while obscuring irrelevant details [7]. The decision to provide specific views is intertwined with the stakeholders' roles, expertise, responsibilities, and specific information needs. This method of tailoring access to only the essential parts of the model not only enhances efficiency and clarity for the stakeholders but also acts as an initial layer of information security by limiting information exposure. However, this layer is simply the beginning, since the collaborative nature of working on shared models demands fine-grained access control mechanisms. These can aid in preventing unintentional or unauthorized changes that could jeopardize the model's confidentiality and the integrity carried by it [8].

While a recent study by David et al. [9] acknowledges the need for these capabilities by industrial practitioners in collaborative settings, it also highlights a gap between need and actual adoption by practitioners. This gap can partly stem from the challenges associated with the development of these capabilities, which is not only time- and resource-intensive, but also fraught with potential errors and technical complexities. Unlike generic tool enhancements that apply broadly to modeling environments, these features are intertwined with the underlying modeling languages. This necessitates development efforts tailored to each specific language and for any alterations to the language itself necessitates subsequent modifications to these features, further exacerbating the challenge.

To address this gap, in this doctoral thesis, we propose a model-driven approach that facilitates the development of blended modeling environments featuring multiple views and enforcing modeled information security. The approach is implemented as a framework that leverages automation to provide the core building blocks of this kind of modeling environment. More specifically, this includes i) support for the automatic provision of synchronization mechanisms between graphical and textual notations in blended modeling, ii) support for the automatic provision of synchronization mechanisms between view models and base model in multi-view modeling, and iii) support for the consistent definition and automatic enforcement of access permissions. By automating the provision of these mechanisms, the framework streamlines the development process of the aforementioned collaborative capabilities and their continuous evolution for arbitrary modeling languages. As such, it lowers the barriers to their adoption, encouraging their use in collaborative MDSE initiatives.

Thesis outline: This doctoral thesis is structured in two parts. Part I serves as an overview to the thesis, structured in the following manner. Chapter 2 discusses the background and related work. Chapter 3 describes the research problem and motivation, introduces the research goals, and presents the research methodology employed. Chapter 4 presents the contributions and papers included in this thesis. Chapter 5 reflects on the research goals and on the collective advantages and limitations of the corresponding contributions, paving the way for a broader understanding of their impact. Chapter 6 concludes Part I by summarizing the key findings and drawing conclusions. Part II includes the collection of included papers.

Chapter 2

Background and related work

In this chapter, we provide the background for the work presented in this thesis together with a list of related works that aid in understanding the concepts used throughout this thesis.

2.1 Model-driven software engineering

Model-Driven Software Engineering (MSDE) is a software engineering approach that addresses the complexity of software systems by shifting the focus of software development from code-driven approaches to model-driven approaches [10]. The core principle of MDSE is to leverage *abstraction* for constructing models that describe software systems and based on these models, employ *automation* to derive the actual software. Models are considered first-class entities and are the backbone of the entire MDSE process. They provide an abstract representation of a software systems and enable individuals to focus on essential aspects of complex problems by omitting those irrelevant to their specific application. Other important artifacts, such as code and documentation, can be produced automatically from these models, relieving developers from issues such as underlying platform complexity or the inability of third-generation languages to express domain concepts.

MDSE typically utilizes a four-layer hierarchical metamodeling architecture, as illustrated in Figure 2.1, to delineate the relationship among four key levels:

object, model, metamodel, and meta-metamodel. The M_0 layer, also known as the instance layer, represents the lowest level of abstraction and it contains instances of the software system being modeled or broadly refered to as objects. The M_1 layer contains models that are abstractions of the artifacts at the M_0 layer. They serve to simplify and clarify the system by focusing on relevant aspects and omitting unnecessary details. Multiple models can represent a single software system, each emphasizing different characteristics or levels of detail. The M_2 layer contains metamodels that define the syntax and semantics of the models at the M_1 layer. The M_3 layer, the highest level of abstraction in the stack, contains the meta-metamodel, which is a model that defines the structure and semantics of metamodels at the M_2 layer. The meta-metamodel is essentially the language or framework used to define metamodels. It is self-descriptive, meaning the meta-metamodel conforms to itself. Several metamodeling frameworks have been proposed with two of the most widely used ones being the OMG Meta Object Facility (MOF)[1] with the UML metamodel that defines the structure and semantics of the Unified Modeling Language [2], and the Eclipse Modeling Framework (EMF) [3] with the Ecore metamodel.

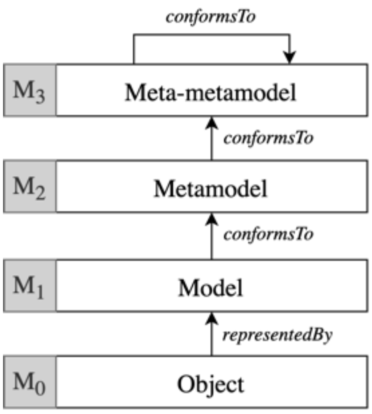


Figure 2.1. The four-layer metamodel architecture

[1] `https://www.omg.org/spec/MOF/2.5.1/PDF`
[2] `https://www.omg.org/spec/UML/2.5.1/PDF`
[3] `https://eclipse.dev/modeling/emf/`

2.1.1 Modeling languages

There are broadly two categories of modeling languages: general-purpose and domain-specific [11]. General-purpose modeling languages, such as UML and Java, are designed to address a broad range of problems. They offer versatile constructs that facilitate modeling across a variety of domains. In contrast, domain-specific modeling languages (DSMLs) are tailored for specific domains, offering a more focused set of language constructs. The use of DSMLs has the advantage of having a concise and tailored language that engineers can more easily understand, thus providing expressive power limited to a particular domain [12]. For instance, while state machines can be implemented in general-purpose programming languages such as C, the process is often tedious and error-prone since it involves lower-level abstractions and potentially even the need for syntactic tricks. The alternative is to define state machines using a DSML designed specifically for this purpose, containing concepts such as *states* and *transitions*, which is more concise and less likely to result in errors. As detailed in Figure 2.2, a DSML is composed of three main components: abstract syntax, concrete syntax and semantics. The abstract syntax which can be represented by a metamodel defines the underlying structure of the language without considering its visual representation, essentially describing the conceptual entities of the domain and their relationships. The concrete syntax refers to the notation or representation of the language, which can for instance be textual or graphical, allowing users to interact with and represent models in a way that is understandable and intuitive within the domain. Semantics give meaning to the constructs defined by the abstract syntax, dictating how models in the DSML are to be interpreted or executed, and are categorized into denotational, operational, translational, or pragmatic [13].

2.1.2 Model transformations

As defined by Kleppe et al. [14], a *model transformation* can be described as the automated process of generating a target model from a source model based on a predefined transformation definition. The *transformation definition* consists of a set of rules that dictate how a model in the source language should be transformed into a model in the target language. Each *transformation rule* describes how

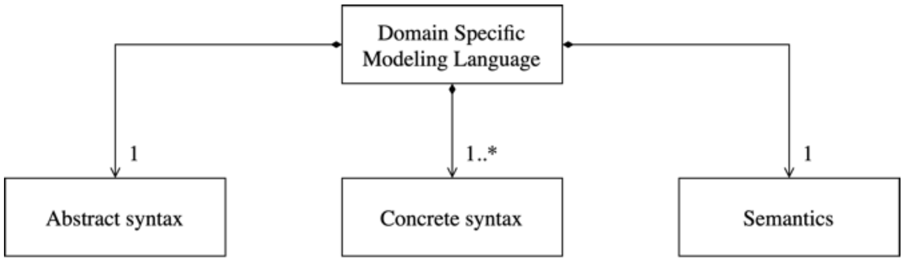


Figure 2.2. Structure of domain-specific modeling languages

one or more constructs in the source language should be transformed into one or more constructs in the target language. To illustrate these concepts, refer to the transformation pattern depicted in Figure 2.3. The transformation definition conforms to a specific transformation language and outlines a series of rules governing the translation of constructs from *metamodel A* to *metamodel B*. A transformation engine utilizes the transformation definition along with *model A* as inputs. It then applies the specified rules from the transformation definition to execute the transformation, resulting in the generation of *model B*. Model transformations can be classified in various ways, depending on their distinct characteristics [15].

Abstraction levels. When considering the abstraction levels of the source and target models, these transformations are divided into two types: *horizontal* model transformations, in which the source and target models are on the same level of abstraction, and *vertical* model transformations, where the source and target models are at different levels of abstraction.

Source and target language. When considering the language in which the source and target models of a transformation are expressed, the transformations can be categorized into *endogenous transformations* between models expressed in the same language, and *exogenous* transformations between models expressed in different languages.

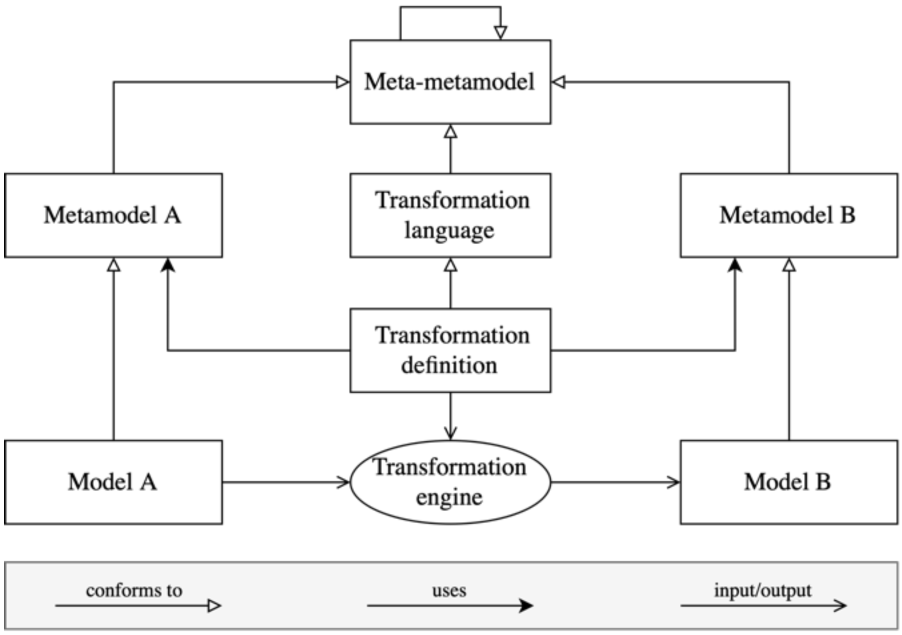


Figure 2.3. Transformation pattern

Target model. Based on where the transformation results are stored relative to the source model, model transformations can be categorized into *in-place* and *out-place* transformations. In in-place transformations, the changes are applied directly to the source model. This approach is typically used when the model needs to be updated or optimized without the necessity of keeping the original version intact. On the other hand, out-place transformations create a new target model separate from the source model, leaving the source model unchanged.

Type of source and target. Based on the type of source and target, model transformations can also be classified into three main categories namely, model-to-model (M2M), model-to-text (M2T) and text-to-model (T2M) transformations [16].

- *Model-to-text.* M2T transformations convert one or more models into textual representations, including but not limited to source code in languages such as C++ or Java, as well as various forms of configuration files.

- *Text-to-model.* Conversely, T2M transformations process textual inputs, such as strings of characters, and convert them into structured models.

- *Model-to-model.* M2M transformations translate between source and target models. They can follow either a declarative, imperative, or hybrid approach. Declarative approaches such as Query/Views/Transformation Relational (QVT-R) [4] emphasize defining relationships between elements in the source and target models without specifying an order of execution, whereas imperative approaches such as Query/Views/Transformation Operational (QVT-O [4] emphasize specifying how the transformation should be executed by specifying the order of the transformations. Alternatively, hybrid approaches such as Atlas Transformation Language (ATL) [17] combine both declarative and imperative constructs. Additionally, depending on the direction of the propagation of changes, M2M transformations can be categorized as *unidirectional* or *bidirectional*.
Bidirectionality can be achieved either by combining a pair of separate unidirectional transformations, or by using languages and approaches that support the definition of bidirectional transformations. For the latter, one approach is using QVT-R, the declarative language of the comprehensive QVT specification [4] introduced by the Object Management Group (OMG)[5]. QVT-R has been applied to several case studies[18, 19], but while it provides many interesting and promising features, the language has not been adopted widely [20]. Tools like Medini [6], ModelMorf [7], and QVT Declarative [8] represent implementations of QVT-R; however they support the standard only partially so far [20, 21]. A comparison between QVT-R and QVT-O [22] points out that while QVT-R supports bidirectional transformations, there exists certain ambiguity about

[4] https://www.omg.org/spec/QVT/1.2/PDF/
[5] https://www.omg.org
[6] http://projects.ikv.de/qvt/wiki
[7] https://web.archive.org/web/20120323171429/http://www.tcs-trddc.com/trddc_website/ModelMorf/ModelMorf.htm
[8] https://projects.eclipse.org/projects/modeling.mmt.qvtd

whether QVT-R is able to specify and support non-bijective transformations [23]. On the other hand, QVT-O, while supporting only unidirectional transformations provides better control over the transformation process. Since QVT-O is similar to modern object oriented languages and imperative programming is more common, developers find it easier to adopt and work with. Another approach for defining bidirectional transformations is using Janus Transformation Language (JTL) [24]. JTL is a declarative model transformation language specifically tailored to support bidirectionality and change propagation even for non-bijective transformations. It employs a QVT-R like syntax language and uses logic programing [25] – more specifically, Answer Set Programming (ASP) – to generate a set of possible target models. The target models are generated from scratch, hence it does not provide incrementality. Due to supporting non-bijective transformations, multiple target models may be generated, leaving users to choose from the proposed solutions. However, the vast number of potential target models, which could be infinite, poses a significant challenge. Latest version of the JTL tool [9] dates back to 2019.

Triple Graph Grammars (TGGs) [26] are another formalism for defining bidirectional transformations. The basic idea behind TGGs is to interpret source and target models as graphs and have a correspondence graph whose nodes reference corresponding elements from both source and target graphs, respectively. TGGs describe model transformations in a highly declarative way. The rules themselves do not contain any information about a transformation direction. However, while triple graph grammars are in theory a natural choice for the realization of bidirectional model transformation, in practice the required correspondence specification can be quite complex, especially for non-bijective transformations. Moflon [10] is a tool capable of generating Java programs starting from diagrammatic specifications of graph transformations. The generated code realizes two separate unidirectional transformations which as in other bidirectional languages should be consistent by construction. While the forward transformation implementation in Moflon can be considered complete with respect to the transformation specification, the backward program restricts the change

[9] `https://github.com/MDEGroup/jtl-eclipse/releases/tag/v0.3.0.2019`

[10] `https://github.com/eMoflon`

propagation to attribute updates and element deletions; hence, is restricted to contexts where the transformation can exploit trace information [24]. Other tools implementing TGGs are TGG-Interpreter [11], Henshin TGG [12], MoTE [13], Atom3 [14], and EMorf [15]. A detailed comparison of Moflon to other TGG tools can be found in the survey by Hildebrandt et al.[27]. A comparison between QVT-R, JTL, and TGGs can be found in the comparative study by Samimi et al.[28].
Lenses [29] are another approach for defining bidirectional transformations. In asymmetric lenses, one of the structures is always a view of the other. Asymmentric lenses consist of two types of functions: i) *get*, which returns the view when applied to a source, and ii) *put* which restores a modified view into existing source. Since asymmetric lenses are restricted to scenarios where one of the two models to be synchronized is an abstraction of the other, they do not seem to be flexible enough for the general MDE setting. In symmetric lenses, each of the two models may contain information that is not present in the other [30], a more common scenario in MDE settings. Symmetric lenses consist of *putr* – which transfers information from left to right — and *putl* – which transfers information from right to left – functions.
In contexts where non-bijective mappings are present, employing bidirectional transformations using the aforementioned techniques becomes particularly challenging. This complexity arises because these methods generally assume a one-to-one correspondence between elements in the source and target models. However, when mappings are non-bijective, it can be difficult to determine unique counterparts for certain elements, complicating the reverse transformation process. This ambiguity can lead to issues in correctly restoring original states or propagating changes effectively, thereby undermining the reliability and predictability of the transformation system [23].

Besides the main three categories of the aforementioned model transformations, a distinct category of transformations known as Higher-Order Transformations (HOTs) [31], involves model transformations wherein the input and/or

[11] http://jgreen.de/tools/tgg-interpreter/
[12] https://projects.eclipse.org/projects/modeling.emft.henshin
[13] https://www.hpi.uni-potsdam.de/giese/update-site/
[14] http://atom3.cs.mcgill.ca/
[15] http://emorf.org/index.html

output models are transformation models themselves. HOTs are used to harness the power of transformations by utilizing them as objects. Consider, for instance, the QVTo metamodel. A QVTo model transformation can be viewed as a model adhering to the QVTo metamodel. HOTs can either utilize this QVTo model transformation as input, generate it as output, or perform both operations.

2.1.3 Mapping modeling languages

A mapping modeling language (MML) refers to a structured and formalized means for specifying mapping correspondences between model elements. A *mapping model* is an instantiation of the MML, incorporating the *mapping rules* that encapsulate the correspondences between elements of the *mapped models*. These mapped models can range from terminal models to metamodels, and even meta-metamodels. Here, we focus on metamodels as mapped models. The term *mapping model* is sometimes used interchangeably with the term *weaving model* [32], even though the links defined in weaving models typically offer richer semantics. To readers unfamiliar with these concepts, mapping and weaving models might appear akin to model transformations; however, they fundamentally differ in their roles and applications within the model-driven engineering process. Mapping models are essentially declarative specifications that describe how elements from one model (the source) relate to elements in another model (the target). These mappings are used to define correspondences and relationships between different models, usually at a high level of abstraction. They do not, by themselves, execute to generate the target model from the source model. On the other hand, model transformations are the mechanisms through which models are automatically converted from one form to another. They are executable specifications that define how a source model is to be processed and transformed into a target model. This involves a set of rules or algorithms that specify how elements in the source model are to be matched, manipulated, and generated in the target model. Model transformations are, therefore, executable and can be directly applied to automate the generation of target models from source models, based on the transformation definition.

Mapping and weaving models are frequently utilized to streamline the generation of model transformations, as the established correspondences within these models guide the transformation definition. The Atlas Model Weaver

(AMW) serves as a notable example of a tool that leverages this process [33, 34, 35, 36] within EMF. AMW supports the definition of relationships, or links, between two metamodels, and the generation of model transformations that adhere to the syntax and semantics of the ATL language. These links are stored in a model known as a weaving model, which is created according to a weaving metamodel. AMW uses element-to-element and structural methods to support the automatic creation of weaving models. Element-to-element methods calculate similarity values between elements of input models, pairwise, using string similarity and dictionary of synonyms, while structural methods use internal properties and element relationships. The defined weaving links can be unidirectional or bidirectional. Bidirectional links can be used to generate both the forward and backward transformations. While this can simplify the initial design since it encapsulates the transformation logic for both directions in one place and can be sufficient for simple and straightforward correspondences, when the transformation logic is complex bidirectionality is a desirable feature but can prove to be complex [33]. A dedicated weaving model for each direction can provide more control and clarity, especially when dealing with non-bijective correspondences. Although this might initially require more effort to set up and maintain, it provides clearer separation. Changes in one direction do not automatically affect the other, which can be relevant if different aspects evolve at different rates or in different ways. Moreover, having a dedicated weaving model can support reuse, as it can be reused in different contexts where only one direction is needed.

Regardless of how they are defined – manually, automatically, or semi-automatically – or their specific applications, mapping modeling languages offer numerous advantages that streamline and improve the engineering process. By offering high-level abstractions, these languages condense complex technical specifics into simpler, more manageable concepts. Such simplification plays a critical role in democratizing the engineering process, enabling participation from non-technical stakeholders. This inclusivity not only fosters broader collaboration but also enriches decision-making by integrating diverse perspectives. Furthermore, mapping modeling languages significantly contribute to knowledge preservation within organizations. By documenting the information in a clear, abstracted manner, they create a repository of knowledge that is easier to understand, update, and transfer between teams and workers. Moreover, the inherent reusability of mapping models underpins their value; they can be repurposed across various contexts with minimal adjustments, emphasizing efficiency and adaptability. This versatility is key to achieving sustainable engineering practices.

2.2 Blended modeling

Ciccozzi et al. [5] formalize the use of multiple modeling notations as *blended modeling* and define it as:

> *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 focuses on the provision of multiple concrete syntaxes, or simply *notations*, for a non-empty set of abstract syntactical concepts. As a result, it is designed to accommodate a variety of notations, each of which is designed to meet specific modeling requirements; in this work, we focus on graphical and textual notations, but the theories and tools presented could be applied to virtually any combination of concrete syntaxes. The advantages of blended modeling include the ability to switch seamlessly between graphical and textual notations at any stage of the modeling process; thus, stakeholders can choose the appropriate notation based on their experience and task at hand. Aside from leveraging the individual benefits of both graphical and textual notations, the ability to visualize and edit the same information simultaneously through synchronized notations allows for flexible separations of concerns, improved interdisciplinary communication, and faster modeling activities.

Several methods and tools appear to offer blended modeling capabilities at first glance. However, many of these tools, including but not limited to FXDiagram[16], MetaUML[17], and PlantUML[18] primarily utilize textual notation

[16] https://jankoehnlein.github.io/FXDiagram/
[17] https://github.com/ogheorghies/MetaUML
[18] https://plantuml.com

for model interaction, using graphical notation merely for visual representation. Some tools such as QuickDBD [19] and DBDiagrams [20] provide limited editing features in their graphical editors. Others support interactions through both textual and graphical notations; however, these are typically specific to certain modeling languages, often designed exclusively for UML and its profiles [6, 37, 38]. In hybrid modeling, some parts of the model are expressed with a graphical syntax, while others with a textual syntax, all atop a single abstract syntax [39]. Projectional approaches such as JetBrains MPS[21] and Melanee [22] can also be mistaken for blended modeling. However, since in projectional approaches the abstract syntax tree (AST) is modified directly upon every editing action and bypasses the stages of the parser-based approaches, in terms of textual notations, projectional modeling tools only imitate the behavior of parser-based textual editors. An in depth investigation of these tools and others can be found in our systematic literature review for blended modeling in commercial and open-source model-driven software engineering tools [40].

The main implication of the definition of blended modeling [5] is that it assumes a *single abstract syntax* supported by multiple concrete syntaxes. While this approach proves effective in specific contexts, it can lead to limitations in others, where the adoption of multiple abstract syntaxes (one for each concrete syntax) can offer significant advantages for the following reasons:

- *Enhanced customization:* When each concrete syntax is paired with its own dedicated abstract syntax, the definition and management of the concrete syntax becomes more straightforward and efficient. This pairing allows for a higher level of syntax-specific customization, ensuring that the concrete syntax can be finely tuned to meet specific user requirements. This is crucial because notations are designed to be highly adaptable to diverse users' needs, serving different purposes and varying levels of detail. Attempting to force a single abstract syntax to support different notations can lead to compromises in customization and usability.

[19] `https://www.quickdatabasediagrams.com/`
[20] `https://dbdiagram.io/d`
[21] `https://www.jetbrains.com/mps/concepts/`
[22] `https://melanee.org/about/`

- *Avoidance of syntactical pollution:* In practice, different users may require different notations of the same type to highlight various aspects of the same modeling concepts, tailored to their unique needs. By allocating a dedicated abstract syntax for each notation, we avoid the syntactical pollution that occurs when one abstract syntax is stretched to accommodate multiple notations, not considered during the initial definition of the abstract syntax. This pollution can dilute the clarity and effectiveness of the notations, making them less intuitive and harder to use for their intended audiences.

- *Existing modeling landscape:* The landscape of modeling languages reveals a proliferation of notation-specific DSMLs that formalize overlapping aspects of the same underlying language. These DSMLs accommodate the diverse needs of various communities, stakeholders, and purposes. Consequently, it becomes evident that prevailing modeling practices often deviate from the conventional notion of blended modeling supporting the definition of multiple concrete syntaxes atop a single abstract syntax. Instead, specific instances arise where the definition of multiple abstract syntaxes becomes imperative to effectively cater to the varied range of notations and stakeholders. Therefore, when implementing blended modeling approaches, it is crucial to acknowledge and address these nuanced scenarios, given their prevalence in current modeling contexts.

Considering these arguments, it becomes evident that incorporating multiple abstract syntaxes into blended modeling frameworks can yield significant benefits. However, this adoption necessitates certain adjustments, foremost among them being the synchronization infrastructure. In conventional blended modeling practices, a prevalent strategy involves employing multiple editors (e.g., graphical and textual) to interact with a single model. For example, Obeo and Typefox have explored the integration of Xtext and Sirius [23], where users can opt to edit the model either graphically or textually, based on the suitability for editing specific model elements. Model editing is facilitated through two distinct editors: the Sirius graphical editor and the Xtext textual editor. Notably,

[23] `https://www.obeodesigner.com/resource/white-paper/WhitePaper_XtextSirius_EN.pdf`

modifications made and saved in one editor are automatically synchronized with the other editor, thanks to built-in synchronization mechanisms. In blended modeling solutions integrating multiple abstract syntaxes, this synchronization infrastructure proves inadequate, as synchronization must occur at the abstract syntax level. Hence, blended modeling solutions integrating multiple abstract syntaxes necessitate the establishment of synchronization mechanisms between all the various abstract syntaxes, potentially through model transformations. In other words, integrating multiple abstract syntaxes can lead to challenging solutions, at least from a technical perspective.

2.3 Multi-view modeling

The necessity for multi-view modeling (MVM) in software engineering is fundamentally driven by the inherent complexity of large-scale software systems. Such systems present challenges that cannot be effectively addressed through a single model [41]. Such a model tends to be either overly simplified, missing essential details, or overly complex, becoming hard to manage and understand. Multi-view modeling provides tailored views that cater to varied domain-specific concerns and offer perspectives that align with the expertise and goals of individual stakeholders [42]. The core of multi-view modeling lies in the viewpoint/view/model paradigm, as formalized by the ISO/IEC 42010 standard [43]. A viewpoint defines a particular abstraction framed by a specific set of constructs and principles, aimed at addressing particular concerns within a system. As such, it dictates the conventions, such as notations, languages, and types of models, essential for developing views. In the context of MDSE, a viewpoint is represented by a metamodel, as illustrated in Figure 2.4, which is a subset of the base metamodel upon which the viewpoint is established. A view materializes from applying a viewpoint to a specific system-of-interest and is represented through one or more models, each an instantiation of the metamodel that represents the viewpoint.

As the different viewpoints in multi-view modeling represent the same system to be modeled, requirements for managing synchronization and consistency amongst the views are inherent. The synchronization approaches are primarily dependent on the construction approach of the viewpoints/views [7]. The

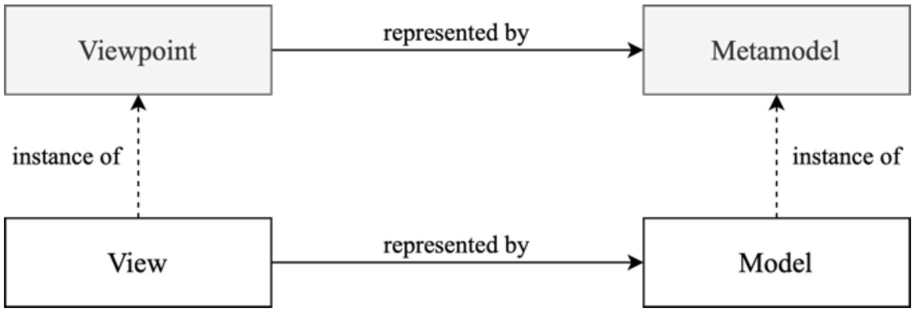


Figure 2.4. Relationship between viewpoint/view to metamodel/model

synthetic approach implements each view as a separate metamodel, with the overall system being obtained as a synthesis of information carried by the various views [44]. Here, it becomes imperative to carefully define the interactions between viewpoints/views to ensure synchronization. Yet, as views increase, maintaining synchronization becomes progressively challenging. Moreover, the addition or modification of views necessitates a revision of synchronization processes. In contrast, the *projective* approach offers users virtual views made up of selected concepts from a single base metamodel [45]. This ensures automatic synchronization, as manipulations are centralized to one single model. However, it demands a well-defined base language semantics and limits customization due to a static base language and predefined views. The *hybrid* approach [46] merges the strengths of both synthetic and projective aproaches. It facilitates view definitions on a base metamodel (resembling the projective approach), but these views manifest as distinct metamodels (similar to the synthetic approach). This ensures inherent synchronization during view definition, alongside the flexibility of introducing views at any development stage.

2.4 Access control

Access control is a fundamental aspect of information security that governs how access to information is granted or denied within an organization or a system. It is designed to ensure that only authorized users, systems, or processes obtain access to the resources they need while preventing unauthorized access. Access control mechanisms are crucial for protecting sensitive information, maintaining data integrity, and ensuring compliance with regulatory standards [47].

Access control operates fundamentally through two key processes being *authentication* and *authorization*. Authentication involves the verification of a user's credentials to confirm their identity. This process requires users to present valid credentials, which are then cross-referenced against a pre-existing database to authenticate their identity. This ensures that only authenticated individuals gain access to designated resources. Authorization involves determining the scope of access and permissible actions for authenticated users. It is governed by access control policies, which encompass rules that define the levels of access granted to various resources [48].

Access control policies are typically categorized into discretionary access control (DAC) policies and non-discretionary access control (NDAC) policies [49]. DAC policies grant the owner of the resource, the autonomy to control other users' accesses to the resource. For instance, services such as Google Drive [24] enable file or folder owners to distribute access to others by assigning specific permissions, like the ability to edit or view, which can be adjusted or withdrawn when needed. NDAC policies are characterized by rules that users cannot modify at their discretion. Instead, these policies enforce controls that can only be altered through central administration. This model is particularly apt for scenarios necessitating rigorous security and hierarchical access controls, such as in Amazon Web Services (AWS) Identity and Access Management (IAM) [25], where access is managed centrally using predefined policies and roles [49, 50].

NDAC policies are essential in complex software systems owing to their ability to enhance security by centralizing access control, allowing only administrators to modify or restrict access. Examples of NDAC policies are multi-level security (MLS), role-based access control (RBAC), attribute-based access control (ABAC), and separation of duty (SoD) [49]. Even though RBAC is technically a form of NDAC, many computer security texts, list it as one of the three primary access control policies, with the other two being DAC and

[24] https://www.google.com/drive/
[25] https://aws.amazon.com/iam/

NDAC. In RBAC, user access is determined by their organizational roles, with privileges organized by role [51]. This approach simplifies security management and policy enforcement by tying access rights to roles rather than individuals. Roles, reflecting users' skills and duties, dictate allowable actions and can be adjusted or reassigned to accommodate changes in responsibilities or organizational needs. This method streamlines privilege management, allowing for easier updates to access rights without modifying each user's settings individually [50].

2.5 UML real-time profile

UML Real-Time (UML-RT) profile is a real-time profile that aims to simplify complex software architecture specification for real-time embedded systems. The UML-RT concepts are inherited from those defined in the Real-Time Object-Oriented Modeling Language (ROOM) [52], and are represented using UML extensibility mechanisms. UML-RT enables both *structure modeling* and *behavior modeling* of real-time systems [53]. The structural part is represented using composite structure diagrams, whereas the behavioral part is represented using state machine diagrams. The fundamental concepts of UML-RT are capsules, which are encapsulated active entities that can be run in parallel. UML-RT relies on state machines for the modeling of capsules' behavior. In the case of a missing state machine, the capsule only operates as a container for other sub-capsules. A behavioral state machine in UML-RT is composed of states, pseudostates, and transitions. States can be simple or composite, and the presence of composite states results in a hierarchical state machine. Pseudostates consist of the initial pseudo state that acts as the starting point of the state machine, and choice and junction pseudostates where guards on outgoing transitions determine which one to execute next. The remaining pseudostates (i.e., entry, exit, and history) are only used in hierarchical state machines. The entry and exit pseudostates are used to enter and exit composite states, while the history pseudostate is used to invoke the last active state prior to the exit of the composite state. Transitions indicate a change of state and can contain triggers that initiate transitions in the form of events, guard conditions that must be evaluated to be true for the initiation of the transition, and effects.

Chapter 3

Research overview

In this chapter, we introduce the research problem and motivation, the research goals of the thesis, and the research methodology that guided our research efforts.

3.1 Research problem and motivation

The motivation for this research work is rooted in the recognition of the challenges associated with the development of blended modeling environments featuring multiple views and enforcing information security.

On the one hand, blended and multi-view modeling requires robust synchronization mechanisms to ensure change propagation across different notations or views. In blended modeling, any modification in one notation must be seamlessly propagated to the others. Establishing the required synchronization mechanisms can be a technically demanding and challenging process [6]. This makes the development and maintenance costly and labor-intensive, especially when modeling languages evolve. Similarly, multi-view modeling, which provides multiple views for different concerns, faces synchronization challenges [54]. Each new view necessitates a specific synchronization setup to reflect changes consistently across the base model and the existing views. This manual process is challenging and it discourages the addition of views.

On the other hand, given the heterogeneous pool of stakeholders and the varying sensitivity of modeled information, the meticulous definition and implementation of fine-grained access permissions is indispensable. Yet, manually defining and managing these permissions – ensuring they are non-conflicting and enforced at all times – is a daunting task. It introduces risks of human error, inconsistency, and scalability challenges. Maintaining security, ensuring compliance, and managing access becomes increasingly difficult and prone to errors [51]. Furthermore, each permission alteration may need to re-iterate the entire process, exacerbating the challenge, particularly due to the frequent nature of such changes.

Effective solutions to tackle these challenges are imperative for facilitating the development and supporting the adoption of blended modeling environments featuring multiple views and enforcing information security.

3.2 Research goals

In light of the outlined research problem and motivation, this research aims at achieving the following overall research goal.

Overall Research Goal (RG): *To facilitate the development of blended modeling environments featuring multiple views and enforcing information security.*

The overall research goal is decomposed into the following three subgoals, each focusing on specific aspects.

- **Research Subgoal 1 (RSG_1): To identify and investigate the current state-of-the-art and -practice on blended modeling solutions in commercial and open-source MDE tools.**

  This subgoal aims to systematically identify and analyze the current state-of-the-art and practical implementations of blended modeling solutions in commercial and open-source modeling tools. This involves assessing existing tools closest to blended modeling, focusing on the user-oriented characteristics relevant to a user of a blended modeling tool and realization-oriented characteristics relevant to a developer of a blended modeling tool. This clarifies assumptions and limitations of existing tools, their effectiveness in delivering blended modeling capabilities, and best practices. With this assessment, we can identify gaps and direct the future development of blended modeling tools.

- **Research Subgoal 2 (RSG_2): To provide support for the automatic provision of synchronization mechanisms for blended and multi-view modeling environments.**

  This subgoal aims to provide support for the automatic provision of synchronization mechanisms for blended and multi-view modeling environments for arbitrary Ecore-based DSMLs. Our research focuses on blended modeling environments, where each notation represents a specific abstract syntax, requiring that synchronization is defined at the level of the abstract syntax (i.e., between metamodels). This principle also applies to multi-view modeling. Given the implications of the hybrid approach, where each view is represented as a separate metamodel, synchronization must similarly be defined at the abstract syntax level.

  The solution should support two main scenarios. For blended modeling, it should support scenarios where metamodels might conceptually completely overlap, partially overlap, or have no overlap. As long as viable mappings between all concepts of the involved metamodels can be established, the solution should generate the necessary synchronization mechanisms. For multi-view modeling, the solution should support the generation of synchronization mechanisms in scenarios where one metamodel acts as a subset of a more extensive one.

- **Research Subgoal 3 (RSG_3): To provide support for the consistent definition and automatic enforcement of access control permissions.**

  This subgoal aims to provide support for the consistent definition and automatic enforcement of access control permissions for ensuring modeled information security for arbitrary Ecore-based DSMLs. Given the involvement of a heterogeneous pool of stakeholders contributing and having access to various sensitive and proprietary information, access permissions should be managed rigorously. The solution should support the definition of consistent and non-contradictory access permissions which customize the stakeholders' ability to create, read, update, or delete model information. Moreover, the solution should support the seamless enforcement of access permissions, ensuring that stakeholders only can interact with the model information as described in the defined permissions. Given the dynamic nature of project teams, where stakeholders may join, leave, or be reassigned to different tasks, the access control solution must adeptly and flexibly handle these changes. Specifically, it should reduce the overhead associated with managing individual user permissions, ensuring flexibility and security in a fast-changing collaborative environment.

3.3 Research methodology

Our research process draws upon constructive research techniques [55, 56] commonly used in the fields of computer science and engineering. Essentially, constructive research emphasizes the construction of an artifact based on existing knowledge and its use in a new manner with the addition of missing links. A fundamental aspect of constructive research is the close relationship of the research to previous theoretical knowledge and the practical relevance of the problem and solution. Essentially, the formulated problem is a topic of practical importance that has not been adequately addressed in the existing literature. Hence, the ideal solution is expected to integrate both theoretical and practical contributions to develop a cohesive whole. Since such solutions require insight from both theoretical and practical standpoints, collaboration between researchers and practitioners is crucial. In our scope, practical insights were predominantly acquired from the industrial partners of the BUMBLE [1] project;

[1]BUMBLE project aimed at providing an innovative system and software development framework based on blended modeling notations/languages (e.g., textual and graphical). Refer to: https://itea4.org/project/bumble.html

we collaborated very tightly with HCL Technologies[2]. Their contributions have been crucial in integrating practical and industrial viewpoints into our research process, as illustrated in Figure 3.1, which was composed of four main steps.

1. **Problem formulation.** The first step involved formulating a research problem with theoretical and practical relevance. We identified current challenges by conducting a systematic review of the existing body of knowledge and collaborating closely with our partners in the BUMBLE [1] project. This process led to identifying a set of research challenges as well as defining our overall research goal and subgoals.

2. **Solution proposal.** The second step involved proposing a comprehensive solution architecture. This process entailed identifying the relevant existing knowledge and resources pivotal for tackling the research challenge. We carried out a thorough analysis of the collected knowledge and artifacts, followed by brainstorming sessions on viable solutions. Then, we reviewed the proposed solutions, focusing on their strengths as well as potential limitations and areas necessitating enhancement. This iterative process continued until we identified the most suitable solution. If new insights or findings arose or issues related to the feasibility or practicality of the originally formulated problem became apparent, we revisited and refined our problem formulation. This was done to ensure a comprehensive understanding of the issue and to mitigate potential ambiguities.

3. **Practical implementation.** The third step involved the practical implementation of the proposed solution. A proof-of-concept prototype was developed; the goal was to confirm the feasibility and practical applicability of the solution. To ensure the prototype's relevance and applicability in industrial settings, we partially integrated it with commercial industrial tools, aligning it with industrial demands and scenarios. Additionally, to extend the solution's reach and impact, we incorporated it in open-source frameworks, thus facilitating accessibility, evaluation, and adoption. During this phase, inconsistencies arising from initial oversight, unforeseen

[2] https://www.hcltech.com

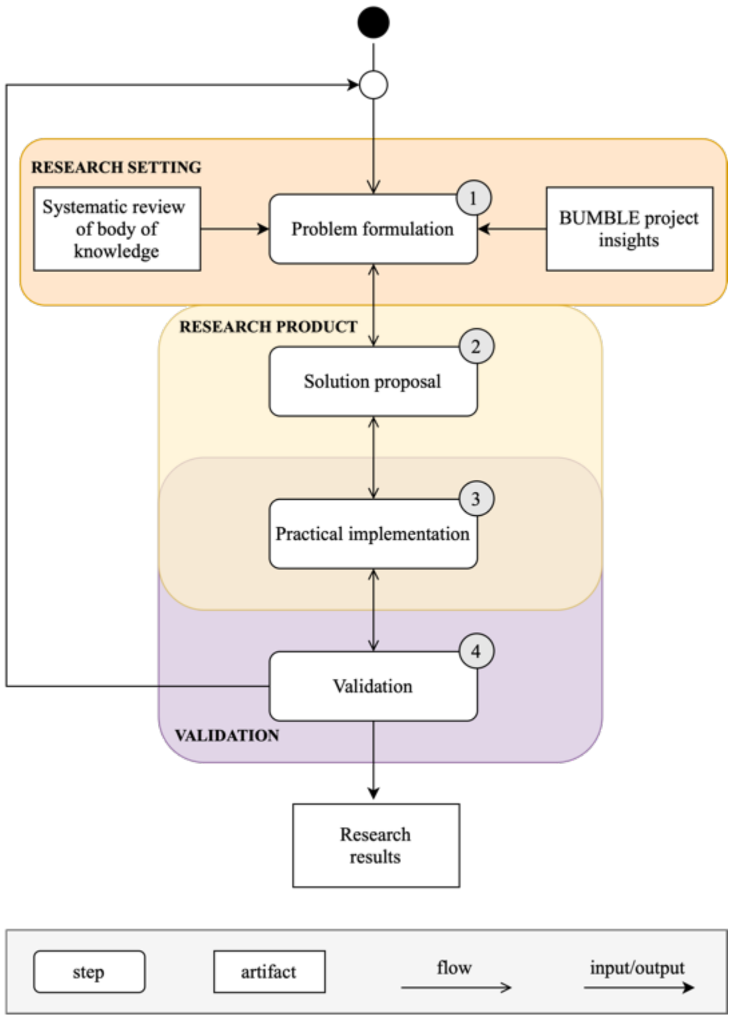


Figure 3.1. Research methodology

practical challenges, or emerging technological barriers that could compromise the solution's feasibility were addressed. We revisited and refined our solution proposal when needed to ensure the outcomes were viable in practical settings.

4. **Validation.** The last step involved the validation of the implemented solution. The core objective was to ensure that the solution, whether integrated into industrial tools or established within open-source frameworks, met the predefined goals and expectations. Validation was conducted by applying the proposed solution in selected use cases, both artificial and industrial. Multiple criteria were considered, such as the solution's ability to address the initially identified problems, its compatibility with existing systems, and its overall usability. The findings derived from this phase provided essential insights into the areas where the solution met expectations, as well as aspects that might require further refinement or improvement. If the validation process uncovered issues related to the solution's practical implementation, including technical limitations, usability challenges, or unforeseen constraints, we returned to the practical implementation phase to address these. Should the validation outcomes be unsatisfactory due to reasons other than implementation issues, we leveraged the insights gained to reevaluate our problem formulation. Conversely, if the validation confirmed our approach as correct, we concluded our research by presenting our results.

Chapter 4

Research results

In this chapter, we discuss our research results. We first detail the thesis contributions and then outline the papers that present each contribution.

4.1 Thesis research contributions

This thesis presents five research contributions (RC_X). RC_1 delivers insights on the current state-of-the-art and -practice on blended modeling solutions (indicated by ◇). RC_2 to RC_5 introduce proposed solutions and their respective proofs-of-concept, which include contributions for specific industrial use cases (indicated by •) and generic contributions that are applicable to a wide range of use cases employing Ecore-based DSMLs (indicated by ∘).

◇ **Research Contribution 1 (RC_1):** Systematic literature review (SLR) on state-of-the-art and -practice on blended modeling solutions in commercial and open-source MDE tools.

• **Research Contribution 2 (RC_2):** Design and implementation of a textual grammar and related textual editor for UML-RT state machines.

• **Research Contribution 3 (RC_3):** Design and implementation of the synchronization mechanisms between graphical and textual notations for a blended modeling environment for UML-RT state machines.

- **Research Contribution 4 (RC_4):** A language-agnostic solution for the automatic provision of synchronization mechanisms for blended and multi-view modeling environments.
- **Research Contribution 5 (RC_5):** A language-agnostic solution for the consistent definition and automatic enforcement of access control permissions.

The mapping of research contributions ti research subgoals is shown in Table 4.1.

	RSG_1	RSG_2	RSG_3
RC_1	✓		
RC_2		✓	
RC_3		✓	
RC_4		✓	
RC_5			✓

Table 4.1. Mapping of research contributions (RCs) to research subgoals (RSGs)

Research Contribution 1 (RC_1): Systematic literature review (SLR) on state-of-the-art and -practice on blended modeling solutions in commercial and open-source MDE tools.

This research contribution focuses on identifying and understanding the potential of current commercial and open-source modeling tools to support blended modeling. For that reason, we designed and carried out a systematic multivocal study encompassing an academic review and a grey literature review. We identified challenges and opportunities in the tooling aspect of blended modeling. Specifically, we investigated the user-oriented – relevant for a user of the blended modeling tool – and realization-oriented characteristics – relevant for a developer of a blended modeling tool – of existing modeling tools that already support multiple types of notations and map their support for other blended aspects, such as inconsistency tolerance, and elevated user experience. We presented the main takeaways and highlighted current challenges and future opportunities related to the concept of blended modeling.

Research Contribution 2 (RC$_2$): Design and implementation of a textual grammar and related textual editor for UML-RT state machines.

This research contribution is the result of a collaboration with HCL Technologies and focuses on the design and implementation of a textual grammar and corresponding editor for UML-RT state machines. HCL's RTist[1] tool initially employed a graphical notation for interacting with UML-RT state machine models. The objective was to complement this with a textual notation, aiming to establish a blended modeling environment for the tool's users. To this end, a specific grammar (i.e., metamodel) was developed for the textual notation as the requirements necessitated different levels of detail compared to the existing graphical notation. A dedicated grammar with a corresponding metamodel was defined to meet the specific needs of the textual notation. The textual grammar was implemented using Xtext, and the design was influenced by direct input from HCL's tool engineers and users, as well as foundational aspects of the UML-RT metamodel. Additionally, the generated textual model editor was customized to address the identified needs and preferences, ensuring that the resulting environment was fully aligned with the user needs and the overall goals of the blended modeling environment. This industrial scenario confirmed our arguments for the need of a blended modeling approach with a dedicated abstract syntax for each notation.

Research Contribution 3 (RC$_3$): Design and implementation of the synchronization mechanisms between graphical and textual notations for a blended modeling environment for UML-RT state machines.

This research contribution is the result of a collaboration with HCL technologies and focuses on developing the synchronization mechanisms between graphical and textual notations for achieving a blended modeling environment for UML-RT state machines. Following the introduction of the textual notation and corresponding grammar for UML-RT state machines in RC$_2$, ensuring seamless synchronization between the existing graphical notation and newly introduced textual notation is required. The solution is achieved by defining model

[1] https://www.hcl-software.com/rtist

transformations in QVT-O that propagate changes across notation-specific models to achieve consistency. The benefits of this effort are two. First, it provides an effective, custom solution for the partner's immediate needs, ensuring seamless synchronization between graphical and textual notations, while simultaneously allowing for the collection of valuable user insights on the resulting blended modeling environment. Second, the insights and methodologies developed during this process allow for a deep understanding of the involved challenges and requirements and lay the groundwork for the subsequent, more ambitious contribution; the automatic generation of synchronization mechanisms.

Research Contribution 4 ($\mathbf{RC_4}$): A language-agnostic solution for the automatic provision of synchronization mechanisms for blended and multi-view modeling environments.

This research contribution presents a language-agnostic solution for the automatic provision of synchronization mechanisms for blended and multi-view modeling environments for Ecore-based DSMLs. Synchronization mechanisms, such as the one presented in RC_3, demonstrate effectiveness, but their manual realization is challenging. This contribution aims to facilitate the development of blended and multi-view modeling environments by supporting the automatic provision of synchronization mechanisms. Unlike RC_3, where the focus was on the realization of a synchronization mechanism tailored for UML-RT state machines, this contribution is language-agnostic and allows for the generation of synchronization mechanisms for arbitrary Ecore-based DSMLs. The solution is achieved by i) designing and implementing a *mapping modeling language* that enables the specification of mapping rules between metamodels on a higher-level of abstraction, as well as providing the capability to automatically generate these mapping models if possible, and by ii) designing and implementing *higher-order transformations (HOTs)* that, leveraging the mapping models, generate the sychronization mechanisms consisting of model transformations conforming to QVTo. The solution has been crafted to accommodate diverse relationships between metamodels.

Research Contribution 5 ($\mathbf{RC_5}$): A language-agnostic solution for the con-

sistent definition and automatic enforcement of access control permissions.

This research contribution presents a language-agnostic solution for the consistent definition of access control permissions for arbitrary Ecore-based DSMLs, and the automatic enforcement of the permissions in tree-based model editors. It employs the role-based access control policy that builds upon the base security layer provided by multi-view modeling. Assigning permissions to roles rather than individuals simplifies management and reduces administrative overhead. The proposed approach supports the consistent definition of fine-grained permissions based on create, read, update, and delete (CRUD) operations for instances of individual meta elements, coupled with automated mechanisms for the generation of customized tree-based model editors that enforce the established permissions. This process ensures a seamless translation of specified policies into functional access controls in tree-based model editors. The solution is achieved by i) the development of a wizard that supports the definition of access permissions for individual meta elements, ii) the definition and application of a set of consistency rules that guarantee the establishment of coherent and non-conflicting permissions, with real-time enforcement during the permission definition process within the wizard, and iii) the customization of tree-based model editor generation processes to incorporate and enforce the specified permissions, thereby producing model editors that align with and uphold the established access permissions.

4.2 Paper contributions

This section provides a summary of the five included publications which encapsulate our five research contributions. Table 4.2 shows the mapping of Papers A-E to the respective contributions presented in each paper.

Paper A: Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study.
Authors: Istvan David, *Malvina Latifaj*, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer, Regina Hebig.

	RC_1	RC_2	RC_3	RC_4	RC_5
P_A	✓				
P_B		✓			
P_C		✓	✓		
P_D				✓	
P_E				✓	✓

Table 4.2. Mapping of papers P_A to P_E to research contributions RC_1 to RC_5

Status: Published in the Software and Systems Modeling journal (SoSyM 2022).
Own contribution: I was the initiator and a key contributor of this work. The detailed plan for the paper was developed through collaborative discussions with the co-authors. I was actively involved in each phase of the systematic literature review and authored a significant portion of the paper's draft.
Abstract: Blended modeling aims to improve the user experience of modeling activities by prioritizing the seamless interaction with models through multiple notations over the consistency of the models. Inconsistency tolerance, thus, becomes an important aspect in such settings. To understand the potential of current commercial and open-source modeling tools to support blended modeling, we have designed and carried out a systematic study. We identify challenges and opportunities in the tooling aspect of blended modeling. Specifically, we investigate the user-facing and implementation-related characteristics of existing modeling tools that already support multiple types of notations and map their support for other blended aspects, such as inconsistency tolerance, and elevated user experience. For the sake of completeness, we have conducted a multivocal study, encompassing an academic review, and grey literature review. We have reviewed nearly 5000 academic papers and nearly 1500 entries of grey literature. We have identified 133 candidate tools, and eventually selected 26 of them to represent the current spectrum of modeling tools.

Paper B: Towards automated support for blended modeling of UML-RT embedded software architectures.
Authors: *Malvina Latifaj*, Federico Ciccozzi, Mattias Mohlin, Ernesto Posse.

Status: Published in the 15th European Conference on Software Architecture (ECSA 2021).
Own contribution: I was the main driver and sole developer of the textual syntax for UML-RT state machines. The plan for the paper was developed through collaborative discussions with the co-authors. I also led the paper writing, with co-authors providing feedback to refine and improve the paper.
Abstract: The Unified Modeling Language for Real Time (UML-RT) is a UML-based domain-specific language for modeling real-time embedded systems. HCL RTist, a model-based development environment for creating complex, event-driven and real-time software with advanced automation features provided by HCL Technologies, provides advanced support for UML-RT. Historically, as for the majority of UML profiles, editing support for UML-RT has also mainly exploited graphical notations (e.g., composite component and state-machine diagrams). Nevertheless, our previous experiments with blended graphical and textual modeling showed that the seamless use of different notations (i.e., graphical and textual) can significantly boost the work of architects and modellers. The results of those experiments together with the exposed wish of RTist customers of being able to design software architectures and applications via multiple notations led us to initiate this work towards an automated support for blended modeling of UML-RT. In this paper we describe the first step of the work – the effort of designing, implementing and integrating a textual notation for UML-RT state-machines in RTist.

Paper C: Blended graphical and textual modeling of UML-RT state-machines: An industrial experience.
Authors: *Malvina Latifaj*, Federico Ciccozzi, Muhammad Waseem Anwar, Mattias Mohlin.
Status: Published in the invitation-only post-proceedings of the 15th European Conference on Software Architecture (ECSA 2021).
Own contribution: I was the main driver and sole developer of the advanced textual editing features for UML-RT state machines' textual editor and synchronization transformations between notations. The plan for the paper was developed through collaborative discussions with the co-authors. I also led the paper writing, with co-authors providing feedback to refine and improve the paper.

Abstract: The ever increasing complexity of modern software systems requires engineers to constantly raise the level of abstraction at which they operate to suppress the excessive complex details of real systems and develop efficient architectures. Model Driven Engineering has emerged as a paradigm that enables not only abstraction but also automation. UML, an industry de-facto standard for modeling software systems, has established itself as a diagram-based modeling language. However, focusing on only one specific notation limits human communication and the pool of available engineering tools. The results of our prior experiments support this claim and promote the seamless use of multiple notations to develop and manipulate models. In this paper we detail our efforts on the provision of a fully blended (i.e., graphical and textual) modeling environment for UML-RT state-machines in an industrial context. We report on the definition of a textual syntax and advanced textual editing for UML-RT state-machines as well as the provision of synchronization mechanisms between graphical and textual editors.

Paper D: Automatic generation of synchronization mechanisms for blended modeling.
Authors: *Malvina Latifaj*, Federico Ciccozzi, Mattias Mohlin.
Status: Published in the Frontiers of Computer Science journal (Frontiers 2023).
Own contribution: I was the main driver and sole developer of the mapping modeling language and higher-order transformations for the generation of synchronization transformations. The plan for the paper was developed through collaborative discussions with the co-authors. I also led the paper writing, with co-authors providing feedback to refine and improve the paper.
Abstract: Blended modeling aims to boost the development of complex multi-domain systems by enabling seamless textual and graphical modeling. The synchronization mechanisms between notations are embodied in model transformations. Manually defining model transformations requires specific knowledge of a transformation language and it is time consuming and error-prone task. Moreover, whenever any of the synchronized languages or notations evolves, transformations become obsolete. Thereby, we propose an automated solution for generating synchronization transformations between domain-specific modeling languages in an industrial setting. Although our main goal was to provide a solution for blended modeling of UML-RT state machines, the pro-

posed approach is language-agnostic. The approach entails i) the specification of mapping rules between two arbitrary domain-specific modeling languages using a mapping modeling language, appositely defined for this, and ii) the automatic generation of synchronization model transformations driven by the mapping rules. We validated the proposed approach on two use cases.

Paper E: Role-based access control for collaborative modeling environments.
Authors: *Malvina Latifaj*, Federico Ciccozzi, Antonio Cicchetti.
Status: Conditionally accepted to the Journal of Object Technology (JOT) at the time of writing.
Own contribution: I was the main driver and sole developer of the solution. The plan for the paper was developed through collaborative discussions with the co-authors. I also led the paper writing, with co-authors providing feedback to refine and improve the paper.
Abstract: Collaborative model-driven software engineering fosters efficient cooperation among stakeholders who collaborate on shared models. Yet, the involvement of multiple parties brings forth valid concerns about the confidentiality and integrity of shared information. Unrestricted access to such information, especially when not pertinent to individual responsibilities, poses significant risks, including unauthorized information exposure and potential harm to information integrity. This work proposes a dual-layered solution implemented as an open-source Eclipse plugin that leverages the role-based access control policy to ensure the confidentiality and integrity of model information in collaborative modeling environments. The first layer limits stakeholders' access to the shared model based on their specific roles, while the second layer refines this access by restricting manipulations to individual model elements. By ensuring that stakeholders access only the information pertinent to their roles and are authorized to manipulate such information in accordance with their expertise and responsibilities, this approach ensures the confidentiality and integrity of shared model information. Furthermore, it alleviates information overload for stakeholders by enabling them to focus only on the model information relevant to their specific roles, thereby enhancing the collaborative efforts.

Chapter 5

Discussion

In this chapter, we provide a discussion on the research subgoals in relation to the corresponding proposed solutions. We further explore the collective impact of the contributions towards fulfilling the overall research goal. Lastly, we contextualize the findings in relation to current limitations and potential enhancements.

5.1 Research subgoals and contributions

In this section, we discuss our research subgoals, contributions, and the papers in which we present these contributions. The relation between these three components is shown in Figure 5.1.

Research Subgoal 1 (RSG$_1$). This subgoal aimed *to identify and investigate the current state-of-the-art and -practice on blended modeling solutions in commercial and open-source MDE tools* and has been addressed in research contribution 1 (RC$_1$) through the execution of a *systematic literature review (SLR)* on the same topic. This contribution has been presented in paper A.

This contribution presents the outcomes of a comprehensive multi-vocal study focusing on the potential, opportunities, and challenges associated with blended modeling solutions. Within the scope of our investigation for this contribution, blended modeling is defined as the seamless integration of multiple notations for a single abstract syntax. Our findings reveal that the majority of examined tools integrate both graphical and textual notations; in most cases with partial overlap. This overlap, rather than serving as a constraint, enhances the utility of each notation by allowing them to complement each other effectively. Among the platforms reviewed, Eclipse emerged as the predominant base for 38 % of the tools, with the remainder distributed across mainly custom platforms. The majority of the examined tools are designed to facilitate collaboration, either offline or in real-time. This is a sound conclusion as the nature of collaboration inherently necessitates the use of diverse notations to accommodate various perspectives and workflows. However, the tools we evaluated only offer limited support for the broader features of blended modeling. Aspects related to modeling flexibility, such as instance-level and language flexibility, have not been addressed in depth yet. Furthermore, our analysis could not uncover substantial information regarding change propagation and traceability, crucial for integrating multiple notations seamlessly.

We further discussed the anticipated need for the applicability of the blended modeling concept within complex engineering domains that might even employ multi-view modeling (MVM) or multi-paradigm modeling (MPM) approaches. The initial premise of blended modeling employing a single abstract syntax may need to be extended to embrace multiple abstract syntaxes to better accommodate complex modeling scenarios. The introduction of multiple abstract syntaxes calls for further research, particularly in terms of coordinating models across different languages. This suggests the emergence of a new generation of modeling tools designed to support an extended version of blended modeling by incorporating semantic-based techniques that support workflows based on multiple abstract syntaxes.

Research Subgoal 2 (RSG$_2$). This subgoal aimed *to provide support for the automatic provision of synchronization mechanisms for blended and multi-view modeling environments* and has been addressed in research contributions 2, 3, and 4 (RC$_2$, RC$_3$, RC$_4$). These contributions have been presented in paper B to E and can be grouped in the following two categories.

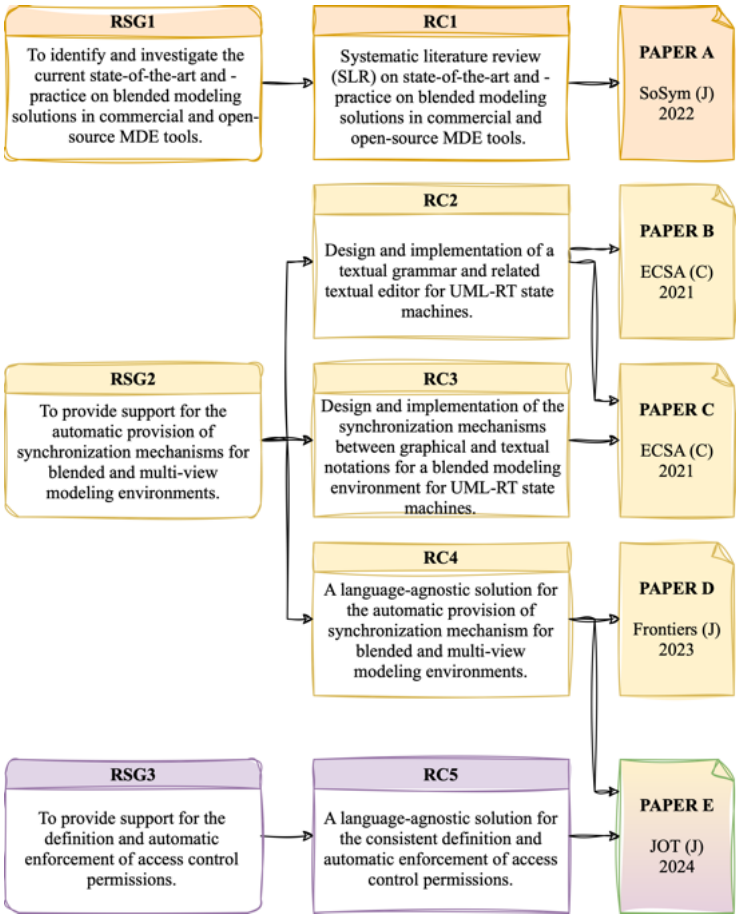
Initial exploration and industry collaboration: As described in the prior paragraph discussing RSG$_1$ and RC$_1$, a key challenge we identified was that with the application of blended modeling in more complex settings, there would be a need to extend the blended modeling concept to multiple abstract syntaxes. The need for blended modeling with multiple abstract syntaxes was similarly observed among industrial partners in the BUMBLE project. Hence, our initial research contributions – RC$_2$ and RC$_3$ – focused on implementing this extended version of blended modeling environments encompassing multiple abstract syntaxes within an industrial setting. This phase, detailed in papers B and C, was conducted in collaboration with HCL Technologies, using their RTist tool for Eclipse – supporting a graphical notation for UML-RT – as a use case. HCL's requirements entailed the inclusion of an additional textual notation for UML-RT state machines and its seamless integration with the existing graphical notation. Hence, we explored extending RTist's capabilities to support blended modeling for UML-RT's state machines, through the development of a textual grammar and notation (RC$_2$) and the development of synchronization mechanisms for seamless integration of graphical and textual notations (RC$_3$). This effort aimed to demonstrate the potential benefits of blended modeling to RTist users and explored the benefits of dedicated abstract syntaxes for each notation. In addition, it helped identify the challenges related to the manual provision of synchronization mechanisms and helped us leverage insights for developing an automated solution for their provision.

Language-agnostic automatic synchronization: RC$_4$ introduced a language-agnostic solution for the automatic provision of synchronization mechanisms across blended and multi-view modeling environments. This solution, applicable to arbitrary Ecore-based DSMLs is presented in papers D and E. Paper D provides the building blocks of the overall approach. More specifically, it presents a mapping modeling language used to define relationships between different notations' abstract syntaxes, and higher-order transformations employed to use the established mapping models to generate model transformations. The solution in paper D focuses on tackling blended modeling scenarios, where metamodels may conceptually completely overlap, partially overlap, or not overlap. In these scenarios, the mapping models are defined manually, since an automatic approach could prove to be inaccurate. Paper E introduces a solution aimed at addressing multi-view modeling scenarios, where one metamodel, referred to as a view metamodel is a subset of the other, referred to as a base metamodel. This solution builds on the approach presented in Paper D, but with two key distinctions. Firstly, it automates the generation of mapping models, leveraging the fact that the view metamodel is an unaltered subset of the base metamodel. Secondly, it introduces an additional higher-order transformation that generates model transformations designed to ensure that upon execution of the model transformation from the view model to the base model, additional information present only in the base model is not lost.

Research Subgoal 3 (RSG$_3$). This subgoal aimed *to provide support for the consistent definition and automatic enforcement of access control permissions* and has been addressed in research contribution 5 (RC$_5$) through *a language-agnostic solution for the consistent definition and automatic enforcement of access control permissions*. This contribution has been presented in paper E.

This contribution supports the consistent and non-contradictory definition of permissions, carried out through a wizard that employs a predefined set of consistency rules. As users define permissions, the wizard applies the consistency rules to dynamically adjust other permissions, ensuring they do not conflict with each other. This feature lightens the cognitive burden on users and guarantees a conflict-free permissions setup. An alternative method might involve the application of conflict resolution rules that prioritize certain permissions over others. However, such a method lacks transparency, as users could find their initial permissions overridden in the model editor due to unseen background processes aimed at resolving conflicts, potentially leading to outcomes not aligned with the user's original intentions. Our approach, in contrast, grants users immediate visibility into how their permission choices affect other permissions and model elements, fostering a more user-centered and transparent interaction. The enforcement of the defined permissions is done in terms of EMF tree-based model editors. The latter are a standard component of the EMF ecosystem and are readily available out-of-the-box for any Ecore-based modeling language.

This availability ensures that any modeling language conforming to Ecore can immediately benefit from these editors without additional setup or integration efforts. The enforcement of permissions is done by tailoring the editors' generation process to take into account the defined permissions and generating a customized model editor for each role associated with a particular view.



RSG1
To identify and investigate the current state-of-the-art and -practice on blended modeling solutions in commercial and open-source MDE tools.
RC1
Systematic literature review (SLR) on state-of-the-art and -practice on blended modeling solutions in commercial and open-source MDE tools.
PAPER A
SoSym (J) 2022
RC2
Design and implementation of a textual grammar and related textual editor for UML-RT state machines.
PAPER B
ECSA (C) 2021
RSG2
To provide support for the automatic provision of synchronization mechanisms for blended and multi-view modeling environments.
RC3
Design and implementation of the synchronization mechanisms between graphical and textual notations for a blended modeling environment for UML-RT state machines.
PAPER C
ECSA (C) 2021
RC4
A language-agnostic solution for the automatic provision of synchronization mechanism for blended and multi-view modeling environments.
PAPER D
Frontiers (J) 2023
RSG3
To provide support for the definition and automatic enforcement of access control permissions.
RC5
A language-agnostic solution for the consistent definition and automatic enforcement of access control permissions.
PAPER E
JOT (J) 2024

Figure 5.1. Mapping of research subgoals to contributions to papers

5.2 Holistic overview of the proposed solution

In this section, we discuss the synergy of the contributions presented in Section 4.1, detailing how their integration drives us toward fulfilling the overall research goal. While the individual contributions span a systematic literature review, industrial contributions, and generic solutions, our discourse here is particularly focused on the latter. The preceding elements, namely the literature review and industrial use cases, served as critical precursors, paving the way for realizing our overall goal. Here, we detail the language-agnostic solution that facilitates the development of blended modeling environments featuring multiple views and enforcing information security. Figure 5.2 illustrates the solution's four-step process.

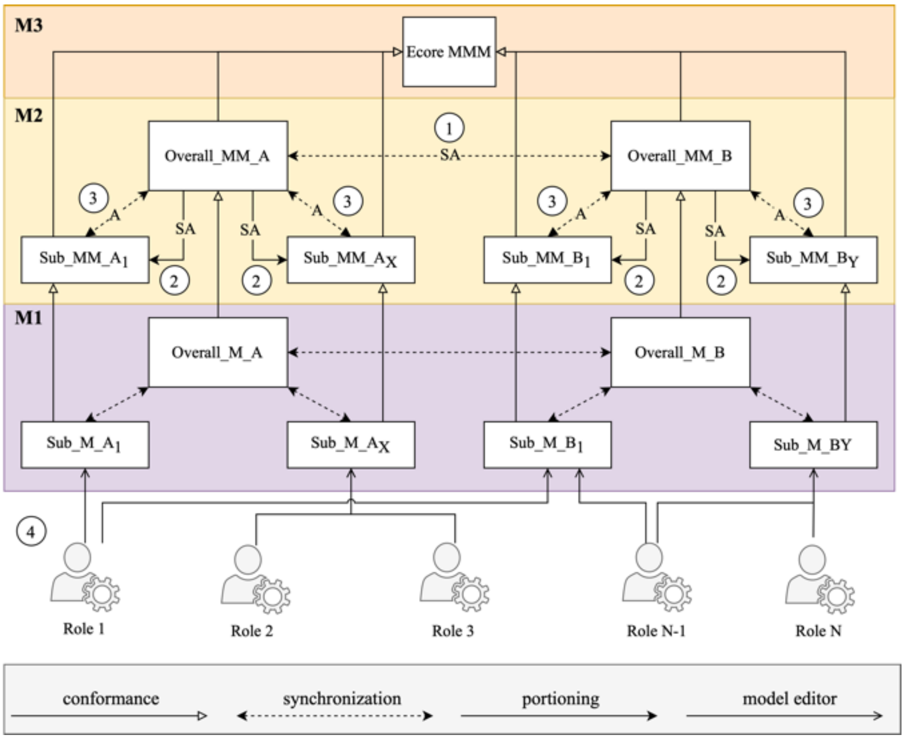


Figure 5.2. Solution workflow

At level M3 we employ the Ecore meta-metamodel. At level M2, we introduce `Overall_MM_A` and `Overall_MM_B`, two Ecore-based metamodels, each designed to support a specific notation.

Step 1. The goal in step ① is to establish synchronization mechanisms for blended modeling. These mechanisms ensure that any modifications made to `Overall_M_A` or `Overall_M_B` – notation-specific models that adhere to `Overall_MM_A` and `Overall_MM_B`, respectively – are accurately reflected and synchronized across both models. Their provision is achieved by employing the solution proposed in RC$_4$. This step is labeled SA (semi-automatic) since it requires human intervention for the definition of mapping correspondences between elements of both metamodels and then employs higher-order transformations for the generation of model transformations. Whenever changes are made to either `Overall_M_A` or `Overall_M_B` at level M1, the execution of the generated model transformations ensures that these changes are reciprocally propagated, thereby supporting the seamless integration between notations in blended modeling contexts.

Step 2. The goal in step ② is to enable the construction of multiple views – namely `Sub_MM_`A_1, `Sub_MM_`A_X, `Sub_MM_`B_1, and `Sub_MM_`B_Y, which are essentially subsets of `Overall_MM_A` and `Overall_MM_B`. The process of selecting elements for each view is manual, whereas the creation of the metamodel that materializes the view (e.g., `Sub_MM_`A_1) occurs automatically. As a result, the procedure in step ② is classified as SA (semi-automatic). In defining the view, it is also determined which roles should be granted access to the view models. After the selection of elements corresponding to each view, the goal in step ② is the definition of access permissions allocated to each role for the given view. This task, while executed by the user, is assisted by a predetermined set of consistency rules, seamlessly integrated into the wizard interface to enforce consistent permissions definition.

Step 3. The goal in step ③ is to establish synchronization mechanisms between the views and the corresponding overall model. For instance, for `Overall_MM_A`, synchronization mechanisms between `Overall_MM_A` -

`Sub_MM_`A_1 and between `Overall_MM_A` - `Sub_MM_`A_X must be defined. Synchronization between views, therefore, is not peer-to-peer; instead, the overall model acts as a central mediator that all view models synchronize with. In this context, since views represent subsets of the overall model, deterministic mapping correspondences between elements are extracted without the need for human intervention. The generated mappings are used as input to higher-order transformations which subsequently generate model transformations; the process is automatic and achieved by applying the solutions proposed in RC_4.

Step 4. The goal in step ④ is to enforce the permissions defined in step ②. These permissions drive the automatic generation of tree-based model editors, represented by arrows in Step ④ in Figure 5.2. The generated editors enforce the defined access control permissions, ensuring that users interact with view model information strictly according to the permissions associated with their assigned roles. For each view, the amount of generated model editors that can be used to interact with the corresponding view models is equal to the number of roles with access to the view. For instance, the number of generated editors for `Sub_M_`A_X is 2 since both *Role 2* and *Role 3* are given access to the view. A user assigned to both roles can interact with the view model using any of the model editors. When a user is assigned an existing role, she automatically gains access to both the view model and the editor associated with that specific role.

Overall Insights. Figure 5.2 illustrates our proposed approach for developing blended modeling environments that feature multiple views and enforce modeled information security. Our solution starts with two Ecore-based metamodels and generates dedicated mechanisms for synchronizing notations. In addition, the solution facilitates the creation of multiple view models. This allows for diverse perspectives on the overall model while ensuring smooth integration between the view and the overall models. The overall model is used as a mediator in the synchronization process, significantly reducing the complexity typically associated with adding new views in peer-to-peer synchronization setups. View models can employ distinct notations, separate from the base model, thus naturally supporting blended modeling. Employing view models serves as a security layer by controlling the exposure of model information to users. For more stringent and fine-grained security, access control mechanisms are implemented, where permissions for each user role are precisely defined and enforced using tree-based model editors.

To exemplify the use of the final modeling environment, consider a scenario in which a user assigned *Role 1* makes modifications to `Sub_MM_`A_1. These modifications are initially propagated to `Overall_MM_A`. If these changes impact `Sub_MM_`A_X, they are also propagated accordingly. Furthermore, the changes extend to `Overall_MM_B` and subsequently to any view built on top of it that is affected by the change, such as `Sub_MM_`B_Y. Consequently, changes initiated by Role 1 users in `Sub_MM_`A_1 are systematically propagated to `Sub_MM_`B_Y through a sequence of transformations. These changes are then accessible and can be modified by users in *Role N-1* or *Role N*.

If the underlying modeling languages evolve, for instance `Overall_MM_A`, several actions become necessary: i) redefining the mappings between `Overall_MM_A` and `Overall_MM_B`, ii) reassessing the defined view metamodels – `Sub_MM_`A_1 or `Sub_MM_`A_X – should they be impacted by changes in `Overall_MM_A`, and iii) reviewing the defined permissions for each view if modifications are needed. Nevertheless, these steps are deemed relatively straightforward if compared to the considerable effort required to achieve the same outcome without our approach.

5.3 Reflections and prospects

In considering the outcomes of this study, it is essential to contextualize the findings, discuss specific decisions, and outline potential prospects for future research.

Tooling ecosystem dependency. The proposed solution is applied to Ecore-based metamodels in the Eclipse Modeling Framework (EMF), capitalizing on its established ecosystem and broad adoption. However, we recognize that other modeling languages such as the Unified Modeling Language (UML) [1] and Systems Modeling Language (SysML) [2] are also widely used in industry.

[1] https://www.uml.org
[2] https://sysml.org

One viable strategy involves utilizing their Ecore-based representation, which is readily available for some languages and would need to be developed for others. This approach requires translating models from various languages into Ecore-based formats, thereby preparing them for integration with our proposed solution. However, we recognize the potential challenges of this method and the community's reluctance to switch tools, often due to significant investments in existing workflows and toolchains. This acknowledgment highlights potential areas for future research to adapt and implement the conceptual framework described in this thesis to other contexts, taking into account the specific challenges associated with different modeling languages and ecosystems.

Definition of mapping models. For the generation of synchronization mechanisms for blended modeling, we focused on manually defined mappings to ensure reliable synchronization between different languages with utmost accuracy. For multi-view modeling, these are automatically derived. While our primary focus was on achieving deterministic mappings, we acknowledge that there can exist challenges involved in the manual definition of these mappings as well, particularly for large languages with substantial differences. One significant challenge arises from the fact that our mapping modeling language is a new language for users, who need not only to learn its syntax and semantics but also to master its usage effectively to ensure that mappings generate the desired outcomes. This learning curve can be steep, especially for users unfamiliar with the concepts underlying the mapping processes. In response to these complexities, our plan includes the development of comprehensive documentation and a detailed set of tutorials. These resources aim to facilitate a deeper understanding of the mapping language, providing step-by-step guidance on its practical use and helping users overcome the initial barriers to effective implementation. To further aid in the elicitation of mappings between languages, another approach involves the use of (semi-)automated matching mechanisms. For instance, syntactic approaches might include exact string matching, substring matching, or using regular expressions. Structural approaches could examine hierarchical structures and associations between elements. Semantic approaches could interpret the meanings of concepts used in different modeling languages to propose mappings that align with the underlying intent. In addition, Large Language

Models (LLMs) have the potential to automate and enhance the mapping generation process by analyzing all these aspects. However, recognizing that LLMs may not always deliver full accuracy, we suggest incorporating a user interface that allows easy user intervention to refine mappings. This interface would enable users to manually tune the mappings, combining automated processes with user expertise to ensure accuracy while enhancing user experience.

Bidirectionality. The synchronization mechanisms in blended and multi-view modeling require transformations that can be applied in both directions so that changes across models can be propagated seamlessly. For doing so there are two possible approaches: i) write two separate unidirectional transformations and ensure by hand that they are consistent, or ii) use a bidirectional approach. While we recognize the inherent consistency of bidirectional model transformations, in this research we have opted for the unidirectional approach. More specifically, we achieve synchronization in the two directions through the creation of two mapping models, one for each direction, followed by the generation of two unidirectional model transformations. This decision was driven by two primary considerations. First, by establishing a separate mapping model for each direction, we enable our varied user base to address the specifics of each direction separately. This provides them with more control over the mapping correspondences and enables them to tackle complex cases such as non-bijection, which could prove challenging and complex when defining a single correspondence specification. Secondly, by establishing two separate mapping models, one for each direction, we provide increased flexibility and the ability to tailor a particular direction independently of the other, if the need arises. At the same time, it increases reusability in other contexts where only one direction might be required.

Execution of model transformations. While the current framework efficiently handles the generation of model transformations, it employs a batch execution approach. More specifically, in blended modeling environments, alterations in any notation-specific model, or multi-view contexts, modifications in the view or base models, necessitate manual intervention to execute model transformations. Our initial focus was solely on the automated provision of the model transformations, rather than on their live execution upon changes to the models involved. However, to improve usability in practical, collaborative environments, future developments should aim towards live execution approaches. Nevertheless, it is important to proceed with caution when implementing such enhancements. While they can significantly streamline collaboration, they also introduce challenges related to inconsistency tolerance.

Access control. While access control encompasses both authentication and authorization, our research focuses solely on authorization. We focus on empowering authorized entities, such as administrators, to define access permissions. Additionally, we provide automated mechanisms to generate model editors that seamlessly enforce these permissions. Our envisioned workflow operates on the premise that users' roles determine the view models they can access and the permissions they receive upon logging in. Users are then confined to interacting with the model through editors tailored to their designated roles. However, the absence of authentication mechanisms poses a limitation to realizing this vision comprehensively. For future work, it is imperative to integrate an authentication mechanism into the existing framework. This enhancement should not only authenticate user identities effectively but also ensure that users are strictly limited to accessing and interacting solely with model editors specifically generated for their roles, as opposed to any generic editor provided out-of-the-box in EMF.

Evaluation. Our proposed solution could benefit from further evaluation on larger-scale models. This would provide important data on scalability and robustness, showing that the solution can manage complex and diverse models as well as maintain performance across varying computational loads. Moreover, a systematic understanding of user experiences for both technical and non-technical stakeholders would provide objective insights into how the solution meets the practical requirements of its users. This aspect is particularly critical for refining user interfaces and functionalities, making the system more intuitive and efficient for all user groups. By integrating these evaluations, our solution can be fine-tuned to better serve its intended purpose and user-base. This kind of evaluation would also highlight potential areas for future enhancements, ensuring that the solution continues to evolve and adapt to new challenges and user needs.

5.4 Beyond the initial scope

As previously discussed, our solution generates the synchronization mechanisms for blended and multi-view modeling, which are defined at the level of the abstract syntax utilizing model transformations. This design choice not only serves the initial purposes but also extends the solution's applicability to a broader range of modeling scenarios beyond the initial focus being blended and multi-view modeling. By defining synchronization at the metamodel level, this approach is suitable for various other scenarios where similar conditions are met. To recap, these conditions include: (i) the metamodels are Ecore-based; (ii) regardless of whether the metamodels completely overlap, partially overlap, or do not overlap at all, it is crucial that viable mappings can be established between the involved metamodels; (iii) alternatively, one metamodel should be an unaltered subset of the other. Below, we present a list of other scenarios where our solution has been or is currently being applied.

Language evolution and model migration. Model migration is a necessary consequence of language evolution. When a modeling language evolves, existing models need to be migrated to ensure compatibility with the new version of the language. Without effective migration strategies, the benefits of language evolution would not be realized as older models become obsolete. Our proposed solution can automatically generate model transformations based on defined mappings, thereby supporting the migration of models to newer language versions. This approach can be instrumental in legacy system modernization efforts. By generating model transformations that facilitate the migration of legacy models to modern platforms, this approach preserves previous investments while harnessing the benefits of new technologies. As a matter of fact, we have demonstrated the applicability of our solution in such scenario by enabling the migration of UML-RT models defined in Rtist on the Eclipse platform to Art models used in RTist in Code within VSCode [57].

Interoperability. Interoperability is defined as "*the degree to which two or more systems, products, or components can exchange information and use the information that has been exchanged*"[3]. When considering interoperability in modeling languages, we are dealing with the exchange of structured information that conforms to specific syntactic and semantic rules defined by their respective metamodels. For instance, a model developed in UML for software design contains different types of information compared to a model created in BPMN for business processes. Effective interoperability ensures that the information from a UML model can be understood and utilized within a BPMN model and vice versa. Interoperability can be achieved through the translation of models between different languages, by employing model transformations. Our proposed solution, on the other hand, can generate the required transformations. In fact, our approach is currently being employed by Ferko et al. [58] in their work on achieving interoperability for digital twins. Digital twins integrate various modeling languages to depict distinct functional units, making interoperability essential for their effective operation. By adopting our systematic approach for generating model transformations, Ferko et al. [58] aim to eliminate the cumbersome and resource-intensive task of manually creating transformations for each distinct language. This strategic implementation not only simplifies the current processes but also enhances scalability and adaptability to future changes in modeling languages.

While the scenarios mentioned are specific instances where our solution has been or is currently applied, the solution's applicability extends beyond these cases to any situation requiring synchronization transformations and meeting the previously outlined conditions.

[3] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Chapter 6

Conclusions

This doctoral thesis aimed to enhance the adoption of collaborative modeling features, which are highly demanded in industrial practices, by tackling the challenges associated with their development. Specifically, the research focused on devising language-agnostic solutions for facilitating the development of collaborative and blended modeling environments featuring multiple views and enforcing information security.

Our research subgoals were successfully achieved through a structured sequence of five contributions, each building upon the other to progressively advance our understanding and eventually address our overall research goal. Initially, we established a comprehensive understanding of the state-of-the-art and -practice in blended modeling solutions (addressed in RC1). This set the stage for subsequent advancements in developing blended modeling capabilities tailored for industrial applications (addressed in RC2 and RC3). Further, we extended our focus to devise language-agnostic automated solutions that streamline traditionally labor-intensive processes (addressed in RC4 and RC5). Specifically, RC4 and RC5 automate i) the provision of synchronization mechanisms between graphical and textual notations in blended modeling, ii) the provision of synchronization mechanisms in multi-view modeling, and iii) the consistent definition and enforcement of access permissions. Such processes typically require extensive manual labor and expertise, a challenge that is further exacerbated if the underlying languages undergo changes. Leveraging automation for these processes has brought forth several advantages. In the following, we highlight the ones we consider to be the most significant and impactful.

- *Streamlining development processes:* automation reduces the need for manual intervention in the provision of synchronization mechanisms for blended and multi-view modeling, and in the implementation of access control. This not only reduces workload on developers but also significantly lowers the likelihood of human error.

- *Adapting to language evolution:* collaborative features are intrinsically linked to modeling languages that are subject to evolution over time. Our framework addresses this challenge head-on, requiring only minimal user input to adapt and evolve based on changes in modeling languages, thus ensuring the continued applicability of these features.

- *Facilitating rapid prototyping:* automation allows developers to create and modify prototypes of modeling environments featuring these capabilities quickly, without the substantial resource investment typically required. Teams can evaluate them, assess user interactions, and refine them with minimal delay and overhead. This speeds up the development cycle and enhances the overall agility of the development team.

- *Democratizing development:* automation democratizes development by lowering the technical barriers that traditionally limit participation. By automating complex tasks, a broader range of stakeholders – including those with less technical expertise – can contribute to and influence the development of modeling environments featuring these capabilities. This inclusivity not only makes development more accessible but also enriches the process with diverse perspectives.

Overall, this thesis proposes a comprehensive framework that facilitates the development of blended modeling environments featuring multiple views and enforcing information security. This framework can reduce the barriers to the adoption of collaborative capabilities and enhance collaborative modeling practices.

Bibliography

[1] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. Collaborative Model-Driven Software Engineering: reflections on the past and visions of the future. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 1–5. ACM, 2017.

[2] Davide Di Ruscio, Mirco Franzago, Ivano Malavolta, and Henry Muccini. Envisioning the future of collaborative model-driven software engineering. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 219–221. IEEE, 2017.

[3] Dimitrios S Kolovos, Louis M Rose, Nicholas Matragkas, Richard F Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, and Massimo Tisi. A research roadmap towards achieving scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, pages 1–10, 2013.

[4] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. Industrial adoption of model-driven engineering: Are the tools really the problem? In *16th International Conference on Model-Driven Engineering Languages and Systems, MODELS Proceedings 2013*, pages 1–17. Springer, 2013.

[5] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended Modelling - What, Why and How. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019*, pages 425–430. IEEE, 2019.

[6] Lorenzo Addazi and Federico Ciccozzi. Blended graphical and textual modeling for UML profiles: A proof-of-concept implementation and experiment. *Journal of Systems and Software*, 175:110912, 2021.

[7] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. Multi-view approaches for software and system modelling: a systematic literature review. *Software and Systems Modeling*, 18(6):3207–3233, 2019.

[8] William Tolone, Gail-Joon Ahn, Tanusree Pai, and Seng-Phil Hong. Access control in collaborative systems. *ACM Computing Surveys (CSUR)*, 37(1):29–41, 2005.

[9] Istvan David, Kousar Aslam, Ivano Malavolta, and Patricia Lago. Collaborative model-driven software engineering—a systematic survey of practices and needs in industry. *Journal of Systems and Software*, 199:111626, 2023.

[10] Douglas C Schmidt. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.

[11] Arianna Fedeli, Nils Beutling, Emanuele Laurenzi, and Andrea Polini. Comparison of general-purpose and domain-specific modeling languages in the IoT domain: A case study from the OMiLAB community. In *BIR 2023 Workshops and Doctoral Consortium, 22nd International Conference on Perspectives in Business Informatics Research*, 2023.

[12] Steffen Zschaler, Dimitrios S Kolovos, Nikolaos Drivalos, Richard F Paige, and Awais Rashid. Domain-specific metamodelling languages for software language engineering. In *International Conference on Software Language Engineering*, pages 334–353. Springer, 2009.

[13] Anneke Kleppe. A language description is more than a metamodel. In *Fourth International Workshop on Software Language Engineering*, pages 1–4, 2007.

[14] Anneke G Kleppe, Jos B Warmer, Jos Warmer, and Wim Bast. *MDA explained: The model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[15] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic notes in theoretical computer science*, 152:125–142, 2006.

[16] Nafiseh Kahani and James R Cordy. Comparison and evaluation of model transformation tools. *Technical Report 2015-627, Queen's University*, pages 1–42, 2015.

[17] Frédéric Jouault and Ivan Kurtev. Transforming models with ATL. In *8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2005*, pages 128–138. Springer, 2005.

[18] Sandra Greiner, Thomas Buchmann, and Bernhard Westfechtel. Bidirectional transformations with QVT-R: a case study in round-trip engineering UML class models and Java source code. In *4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 15–27. IEEE, 2016.

[19] Kevin Lano, Shekoufeh Kolahdouz-Rahimi, and Sobhan Yassipour-Tehrani. Declarative specification of bidirectional transformations using design patterns. *IEEE Access*, 7:5222–5249, 2018.

[20] Matthias Bank, Sebastian Kaske, Thomas Buchmann, and Bernhard Westfechtel. Incremental bidirectional transformations: Evaluating declarative and imperative approaches using the AST2Dag benchmark. In *15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 249–260, 2020.

[21] Richard F Paige. Engineering bidirectional transformations. *Bidirectional Transformations: International Summer School, Oxford, UK, July 25-29, 2016, Tutorial Lectures*, pages 151–187, 2018.

[22] Pavle Guduric, Arno Puder, and Rainer Todtenhofer. A comparison between relational and operational QVT mappings. In *6th International Conference on Information Technology: New Generations (ITNG)*, pages 266–271. IEEE, 2009.

[23] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software & Systems Modeling*, 9:7–20, 2010.

[24] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. JTL: a bidirectional and change propagating transformation language. In *3rd International Conference on Software Language Engineering (SLE 2010)*, pages 183–202. Springer, 2011.

[25] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080. Cambridge, MA, 1988.

[26] Andy Schürr. Specification of graph translators with triple graph grammars. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163. Springer, 1994.

[27] Stephan Hildebrandt, Leen Lambers, Holger Giese, Jan Rieke, Joel Greenyer, Wilhelm Schäfer, Marius Lauder, Anthony Anjorin, and Andy Schürr. A survey of triple graph grammar tools. *Electronic Communications of the EASST*, 57, 2013.

[28] Leila Samimi-Dehkordi, Bahman Zamani, and Shekoufeh Kolahdouz-Rahimi. Bidirectional model transformation approaches a comparative study. In *6th International Conference on Computer and Knowledge Engineering (ICCKE 2016)*, pages 314–320. IEEE, 2016.

[29] J Nathan Foster, Michael B Greenwald, Jonathan T Moore, Benjamin C Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3):17–es, 2007.

[30] Martin Hofmann, Benjamin Pierce, and Daniel Wagner. Symmetric lenses. *ACM SIGPLAN Notices*, 46(1):371–384, 2011.

[31] Massimo Tisi, Frédéric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bézivin. On the use of higher-order model transformations. In *5th European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA 2009)*, pages 18–33. Springer, 2009.

[32] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2):171–188, 2005.

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

[34] Didonet Del Fabro Marcos, Bézivin Jean, Jouault Frédéric, Breton Erwan, and Gueltas Guillaume. AMW: A generic model weaver. *1 eres Journées sur l'Ingénierie Dirigée par les Modeles (IDM 2005)*, 200:105–114, 2005.

[35] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving models with the Eclipse AMW plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe*, volume 2006, pages 37–44. Citeseer, 2006.

[36] Marcos Didonet Del Fabro and Patrick Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Software and Systems Modeling*, 8(3):305–324, 2009.

[37] Salome Maro, Jan-Philipp Steghöfer, Anthony Anjorin, Matthias Tichy, and Lars Gelin. 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*, pages 1–12, 2015.

[38] Anis Charfi, Artur Schmidt, and Axel Spriestersbach. A hybrid graphical and textual notation and editor for UML actions. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 237–252. Springer, 2009.

[39] Ionut Predoaia, Dimitris Kolovos, Matthias Lenk, and Antonio García-Domínguez. Streamlining the development of hybrid graphical-textual model editors for domain-specific languages. *Journal of Object Technology*, 22(2), 2023.

[40] Istvan David, Malvina Latifaj, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer,

and Regina Hebig. Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. *Software and Systems Modeling*, 22(1):415–447, 2023.

[41] Manfred Broy. Multi-view modeling of software systems. In *Formal Methods at the Crossroads. From Panacea to Foundational Support. Lecture Notes in Computer Science*, volume 2757, pages 207–225. Springer, 2003.

[42] Magnus Persson, Martin Törngren, Ahsan Qamar, Jonas Westman, Matthias Biehl, Stavros Tripakis, Hans Vangheluwe, and Joachim Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *13th International Conference on Embedded Software (EMSOFT 2013)*, pages 1–10. IEEE, 2013.

[43] ISO/IEC/IEEE. Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, 2011.

[44] Domenik Bork and Dimitris Karagiannis. Model-driven development of multi-view modelling tools the muviemot approach. In *9th International Conference on Software Paradigm Trends (ICSOFT-PT 2014)*, pages 1–11. IEEE, 2014.

[45] Hugo Bruneliere, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. EMF views: A view mechanism for integrating heterogeneous models. In *Conceptual Modeling. ER 2015. Lecture Notes in Computer Science*, volume 9381, pages 317–325. Springer, 2015.

[46] Antonio Cicchetti, Federico Ciccozzi, and Thomas Leveque. Supporting incremental synchronization in hybrid multi-view modelling. In *Workshops and Symposia on Models in Software Engineering at International Conference on Model Driven Engineering Languages and Systems (MODELS 2011)*, pages 89–103. Springer, 2012.

[47] Henk CA van Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security*, volume 1. Springer Science & Business Media, 2011.

[48] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.

[49] Vincent C Hu, Rick Kuhn, and Dylan Yaga. Verification and test methods for access control policies/models. *NIST Special Publication*, 800:192, 2017.

[50] Vincent C Hu, David Ferraiolo, and D Richard Kuhn. *Assessment of access control systems*. US Department of Commerce, National Institute of Standards and Technology, 2006.

[51] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: Towards a unified standard. In *ACM workshop on Role-Based Access Control*, volume 10, 2000.

[52] Bran Selic. Real-time object-oriented modeling. *IFAC Proceedings Volumes*, 29(5):1–6, 1996.

[53] Ernesto Posse and Juergen Dingel. An executable formal semantics for UML-RT. *Software and Systems Modeling*, 15(1):179–217, 2016.

[54] Jan Reineke and Stavros Tripakis. Basic problems in multi-view modeling. In *20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014)*, pages 217–232. Springer, 2014.

[55] Gordana Dodig Crnkovic. Constructive research and info-computational knowledge generation. In *Model-Based Reasoning in Science and Technology*, pages 359–380. Springer, 2010.

[56] Kari Lukka. The constructive research approach. *Case study research in logistics. Publications of the Turku School of Economics and Business Administration, Series B*, 1(2003):83–101, 2003.

[57] Malvina Latifaj, Federico Ciccozzi, Antonio Cicchetti, and Mattias Mohlin. Cross-platform migration of software architectural UML-RT models. In *17th European Conference on Software Architecture ECSA*, 2023.

[58] Enxhi Ferko, Luca Berardinelli, Alessio Bucaioni, Morris Behnam, and Manuel Wimmer. Towards interoperable digital twins: Integrating SysML into AAS with higher-order transformations. In *3rd International Workshop on Digital Twin Architecture (TwinArch) and Digital Twin Engineering (DTE)*, 2024.

II

Included Papers

Chapter 7

Paper A: Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study

Istvan David, Malvina Latifaj, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer, Regina Hebig.

Published in the Software and Systems Modeling journal (SoSyM 2022).

Abstract

Blended modeling aims to improve the user experience of modeling activities by prioritizing the seamless interaction with models through multiple notations over the consistency of the models. Inconsistency tolerance, thus, becomes an important aspect in such settings. To understand the potential of current commercial and open-source modeling tools to support blended modeling, we have designed and carried out a systematic study. We identify challenges and opportunities in the tooling aspect of blended modeling. Specifically, we investigate the user-facing and implementation-related characteristics of existing modeling tools that already support multiple types of notations and map their support for other blended aspects, such as inconsistency tolerance, and elevated user experience. For the sake of completeness, we have conducted a multivocal study, encompassing an academic review, and grey literature review. We have reviewed nearly 5,000 academic papers and nearly 1,500 entries of grey literature. We have identified 133 candidate tools, and eventually selected 26 of them to represent the current spectrum of modeling tools.

7.1 Introduction

Model-driven engineering (MDE) advocates modeling the engineered system at high levels of abstraction before it gets realized. The resulting models serve crucial roles in ensuring the appropriateness (e.g., correctness, safety, optimality) of the system. To keep the cognitive flow of modeling effective and efficient, stakeholders shall be equipped with proper formalisms, notations, and supporting computer-aided mechanisms. This is especially important in the design of modern systems, as their complexity has been increasing exponentially over the past years [1]. Modeling does not remove complexity from the engineering process, but rather, it replaces the accidental complexity of complex systems with essential complexity that is easier to manage [2]. Nonetheless, as a consequence of the increasing complexity of modern systems, modeling itself is becoming more complex. In this paper, we focus on a specific manifestation of this added complexity stemming from the need for an orchestrated ensemble of modeling notations, aiming to enable seamless interaction with models through any of the notations. Such a need has been reported in multiple academic [3] and industrial domains, e.g., automotive [4], avionics [5], cyber-physical systems [6], and product lines [7]. In such an approach, user experience may also be (temporarily) prioritized over the correctness of the described system, in an effort to enable a smooth process of expressing the stakeholder's cognitive models in terms of the modeling language. This approach is referred to as *blended modeling* [8].

7.1.1 What is blended modeling?

Blended modeling was first introduced by Ciccozzi et al. [9] 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.*

That is, blended modeling is characterized by the following three features.

Multiple notations. This is not to be confused with multiple *languages*. In our terminology, a language is composed of (i) a metamodel (abstract syntax), and (ii) a set of notations (concrete syntax). Blended modeling does not impose different metamodels.

Seamless interaction. Different notations have to be carefully integrated and orchestrated to allow for using the most appropriate notation for specific modeling tasks. This requires intuitive navigation between notations, proper change propagation between them, and in many cases, traceability.

Flexible consistency management. This aspect entails both vertical inconsistencies [10] (e.g., inconsistencies between the instance model and its metamodel); and horizontal inconsistencies (e.g., inconsistencies between two notations used to manipulate instances of the same metamodel).

7.1.2 What is *not* blended modeling?

Multi-view modeling is not blended modeling. As shown in Fig. 7.1, Multi-View Modeling (MVM) [11] and blended modeling share the trait of *multi-notation*. The main differences are, that (i) MVM further assumes *multiple languages*, while (ii) blended modeling assumes relaxed consistency rules instead. These differences stem from the different aims of the two approaches. MVM is concerned with constructing the appropriate views for stakeholders with varying backgrounds. Blended modeling focuses on the elevated UX with respect to an ensemble of notations, assuming a single underlying model. Prior work has reported challenges in relaxed consistency in multi-language settings such as MVM [12]. Blended modeling enables relaxed consistency by restricting the number of languages to one. For example, the SCADE[1] tool suite provides the user with different languages for different purposes within the same model development environment. These languages facilitate multi-view modeling of the overall system and necessitate different abstract syntaxes. Therefore, working with SCADE cannot be considered blended modeling.

Multi-paradigm modeling is not blended modeling. In addition to assuming multiple languages, Multi-Paradigm Modeling (MPM) [13] further assumes potentially different semantics behind the languages, giving rise to *multi-formalism* (Fig. 7.1). This added complexity positions MPM even further from blended

[1] `https://www.ansys.com/products/embedded-software/ansys-scade-suite`

modeling, and vastly exacerbates consistency management, as reported in prior work [14]. For example, Matlab/Simulink is a typical combination of formalisms for system design, in which the overall system is graphically designed in Simulink[2], which follows causal block diagrams (CBD) semantics; and the low-level functions in the system are textually described in Matlab[3], which relies on matrix semantics for complex computations. While some level of navigation is provided between the two formalisms within the Matlab modeling and development environment, relaxed consistency is completely missing. Therefore, working with Matlab/Simulink cannot be considered blended modeling.

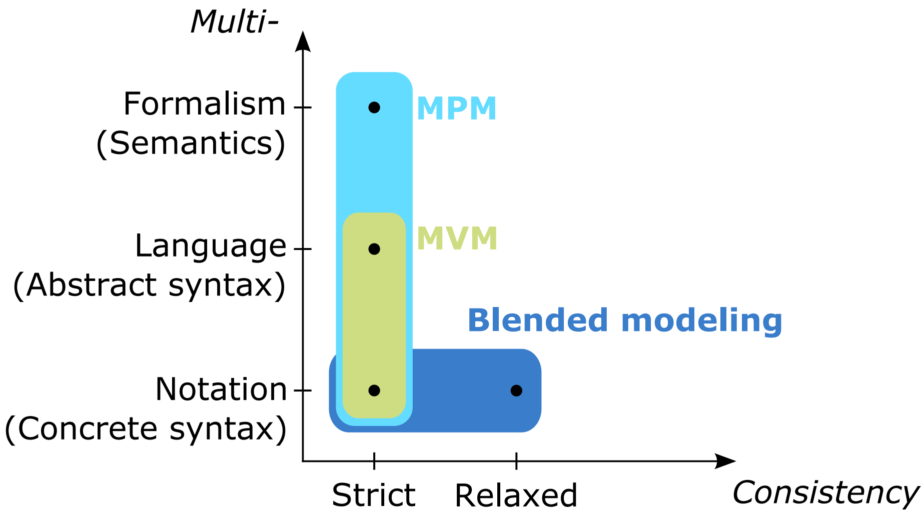


Figure 7.1. Blended modeling in the context of MVM and MPM.

7.1.3 Motivation and aim

Blended modeling is an emerging new concept, thus, a map of current commercial and open-source tools is needed to properly position it in the research-and-development landscape.

[2] `https://se.mathworks.com/products/simulink.html`
[3] `https://www.mathworks.com/products/matlab.html`

In this article, we report the design, execution, and results of our mapping study on tools that are prime candidates to support blended modeling. Our study shows that these are typically tools with multiple notations for a single underlying abstract syntax, but they lack proper inconsistency tolerance mechanisms or fail to leverage such features for an improved user experience. The aim of our study was to identify, classify, and analyze (i) the user-oriented, and (ii) the realization-oriented characteristics of these tools. To infer this information while ensuring external validity, we surveyed both the academic (peer-reviewed) literature and the grey literature [15], consisting of websites, blogs, and user manuals of engineering tools, following the guidelines for multi-vocal reviews in software engineering [16]. To be able to treat both types of literature uniformly, we made *tools* the primary units of our study, instead of papers. This is motivated by the inherent limitations of grey literature in terms of providing high-fidelity research data. Websites and end-user documentation do not aim to provide such information. We formulated a surveying protocol based on well-established guidelines and we have meticulously followed this protocol in the execution of our study. Eventually, we screened 4,975 academic papers and included 77 of them. Additionally, 1,494 grey literature entries were processed. Out of the academic papers, 68 distinct tools were extracted and complemented by 68 tools extracted from the grey literature. After removing duplicates, the set of 133 tools was reviewed according to the tool selection criteria (see Section 7.3) and we eventually identified 26 tools to be analyzed in detail. Although this list of tools is not exhaustive, we are reasonably confident about its representativeness of the domain of interest.

The results of this study provide a clear overview of the state of the art and practice of the domain of modeling tools closest to blended modeling. The tool characteristics reported in this paper can be particularly useful for tool providers in identifying the limitations of their tools in supporting blended modeling. Researchers of the three main dimensions of blended modeling (multi-notation, seamless integration of languages, inconsistency tolerance) could use this work to better contextualize their research, and position their work better in terms of applicability.

7.1.4 Structure

The rest of this paper is structured as follows. First, in Section 7.2, we give an overview of the background concepts of blended modeling and review the related work. In Section 7.3, we define the methodological framework for carrying out this study. In Section 7.4, we elaborate on the findings of this study, in particular on the results pertaining to the research questions of this study: the user-oriented and realization-oriented characteristics of the tools of interest. In Section 7.5, we provide orthogonal insights on the aggregated data. We discuss the results in Section 11.7, and the threats to validity in Section 7.7. Finally, we summarize this paper by drawing the conclusions in Section 11.8.

7.2 Background

In this section, we provide the foundational background concepts to contextualize our study. More specifically, we describe the core ingredients of blended modeling: multiple notations (Section 7.2.1), seamless interaction (Section 7.2.2), and flexibility in managing inconsistencies (Section 7.2.3). Additionally, we discuss the secondary literature related to our study (Section 7.2.4).

7.2.1 Multiple notations

Interacting with the (abstract) model through multiple notations (concrete syntaxes) is one of the three distinguishing features of blended modeling. A vast body of knowledge on the topic has been produced, especially in relation to multi-view modeling, and multi-paradigm modeling.

Multi-view modeling

Multi-view modeling (MVM) tackles the complexity of modeling heterogeneous systems by decomposing the models into multiple views, that are concerned with specific aspects of the system [11]. The ISO/IEC/IEEE 42010:2011 standard [17] defines a view as a set of concerns of specific stakeholders and viewpoints as the specification of conventions utilized to construct a view. The five mutually non-exclusive enabling mechanisms of multi-view modeling are

(i) *synthetic*, where views are specified by means of different domain specific modeling languages and synthesized together; (ii) *separate*, a stricter version of synthetic, where synthesis does not take place; (iii) *projective*, where a single metamodel allows for the definition of multiple virtual views; (iv) *orthographic*, where views are orthographic projections of a single underlying model; or (v) *hybrid*, where views represent only a portion of the common metamodel [18]. MVM has been shown to be an effective approach in several complex domains, such as cyber-physical systems [1], and cloud-based software-intensive systems [19]. The principles of MVM are similar to those of blended modeling. However, its goal is different. While MVM is oriented towards the identification of multiple views and the management of consistency between them, blended modeling focuses on enabling an elevated user experience while working with multiple notations at the same time.

Multi-paradigm modeling

Multi-paradigm modeling (MPM) advocates modeling every aspect of the system explicitly, at the most appropriate level of abstraction, and using the most appropriate formalism [13, 20]. As such, MPM facilitates the modeling of complex systems that could not be described through a single formalism and at a common level of abstraction due to the heterogeneity of the different components. It combines three research areas: (i) meta-modeling used for the specification of formalisms, (ii) multi-formalism used for the coupling of models specified in different formalisms and their transformations, and (iii) model abstraction used for the relationships among models described in different formalisms [21]. The principles of MPM are similar to those of blended modeling, as both approaches promote employing a variety of notations to model the problem at hand. However, MPM achieves this by employing a variety of separate formalisms, i.e., multiple notations with possibly different semantics. Blended modeling assumes a single abstract syntax, and therefore, single semantics. This simplification allows for greater flexibility in terms of temporarily inconsistent designs.

7.2.2 Seamless interaction

Usability in terms of the ability to seamlessly interact with models through multiple different notations is one of the three distinguishing features of blended modeling. In this section, we review how state-of-the-art approaches typically support seamless interaction. We focus on UML tools here since they have received significant attention from research and tool providers of the software engineering domain in the past. We also mention examples for other modeling languages where appropriate.

Text-based modeling with graphical visualizations

Umple [22] is a modeling tool that supports the creation of UML models using both textual and graphical notations, where the synchronization between the two notations is automated and on the fly. However, the graphical editor does not offer full editing capabilities, and the existing editing capabilities are only available on class diagrams but not on state machines, composite structures, or feature diagrams. FXDiagram[4] is a JavaFX-based framework that can be integrated into Eclipse as well as intelliJ IDEA. It supports the creation of graph diagrams (nodes and edges) and it is typically used for graphical visualization of textual DSLs but does not provide editing functions. MetaUML[5] is a GNU GPL library for typesetting UML diagrams, using a textual notation. This notation is used for rendering read-only graphical UML diagrams. PlantUML[6] is very similar but supports also non-UML diagrams. ZenUML[7] supports sequence diagrams and flowcharts, again defined using a textual notation that is translated into read-only graphical views. The generation of the sequence diagrams is automatic, as the conversion happens on the browser. Excalibur [23] is a tool that relies on Xtext for textual specification and Sirius for graphical views of the textual specification. The model elements are defined using Messir textual DSL and the generated graphical visualization is read-only. Chart Mage[8] is a

[4] https://jankoehnlein.github.io/FXDiagram
[5] https://github.com/ogheorghies/MetaUML
[6] https://plantuml.com
[7] https://www.zenuml.com
[8] http://chartmage.com/index.html

web-based tool that supports automatic and on-the-fly generation of sequence diagrams and flowcharts using a textual notation. DotUML[9] is a javascript application that supports the generation of a subset of UML diagrams (i.e., use-case, sequence, class, state, and deployment) from a textual notation. For all of the aforementioned tools, concrete syntaxes are predefined and not customizable, and the graphical notation is read-only, generated using the textual notation.

Mixed textual and graphical modeling

Addazi and Ciccozzi [8] present a proof-of-concept implementation for UML and UML profiles modeling using blended textual and graphical notations. The stack of technologies used includes Eclipse Modeling Framework (EMF)[10], Xtext[11], and Papyrus [24]. Their solution includes a single underlying abstract syntax, two notations (i.e., graphical and textual), and one single persistent resource that is the UML resource. This architecture enables synchronization by means of serialization/deserialization operations across Xtext and UML models. In addition, the authors conduct an experiment to demonstrate that their solution on blended modeling increases user performance compared to single notation modeling. Maro et al. [25] introduce a solution that integrates graphical and textual editors for a specific UML profile-based DSL. Being that the graphical editor is already provided, this work focuses on obtaining the textual editor and switching between views (i.e., graphical and textual). To obtain the textual editor, the UML profile-based DSL is first transformed into an Ecore model using an ATL transformation, and then this Ecore model is consumed by the Xtext plugin to generate the textual editor. Switching between views is achieved by employing ATL transformations. Scheidgen [26] provides embedded textual editors for graphical editors as an add-on feature. For each selected model element that needs to be edited, the embedded textual editor creates an initial representation that can be changed by the user and using parsing operations, new edited model elements are created. However, the synchronization is on-demand as the changes in the underlying model are

[9] `https://dotuml.com`
[10] `https://www.eclipse.org/modeling/emf`
[11] `https://www.eclipse.org/Xtext`

not carried out until they are committed by the user and the textual editor is closed. Lazăr [27] makes use of the Eclipse modeling environment to integrate the existing UML tree-based editor with the textual editor for Alf language[12] and to create fUML[13] models. However, the synchronization is on-demand as the changes are carried out upon the occurrence of a save action by the user. Charfi et al. [28] define a hybrid language that integrates textual and graphical notations in one concrete syntax. The contribution consists of a visual notation for the most used UML actions and an editor that supports the proposed notation. The hypothesis that the hybrid notation can perform better than the textual notation is backed by an experiment that takes into consideration the learnability of the hybrid notation, the prevented errors, and the circumstances in which the hybrid notation is a better fit than the textual notation. However, this approach is restricted to UML actions only. Van Rest et al. [29] implement an approach for the robust synchronization of graphical editors generated with the Graphical Modeling Framework (GMF)[14] and textual editors generated with Spoofax[15]. This approach allows error recovery during synchronization and preserves the textual and graphical layout in case of errors. However, layout preservation is not supported at all times, as during cut-paste operations, the elements and their associated layouts are deleted and then recreated, therefore losing the original layout.

Projectional editing

Projectional editing is an approach where the abstract syntax tree (AST) is modified directly upon every editing action and bypasses the stages of the parser-based approach, where the parser must first check the correctness of the syntactic aspects, and then construct the AST based on the changes in the notation [30]. This course of action allows the definition of multiple notations (e.g., tables, diagrams, formulas) that cannot be supported by parser-based approaches, and supports multiple views of the same program, simultaneously. Moreover, a considerable amount of the ambiguities caused during the parsing process are

[12] https://www.omg.org/spec/ALF
[13] https://www.omg.org/spec/FUML
[14] https://www.eclipse.org/modeling/gmp
[15] http://strategoxt.org/Spoofax

tackled. Projectional editing is a realization of the intentional programming paradigm [31], and as such, it encourages the combination of a variety of different notations. Some of the state-of-the-art language workbenches that adopted this principle for providing domain-specific tool engineers with efficient tools [32] are JetBrains MPS[16] and MelanEE[17]. However, even though they provide a greater amount of notations, their support for textual notations is limited compared to parser-based approaches, as it is only a projection that resembles text. In particular, no possibly inconsistent intermediate states are allowed, which consequently restricts the user accustomed to classical text editors and their corresponding free editing features.

7.2.3 Inconsistency management

Approaches, such as multi-view modeling (MVM) and multi-paradigm modeling (MPM) advocate modeling the engineered system using the most appropriate notations, formalisms, and abstractions. This allows multiple users to be involved in the modeling of the system, and thus, introduces parallelism, which is beneficial for the overall efficiency of the engineering endeavor. Parallelism, however, gives rise to inconsistencies between the design artifacts, compromising the ultimate correctness of the system. Inconsistency has been shown to be an effective heuristic for managing the ultimate correctness of the system [14]. Techniques, such as blended modeling, make use of this assertion by focusing on the early detection of inconsistencies [33] and establishing the proper tolerance mechanisms. The notion of consistency models and their various alternatives have been well-researched already in early distributed systems. Lamport [34] is the first to describe how multi-processor systems should be constructed to ensure proper execution of programs. His notion of *sequential consistency* allows a relaxation of the locking model by assuming a total order of modifications that distributed nodes are guaranteed to observe. Adve and Gharachorloo [35] describe various relaxations of the sequential consistency model, based on architectural choices on the hardware and software level. *Eventual consistency* has been suggested by Vogel et al. [36] to enable a weaker notion of consis-

[16] https://www.jetbrains.com/mps
[17] http://www.melanee.org

tency between distributed participants, by embracing that real consistency can never be achieved. In such settings, distributed participants are characterized by the BASE properties: basic availability, soft state, and eventual consistency. Lately, *strong eventual consistency* (SEC) has been suggested [37] to combine the liveliness guarantees of eventual consistency with the safety guarantees of strong consistency. Conflict-free replicated data types [38] are the prime examples of their applications. Inconsistencies are a well-researched area in software engineering [39], too. Consistency between models can be categorized into two orthogonal dimensions [40]: horizontal and vertical consistency; and syntactic and semantic consistency. Horizontal consistency is concerned with models on the same level of abstraction, whereas vertical consistency is defined between models on different levels of abstraction (typically in model-metamodel contexts) [41]. The majority of inconsistency management techniques rely on syntactic concepts, e.g. synchronization by bi-directional model transformations [42], triple-graph grammars [43], and by version control systems and related mechanisms [44, 45]. However, semantic techniques have been shown to be beneficial in heterogeneous engineering settings [10]. View consistency has been researched in the context of MVM, e.g., in the Vitruvius approach [46], which provides languages for consistency preservation, and defines a model-driven development process for enacting consistency rules. Finkelstein et al. [47] suggest that inconsistencies are organic elements of any engineering process, and instead of simply removing them from the system, one should apply proper inconsistency management techniques [48]. Such inconsistency management techniques typically entail the activities of detecting, resolving, preventing, and tolerating inconsistencies [49]. Blended modeling heavily relies on the tolerance of inconsistencies. Balzer et al. [50] suggest augmenting inconsistency instances with a state. Inconsistency rules are first deconstructed into appearance and disappearance rules spanning a temporal interval; then, tolerance rules are put in place to trigger repair actions based on temporal constructs. Easterbrook et al. [51] propose a similar technique for temporal inconsistency tolerance in the context of MVM. Inconsistency tolerance is achieved via pairs of pre- and post-conditions relying on a user-defined consistency metric. David et al. [52] introduce various patterns of inconsistency tolerance for implementing such systems.

7.2.4 Related secondary literature

This paper reports on the first systematic study on blended modeling. There are, however, secondary studies close to our work that are similar in topic, but differ in terms of motivation and objectives, and are generally limited to a narrower scope. Torres et al. [53] conduct a systematic literature review with the aim to identify a list of available tools to support model management and provide a categorization of these tools into (i) tools that can provide consistency checking on models of different domains, (ii) tools that can provide consistency checking on models of the same domain, and (iii) tools that do not provide any consistency checking. Furthermore, the authors identify the inconsistency types, strategies to keep the consistency between models of different domains, and the challenges to manage models of different domains. The information retrieved from the primary studies is also complemented with additional data sources (e.g., the official website of the tool). Our study focuses on a broader scope, especially multi-notation and seamless interaction. Torres et al. observe that 35% of their analyzed tools do not provide any consistency checking features, whereas in our study we observe that 64% of the analyzed tools do not support models inconsistencies. Moreover, Torres et al. identify different strategies that have been used to keep models consistent, e.g., by using standard file formats for the models, explicitly modeling dependencies among model elements, mapping model elements to a shared ontology, etc. Our study complements such results by highlighting which inconsistency management strategies involve a manual effort (like keeping a dependency matrix always up to date), a semi-automated procedure (e.g., by specifying a priori consistency constraints and checking them during development), or a completely automated one (e.g., via the automated application of inconsistency resolution procedures). Iung et al. [54] conduct a systematic mapping study with the aim to identify tools, language workbenches, or frameworks for DSL development. The authors identify 59 tools and they use the feature model proposed by Erdweg et al. [55] for their comparison. The study focuses on the technologies/tools used for DSL development, their license types, the application domains, and the features of the DSL creation process that these tools support. 48 tools support only one notation (graphical or textual), seven tools support two notations (graphical and textual), two tools support three notations, and two tools support four notations. Our study focuses on a broader scope, by extending the set of features on which the comparison is based with features such as synchronization mechanisms, collaborative features, or conformance relaxation. We also contextualize our work on a broader timeline, while the authors focus on the period between 2012–2019. In line with the results of our study, Iung et al. observed that the notations that were more frequently used in combination are *textual* and *graphical*, with the tabular one complementing them. In [54] two language workbenches are identified as particularly relevant for blended modeling: (i) GEMOC Studio, which provides real-time bidirectional synchronization in their generated editors, and (ii) the Whole Platform, which allows language engineers to choose among four different types of notation (i.e., textual, graphical, tabular, and symbolic), and to visualize the different translations among them at the model level. Franzago et al. [56] and David et al. [57] map the state-of-the-practice of collaborative model-based software engineering. The authors identify and classify collaborative MDSE approaches based on the different categories such as characteristics of the collaborative model editing environments, model versioning mechanisms, model repositories, support for communication and decision making, and more. Additionally, the authors identify limitations and challenges with respect to the state of the art in collaborative MDSE approaches. Regarding model management, they provide a taxonomy for the management support of collaborative MDSE approaches, collaboration support, and communication support. This study covers some of the aspects that we cover in our systematic mapping study (e.g., conflict detection). However, while this study is mostly focused on the characteristics of the collaborative approaches, we aim toward a classification of tools based on a broader set of features such as synchronization mechanisms and their generation, or conformance relaxation. The results of Franzago et al. and David et al. for collaborative modeling that are confirmed in this study are about: (i) the types of notations, with graphical as the most supported one, followed by textual, (ii) the prevalence of custom/other modeling platforms with respect to Eclipse EMF, (iii) the growth of web-based approaches, (iv) the growth of preventive conflict management, and (v) the prevalence of mechanisms for (semi-)automatically resolving conflicts. We anticipate that 15 out of the 26 tools analyzed in this study support collaborative modeling, with the majority of tools providing offline collaboration (i.e., a la Git), rather than real-time collaboration (i.e., a la

Google Docs); this result is different for academia where, according to Franzago et al. and David et al., researchers focus primarily on real-time collaboration. Another difference with respect to the state of the art in collaborative modeling is that blended modeling tools are primarily parser-based, whereas collaborative modeling approaches tend to be equally distributed between parser-based and projectional approaches. Interestingly, while researchers are recently investigating more on eventual consistency for collaborative modeling [57], in our study we observe that blended modeling tools provide limited support for consistency tolerance that would allow deviations between different notations describing the same model. Granada et al. [58] map model-based language workbenches that can be used to generate editors for visual DSLs and point out their features and functionalities. The authors identify eight language workbenches for the generation of editors for visual DSL. The features taken into consideration for their analysis are the following: scope, framework, the distinction between abstract and concrete syntax, abstract syntax, concrete syntax, editing capabilities, use of models, automation, usability, and methodological basis. The conclusions point out that the most complete commercial language workbenches are MetaEdit+[18] and ObeoDesigner[19], while the most complete open-source ones are Eugenia[20], GMF[21], Graphiti[22], and Sirius[23]. Our study differs in scope, as we focus on tools that provide multiple notations, not only on tools that can be used to develop editors for a single visual DSL. Indeed, none of the tools identified by Granada et al. support the definition of more than one (visual) concrete syntax for the same abstract syntax; this means that language engineers willing to develop blended modeling environments should either use a dedicated language workbench for blended modeling or suitably combine the languages produced by two or more of the language workbenches mentioned above. Do Nascimento et al. [59] perform a large-scale systematic mapping study on DSLs and their related tools. The tools are categorized into (i) tools for using DSLs, (ii) tools for creating DSLs, and (iii) language workbenches. Our study differs

[18] https://www.metacase.com/products.html
[19] https://www.obeodesigner.com/en
[20] https://www.eclipse.org/epsilon/doc/eugenia
[21] https://www.eclipse.org/modeling/gmp
[22] https://www.eclipse.org/graphiti
[23] https://www.eclipse.org/sirius

from this work, as we focus on DSLs tool comparison, while the authors provide a brief categorization of DSL tools, and do not go into the details of conducting a comparison of the technical features. It is interesting to note that Do Nascimento et al. observed that tool support for a single DSL is well-studied in the literature, but at that time (2012) there was little knowledge about how to support multiple DSLs and notations in a single modeling environment. They claim that supporting multiple DSLs and multiple notations is fundamental when describing large-scale industrial systems and that methods and tool support are needed for the success of multi-DSL development. Based on the results of our study and the ones on multi-notation modeling (see Section 7.2.1, we can confirm that in the last years, the MDE scientific community actively worked and contributed to filling this research gap.

There are additional studies related to our research that are not systematic in nature, but their takeaways are still relevant. Negm et al. [60] compare 14 language workbenches based on (i) structure (grammar-driven or model-driven), (ii) editor (parser-based or projectional), (iii) language notations (textual, tabular, symbols, or graphical), (iv) semantics (translational or interpretive), and (v) composability language aspects. However, this study is limited to language workbenches and does not cover aspects such as synchronization mechanisms and their generation, or collaborative features. Some of the results obtained by Negm et al. are relevant for blended modeling as well. Firstly, out of nine analyzed parser-based language workbenches, only one (i.e., Ensõ) supports both textual and graphical concrete syntaxes; this capability is achieved by having a bidirectional mapping between tokens in the textual representation of the model and elements in the object graph. Moreover, all four considered projection-based language workbenches support multiple concrete syntaxes, with the *Whole* platform and MPS supporting four different syntaxes: textual, graphical, tabular, and symbolic. The main advantage of projection-based workbenches is that they can rely on a shared common representation of all modeling elements (e.g., the AST in MPS), whereas parser-based workbenches have a dedicated parser for each concrete syntax. One of the claimed advantages of parser-based language workbenches (especially the textual ones) is the flexibility with respect to the models' conformance; the textual representation of parser-based models can still be opened and inspected, whereas projectional editors work directly on the abstract representation of the model. Similarly, according to Negm et al., textual parser-based workbenches avoid tool lock-in since the modeler is not limited to using any specific editor and can be easily integrated with other tools. Erdweg et al. [55] conduct a comparison study of 10 language workbenches participating in the 2013 edition of the Language Workbench Challenge (LWC). The comparison of the language workbenches is based on a feature model that includes: notation, semantics, editor support, validation, testing, and composability, where some of them support multiple notations (fully or partially). The conclusions state that no language workbench realizes all features. However, this study is limited to the language workbenches presented in LWC'13. For what concerns blended modeling, the results obtained by Erdweg et al. are in line with the ones reported by Negm et al. [60], where projectional language workbenches are better supporting the combination of different concrete syntaxes, with Enzõ and MPS again as the ones supporting all types of concrete syntaxes. Erdweg et al. also highlight the need for integrating "different notational styles", which is at the core of blended modeling. Merkle [61] conduct a comparison study of textual language workbenches categorizing them into pure text-based and projectional-based with a textual projection. The language workbenches compared in this study are Xtext[24], TEF[25], EMFText[26], and MPS[27]. The language workbenches are compared based on workflow, abstract/concrete syntax, and editor. However, this study is limited to textual language workbenches, while our focus is on tools that provide multiple notations. In the study by Merkle, the only language workbench supporting a combination of concrete syntaxes is TEF (Textual Editing Framework[28]), an Eclipse-based language workbench focusing primarily on textual editors, but with the possibility of embedding them into other editors supporting other concrete syntaxes [26]. Internally, TEF follows the *background parsing* strategy for the textual concrete syntax, where textual models are always represented and edited as plain text, and their parsing

[24] `https://www.eclipse.org/Xtext`

[25] `https://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html`

[26] `https://github.com/DevBoost/EMFText`

[27] `https://www.jetbrains.com/mps`

[28] `http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html`

is demanded by a background process. TEF also provides some basic form of blending, where modelers can bring up a textual editor from either a graphical or a tree-based editor (e.g., by opening a small overlay window); however, TEF-based modeling tools cannot be considered as blended since model updates the embedded textual editor is not seamlessly integrated into its host editor, and model updates are propagated on-demand to the host editor only when the modeler closes the textual editor.

7.3 Study design

The goal of this study is to characterize the state of the art and the state of the practice of modeling tools in relation to blended modeling. More specifically, we formulate such high-level goal by using the Goal-Question-Metric perspectives [62], shown in Table 7.1.

Purpose	Identify, classify, and analyze
Issue	the user-oriented and implementation-oriented characteristics of
Object	existing modeling tools
Context	in relation to the principles of blended modeling,
Viewpoint	from a researcher's and practitioner's point of view.

Table 7.1. Goal of this study.

7.3.1 Process

This research was carried out by following the process shown in Figure 7.2. Our process can be divided into three main phases, all well-established in systematic secondary studies [63, 64, 65, 66]: planning, conducting and documenting. In the following, we present the three phases of the process.

Planning

This phase aims at defining the plan for carrying out all the activities of this study. More specifically, we first identified *related secondary studies*, i.e.,

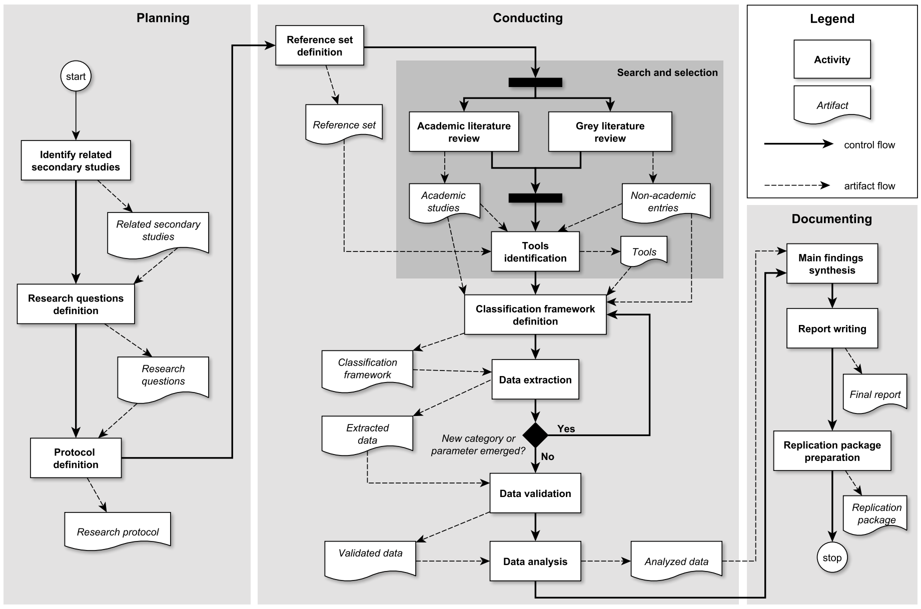


Figure 7.2. Overview of the whole review process

surveys and literature reviews with a scope similar to the current review's scope (Section 7.2.4). Subsequently, we formulated the *research questions* (Section 7.3.2), and compiled the *research protocol*.

The research protocol is a document reporting the methodological details of this study. Specifically, the research protocol contains a detailed description of all the steps we followed in the subsequent *Conducting* and *Documenting* phases. To mitigate potential threats to validity and any bias, the research protocol was defined *prior to* conducting the study, and it was reviewed by two experts. The experts were asked to provide feedback on the protocol, particularly on possible unidentified threats to validity, problems in the overall construction of the review, and the appropriateness of the proposed research protocol and final reports for the aim of this study. Both experts are well-established professors of Computer Science, with substantial experience in empirical research.

Conducting

In this phase, the mapping study is carried out according to the research protocol. More specifically, we carry out the following activities.

Reference set definition. The goal of this activity is to identify the modeling tools that could be part of the final set of modeling tools. This set will serve as a guideline for the subsequent steps of the study design, especially formulating the inclusion and exclusion criteria. The inclusion and exclusion criteria will be tested against this set, and thus, the reference set is subject to change until the criteria are not final. We identify the initial reference set based on (i) the modeling tools mentioned in related secondary studies (Section 7.2.4); (ii) the authors' experiences with tools partially supporting blended modeling (e.g., [9, 8]); (iii) searches in generic web search engines; and (iv) knowledge garnered from existing networks of experts, e.g., by accessing forums and mailing lists (e.g., the Eclipse EMF community forum[29]). The results of the subsequent *Search and selection* activity are eventually compared to the reference set for validation purposes. The eventual reference set is composed of the following tools: MagicDraw [67], Eclipse Papyrus [24], MetaEdit+[30], Umple [22], and the Open Source AADL Tool Environment (OSATE) [68].

Search and selection. **(Section 7.3.3)** The goal of this activity is to identify as many (possibly blended) modeling tools as possible. Two parallel activities are carried out: the *Academic literature review*, and the *Grey literature review*. In both search activities, we perform a combination of automated search, manual search, and backward-forward snowballing [69]. These activities yield two types of artifacts: (i) *Academic studies* (e.g., articles published in scientific journals, and proceedings of scientific conferences) and (ii) *Non-academic entries* (e.g., blog posts, technical reports). Because the subject of this study are the *tools* these artifacts describe, both types of artifacts are screened for a specific *Tool* in the *Tools identification* activity. Here, we manually analyze all academic studies

[29] https://www.eclipse.org/forums/index.php?t=thread&frm_id=108
[30] https://www.metacase.com/products.html

and non-academic entries and identify every modeling tool mentioned in their contents. Moreover, in this activity, we keep track of pointers and links referring to the relevant documentation about each tool (e.g., its official documentation, its wiki-based knowledge base, etc.).

Classification framework definition. **(Section 7.3.4)** The goal of this activity is to define the set of categories and their possible values to classify the identified modeling tools.

Data extraction. **(Section 7.3.5)** The goal of this activity is to collect relevant information about each modeling tool. In this activity, multiple researchers collaboratively (i) read the full text of the relevant documentation of each modeling tool, and (ii) populate the data extraction form with the collected data. Upon the emergence of a new category or new possible value in the domain of previously defined categories, the classification framework can be dynamically adapted. In such cases, the previously extracted data entries are updated in accordance with the new framework.

Data validation. **(Section 11.5)** To ensure the validity of the extracted data, the tool vendors and knowledgeable experts are contacted to review the data extracted in the previous step.

Data analysis. **(Section 7.3.7)** The goal of this activity is to analyze the extracted data in accordance with the research questions. The activity involves both quantitative and qualitative analyses.

Documenting

The main activities performed in this phase are: (i) a thorough elaboration on the data analyzed in the previous phase with the aim of discovering the main findings of the study; (ii) reporting the possible threats to validity, especially the ones identified during the definition of the review protocol; and (iii) producing the final report. The final report is evaluated by external reviewers and forms the basis of this article. The complete *replication package* is available online[31] to allow independent researchers to replicate and verify our study, and to reuse our

[31] `https://zenodo.org/record/6402743`

data for other purposes. The replication package includes the research protocol, the list of all academic and non-academic entries considered in the search and selection phase, the complete list of all identified tools, raw data, the scripts for data analysis, and the details on technical requirements.

7.3.2 Research questions

The research questions of this study are reported below.

RQ1. *What are the **user-oriented characteristics** of modeling tools most suitable for supporting blended modeling?*

Modeling tools are designed and developed to be adopted by specific users, application domains, and usage scenarios.

By answering this research question, we aim to identify the external characteristics of modeling tools, pertaining to their adoption and usage [9]. Typical examples include: supported (types of) notations, human-computer interfaces, application domains, and addressed user groups.

Practitioners can benefit from the answer to this research question by understanding how specific state-of-the-art tools address their problems, what are their limitations in terms of blended modeling, and how they can be improved.

RQ2. *What are the **realization-oriented characteristics** of modeling tools most suitable for supporting blended modeling?*

With the advent of model-based approaches and domain-specific modeling, in particular, several modeling tools are being developed to support certain levels of blending, formalisms, and semantics. Moreover, until the recent spread of mainstream language workbenches (e.g., Xtext[32], Sirius[33], MPS[34], etc.), the development of such modeling tools had been relatively ad-hoc.

[32] `https://www.eclipse.org/Xtext`
[33] `https://www.eclipse.org/sirius`
[34] `https://www.jetbrains.com/mps`

By answering this research question, we aim to identify the internal characteristics of modeling tools, and that, in terms of (i) their features and (ii) the techniques employed to implement those features. Typical examples include: implementation platforms, consistency mechanisms, change propagation, traceability, and the linguistic level of model-to-model correspondence are investigated.

Researchers can benefit from the answer to this research question by understanding the state of the practice on the techniques of blended modeling tools, including the gaps to fill.

The identified research questions drive the whole study, with a special influence on (i) the search and selection, (ii) data extraction (including the definition of the classification framework), and (iii) data and main findings synthesis.

7.3.3 Search and selection

The goal of the search and selection phase is to retrieve a representative set of modeling tools supporting multiple modeling notations, as demanded by the principles of blended modeling. First, we perform a *systematic review* of both the academic (i.e., scientific articles published at peer-reviewed academic venues) and grey literature (i.e., websites, online blogs, etc.), and discuss the results in Section 7.3.3. The output of these two activities (i.e., academic studies and non-academic entries) is then further analyzed in order to identify the modeling tools either considered, mentioned, or discussed in them (Section 7.3.3).

Systematic reviews

We follow the same overall process when reviewing both the academic and grey literature. In this phase, it is fundamental to achieve a good trade-off between the coverage of existing results on the considered topic and having a manageable number of studies to be analyzed [63, 16]. To achieve the above-mentioned trade-off, our search and selection process has been designed as a multi-stage process; this gives us full control over the number and characteristics of the entries being either selected or excluded during the various stages. In the following, we present each step of our systematic review process. In the remainder of this report, we refer to both academic studies and non-academic entries as *primary studies*, unless specifically noted otherwise. The systematic review is divided into three subsequent and complementary steps of (i) automatic search, (ii) application of selection criteria, and (iii) snowballing.

Automatic search. In this step, we automatically inspect all the results returned from a query execution on (i) Google Scholar for academic studies and (ii) the Google Search engine for grey literature. The automatic searches for both academic and non-academic literature are executed in November 2020.

For the *academic literature*, we use *Google Scholar*. We use Google Scholar as the data source for the following main reasons: (i) it is one of the largest and most complete databases and indexing systems for scientific literature; (ii) as reported in [69], the adoption of this data source has proved to be a sound choice to identify the initial set of literature studies for the snowballing process (Section 7.3.3), producing a reasonable number of false positives, but no false negatives (thus, no information is lost); (iii) the query results can be automatically processed via already existing tools. Below we report the search string used in this study. In order to cover as many potentially relevant studies as possible, we defined the search string so that it includes academic studies on blended modeling. The search string can be divided into three main components: the first component captures the model-driven paradigm, the second one captures the focus on multiple entities (e.g., multiple notations) and blending, and the third one is used for ensuring that our results focus on software aspects. To keep the results of this initial search as focused as possible, the query has been applied to the title of the targeted studies.

```
("modeling" OR "modelling" OR "model based"
  OR "model driven")
    AND
("multi*" OR "blended")
    AND
("notation*" OR "syntax*" OR "editor"
  OR "tool" OR "software")
```

The search string has been tested by executing pilot searches on Google Scholar. At the time of writing, Google Scholar produced a total of 280 hits when searching with the reported search string.

For the *grey literature*, we target the regular *Google Search Engine*. The search engine is selected in accordance with the recommendations for including grey literature in software engineering multi-vocal reviews [16]. The search string used for the academic literature yields mostly academic results even in a general web search. We have, therefore, adapted our search strategy to find non-academic sources. In particular, we identified a number of relevant hits through manual searches early on. These manual hits could be classified as either *lists* (e.g., Wikipedia's "List of Unified Modeling Language tools") or *tool-specific pages* (e.g., tool vendor pages or blog posts about how specific tools are used). We experimented with several search strings to ensure that we find all relevant hits. In particular, we tried to combine different modeling languages and diagram types into one large all-encompassing search string to simplify our search and make it easier to extract results. However, on prototyping this approach, we realized that the `OR` clauses that we used did not have the desired effect and we did not find the tools we expected, and in particular, not the lists that we expected. In comparison, a search string such as `(MARTE) AND (tool OR editor OR notation OR modelling)` yields 162 results on Google, whereas our combined search string that included MARTE[35] and many other languages only yielded 150 results.

Therefore, we decided to carry out an independent search for popular modeling languages. We ran the different searches independently and merged the results later on. We selected the relevant modeling languages using a mixture of expert knowledge, browsing the web pages of well-known modeling tools from the reference set and beyond (e.g., Eclipse Capella[36] and Enterprise Architect[37]), using lists such as Wikipedia's page on `Modeling Languages`. We narrowed down the resulting list of around 40 potential modeling languages by searching for `(Language Name) AND (tool OR editor OR notation OR modelling)` in Google, and analyzing the first ten non-academic hits (i.e.,

[35] https://www.omg.org/omgmarte
[36] https://www.eclipse.org/capella
[37] https://sparxsystems.com/products/ea

search results that were not academic papers). Since the search term explicitly contains the terms `tool` and `editor`, we expected that the Google search engine would include such a tool within the first ten non-academic hits if it exists, and has any practical relevance. Experiments where we checked later result pages for selected searches confirmed this expectation. We thus only included modeling languages for which Google does report a link to a modeling tool. Otherwise, we disregarded it. To address the large number of hits we would get this way, we limited the search results for each included search string to the first 50 unique results (if less than 50 hits are reported, we collect all of them), which is based on the suggestion from Garousi et al. [16]. The eventual result set included 1,494 hits, typically containing blog posts, user manuals, websites, technical reports, white papers, academic articles, etc.

Application of selection criteria. In this step, the identified potentially relevant entries undergo rigorous filtering based on the application of a set of selection criteria. Following the guidelines for systematic literature review for software engineering [63], we define the set of inclusion and exclusion criteria *a priori*, in order to reduce the likelihood of bias. The potentially relevant entries are rigorously examined by adopting multiple selection rounds in an adaptive reading depth fashion [70]. Specifically, in the first round, the title of the entry is examined. This first step enables us to discard all those papers or web pages that clearly do not fall within the scope of this study. In the second exclusion round, the introduction and conclusion sections are inspected (if present). Finally, the entries are further inspected by considering their full text, in order to ensure that only the ones relevant to answering the research questions are selected. While processing the full text of a paper/web page, we also keep track of all the mentioned modeling tools and consider them in the tools' identification phase (Section 7.3.3).

In the following, we detail the set of inclusion and exclusion criteria that guide the selection of the academic and non-academic entries for our systematic review.[38] A potentially relevant entry is selected if it (i) satisfies *all* inclusion criteria and (ii) does not satisfy *any* of the exclusion criteria. The selection

[38]The identifiers used in this section are consistent with those used in the replication package to enable better traceability.

criteria are divided into three categories, namely: *generic* (i.e., they apply for both academic and non-academic studies), *academic-specific*, and *grey-specific*. The decision of adopting three categories of criteria originates from the different nature of the sources we considered (i.e., Google Scholar and the Google Search Engine). By defining three different sets, it is possible to design selection criteria specifically tailored to the specific characteristics of academic and non-academic entries, and hence, improve the overall quality of the selection process.

Generic inclusion criteria:

GEN-I1) Entries on modeling tools, i.e., where models are used as first-class entities and used as a substantial abstraction from the problem domain (e.g., OSATE [68] for modeling hardware/software systems according to the AADL modeling language).

GEN-I2) Entries discussing at least two different notations (possibly for the same abstract syntax). The notations can be of the same type (e.g., both textual).

Generic exclusion criteria:

GEN-E1) Entries on non-modeling tools. For example, articles on IDEs, programming tools, drawing tools, etc.

GEN-E2) Entries that are not in English.

GEN-E3) Duplicates of already included entries.

GEN-E4) Entries that are not available, and hence not analyzable (e.g., the full text of a scientific article is not accessible or the link to a web page is broken).

Exclusion criteria specific to academic sources:

A-E1) Studies in the form of full proceedings and books since they are too broad for being thoroughly analyzed in this phase of the study.

A-E2) Studies that have not been peer-reviewed, as peer-reviewing is the *de facto* standard of quality assurance for scientific literature.

Exclusion criteria specific to grey literature:

G-E1) Web pages reporting exclusively the basic principles of modeling techniques, without mentioning any modeling tool.

G-E2) Web pages reporting exclusively abstract best practices while applying modeling techniques.

G-E3) Web pages reporting an implementation without a discussion of its benefits and/or drawbacks.

G-E4) Academic literature, since such type of studies is considered by a different process in our protocol.

G-E5) Videos, podcasts, and webinars since they are too time-consuming to be considered for this phase of the study.

Snowballing. In this step, we complement the preliminary set of academic studies by applying the snowballing procedure [69]. To mitigate a potential bias with respect to the construct validity of the study, backward and forward snowballing is used to complement the automatic search of the academic literature [71]. In particular, this process is carried out by considering the scientific publications selected in the initial automatic search, and subsequently selecting relevant studies among those cited by one of the initially selected ones (backward snowballing). Then, we also perform forward snowballing, i.e., selecting relevant studies among those citing one of the initially selected academic studies [69]. In this context, the *Google Scholar*[39] bibliographic database is adopted to retrieve the studies citing the ones selected through the initial search phase. The final decision about the inclusion of the newly considered publications in the study is based on the application of the selection criteria presented in Section 7.3.3.

[39] `https://scholar.google.com`

Tool identification

In the tool identification activity, each primary study is manually analyzed and the mentioned modeling tools are identified. This is achieved by investigating the full text of each primary study, and collecting every modeling tool mentioned in it, independently of whether it is blended or not. Then, the set of identified modeling tools is filtered for duplicates, which are subsequently merged, regardless of whether the tool originates from an academic or a non-academic source. After the merge, we obtained a total of 133 modeling tools. For each tool, we have collected the following information: (i) name, (ii) link/reference to official documentation, (ii) organization(s) implementing, maintaining, and supporting the tool, and (iii) tracing information towards all primary studies mentioning the tool. In order to ensure that the identified tools support us in answering the research questions of this study, we further filter the list of all modeling tools according to a set of selection criteria. Below we report the inclusion and exclusion criteria.

- TI1) The tool allows its users to edit the same model in multiple notations. The user can switch between these notations easily and without an extra processing step (i.e., the tool supports some level of blended modeling). The tool allows a certain degree of temporary inconsistencies. Notations like an overview tree for navigation purposes or any textual representation used for file persistency purposes only are not considered (e.g., XMI).
- TI2) The tool is publicly available (either as an open-source or commercial product).
- TI3) The documentation of the tool is publicly available.
- TE1) The tool is a language workbench. (Our study focuses on modeling tools themselves.)
- TE2) The tool is not available for download as a binary that can be run on current operating systems from an official website or an affiliated platform supporting it (e.g., a GitHub repository).
- TE3) The documentation of the tool is not in English.

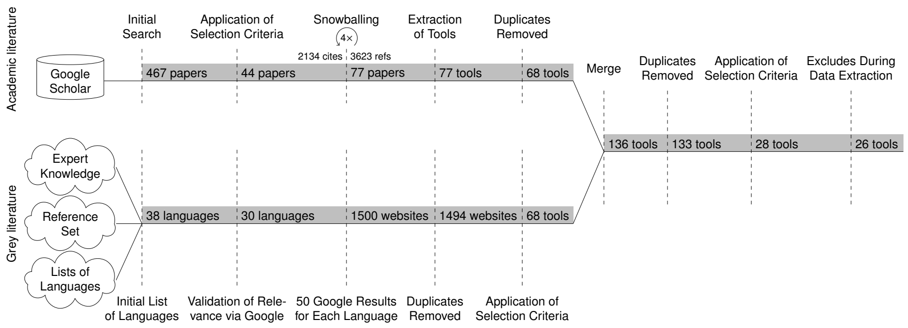


Figure 7.3. Overview of the conducted search and selection steps

A potentially relevant modeling tool is included if it satisfies *all* inclusion criteria (TI1-TI3), and discarded if it satisfies *any* exclusion criterion (TE1-TE2).

To minimize bias, this activity is performed by five researchers and organized as follows. First, two researchers are randomly assigned to each of the potentially relevant tools. Then, the researchers independently apply the tool selection criteria to their assigned tools; each researcher could mark a tool as `included`, `excluded`, `maybe`. For the 12 of 133 tools where at least one researcher indicates an uncertainty (`maybe`), the conflicts are resolved with the intervention of a randomly–assigned third researcher and, when needed, discuss plenary among all researchers involved in this study. After the final set of modeling tools has been established, we check whether each tool in the reference set is also included in this final set of tools. If all tools in the reference set are indeed included in the final set of tools, we continue with the subsequent phases of the protocol (i.e., data extraction). Otherwise, a dedicated meeting is set up, and a refinement of the systematic review process is designed and conducted again. Eventually, the final list of modeling tools contains all tools of the reference set.

Figure 7.3 shows the different steps performed in the search and selection phase. Out of the 467 papers in the initial scientific search, 44 papers were included in the snowballing process. The snowballing was performed four times before no more new papers were included. During this process, a total of 2,134 cited and 3,623 referenced papers were reviewed. In summary, 68 distinct tools were extracted from the included papers. For the grey literature part, 30 relevant languages were identified as described above, for which the different search terms yielded 1,494 distinct websites. After applying the selection criteria, 68 tools were included in the tools set. Merging the academic and grey literature parts resulted in 133 distinct tools, of which 30 tools were selected according to the tool selection criteria. Two tools had to be excluded during the data extraction process due to lack of availability or semantically out of scope (see Section 7.3.4). Eventually, 26 modeling tools were sampled, shown in the Referred Tools section at the end of this paper.

Table 7.2. Categories of the classification framework, and their domain.

Category	Definition	Type/Domain
GENERIC		
META		
Tool ID The name of the tool.	The internally used ID of the tool. Free text	
Analyzed release	The version of the release the analysis was carried out on.	Free text
TOOL		
First release	Date of the first available release.	Date
Latest release	Date of the latest available release.	Date
Motivation	The self-declared motivation of the tool.	Free text
Open-source	Whether the tool's sources are available openly.	{Yes, No}
Web-based	Whether the tool is web-based.	{Yes, No}
Collaboration	The degree and type of support for collaboration.	{No, Asynchronous, Synchronous}
RQ1: USER-ORIENTED CHARACTERISTICS		
NOTATIONS		
Notation types	Types of notations supported by the tool.	{Textual, Graphical, Tabular, Tree-based, Mixed textual-graphical}
Notation instances (number of)	Sum number of instances of notation types.	Numeric
Embedded notations	Whether there are notations that are embedded into each other.	{Yes, No}
Overlap	The degree of overlap between notations.	{None, Partial, Complete}
VISUALIZATION AND NAVIGATION		
Visualize multiple notations	The ability to visualize more than one notations.	{Yes, No}
Synchronous navigation	Whether the tool supports a synchronous navigation of multiple visualized notations.	{Yes, No}
Navigation among notations	The dynamics of navigation between different notations.	{Immediate, Complex}
FLEXIBILITY		
Flexibility - models	Whether the tool supports temporary inconsistency at the level of the instance models.	{Yes, No}
Flexibility - language	Whether the tool supports temporary inconsistency at the level of the language.	{Yes, No}
Flexibility - persistence	Whether the tools can persist inconsistent models.	{Yes, No}
RQ2: REALIZATION-ORIENTED CHARACTERISTICS		
MAPPING AND PLATFORMS		
Mapping	The way concrete and abstract syntax are mapped.	{Parser-based, Projectional}
Platform	The platform the tool is built on.	{Eclipse, Other}
CHANGE PROPAGATION AND TRACEABILITY		
Change propagation	The dynamics of propagating changes across notations.	{Sequential, Concurrent}
Traceability	Whether the tool supports explicit traceability between notations.	{Yes, No}
INCONSISTENCY MANAGEMENT		
Inconsistency visualization	The degree and way the tool visualizes inconsistencies.	{No, Internal, External}
Inconsistency management type	The way the tool manages inconsistencies.	{On-the-fly, On-demand, Preventive}
Inconsistency management automation	The degree of automation of inconsistency management activities.	{Manual, Partial, Automated, Not applicable}

7.3.4 Classification framework definition

Table 7.2 shows the classification framework of this study. The classification framework is composed of three distinct facets; the first facet is about generic characteristics of modeling tools (e.g., release dates, vendor, main motivation for blending notations); the second and third facets directly address research questions RQ1 and RQ2.

We partially reuse the results of previous work [9] related to blended modeling for defining the initial version of the classification framework. Then, as suggested in [64], the customization of the classification framework is performed as follows: (i) firstly we select a random sample of 10 modeling tools, (ii) then two researchers independently extract the data from the 10 modeling tools by using the initial version of the classification framework, (iii) the two researchers then discuss the results of the data extraction with a third researcher, with a special focus on too generic/abstract parameters, parameters which did not fully fit with the characteristics of the tools, parameters with redundant values, and recurrent missing concepts, (iv) the classification framework is customized according to the discussion, and lastly (v) the final version of the classification framework is applied to all remaining modeling tools. It is important to note that when analyzing the remaining 26 tools, the classification framework can still be enriched/updated based on the characteristics of the currently analyzed tool. The details about how we extracted data for each modeling tool are provided in the next section.

7.3.5 Data extraction

The main goal of this activity is to extract relevant data about each modeling tool for answering the research questions. The inputs to this activity are: (i) the set of 28 modeling tools, out of which 26 remained after excluding two additional tools during this phase; and (ii) the textual contents of the academic studies and non-academic entries referring to the tools, and the tools' official documentation (when publicly available). Moreover, when we are not able to collect all relevant data for some specific aspects of a tool (e.g., the internal consistency mechanisms of a proprietary tool) we perform a series of ad-hoc Web searches and contact the support team of the tool for collecting the missing data. For the sake of external verifiability, full tracing information is kept between the extracted data and the considered data sources and it is included in the replication package of the study.

To carry out a rigorous data extraction process, and to ease the control and the subsequent analysis of the extracted data, a predefined data extraction form is designed prior to the data extraction process. The structure of the data extraction form is based on the various categories of the classification framework.

7.3.6 Data validation

To ensure the validity of the extracted data, the tool vendors are contacted and the data and the explanation of the reference framework are made available to them. If a tool does not have a clearly identified vendor, we identify knowledgeable experts who published scientific papers related to the tool. The vendors and experts are asked to identify any invalid data related to their tool. The contact is initiated via email with the vendors and experts having an option to ask and discuss the details with our research team. The majority of interactions happened in email. Some vendors and experts preferred a live discussion during a video call, which we also accommodated. Eventually, we have contacted vendors and experts of 24 tools. The authors of this paper have developed or extensively contributed to the remaining 2 tools, and validated them internally. The validation phase ran for three weeks, between February 28 and March 22, 2022. 69% of tool vendors or experts replied either with minor change suggestions or with the approval of the extracted data. Based on their responses, 3.8% of the data (20 of 520 records) has been updated. The most changes, five, were observed in the model-level flexibility category.

7.3.7 Data analysis

The data analysis activity involves collating and summarizing the data, aiming at understanding, analyzing, and classifying the state of the art of modeling tools [65, § 6.5]. The data synthesis is divided into two main phases: vertical analysis and horizontal analysis. In both cases, we perform a combination of content analysis [72] (mainly for categorizing and coding tools under broad thematic categories) and narrative synthesis [73] (mainly for detailed explanation and interpretation of the findings coming from the content analysis). When performing *vertical analysis*, we analyze the extracted data to find trends and collect information about *each category* of the classification framework. When performing *horizontal analysis*, we analyze the extracted data to explore possible relations *across different categories* of the classification framework.

Vertical analysis

Depending on the parameters of the classification framework, in this research, we apply both quantitative and qualitative synthesis methods, separately. When considering quantitative data, depending on the specific data to be analyzed, we apply descriptive statistics for a better understanding of the data. When considering qualitative data, we apply the *line of argument* synthesis [64], that is: firstly we analyze each tool individually to document it and tabulate its main features with respect to each specific parameter of the classification framework, then we analyze the set of tools as a whole, to reason on potential patterns and trends. When both quantitative and qualitative analyses are completed, we integrate their results to explain quantitative results by using qualitative results [65, § 6.5]. The results are discussed in Section 7.4.

Horizontal analysis

Following the best practice of previous secondary studies [57, 56, 74, 75], we explore significant phenomena across pairs of categories as well. We use contingency tables annotated with the Chi-square statistic at $\alpha = 0.05$, for identifying statistically significant cases. Following the directions of Haviland [76], we report the p-values of the conventional Chi-square test without Yates's correction for continuity. The results are discussed in Section 7.5.

7.4 Results

In this section, we elaborate on the findings of this study. First, we discuss the general findings in Section 7.4.1. Then, we elaborate on the two research questions of our study: the user-oriented characteristics (RQ1) and the realization-oriented characteristics (RQ2) of the sampled tools, in Section 7.4.2 and 7.4.3,

Table 7.3. Relationships between blended aspects (BA) and the research questions (RQ) of this study.

	RQ1: User-oriented characteristics (Section 7.4.2)	**RQ2: Realization-oriented characteristics (Section 7.4.3)**
BA1: Multi-notation	Notations (Section 7.4.2)	Mapping and platforms (Section 7.4.3)
BA2: Seamless interaction	Visualization and navigation (Section 7.4.2)	Change propagation, traceability (Section 7.4.3)
BA3: Flexibility	Model/language/persistence flexibility (Section 7.4.2)	Inconsistency management and tolerance (Section 7.4.3)

respectively. In both cases, we contextualize our findings in terms of the three core blended modeling aspects: multi-notation, seamless interaction, and flexibility, as shown in Table 7.3.

7.4.1 Overview

In this section, we review some of the general findings regarding the analyzed blended modeling tools. The list of the included tools is shown in Table 7.4.

Project age and timeline. The tools and their respective projects spread over 25 years, with SOM/ADOxx [91] being the oldest tool (first release in 1996) in our sample. On average, the age of the tool projects is 10.6 years ($\sigma = 5.9$). The means of the first and last releases are 2008.8 ($\sigma = 5.9$) and 2019.4 ($\sigma = 1.8$), respectively. These numbers suggest a sample of mature enough tools with sufficient recency in terms of the latest release. Fig. 7.4 provides a visual overview of the age and timeline of tool projects.

Motivations. The self-declared motivations of the tools vary greatly. We have recorded the mission statements of the tools and clustered them. General-purpose modeling tools are typical in our sample, usually offering multi-notation support

Table 7.4. The list of included tools.

Tool			Releases			Info	
ID	Name	Vendor/Maintainer	First	Latest	Analyzed	Open-source	Self-declared motivation
[77]	ADOIT: Community Edition	BOC Products & Services AG	2003	2020	ADOIT:CE based on ADOIT 12.0	No	Enterprise architecture management
[78]	Archi	Beauvoir, P and Sarrodie, JB	2010	2021	4.8.1	Yes	Enterprise architecture
[79]	ARIS	Software AG	2009	2017	2.4d - 7.1.0.1161389	No	Business process modeling
[80]	ASCET Developer	ETAS	2002	2020	7.6.0 Build ID 209	No	"Easily combine texts and graphics suiting your programming needs."
[81]	AToMPM	Université de Montréal	2013	2020	0.8.5	Yes	Multi-paradigm modeling on the web
[T06]	BlendedProfile	Mälardalen University	2018	2020	0.3	Yes	Blended modelling for UML profiles
[82]	Boston	Viev	2015	2020	5.0	No	Fact-based modeling via Object-Role Modeling (ORM)
[83]	Cardanit	ESTECO SpA	2013	2020	Online @07.04.2021.	No	Modeling BPMN with diagrams and tabular views
[84]	Certware	NASA	2013	2016	2.0	Yes	Safety case modeling
[85]	DBDiagrams	Holistics Software	2018	2021	Online @07.04.2021.	No	Visualize textual DB schema definition
[24]	Eclipse Papyrus	The Eclipse Foundation	2008	2020	5.0.0	Yes	Generic-purpose MBSE tool, based on UML and providing support for DSLs via UML Profiles
[86]	Eclipse Process Framework Project	The Eclipse Foundation	2006	2018	1.5.2	Yes	Software process modeling
[67]	MagicDraw	CATIA No Magic	1998	2021	MagicDraw 2021x LTR Enterprise	No	Modelling tool that facilitates analysis and design of Object Oriented (OO) systems and databases. It provides code engineering mechanism (with full round-trip support for Java, C++, C#, CL (MSIL) and CORBA IDL programming languages), as well as database schema modeling, DDL generation and reverse engineering facilities.
[87]	mbdeddr	itemis AG	2012	2018	2018.2.0 based on MPS 2018.2.6	Yes	"Boosting productivity and quality by using extensible DSLs, flexible notations and integrated verification tools."
[T15]	MEMO4ADO	OMiLAB	2015	2018	1.10	No	Multi-Perspective Enterprise Modeling
[88]	Modelio	Modelisoft	2011	2020	4.1.0 (202001232131)	Yes	Generic modeling tool for UML, BPMN, ArchiMate, SysML, etc
[68]	OSATE	Carnegie Mellon University	2004	2021	2.9.1	Yes	AADL is a language, with different representations. A textual representation provides a comprehensive view of all details of a system, and graphical if one want to hide some details, and allow for a quick navigation in multiple dimensions.
[89]	QuickDataBaseDiagrams	Dovetail Technologies Ltd	2002	2021	Online @07.04.2021.	No	Modeling DB schemas by text and diagram
[90]	SequenceDiagramOrg	-	2014	2021	Online - 9.1.1	No	Improve the efficiency when creating and working with sequence diagrams by combining text notation scripting and drawing by clicking and dragging in the same model.
[91]	SOM/ADOxx	OMiLAB	1996	2014	SOM 3.0 on ADOxx 1.5	No	Semantic Object Model. Comprehensive approach for object-oriented and semantic modeling of business systems.
[92]	Swimlanes	-	2014	2021	Online @07.04.2021.	No	Visualize sequence diagrams
[93]	TopBraid Composer Maestro Edition	TopQuadrant, Inc	2006	2021	7.1.0	No	"TopBraid Composer™ Maestro Edition (TBC-ME) is a comprehensive Knowledge Graph modeling and SPARQL query tool. In use by thousands of commercial customers, Composer offers robust and comprehensive support for building and testing configurations of rich knowledge graphs."
[94]	UMLet	TU Wien	2002	2018	14.3 Standalone	Yes	Allow textual+visual modeling of UML diagrams
[95]	UMLetino	TU Wien	2013	2018	14.3	Yes	Allow textual+visual modeling of UML diagrams
[22]	Umple	University of Ottawa	2008	2020	Online - 1.30.1.5099 .60569f335	Yes	Support the convenient modeling across different formalisms. No particular domain targeted, thus, it's a pretty abstract tool.
[96]	USE	Universität Bremen	2007	2020	6.0.0	Yes	System modeling via a subset of UML + OCL

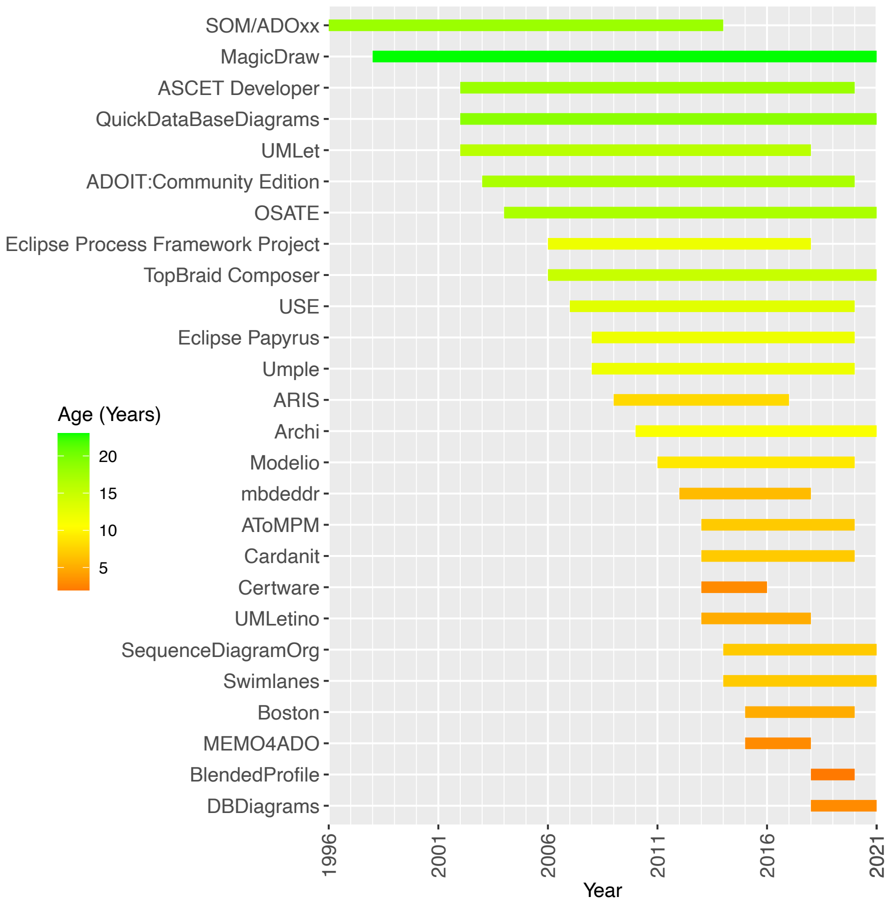


Figure 7.4. Overview of the age of the tool projects, spanned by their respective first and last releases.

for UML-based modeling, e.g. Modelio [88], USE [96], and Papyrus [24]. Some of these tools are very specific about their intentions to combine or augment the traditional graphical notation of UML with textual elements, such as UMLet [94] and ETAS ASCET Developer [80]. Among the tools with specific modeling purposes are the ones aiming at process modeling (e.g., SOM/ADOxx [91], ARIS [79]), database modeling (e.g., DBDiagram [85], QuickDBD [89]), and enterprise architecture (e.g., Archi [78], ADOIT [77]).

Web-based implementation. We have found that the majority of the sampled tools, 17 of 26 (65%), are exclusively desktop-based applications, as shown in Table 7.5.

Table 7.5. The web-based nature of tools.

Web-based	#Tools	Tools
No	17 (65%)	[78], [79], [80], [T06], [82], [84], [24], [86], [67], [87], [T15], [88], [68], [91], [93], [94], [96]
Yes	9 (35%)	[77], [81], [83], [85], [89], [90], [92], [95], [22]

Open-source. Half of the sampled tools are released as open-source software (Table 7.6), allowing access to the source code of the tool.

Table 7.6. The open-source nature of tools.

Open-source	#Tools	Tools
No	13 (50%)	[77], [79], [80], [82], [83], [85], [67], [T15], [89], [90], [91], [92], [93]
Yes	13 (50%)	[78], [81], [T06], [84], [24], [86], [87], [88], [68], [94], [95], [22], [96]

Collaboration. Collaborative modeling is the joint creation of a shared representation of a system through means of modeling [56, 57]. Collaboration enables an orchestrated interplay among stakeholders of different domains, and thus, very often, collaboration raises the need for multiple different notations. In real-time collaborative settings, the groupwork of stakeholders happens synchronously. Off-line collaborative settings do not assume synchronicity, but rather stakeholders who work on shared models at different times. As shown in Table 7.7, the majority of tools, 15 of 26 (58%), provides some means of collaboration. Specifically, off-line techniques are typical, accounting for 9 of 15 collaborative tools (60%) or 9 of 26 tools overall (35%), respectively. Finally, 11 of 26 sampled tools (42%) do not support any means of collaboration.

Table 7.7. Support for collaboration.

Collaboration	**#Tools**	**Tools**
No	11 (42%)	[79], [T06], [84], [24], [86], [T15], [90], [91], [92], [93] [96]
Yes: Off-line	9 (35%)	[78], [80], [67], [87], [88], [68], [94], [95], [22]
Yes: Real-time	6 (23%)	[77], [81], [82], [83], [85], [89]

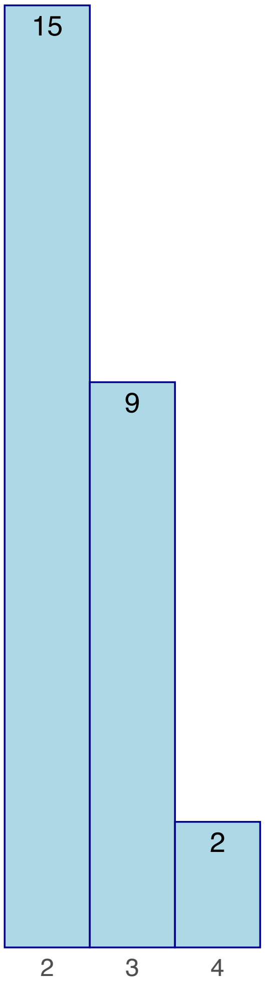
7.4.2 User-oriented characteristics (RQ1)

In this section, we discuss the findings related to the user-oriented characteristics of the sampled tools. We contextualize our findings in terms of the three aspects of blended modeling tools: the support for multiple notations (Section 7.4.2), seamless interaction (Section 7.4.2), and flexibility (Section 7.4.2).

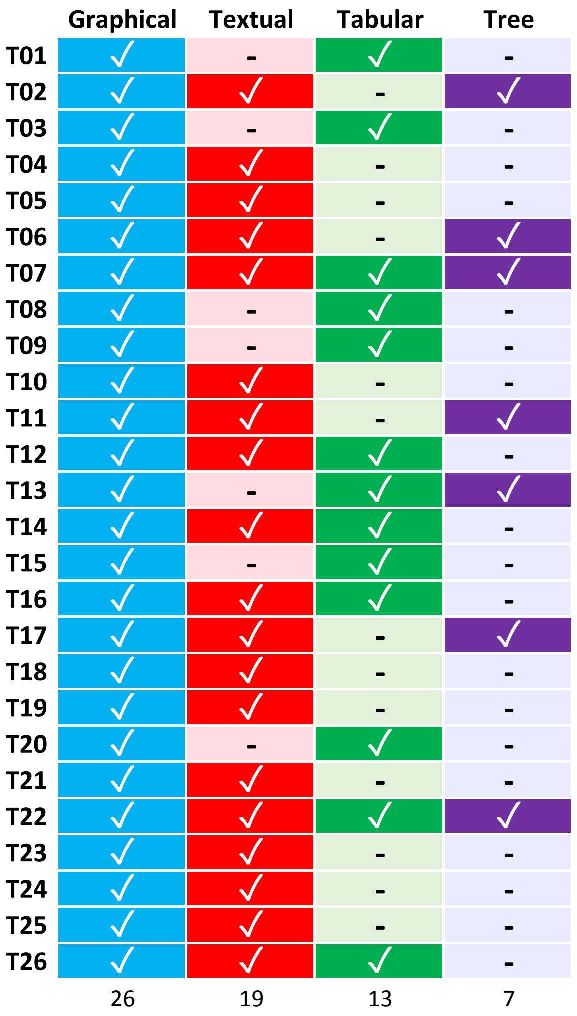
Notations

Notation types. As shown in Fig. 7.5a and Table 7.8, the majority of tools, 5 of 26 (19%), support two types of notation, with additional nine tools supporting three types, and two tools supporting four types.

Every tool, 26 of 26 (100%), features a graphical notation. Textual notations are supported by 19 tools. Additional 13 tools were found with a support for tabular notations, and seven with a support for tree-like notations. This information is detailed in Fig. 7.5b and Table 7.9.



(a) Number of notation types.



	Graphical	Textual	Tabular	Tree
T01	✓	-	✓	-
T02	✓	✓	-	✓
T03	✓	-	✓	-
T04	✓	✓	-	-
T05	✓	✓	-	-
T06	✓	✓	-	✓
T07	✓	✓	✓	✓
T08	✓	-	✓	-
T09	✓	-	✓	-
T10	✓	✓	-	-
T11	✓	✓	-	✓
T12	✓	✓	✓	-
T13	✓	-	✓	✓
T14	✓	✓	✓	-
T15	✓	-	✓	-
T16	✓	✓	✓	-
T17	✓	✓	-	✓
T18	✓	✓	-	-
T19	✓	✓	-	-
T20	✓	-	✓	-
T21	✓	✓	-	-
T22	✓	✓	✓	✓
T23	✓	✓	-	-
T24	✓	✓	-	-
T25	✓	✓	-	-
T26	✓	✓	✓	-
	26	19	13	7

(b) Support for specific notation types.

Figure 7.5. Number and combinations of notation types.

Table 7.8. Number of supported notation types.

#Notation types	#Tools	Tools
2	15 (58%)	[77], [79], [80], [81], [83], [84], [85], [T15], [89], [90], [91], [92], [94], [95], [22]
3	9 (35%)	[78], [T06], [86], [24], [67], [87], [88], [68], [96]
4	2 (8%)	[82], [93]

Table 7.9. Support for specific notation types.

Notation type	#Tools	Tools
Graphical	26 (100%)	[77], [78], [79], [80], [81], [T06], [82], [83], [84], [85], [24], [86], [67], [87], [T15], [88], [68], [89], [90], [91], [92], [93], [94], [95], [22], [96]
Textual	19 (73%)	[78], [80], [81], [T06], [82], [85], [86], [24], [87], [88], [68], [89], [90], [92], [93], [94], [95], [22], [96]
Tabular	13 (50%)	[77], [79], [82], [83], [84], [86], [67], [87], [T15], [88], [91], [93], [96]
Tree	7 (27%)	[78], [T06], [82], [24], [67], [68], [93]

Embedded notations. We found a single occurrence of embedded notations, i.e., a host notation being enriched by fragments of another notation (Table 7.10). While the host notation is prevalent during the entirety of the interaction, the embedded notation is accessible in a specific subset of the host notation. For example, in the `Statecharts + Class Diagrams` (SCCD) formalism [97], Class Diagram fragments are used to augment the Statecharts formalism, and provide structural information to compose complex systems.

Table 7.10. Support for embedded languages.

Embedded languages	#Tools	Tools
No	25 (96%)	[77], [78], [79], [80], [81], [T06], [82], [83], [84], [85], [24], [86], [67], [T15], [88], [68], [89], [90], [91], [92], [93], [94], [95], [22], [96]
Yes	1 (4%)	[87]

Overlap. The majority of tools, 21 of 26 (81%), comes with notations that are not fully overlapping. This means that different notations provide different modeling aspects in these tools. An example of full overlap is where a graphical state machine language can render a state machine model with every structural feature; whereas a table only shows which states have transitions to which states.

Table 7.11. Overlap between notations.

Overlap	#Tools	Tools
Partial	21 (81%)	[77], [78], [79], [80], [81], [T06], [82], [83], [84], [24], [86], [67], [T15], [88], [68], [91], [93], [94], [95], [22], [96]
Complete	5 (19%)	[85], [87], [89], [90], [92]

Visualization and navigation

Usability aspects in general are hard to measure. To gain reliable results, it is necessary to conduct a complex user study with concrete tasks, a larger number of participants, interviews and/or surveys, and a thorough evaluation of the answers. This is not feasible in the context of this study and therefore, we decided to focus on usability aspects that are i) easily measured objectively and ii) specific to blended modeling. We do not consider the usability of modeling languages themselves as discussed in [98] and [99]. Instead, we focus on the usability of the tools in terms of the topics that are crucial for blended modeling. The idea of blended modeling is to use the notation that is best suited for the current task at hand. This makes it necessary to switch frequently between the available notations. Therefore, for pleasant usability with good support for the user, a tool must offer the possibility to visualize multiple notations side by side and/or provide seamless navigation between notations, or even synchronized navigation. To clarify this more focused view of usability, we use the term "seamless interaction".

Visualization of multiple syntaxes. In general, a blended modeling tool must have the ability to support multiple concrete syntaxes of the same abstract syntax. This parameter, in particular, addresses the possibility of simultaneously viewing multiple notations within a modeling tool, e.g., side-by-side or in an integrated manner such as projectional editors as mbeddr [87] do. All 26 identified tools support the simultaneous view of two or more notations.

Synchronized navigation. In addition to the previous parameter, this parameter investigates whether the navigation across multiple notations in the models' editors is synchronized. For instance, this can be the case in a side-by-side view, if an element in one notation is selected, also its corresponding element in the other notation is selected. Another example of such synchronized navigation is the usage of the double click feature to jump between different views showing corresponding elements but belonging to different notations. As shown in Table 7.12, more than half of the tools, 16 of 26 (62%), provides synchronized navigation facilities.

Table 7.12. Support for synchronized navigation.

Sync'd navigation	#Tools	Tools
Yes	16 (62%)	[78], [79], [81], [T06], [82], [83], [85], [86], [67], [87], [88], [89], [90], [92], [93], [22]
No	10 (38%)	[77], [80], [84], [24], [T15], [68], [91], [94], [95], [96]

Navigation among notations. Blended modeling tools introduce the benefit that the same model can be viewed and modified using different notations. To enable a fluent modeling experience, the effort required to navigate across notations should be minimal. This binary parameter classifies the effort. It can be either *immediate* (e.g., a click or a keyboard shortcut), or it can involve more complex steps, such as the navigation through multiple (context) menus or wizards. The majority of tools, 20 of 26 (77%), provide immediate navigation from one notation to the other, suggesting a better user experience in terms of seamless interaction.

Flexibility

Flexibility is the user-related embodiment of tolerating vertical and horizontal inconsistencies [10] at various levels of abstraction in the modeling stack and various modeling facilities. In this study, we specifically consider three types of flexibility, as follows.

Table 7.13. Navigation among notations.

Navigation	**#Tools**	**Tools**
Immediate	20 (77%)	[78], [79], [80], [81], [T06], [82], [83], [85], [24], [86], [67], [87], [T15], [88], [89], [90], [91], [92], [93], [22]
Complex	6 (23%)	[77], [84], [68], [94], [95], [96]

Flexibility – models. As shown in Table 7.14, the majority of tools, 19 of 26 (73%), does not provide flexibility at the model level. This means that there are no inconsistency tolerance mechanisms in place that would allow deviations between different notations describing the same model. However, a small set of six tools support model-level flexibility.

Table 7.14. Support for model-level flexibility.

Flexibility: models	**#Tools**	**Tools**
No	19 (73%)	[77], [78], [79], [81], [T06], [83], [84], [24], [67], [87], [88], [68], [90], [91], [92], [93], [94], [95], [96]
Yes	7 (27%)	[82], [86], [80], [85], [T15], [89], [22]

Flexibility – language. The majority of tools, 22 of 26 (85%), does not provide flexibility at the language-level. (Table 7.15) This means that vertical inconsistencies between model and language (e.g., broken conformance or typing relationships) are not tolerated. We found three exceptions, which are, however, different from the ones with support for model-level flexibility discussed above: mbeddr [87], OSATE [68], TopBraid Composer [93]. Only a single tool, Umple [22], supports both model- and language-level flexibility.

Flexibility – persistence. The majority of tools, 22 of 26 (85%), does not support persisting inconsistent models. (Table 7.16) Out of the ones with support for persistence-level flexibility, ETAS ASCET Developer [80] and Umple [22] support model-flexibility and flexibility at both levels, respectively. The other

Table 7.15. Support for language-level flexibility.

Flexibility: language	#Tools	Tools
No	22 (85%)	[77], [78], [79], [80], [81], [T06], [82], [83], [84], [85], [86], [24], [67], [T15], [88], [89], [90], [91], [92], [94], [95], [96]
Yes	4 (15%)	[87], [68], [93], [22]

two tools with support for persistence-level flexibility are MagicDraw [67] and SequenceDiagramOrg [90].

Table 7.16. Support for persistence flexibility.

Flexibility: persistence	#Tools	Tools
No	22 (85%)	[77], [78], [79], [81], [T06], [82], [83], [84], [85], [24], [86], [87], [T15], [88], [68], [89], [91], [92], [93], [94], [95], [96]
Yes	4 (15%)	[80], [67], [90], [22]

7.4.3 Realization-oriented characteristics (RQ2)

In this section, we discuss the findings related to the implementation characteristics of the sampled tools. We contextualize our findings in terms of the three aspects of blended modeling tools: the support for multiple notations (Section 7.4.3), seamless interaction (Section 7.4.3), and flexibility (Section 7.4.3).

Mapping and platforms

Mapping. The mapping between abstract syntax and notation is typically implemented either in a parser-based or in a projectional fashion. In *parser-based* approaches, the user modifies the models via different notations, and a parser produces the abstract syntax tree. In *projectional* approaches, however, the abstract syntax tree is modified directly. Since projectional editors bypass the stages of parser-based editors, they provide support for notations that cannot be easily parsed, but at the same time deliver a different editing experience for textual notations. As shown in Table 7.17, the majority of tools, 22 of 26 (85%), implement a parser-based editor, while four come with projectional facilities.

Table 7.17. Type of mapping.

Mapping	**#Tools**	**Tools**
Parser-based	22 (85%)	[78], [79], [80], [81], [T06], [82], [83], [84], [85], [24], [86], [88], [68] [89], [90], [91], [92], [93], [94], [95], [22], [96]
Projectional	4 (15%)	[77], [67], [87], [T15]

Platforms. Eclipse is the only frequently encountered platform in our sample. As shown in Table 7.18, 10 of 26 tools (38%) are built on top of Eclipse, and 18 are built on other, mainly custom platforms. mbdeddr [87] is the only MPS-based tool in our sample. One tool, MagicDraw [67], also supports more than one platform.

Table 7.18. Platforms of implementation.

Platform	**#Tools**	**Tools**
Other	17 (65%)	[77], [79], [81], [82], [83], [85], [67], [T15], [87], [88], [89], [90], [91], [92], [95], [22], [96]
Eclipse	10 (38%)	[78], [80], [T06], [84], [24], [86], [67], [68], [93], [94]

Change propagation and traceability

Change propagation and traceability are the realization-oriented manifestations of the *seamless integration* blended modeling aspect. (See Table 7.3.) During the data extraction phase, however, we have failed to obtain any useful information in these two categories. In the vast majority of cases (exception for ADOIT [77], ARIS [79], the Eclipse Process Framework [86], and MagicDraw [67]), we have not found explicit discussions of these concerns, nor any evidence of these concerns being explicit in the tool. We report these negative results to maintain the symmetry of our classification framework. We suggest replications of this study to be carried out in a conceptual way [100], i.e., attempting to answer the research questions using different methods.

Inconsistency management

Inconsistency visualization. As shown in Table 7.19, the majority of tools, 15 of 26 (58%), does not provide any visualization for inconsistencies. Out of the remaining 11 tools, eight implement an internal visualization mechanism, and three rely on external services.

Table 7.19. Support for inconsistency visualization.

Inconsistency visualization	#Tools	Tools
No	15 (58%)	[77], [79], [81], [T06], [83], [84], [86], [67], [68], [90], [91], [92], [94], [95], [96]
Internal	8 (31%)	[78], [82], [80], [85], [87], [88], [89], [22]
External	3 (11%)	[24], [T15], [93]

Inconsistency management type. The two fundamental approaches to manage inconsistencies are prevention, and allow-and-resolve [14]. Preventive techniques effectively prohibit the emergence of inconsistencies, either by serializing user operations (e.g., via locking), or by constructing the underlying data structures in a way that they can never be inconsistent (e.g., in conflict-free replicated data types (CRDT) [38]. Allow-and-resolve approaches embrace the existence of inconsistencies [47] instead of preventing them. This allows treating inconsistencies with highly sophisticated operations for tolerance [50, 52], and resolution [101, 102, 103]. As shown in Table 7.20, half of the tools, 13 of 26 (50%), prevent inconsistencies. The remaining tools either manage inconsistencies on-the-fly (11 of 26 – 42%) or on-demand (2 of 26 – 8%).

Table 7.20. Support for different inconsistency management types.

Inconsistency mgmt type	**#Tools**	**Tools**
Preventive	13 (50%)	[77], [79], [T06], [83], [86], [67], [T15], [88], [92], [94], [95], [22], [96]
On-the-fly	11 (42%)	[78], [80], [81], [82], [84], [85], [87], [89], [90], [91], [93]
On-demand	2 (8%)	[24], [68]

Inconsistency management automation. As shown in Table 7.21, 13 of 26 tools (50%) do not provide inconsistency resolution due to their preventive inconsistency management approach. These tools are identical to the ones of the *preventive* category in Table 7.20. Out of the remaining 13 tools, 11 provide some level of automation for resolving inconsistencies, while two tools rely on manual resolution.

Table 7.21. Level of automation of inconsistency management.

Inconsistency automation	**#Tools**	**Tools**
Not applicable	13 (50%)	[77], [79], [T06], [83], [86], [67], [T15], [88], [92], [94], [95], [22], [96]
Fully automated	6 (23%)	[82], [68], [89], [90], [91], [93]
Semi-automated	5 (19%)	[78], [84], [85], [87], [24]
Manual	2 (8%)	[80], [81]

7.5 Orthogonal findings

We have analyzed the extracted data for horizontal findings, orthogonal to the vertical analysis reported in the previous section. Specifically for this purpose, we have generated contingency tables for each pair of categories of the classification framework and looked for relevant emerging correlations. In this section, we discuss these findings and contextualize them in terms of the aspects of blended modeling: the support for multiple notations (Section 7.5.1), seamless interaction (Section 7.5.2), and flexibility and inconsistency management (Section 7.5.3); and in the additional aspect of technological trends that are independent from the blended aspects (Section 7.5.4).

7.5.1 Number of notation types and overlap of notations

As shown by the data in Section 7.4.2, the sampled tools support 2.5 types of notation on average. In about 81% of the cases, the overlap between the specific notations is only partial, thus providing a richer way to build models.

Notation types count vs Web-based nature. The number of types of notation tends to be higher in desktop tools. Tools with more than two types of notation are exclusively desktop-based. While every web-based tool in our sample provides a maximum of two types of notations, 11 of 17 desktop tools (65%) provide three or more types of notations. We have measured a statistically significant difference at $p = 0.0064$.[40]

Overlap of notations vs Web-based nature. We found significantly more completely overlapping notations in web-based tools than in desktop-based tools. 4 of 9 web-based tools (44%) come with completely overlapping notations. This ratio is 5.9% in desktop-based tools ($p = 0.0176$). This is in line with the previous observation of web-based tools typically providing fewer types of notation. It is plausible to assume that in desktop tools, the higher number of

[40]For the remainder of the paper, $\alpha = 0.05$, unless specifically noted otherwise. Following the directions of Haviland [76], we report the p-values of the conventional Chi-square test without Yates's correction for continuity.

notation types might result in relevant differences between the notations and, thus, less overlap among them.

Notation types count vs Open-source nature. The number of notation types tends to be higher in open-source tools than in commercial ones. Three or more types of notations are supported in 8 of 13 open-source tools (62%), while this number is only 3 of 13 (23%) in commercial tools. However, a deeper look also reveals that the only two tools supporting four types of notations are commercial ones ([82], [93]). While 2 of 13 commercial tools (15%) provide four types of notations, 10 of 13 (77%) of them support only two. These differences are significant at $p = 0.0105$. It is plausible to assume that while commercial tool vendors have the capabilities to develop sophisticated tools with many types of notations, they still opt for a more streamlined user experience either due to explicit user requirements, or to minimize the technological risks and improve the maintainability of the tools.

7.5.2 Seamless interaction

In terms of seamless interaction, we have found significant relationships between the navigation among notations, their synchronicity, and the presence of inconsistency visualization.

Navigation among notations vs Synchronous navigation. We have observed a statistically significant difference (p =4E-4) between the *complexity of navigation among notations*, and the *synchronicity of navigation*. The two features go hand in hand. 16 of 16 tools (100%) with support for synchronous navigation also support immediate navigation across different notations. In contrast, only 4 of 10 tools (40%) without synchronous navigation support immediate navigation. That is, in over half of such tools, navigation between notations becomes a complex and tedious task, significantly impacting the user experience in terms of seamless interaction. Synchronous navigation is more frequently observed in tools with completely overlapping notations. While 5 of 5 tools (100%) with completely overlapping notations operate with synchronous navigation, this ratio is only 11 of 21 (52%) in tools with partially overlapping notations.

Navigation among notations vs Inconsistency visualization. We observed that 11 of 11 tools (100%) that support inconsistency visualization operate with immediate navigation; while tools without inconsistency visualization support immediate navigation only in 9 of 15 cases (60%). The difference is significant at $p = 0.0168$.

7.5.3 Flexibility and inconsistency management

As discussed in Section 7.4.2, flexibility, in general, is sporadically supported by the tools we have sampled. We have found that the three types of flexibility features (model-level, language-level, persistence-level), often correlate with inconsistency management aspects.

Model-level flexibility vs Inconsistency visualization. Inconsistency visualization is significantly better supported in tools with model-level flexibility. We have found that 7 of 7 tools (100%) with model-level flexibility also support inconsistency visualization, while this ratio drops to 4 of 19 (21%) in tools without model-level flexibility ($p = 3\text{E}{-4}$). It is plausible to assume that inconsistency visualization is an enabler to model-level flexibility. Visualizing inconsistencies certainly helps the stakeholders to keep track of inconsistencies and reason about the most appropriate time and approach to resolving them.

Inconsistency visualization vs Collaboration. Tools with internal inconsistency visualization features are also collaborative tools. This holds for 8 of 26 tools (31%). Conversely, the 11 of 26 tools (42%) without collaborative features do not support internal means of inconsistency visualization. The ratio of collaborative and non-collaborative tools is split almost evenly when inconsistency visualization is not present. 15 of 26 tools (58%) come without inconsistency visualization, out of which seven (27%) support collaboration and eight (31%) lack collaborative features. These relationships are significant at $p = 0.0047$. These observations can be explained by the strong relationship between collaboration and inconsistencies: as the lack of collaboration might severely reduce the cases when inconsistencies can appear, tools vendors whose tools do not support collaboration might be less interested in developing internal inconsistency visualization techniques.

7.5.4 Technological trends

We have further identified some purely technological trends, orthogonal to the three facets of blended modeling, mainly related to the web-based nature of tools (Table 7.5), their collaborative features (Table 7.7), and their platforms of implementation (Table 7.18).

Collaboration on the web. The type of collaboration tends to correlate with the type of client software. 5 of 6 tools (83%) that operate with synchronous (real-time) collaboration, are implemented as web-based tools. In contrast, 7 of 9 tools (78%) that operate with asynchronous (off-line) collaboration, are implemented as desktop tools. The type of client software is nearly evenly split in collaborative tools between web clients (7 of 15 – 47%) and desktop clients (8 of 15 – 53%). However, 9 of 11 non-collaborative tools (82%) are built as desktop applications, and we found only two web-based non-collaborative tools. These differences are significant at $p = 0.0164$. These observations are in line with the observations of our previous work [57], especially on the apparent mobilization of collaborative modeling.

Modeling platforms are primarily desktop-based. We observed that neither of the web-based tools in our sample is implemented on a platform that explicitly aims to provide *modeling* capabilities. In contrast, 11 of 18 desktop tools (61%) are implemented on top of a modeling platform, such as Eclipse (10 of 18 – 56%), JetBrains MPS (1 of 18 – 6%), and other, custom platforms (7 of 18 – 39%). While the web-based tools in our sample leverage web frameworks that provide reusable elements to build front-end and back-end functionality, the lack of *modeling* frameworks tailored to the web are apparent. These differences are significant at $p = 0.0097$.

7.6 Discussion

The corpus of this paper consists of 68 academic papers and 68 entries of grey literature survey, which eventually resulted in 26 identified tools. Based on the rigorously constructed research protocol, we are reasonably confident in the representativeness of our sample for the field under study.

7.6.1 Takeaways

The main takeaway of our investigation is that the state-of-the-art and state-of-the-practice tools only provide *partial and accidental support for blended modeling*. This is not a surprising result, considering the novel and emerging nature of the concept of blended modeling. We have found adequately scaling tools in terms of the number of supported notation types. 11 of 26 tools (42%) provide more than the minimal two notation types (Table 7.8). Various aspects related to flexibility, however, pose a potentially serious obstacle for multi-notation tools to become true blended modeling tools. Only 7 of 26 tools (27%) provide flexibility at the instance model level, i.e., tolerance of horizontal inconsistencies between models (Table 7.14). 4 of 26 tools (15%) support flexibility at the language level, i.e., tolerance of vertical inconsistencies, such as conformance or type discrepancies (Table 7.15). In terms of user experience (UX), and especially seamless interaction, we noticed encouraging signs in cross-notation navigability and inconsistency management automation. 16 of 26 tools (62%) support a synchronized navigation across their supported notations (Table 7.12) and, in 20 of 26 tools (77%), immediate navigation is also available (Table 7.13). This enables a better concert of notations, allowing using them in a truly complementary fashion. 11 of 13 tools (85%) that allow inconsistencies to occur treat them with a substantial level of automation; only 2 of 13 (15%) of such tools (a grand total of 8% – 2 of 26) rely on manual resolution of inconsistencies (Table 7.21).

In terms of *user-oriented characteristics* (RQ1), we observed a strong dominance of graphical notations, supported by 26 of 26 tools (100%), followed by textual (19 of 26 – 73%), tabular (13 of 26 – 50%), and tree-based ones (7 of 26 – 27%) (Table 7.9 and Fig. 7.5b). Only 5 of 26 tools (19%) feature a combination of notations that are completely overlapping in terms of modeling language concepts (Table 7.11). This means that multi-notation tools tend to leverage the complementary nature of different types of notation. This is a welcome direction as it opens up for opportunities of a richer modeling experience, paramount in approaches such as MVM and MPM and, as such, it motivates the efforts of blended modeling.

In terms of *realization-oriented characteristics (RQ2)*, we observed the dominance of parser-based solutions, employed in 22 of 26 tools (85%)(Table 7.17). Evidence suggests that projectional editors align better with multi-view and multi-notation principles [32, 104, 30], which are now the typical modeling settings for complex systems [1]. The average age of tools in our sample is 10.6 years ($\sigma = 5.9$), dating the typical modeling tool earlier than the uptick in research interest in projectional editors [41]. We foresee the support for projectional editors to grow as modeling tools are becoming more complex in their denotational and semantic functionalities. We observed a relatively high support for automation of inconsistency management (Table 7.21). Inconsistency management, and tolerance in particular (Tables 7.14-7.16), are key enablers to the flexibility of modeling tools. Only 2 of 26 tools (8%) come without some level of automation in resolving conflicts and these are either research tools, such as [81], or tools that are explicitly not supporting groupwork, such as [80].

7.6.2 Challenges and opportunities

By mapping the state-of-the-art and state-of-the-practice, we have identified challenges and opportunities related to the concept of blended modeling in relation to tools.

Multi-formalism. Our study assumed one single underlying abstract syntax and a single underlying formalism, but even with this simplification, the support

[41]A directed search on Google Scholar using the `(intitle:"projectional editing" OR intitle:"projectional editor" OR intitle:"projectional editors" OR ("projectional editing" OR "projectional editor" OR "projectional editors")` search string suggests an increasing publication output starting from 2013.

for multi-notation is sporadic. Multi-formalism, and especially multi-semantics, exacerbates this problem as we anticipate the interest in blended modeling gradually shifting towards more complex domains [18, 13, 10]. We see an opportunity for tool builders and integrators in complex engineering domains that inherently work in an MVM/MPM setup, such as mechatronics, automotive, and robotics, to incorporate blendedness as an enabling concept into their existing tool ecosystems. However, this should be preceded by academic research on extending blended modeling, especially on topics such as coordination between models of different languages [105], and synchronization of abstract and concrete syntax in DSLs [106]. Nevertheless, we expect an early maturation and rapid take-off of blended modeling techniques in an array of applied modeling settings. Therefore, we advise technology transfer entities to closely follow academic and semi-academic advancements to propel the transition of the concept to applied industrial settings.

Seamless interaction. As a primary user experience (UX) concern, seamless interaction can make a substantial difference in user satisfaction [107] towards modeling tools. The user-oriented aspects of our study (Section 7.4.2) show that current tools are often equipped with related features (e.g., synchronous navigation among notations). Such tools have the opportunity to provide holistic support for blended modeling. The evaluation and comparison of realization-oriented aspects, however, is certainly a challenge, as demonstrated in Section 7.4.3. The scope of our study did not include the development of methods that would allow extracting information about user experience and seamless integration of the different modeling paradigms in blended modeling tools. In general, the evaluation of such user-facing aspects remains a challenge. We encourage researchers to develop methods suitable for extracting the types of information outlined in Section 7.4.3; and to further enrich the user-facing aspects, based on Section 7.4.2. We suggest facilitating dedicated evaluation events, e.g., hands-on workshops at major conferences, where crowdsourcing models for hands-on experimentation and evaluation are feasible because of the volume of the co-located participants and their significant expertise, such as the Hands-on Workshop on Collaborative Modeling (HoWCoM)[42], and the work-

[42] http://howcom2021.github.io/

shop on Human Factors in Modeling / Modeling of Human Factors (HuFaMo)[43] at MODELS[44], as well as the Conference on Human Factors in Computing Systems (CHI)[45]. Explicitly modeled user interfaces [108] and API protocols [109] provide especially good foundations for developing software tools that allow seamless switching between notations. Seamless interaction across textual and graphical notations is especially challenging [105] due to the differences between their respective grammar-based and metamodel-based approaches [110]. Projectional editing [30] provides appropriate means to overcome these limitations, thus, we advise researchers to investigate seamless interaction from this standpoint as well.

Flexibility. The flexibility of modeling tools in terms of (temporarily) tolerating inconsistencies, such as violations of well-formedness rules and inter-notation/inter-view discrepancies, is best approached by employing state-of-the-art inconsistency models, such as eventual and strong eventual consistency [38]. Although the scope of this study does not entail the particularities of inconsistency management, we have identified traces and patterns of shortcomings in this aspect. While the majority of tools operate in a preventive inconsistency management fashion (Section 7.4.3), they implement prevention in the traditional way, i.e., by prohibiting consistency-breaking operations. Such approaches stem from the limitations of strict consistency, whereas novel developments in the field offer much better inconsistency management and, by extension, better flexibility. Strong eventual consistency (SEC) [38], for example, offers a convenient trade-off between the strictness of strong consistency and the guarantees of eventual consistency. As such, SEC is especially well-suited for tools whose developers are more comfortable with preventive inconsistency management models. Such avenues have been explored in multiple collaborative modeling frameworks, such as lowkey[46], and C-Praxis [111]. We see an opportunity in developing advanced inconsistency tolerance methods that work at the semantic level of models, especially if blended modeling is extended to support multiple

[43] https://www.monash.edu/it/humanise-lab/hufamo21
[44] http://www.modelsconference.org/
[45] https://chi2021.acm.org/
[46] https://github.com/geodes-sms/lowkey

abstract syntaxes or multiple semantics. Recently, inconsistency management between the data and (meta)model level has been investigated, e.g., by Zaher et al. [112]. Such directions align well with the persistence flexibility aspect of modeling tools, which is sporadically supported currently. In general, we encourage tool builders to treat inconsistencies as first-class citizens and, instead of overspending on resources to prevent them, we suggest appropriately managing them [47, 14].

The many facets of web-based tools. The interconnected nature of web-based tools and the advanced communication and networking standards of the Internet align well with building collaborative modeling tools. We observed a tendency of tool builders to use web technologies more in collaborative tools (Section 7.5.4). However, we also observed that web-based tools come with significantly less types of notations (Section 7.5.1), and that *modeling* platforms and frameworks are built for desktop applications (Section 7.5.4). It is possible that the shortage of modeling frameworks and language workbenches with a web-based focus limits the ability of tool vendors to provide rich modeling tools with numerous types of notations and advanced modeling facilities. Modeling platforms such as Eclipse already started providing support for deploying modeling tools onto the web, but this is merely a workaround. We foresee an increasing industrial interest in web-based modeling frameworks, such as WebGME [113], providing researchers of language engineering and language workbenches with opportunities.

Tools performance assessment. The current generation of modeling tools is facing challenges to manage large-scale complex models [114, 115]. Given the presence of multiple different notations in blended modeling, estimating tool performance when dealing with large-scale and complex models is crucial for the future technical sustainability of blended modeling. However, in our data analysis, we did not observe that tool builders discuss the performance of their blended modeling tools. We conjecture that this lack of communication is mainly because (i) tool performance is still an open problem in MDE [114], and (ii) there are still no standard benchmarks for objectively and fairly comparing the performance of different modeling tools. We suggest that researchers investigate a shared and open benchmark for assessing the performance of modeling tools when dealing with models of different levels of size and complexity (i.e., from a few up to millions of modeling elements). To avoid bias concerning specific DSMLs or application domains, populating such benchmark should be a community effort, where researchers and tool builders coming from different domains collaborate and contribute their models, language definitions, and requirements (e.g., expected time to open a model with 1M elements, expected time to propagate a model change from a visual syntax to the corresponding textual one, etc.). Having such shared benchmarks will provide practitioners with an evidence-based instrument for comparing similar modeling tools and choosing the best one according to their project and organizational needs. Also, a shared benchmark will help MDE researchers in designing and conducting empirical studies assessing the performance of (blended) modeling tools, thus providing objective and replicable knowledge for addressing the grand challenge of scalability in Model-Driven Engineering [114].

7.7 Threats to validity

The study reported in this paper has been carried out based on a carefully designed protocol. To minimize the threats to validity, we have designed our protocol based on well-established guidelines for systematic studies in software engineering [65, 64, 116] and those for including grey literature by Garousi et al. [117]. We have assessed the quality of our study following the guidelines by Petersen et al. [66] and achieved a 63.6% result. This score is significantly higher than the median and absolute maximum scores (33% and 48%, respectively) reported in [66]. This high score can be mainly attributed to the detailed search strategy; the involvement of external senior consultants in the study design phase; and the involvement of multiple authors in the screening phase, minimizing the number of false inclusions and exclusions. In the following, we discuss the possible threats to the validity of our study and elaborate on how we have mitigated them.

7.7.1 External validity

External validity concerns the generalizability of the results [64] and it is primarily associated with the sampling method. The most severe threat to external validity is the lack of representativeness of the selected tools to the field of interest in general. We have mitigated this threat by an appropriately constructed protocol with two orthogonal concerns. First, our search strategy included manual and automated search steps, with exhaustively iterative backward and forward snowballing. Second, we have carried out this search both for the academic and the grey literature [117]. Another class of threats to external validity can be attributed to the inclusion and exclusion criteria used in the screening. To mitigate these threats, we defined exclusion criteria specific to the type of literature (white or grey) being surveyed. Some threats remain, for example, due to the exclusion of non-peer-reviewed academic material (`A-E2` in Section 7.3), and the exclusion of proprietary tools that do not allow experimentation with at least a trial version (`GEN-E4`). We consider these threats minimal.

7.7.2 Internal validity

Internal validity is the extent to which claims are supported by data and it is primarily associated with the study design. We have mitigated this risk by the thorough construction and validation of our protocol. The protocol has been developed by multiple authors with relevant expertise on the topics related to blended modeling. Additionally, the protocol has been validated by an external reviewer with significant expertise in empirical research. We have employed rigorous descriptive statistical methods for orthogonal analysis and validation of the data to further mitigate the threats.

7.7.3 Construct validity

Construct validity is concerned with the generalizability of the measures of the study to the investigated concepts, and it is primarily associated with the categories and parameters employed during the data extraction and the subsequent analysis. We have mitigated the threats by mapping the research questions to typical parameters before constructing our search strategy. Consequently, we are reasonably confident about the construction validity of the search strings used in the automatic search steps. We have further minimized the threats in the screening phase by refining the inclusion and exclusion criteria in multiple iterations, to reach unambiguous definitions. Each study was assigned to two researchers randomly, and a third researcher was involved to oversee the results and make the final decisions on the inclusion.

7.7.4 Conclusion validity

Conclusion validity is the degree of credibility of the conclusions, based on the relationship between cause and effect. Specifically, in our case, conclusion validity is concerned with the relationship between the conclusions communicated in Sections 7.4.2–11.7 and the extracted data. We mitigated the main threats in two steps. First, considering that different researchers might interpret the same data in different ways, we have documented our research protocol in great detail and made it available along with our datasets and statistical analysis scripts in the publicly available replication package.[31] Second, we have constructed conclusions based only on the available data. Any hypotheses and conjunctures were explicitly marked as such.

7.8 Conclusions

In this paper, we have reported the results of our systematic, multi-vocal study on the potential, opportunities, and challenges of the emerging approach of blended modeling. We have reviewed nearly 5,000 academic papers, and nearly 1,500 entries of grey literature. Based on these, we have identified 133 candidate tools, and eventually selected 26 state-of-the-art and state-of-the-practice modeling tools which represent the current spectrum of modeling tools. We defined a classification framework for these tools which we used to map their support for other blended aspects, such as navigation and inconsistency tolerance.

Our findings show that current tooling only provides partial support for the features of blended modeling, in particular for inconsistencies between different notations of the same model. The existing support for automated consistency management is encouraging. We also observe that the overlap between notations is not complete. Projectional editing seems to be a promising avenue for future blended modeling, but most existing tools we reviewed are not projectional. Concerning the challenges, we observe that support for multi-formalism and multi-semantics is still largely lacking. We also see opportunities for improvements when it comes to the seamless integration of the different modeling notations and the evaluation of the user experience. Finally, we identify incorporating “softer” models of consistency that directly use the semantics of the models to achieve eventual consistency as a promising area of future research. We foresee a new generation of modeling tools that will take blended modeling further by introducing semantic techniques that will allow basing the modeling workflow on multiple different abstract syntaxes. As for future work, we are working on implementing a generator that produces blended modeling tools for arbitrary domain-specific languages. These tools will be based on the takeaways of this study as well as on a prototype implementation that already embraces the blended principles by Addazi et al. [8]. We intend to keep our dataset up-to-date and report increments on the efforts made on improving blended modeling. Finally, we plan to develop methods for the evaluation of the user experience of blended modeling tools based on hands-on events and workshops.

Referred Tools

[T01] BOC Products & Services AG (2021) ADOIT:Community Edition. https://www.adoit-community.com/en/, Retrieved: 22/05/2021.

[T02] Beauvoir, P and Sarrodie, JB (2021) Archi. https://www.archimatetool.com/, Retrieved: 22/05/2021.

[T03] Software AG (2021) ARIS. https://www.ariscommunity.com/, Retrieved: 22/05/2021.

[T04] ETAS (2021) ASCET Developer. https://www.etas.com/en/products/ascet-developer.php, Retrieved: 22/05/2021.

[T05] Université de Montréal (2021) AToMPM. https://atompm.github.io/, Retrieved: 22/05/2021.

[T06] Mälardalen University (2021) Blended Profile. http://www.es.mdh.se/ModComp/demo.html, Retrieved: 09/08/2021.

[T07] Viev (2021) Boston Professional. https://www.viev.com/index.php/products-menu/boston-professional, Retrieved: 22/05/2021.

[T08] ESTECO SpA (2021) Cardanit. https://www.cardanit.com/, Retrieved: 22/05/2021.

[T09] NASA (2021) certware. https://nasa.github.io/CertWare/, Retrieved: 22/05/2021.

[T10] Holistics Software (2021) DBDiagram. https://dbdiagram.io/home, Retrieved: 22/05/2021.

[T11] The Eclipse Foundation (2021a) Eclipse Papyrus. https://www.eclipse.org/papyrus/, Retrieved: 22/05/2021.

[T12] The Eclipse Foundation (2021b) Eclipse Process Framework Project. https://projects.eclipse.org/projects/technology.epf, Retrieved: 22/05/2021.

[T13] CATIA No Magic (2021) MagicDraw. https://www.3ds.com/products-services/catia/products/no-magic/, Retrieved: 22/05/2021.

[T14] itemis AG (2021) mbeddr. http://mbeddr.com/, Retrieved: 22/05/2021.

[T15] OMiLAB (2021) MEMO4ADO. https://austria.omilab.org/psm/content/memo4ado/info, Retrieved: 08/09/2021.

[T16] Modelisoft (2021) Modelio. https://www.modelio.org/, Retrieved: 22/05/2021.

[T17] Carnegie Mellon University (2021) OSATE. https://osate.org/, Retrieved: 22/05/2021.

[T18] Dovetail Technologies Ltd (2021) QuickDataBaseDiagrams. https://www.quickdatabasediagrams.com/, Retrieved: 22/05/2021.

[T19] - (2021a) SequenceDiagram.org. https://sequencediagram.org, Retrieved: 22/05/2021.

[T20] OMiLAB (2021) SOM/ADOxx. https://austria.omilab.org/psm/content/som/info?view=home, Retrieved: 22/05/2021.

[T21] - (2021b) Swimlanes.io. https://swimlanes.io/, Retrieved: 22/05/2021.

[T22] TopQuadrant, Inc (2021) TopBraid Composer. https://www.topquadrant.com/products/topbraid-composer/, Retrieved: 22/05/2021.

[T23] TU Wien (2021a) UMLet. https://www.umlet.com/, Retrieved: 22/05/2021.

[T24] TU Wien (2021b) UMLetino 14.3. https://www.umletino.com/, Retrieved: 22/05/2021.

[T25] University of Ottawa (2021) Umple. https://cruise.umple.org/umple/, Retrieved: 22/05/2021.

[T26] Universität Bremen (2021) USE – The UML-based Specification Environment. http://useocl.sourceforge.net/w/index.php/Main_Page, Retrieved: 22/05/2021.

Bibliography

[1] Magnus Persson, Martin Törngren, Ahsan Qamar, Jonas Westman, Matthias Biehl, Stavros Tripakis, Hans Vangheluwe, and Joachim Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *13th International Conference on Embedded Software (EMSOFT 2013)*, pages 1–10. IEEE, 2013.

[2] Colin Atkinson and Thomas Kühne. Reducing accidental complexity in domain models. *Softw. Syst. Model.*, 7(3):345–359, 2008.

[3] Manfred Broy. Software and System Modeling: Structured Multi-view Modeling, Specification, Design and Implementation. In *Conquering Complexity*, pages 309–372. Springer, 2012.

[4] Lars Huning, Timo Osterkamp, Marco Schaarschmidt, and Elke Pulvermüller. Seamless integration of hardware interfaces in UML-based MDSE tools. In *Proceedings of the 16th International Conference on Software Technologies, ICSOFT 2021, Online Streaming, July 6-8, 2021*, pages 233–244. SCITEPRESS, 2021.

[5] Zonghua Gu, Shige Wang, Sharath Kodase, and Kang G. Shin. An end-to-end tool chain for multi-view modeling and analysis of avionics mission computing software. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), 3-5 December 2003, Cancun, Mexico*, pages 78–81. IEEE Computer Society, 2003.

[6] Gang Yang, Xingshe Zhou, and Yuanyuan Lian. Constraint-Based Consistency Checking for Multi-View Models of Cyber-Physical System. In

2017 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C 2017, pages 370–376. IEEE, 2017.

[7] Michael Schulze, Jens Weiland, and Danilo Beuche. Automotive model-driven development and the challenge of variability. In *16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2-7, 2012, Volume 1*, pages 207–214. ACM, 2012.

[8] Lorenzo Addazi and Federico Ciccozzi. Blended graphical and textual modeling for UML profiles: A proof-of-concept implementation and experiment. *Journal of Systems and Software*, 175:110912, 2021.

[9] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended Modelling - What, Why and How. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019*, pages 425–430. IEEE, 2019.

[10] Ken Vanherpen, Joachim Denil, Istvan David, Paul De Meulenaere, Pieter J. Mosterman, Martin Törngren, Ahsan Qamar, and Hans Vangheluwe. Ontological reasoning for consistency in the design of cyber-physical systems. In *1st International Workshop on Cyber-Physical Production Systems, CPPS@CPSWeek 2016*, pages 1–8. IEEE, 2016.

[11] Reinhard von Hanxleden, Edward A. Lee, Christian Motika, and Hauke Fuhrmann. Multi-view Modeling and Pragmatics in 2020 – Position Paper on Designing Complex Cyber-Physical Systems. In *Large-Scale Complex IT Systems. Development, Operation and Management - 17th Monterey Workshop 2012*, volume 7539 of *LNCS*, pages 209–223. Springer, 2012.

[12] Jan Reineke, Christos Stergiou, and Stavros Tripakis. Basic problems in multi-view modeling. *Softw. Syst. Model.*, 18(3):1577–1611, 2019.

[13] Pieter J. Mosterman and Hans Vangheluwe. Computer Automated Multi-Paradigm Modeling: An Introduction. *Simul.*, 80(9):433–450, 2004.

[14] Istvan David. *A Foundation for Inconsistency Management in Model-Based Systems Engineering*. PhD thesis, University of Antwerp, Belgium, Middelheimlaan 1, 2020 Antwerpen, Belgium, 2019.

[15] Hannah R Rothstein and Sally Hopewell. Grey literature. *The handbook of research synthesis and meta-analysis*, 2:103–125, 2009.

[16] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Inf. Softw. Technol.*, 106:101–121, 2019.

[17] ISO/IEC/IEEE. Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, 2011.

[18] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. Multi-view approaches for software and system modelling: a systematic literature review. *Software and Systems Modeling*, 18(6):3207–3233, 2019.

[19] Jonathan Corley, Eugene Syriani, Hu}seyin Ergin, **and** Simon Van~ Mierlo. *Modern Software Engineering Methodologies for Mobile and Cloud Environments*, chapter Cloud-based Multi-View Modeling Environments, pages 120–139. Number 7. IGI Global, 2016.

[20] Paulo Carreira, Vasco Amaral, and Hans Vangheluwe. *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer Nature, 2020.

[21] Hans Vangheluwe, Juan de Lara, and Pieter J Mosterman. An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems)*, pages 9–20, 2002.

[22] Timothy C Lethbridge, Vahdat Abdelzad, Mahmoud Husseini Orabi, Ahmed Husseini Orabi, and Opeyemi Adesina. Merging modeling and programming using Umple. In *International Symposium on Leveraging Applications of Formal Methods*, pages 187–197. Springer, 2016.

[23] Benoît Ries, Alfredo Capozucca, and Nicolas Guelfi. 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*, pages 103–107. ACM, 2018.

[24] The Eclipse Foundation. Eclipse Papyrus. https://www.eclipse.org/papyrus/, 2021. Retrieved: 22/05/2021.

[25] Salome Maro, Jan-Philipp Steghöfer, Anthony Anjorin, Matthias Tichy, and Lars Gelin. 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*, pages 1–12, 2015.

[26] Markus Scheidgen. Textual modeling embedded into graphical modeling. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 153–168. Springer, 2008.

[27] Codruţ-Lucian Lazăr. Integrating Alf editor with Eclipse UML editors. *Studia Universitatis Babes-Bolyai, Informatica*, 56(3), 2011.

[28] Anis Charfi, Artur Schmidt, and Axel Spriestersbach. A hybrid graphical and textual notation and editor for UML actions. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 237–252. Springer, 2009.

[29] Oskar van Rest, Guido Wachsmuth, Jim R. H. Steel, Jörn Guy Süß, and Eelco Visser. Robust Real-Time Synchronization between Textual and Graphical Editors. In *Theory and Practice of Model Transformations - 6th International Conference, ICMT@STAF 2013*, volume 7909 of *LNCS*, pages 92–107. Springer, 2013.

[30] Markus Völter, Janet Siegmund, Thorsten Berger, and Bernd Kolb. Towards User-Friendly Projectional Editors. In *Software Language Engineering - 7th International Conference, SLE 2014*, volume 8706 of *LNCS*, pages 41–61. Springer, 2014.

[31] Charles Simonyi. The Death of Computer Languages, The Birth of Intentional Programming. Technical Report MSR-TR-95-52, September 1995.

[32] Thorsten Berger, Markus Völter, Hans Peter Jensen, Taweesap Dangprasert, and Janet Siegmund. Efficiency of projectional editing: a

controlled experiment. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016*, pages 763–774. ACM, 2016.

[33] Istvan David, Joachim Denil, and Hans Vangheluwe. Process-oriented Inconsistency Management in Collaborative Systems Modeling. In *16th International Industrial Simulation Conference 2018, ISC 2018*, pages 54–61. Eurosis, 2018.

[34] Leslie Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.

[35] Sarita V. Adve and Kourosh Gharachorloo. Shared Memory Consistency Models: A Tutorial. *Computer*, 29(12):66–76, 1996.

[36] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.

[37] Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno M. Preguiça, Mahsa Najafzadeh, and Marc Shapiro. Putting consistency back into eventual consistency. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015*, pages 6:1–6:16. ACM, 2015.

[38] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011*, volume 6976 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2011.

[39] George Spanoudakis and Andrea Zisman. Inconsistency management in software engineering: Survey and open research issues. In *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*, pages 329–380. World Scientific, 2001.

[40] Gregor Engels, Jochen Malte Küster, Reiko Heckel, and Luuk Groenewegen. A methodology for specifying and analyzing consistency of

object-oriented behavioral models. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001*, pages 186–195. ACM, 2001.

[41] Ken Vanherpen. *A Contract-based Approach for Multi-viewpoint Consistency in the Concurrent Design of Cyber-physical Systems*. PhD thesis, University of Antwerp, Belgium, Middelheimlaan 1, 2020 Antwerpen, Belgium, 2018.

[42] Perdita Stevens. Maintaining consistency in networks of models: bidirectional transformations in the large. *Softw. Syst. Model.*, 19(1):39–65, 2020.

[43] Holger Giese and Robert Wagner. Incremental Model Synchronization with Triple Graph Grammars. In *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006*, volume 4199 of *LNCS*, pages 543–557. Springer, 2006.

[44] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. Consistency-preserving edit scripts in model versioning. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013*, pages 191–201. IEEE, 2013.

[45] Steven Kelly. Collaborative modelling with version control. In *Software Technologies: Applications and Foundations - STAF 2017 Collocated Workshops*, volume 10748 of *LNCS*, pages 20–29. Springer, 2017.

[46] Heiko Klare, Max E. Kramer, Michael Langhammer, Dominik Werle, Erik Burger, and Ralf H. Reussner. Enabling consistency in view-based system development - the vitruvius approach. *J. Syst. Softw.*, 171:110815, 2021.

[47] Anthony Finkelstein. A Foolish Consistency: Technical Challenges in Consistency Management. In *Database and Expert Systems Applications, 11th International Conference, DEXA 2000*, volume 1873 of *LNCS*, pages 1–5. Springer, 2000.

[48] Anthony Finkelstein, Dov M. Gabbay, Anthony Hunter, Jeff Kramer, and Bashar Nuseibeh. Inconsistency Handling in Multperspective Specifications. *IEEE Trans. Software Eng.*, 20(8):569–578, 1994.

[49] Bashar Nuseibeh, Steve M. Easterbrook, and Alessandra Russo. Making inconsistency respectable in software development. *J. Syst. Softw.*, 58(2):171–180, 2001.

[50] Robert Balzer. Tolerating Inconsistency. In *Proceedings of the 13th International Conference on Software Engineering*, pages 158–165. IEEE/ACM, 1991.

[51] Steve Easterbrook, Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh. Coordinating Distributed ViewPoints: the anatomy of a consistency check. *Concurrent Engineering*, 2(3):209–222, 1994.

[52] Istvan David, Eugene Syriani, Clark Verbrugge, Didier Buchs, Dominique Blouin, Antonio Cicchetti, and Ken Vanherpen. Towards Inconsistency Tolerance by Quantification of Semantic Inconsistencies. In *Proceedings of the 1st International Workshop on Collaborative Modelling in MDE (COMMitMDE 2016) co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2016)*, volume 1717 of *CEUR Workshop Proceedings*, pages 35–44. CEUR-WS.org, 2016.

[53] Weslley Torres, Mark G. J. van den Brand, and Alexander Serebrenik. A systematic literature review of cross-domain model consistency checking by model management tools. *Softw. Syst. Model.*, 20(3):897–916, 2021.

[54] Aníbal Iung, João Carbonell, Luciano Marchezan, Elder Macedo Rodrigues, Maicon Bernardino, Fabio Paulo Basso, and Bruno Medeiros. Systematic mapping study on domain-specific language development tools. *Empir. Softw. Eng.*, 25(5):4205–4249, 2020.

[55] Sebastian Erdweg et al. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. *Comput. Lang. Syst. Struct.*, 44:24–47, 2015.

[56] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. Collaborative Model-Driven Software Engineering: reflections on the past and visions of the future. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 1–5. ACM, 2017.

[57] Istvan David, Kousar Aslam, Sogol Faridmoayer, Ivano Malavolta, Eugene Syriani, and Patricia Lago. Collaborative Model-Driven Software Engineering: A Systematic Update. In *24th International Conference on Model Driven Engineering Languages and Systems, MODELS 2021*, pages 273–284. IEEE, 2021.

[58] David Granada, Juan M. Vara, Francisco J. Pérez Blanco, and Esperanza Marcos. Model-based Tool Support for the Development of Visual Editors - A Systematic Mapping Study. In *Proceedings of the 12th International Conference on Software Technologies, ICSOFT 2017*, pages 330–337. SciTePress, 2017.

[59] Leandro Marques do Nascimento, Daniel Leite Viana, PAS Neto, DA Martins, Vinicius Cardoso Garcia, and SR Meira. A systematic mapping study on domain-specific languages. In *The Seventh International Conference on Software Engineering Advances (ICSEA 2012)*, pages 179–187, 2012.

[60] Eman Negm, Soha Makady, and Akram Salah. Survey on Domain Specific Languages Implementation Aspects. *International Journal of Advanced Computer Science and Applications*, 10, 2019.

[61] Bernhard Merkle. Textual modeling tools: overview and comparison of language workbenches. In *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA 2010*, pages 139–148. ACM, 2010.

[62] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, volume 2, pages 528–532. Wiley, 1994.

[63] Barbara A. Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. *Inf. Softw. Technol.*, 55(12):2049–2075, 2013.

[64] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. *Experimentation in Software Engineering*. Springer, 2012.

[65] Barbara A. Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering, Version 2.3. EBSE Technical Report EBSE-2007-01, Keele University and University of Durham, 2007.

[66] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Inf. Softw. Technol.*, 64:1–18, 2015.

[67] CATIA No Magic. MagicDraw. `https://www.3ds.com/products-services/catia/products/no-magic/`, 2021. Retrieved: 22/05/2021.

[68] Carnegie Mellon University. OSATE. `https://osate.org/`, 2021. Retrieved: 22/05/2021.

[69] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14*, pages 38:1–38:10. ACM, 2014.

[70] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008*, Workshops in Computing. BCS, 2008.

[71] Trisha Greenhalgh and Richard Peacock. Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources. *BMJ*, 331(7524):1064–1065, 2005.

[72] Roberto Franzosi. *Quantitative narrative analysis*. Number 162. Sage, 2010.

[73] Mark Rodgers, Amanda Sowden, Mark Petticrew, Lisa Arai, Helen Roberts, Nicky Britten, and Jennie Popay. Testing methodological guidance on the conduct of narrative synthesis in systematic reviews: effectiveness of interventions to promote smoke alarm ownership and function. *Evaluation*, 15(1):49–73, 2009.

[74] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. Architecting with microservices: A systematic mapping study. *J. Syst. Softw.*, 150:77–97, 2019.

[75] Federico Ciccozzi, Ivano Malavolta, and Bran Selic. Execution of UML models: a systematic review of research and practice. *Softw. Syst. Model.*, 18(3):2313–2360, 2019.

[76] Mark G Haviland. Yates's correction for continuity and the analysis of $2\times$ 2 contingency tables. *Statistics in medicine*, 9(4):363–367, 1990.

[77] BOC Products Services AG. ADOIT:Community Edition. https://www.adoit-community.com/en/, 2021. Retrieved: 22/05/2021.

[78] Phillip Beauvoir and Jean-Baptiste Sarrodie. Archi. https://www.archimatetool.com/, 2021. Retrieved: 22/05/2021.

[79] Software AG. ARIS. https://www.ariscommunity.com/, 2021. Retrieved: 22/05/2021.

[80] ETAS. ASCET Developer. https://www.etas.com/en/products/ascet-developer.php, 2021. Retrieved: 22/05/2021.

[81] Eugene Syriani and Hans Vangheluwe. AToMPM. https://atompm.github.io/, 2021. Retrieved: 22/05/2021.

[82] Viev. Boston Professional. https://www.viev.com/index.php/products-menu/boston-professional, 2021. Retrieved: 22/05/2021.

[83] ESTECO SpA. Cardanit. https://www.cardanit.com/, 2021. Retrieved: 22/05/2021.

[84] NASA. certware. https://nasa.github.io/CertWare/, 2021. Retrieved: 22/05/2021.

[85] Holistics Software. DBDiagram. https://dbdiagram.io/home, 2021. Retrieved: 22/05/2021.

[86] The Eclipse Foundation. Eclipse Process Framework Project. https://projects.eclipse.org/projects/technology.epf, 2021. Retrieved: 22/05/2021.

[87] itemis AG. mbeddr. http://mbeddr.com/, 2021. Retrieved: 22/05/2021.

[88] Modelisoft. Modelio. https://www.modelio.org/, 2021. Retrieved: 22/05/2021.

[89] Dovetail Technologies Ltd. QuickDataBaseDiagrams. https://www.quickdatabasediagrams.com/, 2021. Retrieved: 22/05/2021.

[90] -. SequenceDiagram.org. https://sequencediagram.org, 2021. Retrieved: 22/05/2021.

[91] OMiLAB. SOM/ADOxx. https://austria.omilab.org/psm/content/som/info?view=home, 2021. Retrieved: 22/05/2021.

[92] -. Swimlanes.io. https://swimlanes.io/, 2021. Retrieved: 22/05/2021.

[93] TopQuadrant, Inc. TopBraid Composer. https://www.topquadrant.com/products/topbraid-composer/, 2021. Retrieved: 22/05/2021.

[94] Martin Auer and Thomas Tschurtschenthaler. UMLet. https://www.umlet.com/, 2021. Retrieved: 22/05/2021.

[95] Martin Auer and Thomas Tschurtschenthaler. UMLetino 14.3. https://www.umletino.com/, 2021. Retrieved: 22/05/2021.

[96] Universität Bremen. USE – The UML-based Specification Environment. http://useocl.sourceforge.net/w/index.php/Main_Page, 2021. Retrieved: 22/05/2021.

[97] Simon Van Mierlo, Yentl Van Tendeloo, Bart Meyers, Joeri Exelmans, and Hans Vangheluwe. SCCD: SCXML extended with class diagrams. In *Proceedings of the Workshop on Engineering Interactive Systems with SCXML*, volume 2, pages 1–2, 2016.

[98] Daniel L. Moody. The `Physics` of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Trans. Software Eng.*, 35(6):756–779, 2009.

[99] Ankica Barisic, Vasco Amaral, and Miguel Goulão. Usability Evaluation of Domain-Specific Languages. In *8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012*, pages 342–347. IEEE, 2012.

[100] Alan R. Dennis and Joseph S. Valacich. A replication manifesto. *AIS Trans. Replication Res.*, 1:1, 2015.

[101] Tom Mens, Ragnhild Van Der Straeten, and Maja D'Hondt. Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis. In *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*, volume 4199 of *LNCS*, pages 200–214. Springer, 2006.

[102] Christian Nentwich, Wolfgang Emmerich, and Anthony Finkelstein. Consistency Management with Repair Actions. In *Proceedings of the 25th International Conference on Software Engineering*, pages 455–464. IEEE, 2003.

[103] Jürgen Gausemeier, Wilhelm Schäfer, Joel Greenyer, Sascha Kahl, Sebastian Pook, and Jan Rieke. Management of cross-domain model consistency during the development of advanced mechatronic systems. In

DS 58-6: Proceedings of ICED 09, the 17th International Conference on Engineering Design, volume 6 of *ICED*, pages 1–12, 2009.

[104] Markus Voelter. Language and IDE Modularization and Composition with MPS. In *Generative and Transformational Techniques in Software Engineering IV, International Summer School, GTTSE 2011*, volume 7680 of *LNCS*, pages 383–430. Springer, 2011.

[105] Luc Engelen and Mark van den Brand. Integrating Textual and Graphical Modelling Languages. *Electron. Notes Theor. Comput. Sci.*, 253(7):105–120, 2010.

[106] István Ráth, András Ökrös, and Dániel Varró. Synchronization of abstract and concrete syntax in domain-specific modeling languages. *Software & Systems Modeling*, 9(4):453–471, 2010.

[107] Barbara H. Wixom and Peter A. Todd. A Theoretical Integration of User Satisfaction and Technology Acceptance. *Inf. Syst. Res.*, 16(1):85–102, 2005.

[108] Eugene Syriani, Daniel Riegelhaupt, Bruno Barroca, and Istvan David. Generation of Custom Textual Model Editors. *Modelling*, 2(4):609–625, 2021.

[109] Simon Van Mierlo, Yentl Van Tendeloo, Istvan David, Bart Meyers, Addis Gebremichael, and Hans Vangheluwe. A multi-paradigm approach for modelling service interactions in model-driven engineering processes. In *Proceedings of the Model-driven Approaches for Simulation Engineering Symposium, SpringSim (Mod4Sim) 2018*, pages 6:1–6:12. ACM, 2018.

[110] Terje Gjøsæter, Andreas Prinz, and Markus Scheidgen. Meta-model or Grammar? Methods and Tools for the Formal Definition of Languages. In *Nordic Workshop on Model Driven Engineering (NW-MoDE 2008)*, pages 67–82, 2008.

[111] Jonathan Michaux, Xavier Blanc, Marc Shapiro, and Pierre Sutra. A semantically rich approach for collaborative model edition. In *Proceed-*

ings of the 2011 ACM Symposium on Applied Computing (SAC), pages 1470–1475. ACM, 2011.

[112] MohammadAmin Zaheri, Michalis Famelis, and Eugene Syriani. Towards Checking Consistency-Breaking Updates between Models and Generated Artifacts. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS 2021 Companion*, pages 400–409. IEEE, 2021.

[113] Miklós Maróti, Tamás Kecskés, Róbert Kereskényi, Brian Broll, Péter Völgyesi, László Jurácz, Tihamer Levendovszky, and Ákos Lédeczi. Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. *MPM@ MoDELS*, 1237:41–60, 2014.

[114] Antonio Bucchiarone, Jordi Cabot, Richard F. Paige, and Alfonso Pierantonio. Grand challenges in model-driven engineering: an analysis of the state of the research. *Softw. Syst. Model.*, 19(1):5–13, 2020.

[115] Dimitrios S Kolovos, Louis M Rose, Nicholas Matragkas, Richard F Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, and Massimo Tisi. A research roadmap towards achieving scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, pages 1–10, 2013.

[116] He Zhang, Muhammad Ali Babar, and Paolo Tell. Identifying relevant studies in software engineering. *Information and Software Technology*, 53(6):625–637, 2011.

[117] Vahid Garousi and João M. Fernandes. Highly-cited papers in software engineering: The top-100. *Inf. Softw. Technol.*, 71:108–128, 2016.

Chapter 8

Paper B: Towards automated support for blended modeling of UML-RT embedded software architectures

Malvina Latifaj, Federico Ciccozzi, Mattias Mohlin, Ernesto Posse.
Published in the 15th European Conference on Software Architecture (ECSA 2021).

Abstract

The Unified Modeling Language for Real Time (UML-RT) is a UML-based domain-specific language for modeling real-time embedded systems. HCL RTist, a model-based development environment for creating complex, event-driven and real-time software with advanced automation features provided by HCL Technologies, provides advanced support for UML-RT. Historically, as for the majority of UML profiles, editing support for UML-RT has also mainly exploited graphical notations (e.g., composite component and state-machine diagrams). Nevertheless, our previous experiments with blended graphical and textual modeling showed that the seamless use of different notations (i.e., graphical and textual) can significantly boost the work of architects and modelers. The results of those experiments together with the exposed wish of RTist customers of being able to design software architectures and applications via multiple notations led us to initiate this work towards an automated support for blended modeling of UML-RT. In this paper we describe the first step of the work – the effort of designing, implementing and integrating a textual notation for UML-RT state-machines in RTist.

8.1 Introduction

Software steers our daily life and methods for architecting it in more effective and efficient ways have been the focal point of research in software architecture for decades now. Automation is an inescapable ingredient of efficient architecting and software is not an exception. In the architecting of software, automation boosts the throughput, as well as improves the quality of the results, of virtually any architecting task, from requirements specification to maintenance, through design, development and validation. While automation relieves the architect from tedious and time-consuming activities, abstraction in terms of modeling allows to focus on the problem at hand from a more human-oriented perspective than programming. Abstraction and automation are considered to be the core pillars of Model-Driven Engineering. Models are first-class entities of the architecting and engineering process that abstract from certain aspects of the problem at hand; models are automatically manipulated via transformations for multiple purposes, from validation to code generation, but also communication among different stakeholders.

Commonly, domain-specific abstractions described in Domain Specific modeling Languages (DSML) [1] are leveraged to allow domain experts, who may or may not be software experts, to express complex functions in a domain-focused and human-oriented way than if using traditional programming languages. DSMLs formalise (for computer-based analysis and synthesis purposes) the *communication* language of architects at the level of domain-specific concepts such as an engine and wheels for a car. UML is the most used architecture description language in industry [2], the de-facto modeling standard in industry [3], and an ISO/IEC (19505-1:2012) standard. It is general-purpose, but it provides powerful profiling mechanisms to constrain and extend the language to achieve UML-based DSMLS, called UML profiles; in this paper, we focus on the UML real-time profile (UML-RT)[4], and its implementation in an industrial tool, HCL RTist[1].

[1] `https://www.hcltechsw.com/rtist`

8.1.1 Problem, motivation, and the RTist case

Domain-specific modeling tools, like RTist, traditionally focus on one specific editing notation (such as text, diagrams, tables or forms). This limits human communication, especially across stakeholders with varying roles and expertise. Moreover, architects and engineers may have different notation preferences; not supporting multiple notations negatively affects their throughput. Besides the limits to communication, choosing one particular kind of notation has the drawback of limiting the pool of available tools to develop and manipulate models that may be needed. For example, choosing a graphical representation limits the usability of text manipulation tools such as text-based diff/merge, which is essential for team collaboration. When tools provide support for both graphical and textual modeling, it is mostly done in a mutual exclusive manner. Most off-the-shelf UML modeling tools, such as IBM Rational Software Architect[2] or Sparx Systems Enterprise Architect[3], focus on graphical editing features and do not allow seamless graphical–textual editing. This mutual exclusion suffices the needs of developing small scale applications with only very few stakeholder types. RTist is not an exception. It provides support for modeling UML-RT architectures and applications based on graphical *composite structure diagrams*, to model structure, and *state-machine diagrams*, to model behavior. In addition, the implementation of UML-RT in RTist provides support for leveraging C/C++ action code for the description of fine-grained, algorithmic, behaviors within graphical state-machines. That is needed to enable the definition of full-fledged UML-RT models from which executable code can be automatically generated. While providing means to model graphical entities and "program" algorithmic behaviours textually, the two are disjoint, since the modeling of UML-RT is graphical only and the textual C/C++ is injected in graphical models as "foreign" entity and with almost no overlapping with graphical model elements. The aim is instead to achieve a modeling tool that is able to make different stakeholders to work on overlapping parts of the models using different modeling notations (e.g., graphical and textual) in an automated manner.

[2] http://www-03.ibm.com/software/products/en/ratsadesigner/
[3] https://sparxsystems.com/

8.1.2 Paper contribution

In this paper we describe the first step towards providing a fully blended graphical-textual modeling environment for UML-RT in RTist. Our experiments in a previous study with blended graphical-textual modeling showed that the seamless use of different notations can significantly boost the architecting of software using UML profiles [5]. The results of those experiments together with the exposed wish of RTist customers of being able to design software via multiple notations led us to initiate this work towards an automated support for blended modeling of UML-RT in RTist. In this paper we focus on the design, implementation and integration of a textual notation for UML-RT state-machines in RTist.

8.2 Blended modeling: what and why

We have previously defined the notion of *blended modeling* [6] as:

> *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.*

A seamless blended modeling environment, which allows stakeholders to freely choose and switch between graphical and textual notations, can greatly contribute to increase productivity as well as decrease costs and time to market. Such an environment is expected to support at least graphical and textual modeling notations in parallel as well as properly manage synchronisation to ensure consistency among the two. The possibility to visualise and edit the same information through a set of diverse perspectives always in sync has the potential to greatly boost communication between stakeholders, who can freely select their preferred notation or switch from one to the other at any time. Besides obvious notation-specific benefits, such as for instance the possibility to edit textual models in any textual editor outside the modeling environment, a blended framework would disclose the following overall benefits.

Flexible separation of concerns and better communication. Providing graphical and textual modeling editors for different aspects and sub-parts (even overlapping) of a DSML like UML-RT enables the definition of concern-specific architectural views characterised by either graphical or textual modeling (or both). These views can interact with each other and are tailored to the needs of their intended stakeholders. Due to the multi-domain nature of modern software systems (e.g., cyber-physical systems, Internet-of-Things), this represents a necessary feature to allow different domain experts to describe specific parts of a system using their own domain-specific vocabulary and notation, in a so called *multi-view modeling* [7] fashion. The same information can then be rendered and visualised through other notations in other perspectives to maximise understanding and boost communication between experts from different domains as well as other stakeholders in the development process.

Faster modeling activities. We have experimented with blended modeling of UML profiles [5] and the seamless combination of graphical and textual modeling has shown a decreased modeling effort in terms of time thanks to the following two factors:

1. Any stakeholder can choose the notation that better fits her needs, personal preference, or the purpose of her current modeling task, at *any time*. For instance, while structural model details can be faster to describe by using diagrammatic notations, complex algorithmic model behaviours are usually easier and faster to describe using textual notations (e.g., Java-like action languages).

2. Text-based editing operations on graphical models[4], such as copy&paste and regex search&replace, syntax highlighting, code completion, quick fixes, cross referencing, recovery of corrupted artefacts, text-based diff and merge for versioning and configuration, are just few of the features offered by modern textual editors. These would correspond to very complex operations if performed through graphical editors; thereby, most of them are currently not available for diagrams. Seamless blended modeling

[4]Please note that by *graphical/textual model*, we intend a model rendered using a graphical/-textual notation.

would enable the use of these features on graphically-described models through their textual editing view. These would dramatically simplify complex model changes; an example could be restructuring of a hierarchical state-machine by moving the insides of a hierarchical state. This is a demanding re-modeling task in terms of time and effort if done at graphical level, but it becomes a matter of a few clicks (copy and paste) if done at textual level.

8.3 A textual notation for UML-RT state-machines

In this paper, we introduce a textual notation for UML-RT state-machines that is intended to be part of future RTist release. UML-RT is a real-time profile that aims to simplify the ever-increasing complex software architecture specification for real-time embedded systems. The UML-RT concepts are inherited from the ones defined in the Real-time Object-Oriented modeling Language (ROOM) [4] and represented using UML extensibility mechanisms. UML-RT enables both *structure modeling* and *behavior modeling* of real-time systems [8]. The structural part is represented using composite structure diagrams, whereas the behavioral part is represented using state-machine diagrams. The fundamental concepts of UML-RT are capsules, which are encapsulated active entities that can execute in parallel. UML-RT relies on state-machines for the modeling of capsules' behaviour. In the case of a missing state-machine, the capsule only operates as container for other sub-capsules. A behavioral state-machine in UML-RT is composed of states, pseudo states, and transitions. States can be simple or composite, and the presence of composite states results in a hierarchical state-machine. Pseudo states consist of the initial pseudo state that acts as the starting point of the state-machine, and choice and junction pseudo states where guards on outgoing transitions determine which one to execute next. The remaining pseudo states (i.e., entry, exit, and history) are only used in hierarchical state-machines. Entry and exit pseudo states are used to enter and exit composite states, while history pseudo state is used to invoke the last active state prior to the exit of the composite state. Transitions indicate a change of state and can contain triggers that initiate transitions in the form of events, guard conditions that must evaluate to true for the initiation of the transition, and effects.

8.3.1 Textual language workbench

To complement the existing graphical editor in RTist with a textual notation and editor, a language workbench for such purpose needed to be carefully selected. HCL RTist is an Eclipse-based environment that leverages the Eclipse Modeling Framework (EMF)[5] as a backbone. Thereby, by choosing an EMF-based language workbench, we could leverage EMF as a common data layer. For this reason, we chose Xtext[6], a framework for the development of textual DSMLs, based on EBNF grammars. The textual editor supports an outline view, syntax highlighting, error checking, quick-fix proposals, and many other features provided by Xtext. Furthermore, Xtext provides code completion for keywords and cross-references by increasing the usability of the language and decreasing the learning curve.

8.3.2 Definition of a textual notation

Our goal was to introduce a textual notation (and related editor) to the already existing UML-RT profile supported by RTist. A possible alternative was to use this underlying metamodel consumed by the RTist's graphical editor as an input for an Xtext plugin to automatically generate a textual editor. However easy to implement, this process generates erroneous and unintuitive grammar, too far from the expectations of RTist's users. Manually editing this generated grammar would have been a tedious and potentially error-prone process. Therefore, we decided to design a textual notation, in terms of an Xtext grammar, from scratch. Starting from a wish-list of RTist's customers and architects, and using the UML-RT metamodel portion describing state-machines as blueprint, we manually defined of our UML-RT textual notation for state-machines in Xtext. The steps needed for the definition of the grammar were the following.

Identify reserved keywords. When defining a DSML, it is crucial to identify the reserved keywords used to typify the core concepts of the language.

[5] https://www.eclipse.org/modeling/emf/
[6] https://www.eclipse.org/Xtext/

The importance of these keywords lies in improved readability, higher language familiarity, and efficient parsing as they serve as directives for specific concepts. The chosen keywords for the textual syntax for UML-RT state-machines are the following: *capsule, statemachine, state, initial, junction, choice, entry, exit, entrypoint, exitpoint, history, transition, when, on* and *inherits*.

Elements' ordering strategy. Even though it is not mandatory for our language to have a fixed order of elements, this approach enhances readability and navigation of the textual syntax, as well as increased predictability on where the elements created in other notations will be placed in the textual syntax. Our grammar is based on the vertical distance approach where elements that affect each other's understandability and are closely related [9], are grouped together and have a low vertical distance. Furthermore, being that this grammar prohibits cross-references before element declaration, we take the aforementioned statement into consideration and make sure that elements that need to be cross-referenced will be declared before the cross-reference takes place.

A spoonful of syntactic sugar. The majority of programming languages, including C++, that is used as action code for behavioral state-machines, makes use of statement terminators in the form of semi-colons. Being that one of the main goals when introducing this textual syntax is for developers to use it jointly with the C++ action code, we introduced consistent use of semi-colons for indication statement termination to make the grammar more conforming to C++ and to increase readability. For the same readability reasons and developers' preferences, we also introduce colons after transition names. Furthermore, to make the grammar more compact, we allow the declaration of multiple objects of the same type in one single line of code. Due to the combination of the textual syntax with action code, we need to handle C++ code blocks so we can "isolate" them and make them distinguishable from the rest of the grammar. For this reason, we include backticks in order to enclose code snippets and to make the lexer aware of where the code block begins and ends.

The overall goal during this process was to keep a fixed concrete syntax while simultaneously enhancing the abstract syntax, even though frequently we had to trade-off between ease of expression in the concrete syntax and extra complexity in the abstract syntax.

8.3.3 Scoping

Scoping in Xtext is concerned with the visibility of elements; therefore, the scope provider computation returns the target candidates that are visible in the current context and by a given reference. In order to enforce the UML-RT's modularity it is necessary to specify a custom scope provider. The default behavior of Xtext allows establishing a cross-reference to all the elements of a particular type that are located inside the same Eclipse resource (i.e., project). By customizing the scope provider, we restrict this behaviour, and only allow cross-references for elements declared in the same model file. The rationale behind this decision lies in the fact that multiple model files containing different capsules can be located inside the same resource, and a particular capsule should not be able to cross-reference elements of other capsules. However, a key concept in which UML-RT relies on to reuse and extend parts of existing state-machines is the inheritance mechanism. When capsule A inherits capsule B, the state-machine of capsule A implicitly inherits the state-machine of capsule B. Therefore, to support inheritance, we need to customize the scope provider so that it allows cross-references for elements not only from the capsule itself, but also from the inherited capsule, in case there is one. Another default behavior of Xtext consists in allowing cross-references for all elements of a particular type declared in the same model file, regardless of their level of nesting. This contradicts an important UML-RT concept; compound transitions. Since transitions in UML-RT state-machines can not cross state boundaries, the concept of compound transitions is applied, consisting of multiple segments that are connected by means of pseudo-states. However, with the default behaviour of Xtext, a transition can cross state boundaries. Therefore, the scope provider is customized to restrict that, and provide the desired behavior in conformance with UML-RT concepts, by allowing transitions to only cross-reference pseudo states and states that are on the same level of nesting as the transition, or their immediate entry and exit points.

8.3.4 Integration in RTist

By customizing the Xtext ASTFactory and Linker it becomes possible to map the syntax to the existing environment for UML-RT modeling in RTist. This means that RTist treats a textual state-machine in exactly the same way as a graphical state-machine. The state-machine AST created by the Xtext parser must be inserted into the proper model context, which is the capsule to which the state-machine belongs. The Xtext parser recreates the AST every time the state-machine textual model is modified (or to be more accurate, a short time after each consecutive sequence of text modifications). Each newly created AST will contain cross-references that target elements both within the AST itself as well as elements contained in the RTist model. As soon as the AST is inserted into a capsule, RTist will treat this state-machine in the same way as a state-machine that was modeled graphically (since the model representation is identical).

8.4 Challenges

There are some challenges involved in synchronizing changes across graphical and textual state-machine models. These challenges are identified during the process of integrating the textual notation in RTist and are described in the following sections.

8.4.1 Incoming cross-references

The user may create cross-references that target elements in the state-machine. For example, she may create a dependency from a capsule to one of the states in the state-machine. If attention is not paid when inserting the state-machine into the capsule, such cross-references could break. For instance, if the state-machine is updated by simply deleting the old version and then inserting the new updated version this will happen, since the deletion will trigger clean-up of cross-references (i.e., the tool "thinks" that the state was deleted and hence resets the dependency to avoid a broken reference).

8.4.2 Model information not represented in the textual notation

The textual state-machine notation does not cover the entire UML-RT meta-model. That is, there are certain pieces of information that cannot be specified using the textual syntax, but which other views in RTist may allow to view and edit. One example is UML stereotypes which may be applied to any model element, including elements in a state-machine. When the state-machine model gets updated due to a change in the textual notation it is important not to lose this additional information. Both these problems are related to how the state-machine graphical model is updated when the textual model changes. The solution is to "merge" the changed parsed AST into the RTist graphical model, rather than replacing it (i.e., updating it by a delete followed by an insert).

8.4.3 Model element unique identifiers

It is important to ensure that URIs of elements in the AST are stable. More specifically, the last part of an EMF URI, the so called fragment, which identifies the element within its file, shall remain the same. RTist uses by default random unique IDs as fragments, but this obviously does not work for AST elements, since it would mean that all AST elements get new URIs each time the textual state-machine is modified (and hence it becomes impossible to keep incoming cross-references bound). To solve this, an Xtext fragment provider was implemented. Its job is to assign fragments using fully qualified names instead. The alternative would have been to make the fragment strings visible in the textual notation, which is obviously not an option from a user-friendliness point of view.

8.4.4 Code formatting and comments

Just like it is possible to have additional information in the UML-RT state-machine graphical model that is not carried by its textual counterpart, the opposite is also possible. A textual state-machine model may have lexical entities that the parser would not reflect in the AST. Typical examples include code formatting (i.e., indentations, use of newlines etc) and comments. If a state-machine is modified in another way than through the textual editor, it is necessary to serialize the updated model and then update the textual model in a way that preserves code formatting and comments. This problem is not fully solvable in an automated manner since there usually is no formalized means for how code formatting and comments are used. Xtext provides an approach for serialization that attempts the preservation of as much code formatting and comments as possible. If the user has experienced how this algorithm works in practice, she could probably adjust the code formatting and use of comments to avoid losing information when the model is serialized. Anyhow, we will look into potential semi-automated solutions for this.

8.5 Related work

The Action Language for Foundational UML (Alf) [10] is a textual language standardized by the Object Management Group (OMG) for representing UML models. Since its underlying semantics are indicated by a limited subset of UML named Foundational UML (fUML), the Alf syntax is restricted within its bounds and does not support state-machines as they are not available in the fUML subset. tUML is a textual language for a limited subset of the standard UML metamodel targeted at real-time embedded systems that consists of class diagrams, composite structure diagrams, and state diagrams. The implementation of tUML has been carried out to have a very close proximity to the UML metamodel. Consideration has been given to propose tUML to OMG as an extension of Alf, being that the latter lacks support for state-machines [11]. There also exists a plethora of tools and modeling languages that support textual notations for UML models. Earl Grey [12] is a textual modeling language that supports the creation of UML class and state models. MetaUML [13] is a MetaPost library for creating UML diagrams using textual notations, and it supports class diagrams, package diagrams, component diagrams, use case diagrams, activity diagrams, and state diagrams. The textual notation is not only used to define the elements and their relationships but also their layout properties. PlantUML [14] is an open-source tool that supports the generation of both UML and non-UML diagrams from a textual language. Among the most important UML diagrams they support are sequence diagrams, class diagrams, activity diagrams, state diagrams, and more. Umple [15] is an open-source modeling tool that can be used to add UML abstractions to programming languages (i.e., Java, C++, PHP, and Ruby) and create UML class and state diagrams from a textual notation. The generated graphical view for class diagrams can be edited, while for state-machines, it is read-only. Textual, executable, and translatable UML (txtUML) [16] is an open-source project that supports the creation of models from a textual notation and generates the corresponding graphical visualization. TextUML Toolkit [17] is an open-source IDE that allows the creation of UML2 models from a textual notation. This toolkit is available on Cloudfier, as a plug-in for Eclipse IDE and as a standalone command-line tool.

There have been a handful of attempts at providing textual syntax for UML-RT, and we have been involved with some of them. Calur [18] provides a textual syntax only for UML-RT's action language, not state-machines. Unlike our approach, both eTrice[7] and Papyrus-RT[8] provide a kind of all-or-nothing approach. They both provide syntax for both structure and behaviour, but the entire model is described as either textual or graphical, whereas in our approach the user can select only parts of the model to be represented textually. This allows the user to retain the ability to use existing RTist tooling for graphical modeling. Note also that our textual notation for UML-RT state-machines has been designed and implemented to maximise user experience of architects and engineers, as their throughput thanks to the possibility of blended modeling.

8.6 Outlook

The provision of textual modeling as complement to the existing graphical modeling for UML-RT state-machines is the first step towards a fully blended modeling environment for UML-RT in RTist. The next planned steps concern the extension of the textual notation to the rest of UML-RT concepts as well as the provision of more effective synchronisation mechanisms. Moreover, we plan to investigate the possibility to leverage the Language Server Protocol, and in particular the Graphical LSP in Eclipse[9] to provide a web-based modeling solution too. In addition, we intend to implement this approach using other

[7] https://www.eclipse.org/etrice/
[8] https://www.eclipse.org/papyrus-rt/
[9] https://www.eclipse.org/glsp/

graphical modeling tools (e.g., Sirius, GMF) and compare the results.

The involved architects at HCL have confirmed a satisfactory result achieved via the textual notation, and they were involved in the effort from start to end. Nevertheless, to provide a quantification of the improvements brought by blended modeling in RTist, we plan to run additional experiments with a larger number of stakeholders from multiple companies, including RTist's users. We have built an international consortium across 4 countries and running a project in the ITEA3 cluster programme on blended graphical–textual modeling called BUMBLE[10]. In that context, we will run more extensive controlled experiments and industrial case-studies too. An important element of the dissemination plan consists in leveraging the different opportunities provided in the Eclipse community, including Eclipse conferences (e.g., EclipseCon Europe) and marketing. We will also collaborate with the Eclipse Working Groups, Papyrus and Capella Industry Consortia to reach out to industrial MDE tool users. We plan to disseminate results via research forums (conferences, workshops), corporate presentations, participation to industrial events like expos, on-line community forums for Eclipse, social media, fact sheets and wikis.

[10] https://itea3.org/project/bumble.html

Bibliography

[1] Gunter Mussbacher, Daniel Amyot, Ruth Breu, Jean-Michel Bruel, Betty HC Cheng, Philippe Collet, Benoit Combemale, Robert B France, Rogardt Heldal, and James Hill. The relevance of model-driven engineering thirty years from now. In *Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28–October 3, 2014. Proceedings 17*, pages 183–200. Springer, 2014.

[2] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, 2012.

[3] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 471–480, 2011.

[4] Bran Selic. Real-time object-oriented modeling. *IFAC Proceedings Volumes*, 29(5):1–6, 1996.

[5] Lorenzo Addazi and Federico Ciccozzi. Blended graphical and textual modeling for UML profiles: A proof-of-concept implementation and experiment. *Journal of Systems and Software*, 175:110912, 2021.

[6] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended modelling – what, why and how. In *MPM4CPS workshop*, September 2019.

[7] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. Multi-view approaches for software and system modelling: A systematic literature review. *Software & Systems Modeling*, 2019.

[8] Ernesto Posse and Juergen Dingel. An executable formal semantics for UML-RT. *Software and Systems Modeling*, 15(1):179–217, 2016.

[9] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[10] Object Management Group (OMG). Action Language for Foundational UML (Alf), Version 1.1. OMG Document Number formal/2017-07-04. http://www.omg.org/spec/ALF/1.1, 2017.

[11] Frédéric Jouault and Jérôme Delatour. Towards fixing sketchy UML models by leveraging textual notations: Application to real-time embedded systems. In *OCL@ MoDELS*, pages 73–82, 2014.

[12] Martin Mazanec and Ondrej Macek. On general-purpose textual modeling languages. In *Dateso*, volume 12, pages 1–12. Citeseer, 2012.

[13] Ovidiu Gheorghies. MetaUML: Tutorial, reference and test suite. https://profs.info.uaic.ro/~ogh/files/main.pdf, 2005. Retrieved: 02/05/2021.

[14] PlantUML Language Reference Guide, Version 1.2021.2. http://plantuml.com/guide. Retrieved: 02/05/2021.

[15] Timothy C Lethbridge, Vahdat Abdelzad, Mahmoud Husseini Orabi, Ahmed Husseini Orabi, and Opeyemi Adesina. Merging modeling and programming using Umple. In *International Symposium on Leveraging Applications of Formal Methods*, pages 187–197. Springer, 2016.

[16] Gergely Dévai, Gábor Ferenc Kovács, and Ádám An. Textual, executable, translatable UML. In *OCL@ MoDELS*, pages 3–12. Citeseer, 2014.

[17] TextUML Toolkit. http://abstratt.github.io/textuml/, 2021. Retrieved: 02/05/2021.

[18] Nicolas Hili, Ernesto Posse, and Juergen Dingel. Calur: an action language for UML-RT. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

Chapter 9

Paper C: Blended graphical and textual modeling of UML-RT state-machines: An industrial experience

Malvina Latifaj, Federico Ciccozzi, Muhammad Waseem Anwar, Mattias Mohlin.
Published in the invitation-only post-proceedings of the 15th European Conference on Software Architecture (ECSA 2021).

Abstract

The ever increasing complexity of modern software systems requires engineers to constantly raise the level of abstraction at which they operate to suppress the excessive complex details of real systems and develop efficient architectures. Model Driven Engineering has emerged as a paradigm that enables not only abstraction but also automation. UML, an industry de-facto standard for modeling software systems, has established itself as a diagram-based modeling language. However, focusing on only one specific notation limits human communication and the pool of available engineering tools. The results of our prior experiments support this claim and promote the seamless use of multiple notations to develop and manipulate models. In this paper we detail our efforts on the provision of a fully blended (i.e., graphical and textual) modeling environment for UML-RT state-machines in an industrial context. We report on the definition of a textual syntax and advanced textual editing for UML-RT state-machines as well as the provision of synchronization mechanisms between graphical and textual editors.

9.1 Introduction

The complexity of software systems has been growing at an unbelievable pace for decades now. Relying merely on human efforts to develop high-quality software is presently regarded as a futile attempt. It can be argued that in the face of this complexity, without an efficient and effective architecture, software is inscrutable [1]. To tackle the architectural complexity of software development, Model Driven Engineering (MDE) has emerged as a software engineering paradigm that focuses on raising the level of abstraction when architecting software systems [2, 3]. This is done by promoting modeling languages and models, which are closer to human understanding, instead of code, closer to machines, as core architectural and engineering artefacts. This approach imposes limits to the problem-domain, facilitates the identification of relevant abstractions, and avoids superfluousness. Together with that, MDE pledges automation by exploiting modeling technologies that among others, enable the generation of fully fledged code from architectural models. Domain-specific abstractions facilitating the architectural description of software systems are defined using formal specifications expressed in Domain Specific Modeling Languages (DSMLs), which capture the core aspects of a domain, thus promoting productivity, efficiency and comprehensibility of domain-specific problems. UML is the most used architecture description language in industry [4], the de-facto modeling standard in industry [5], and an ISO/IEC (19505-1:2012) standard. It is general-purpose, but it provides powerful profiling mechanisms to constrain and extend the language to achieve UML-based DSMLS, called UML profiles; in this paper, we focus on the UML real-time profile (UML-RT)[6], as this is the profile implemented in the commercial tool HCL RTist[1] of our industrial partner. We also leverage an open-source implementation of it provided in the Eclipse Papyrus-RT[2] tool.

9.1.1 Problem, motivation, and the RTist case

Domain-specific modeling tools, like RTist, traditionally focus on one specific editing notation (such as text, diagrams, tables or forms). This limits human

[1] `https://www.hcltechsw.com/rtist`
[2] `https://www.eclipse.org/papyrus-rt/.`

communication, especially across stakeholders with varying roles and expertise. Moreover, architects and engineers may have different notation preferences; not supporting multiple notations negatively affects their throughput. Besides the limits on communication, choosing one particular kind of notation has the drawback of limiting the pool of available tools to develop and manipulate models that may be needed. For example, choosing a graphical representation limits the usability of text manipulation tools such as text-based diff/merge, which is essential for team collaboration. When tools provide support for both graphical and textual modeling, it is mostly done in a mutual exclusive manner. Most off-the-shelf UML modeling tools, such as IBM Rational Software Architect[3] or Sparx Systems Enterprise Architect[4], focus on graphical editing features and do not allow seamless graphical–textual editing. This mutual exclusion suffices the needs of developing small-scale applications with only very few stakeholder types. RTist is not an exception. It provides support for modeling UML-RT architectures and applications based on graphical *composite structure diagrams*, to model structure, and *state-machine diagrams*, to model behavior. In addition, the implementation of UML-RT in RTist provides support for leveraging C/C++ action code for the description of fine-grained, algorithmic, behaviors within graphical state-machines. That is needed to enable the definition of full-fledged UML-RT models from which executable code can be automatically generated. While providing means to model graphical entities and "program" algorithmic behaviours textually, the two are disjoint, since the modeling of UML-RT is graphical only and the textual C/C++ is injected in graphical models as a "foreign" entity and with almost no overlapping with graphical model elements. The aim is instead to achieve a modeling tool that is able to make different stakeholders to work on overlapping parts of the models using different modeling notations (e.g., graphical and textual) in an automated manner.

9.1.2 Paper contribution

In this paper we describe our proposed solution for providing a fully blended graphical-textual modeling environment for UML-RT state-machines in an indus-

[3] http://www-03.ibm.com/software/products/en/ratsadesigner/

[4] https://sparxsystems.com/

trial setting. Our experiments in a previous study with blended graphical-textual modeling showed that the seamless use of different notations can significantly boost the architecting of software using UML profiles [7]. The results of those experiments together with the exposed wish of RTist customers of being able to design software via multiple notations led us to initiate this work towards an automated support for blended modeling of UML-RT in RTist. In a prior work [8], we describe the effort of designing, implementing and integrating a textual notation for UML-RT state machines in RTist. In this paper, we extend that work, and address the problem formulated in the previous section by providing the following additional research contributions.

C1. Definition of a 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.

C2. Provision of synchronization mechanism between textual and graphical notations to achieve a seamless blended modeling environment and validation of the solution.

9.1.3 Paper outline

The remainder of the paper is organized as follows. In Section 9.2 we describe the concept of blended modeling and in Section 9.3 we detail the design of our proposed solution. The implementation details of the solution are presented in Section 9.4, whereas the validation is discussed in Section 9.5. The related works are detailed in Section 9.6 and the paper is concluded in Section 9.7 with a brief summary and an overview of the current and upcoming enhancements to the overall blended modeling approach.

9.2 Blended modelling: what and why

We have previously defined the notion of *blended modeling* [9] as:

> *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.

A seamless blended modeling environment, which allows stakeholders to freely choose and switch between graphical and textual notations, can greatly contribute to increase productivity as well as decrease costs and time to market. Such an environment is expected to support at least graphical and textual modeling notations in parallel as well as properly manage synchronisation to ensure consistency among the two. The possibility to visualise and edit the same information through a set of diverse perspectives always in sync has the potential to greatly boost communication between stakeholders, who can freely select their preferred notation or switch from one to the other at any time. Besides obvious notation-specific benefits, such as for instance, the possibility to edit textual models in any textual editor outside the modeling environment, a blended framework would disclose the following overall benefits.

Flexible separation of concerns and better communication. Providing graphical and textual modeling editors for different aspects and sub-parts (even overlapping) of a DSML like UML-RT enables the definition of concern-specific architectural views characterised by either graphical or textual modeling (or both). These views can interact with each other and are tailored to the needs of their intended stakeholders. Due to the multi-domain nature of modern software systems (e.g., cyber-physical systems, Internet-of-Things), this represents a necessary feature to allow different domain experts to describe specific parts of a system using their own domain-specific vocabulary and notation, in a so called *multi-view modeling* [10] fashion. The same information can then be rendered and visualised through other notations in other perspectives to maximise understanding and boost communication between experts from different domains as well as other stakeholders in the development process.

Faster modeling activities. We have experimented with blended modeling of UML profiles [7] and the seamless combination of graphical and textual modeling has shown a decreased modeling effort in terms of time thanks to the following two factors:

1. Any stakeholder can choose the notation that better fits his/her needs,

personal preference, or the purpose of the current modeling task, at *any time*. For instance, while structural model details can be faster to describe by using diagrammatic notations, complex algorithmic model behaviours are usually easier and faster to describe using textual notations (e.g., Java-like action languages).

2. Text-based editing operations on graphical models[5], such as copy and paste and regex search and replace, syntax highlighting, code completion, quick fixes, cross referencing, recovery of corrupted artefacts, text-based diff and merge for versioning and configuration, are just few of the features offered by modern textual editors. These would correspond to very complex operations if performed through graphical editors; thereby, most of them are currently not available for diagrams. Seamless blended modeling would enable the use of these features on graphically-described models through their textual editing view. These would dramatically simplify complex model changes; an example could be restructuring of a hierarchical state-machine by moving the insides of a hierarchical state. This is a demanding re-modeling task in terms of time and effort if done at graphical level, but it becomes a matter of a few clicks (copy and paste) if done at textual level.

9.3 Design solution

In this section, we detail the solution design, illustrated in Fig. 9.1, for the provision of a blended modeling environment for UML-RT state-machines. Note that in order to maximise accessibility to our solution, we describe the solution for an open-source tool, Eclipse Papyrus-RT, which is orthogonal to the one in RTist (which also is Eclipse EMF-based). The starting point is the already existing Ecore-based DSML formalizing the UML-RT profile (i.e., MMG), which is utilized to instantiate graphical models (i.e., MG) in both Papyrus-RT and RTist. Using this DSML as blueprint, we define a textual language (i.e., MMT)

[5]Please note that by *graphical/textual model*, we intend a model rendered using a graphical/-textual notation.

in Xtext[6] that will be used to instantiate textual models (i.e., M_T) of UML-RT state-machines. Moreover, using Xtext's formatting APIs, we also customize the textual editor to preserve essential textual information, such as lines, formatting and hidden regions like comments. This provides our first contribution **C1**. Subsequently, we design and implement the synchronization mechanisms between the two notations by model-to-model (M2M) transformations [11]. These transformations are defined on the basis of implicit mappings between metaelements of the source and target metamodels and implemented in terms of the operational version of the Query/View/Transformation language (QVTo[7]) in Eclipse. QVTo supports only unidirectional transformations, thus, to achieve bidirectionality, we defined two unidirectional transformations; MM_T2MM_G, where the source metamodel is MM_T and target metamodel is MM_G, and MM_G2MM_T, where the source metamodel is MM_G and the target metamodel is MM_T. Both model transformations are horizontal as the source and target model reside in the same abstraction level, and exogenous as the models are expressed in different modeling languages. This makes for our second contribution **C2**. Further details on the definition of the textual syntax and synchronization mechanisms can be found in Section 9.3.1 and 9.3.2. The implementation details are included in Section 9.4, and the validation of the solution is detailed in Section 9.5.

9.3.1 Textual notation for UML-RT state-machines

Textual language workbench. To complement the existing graphical editor in RTist with a textual notation and editor, a suitable language workbench needs to be carefully selected. HCL RTist and Papyrus-RT are Eclipse-based environments that leverage the Eclipse Modeling Framework (EMF)[8] as backbone. Thereby, by choosing an EMF-based language workbench, we could leverage EMF as a common data layer. For this reason, we chose Xtext, a framework for the development of textual DSMLs, based on EBNF grammars. The textual editor supports an outline view, syntax highlighting, error checking, quick-fix proposals, and many other features provided by Xtext. Furthermore, Xtext

[6] https://www.eclipse.org/Xtext/
[7] https://wiki.eclipse.org/QVTo
[8] https://www.eclipse.org/modeling/emf/

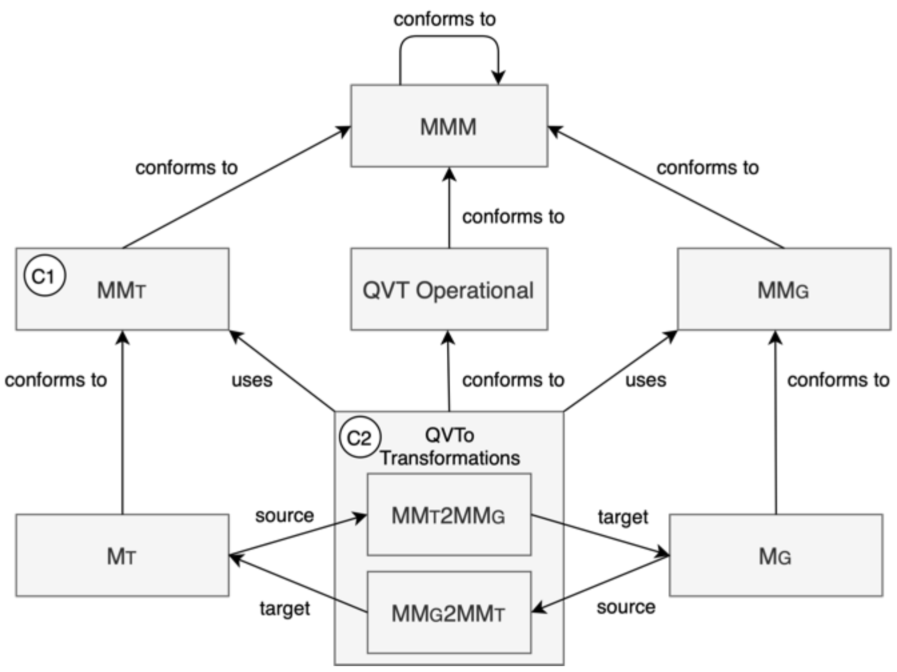


Figure 9.1. Synchronization solution design

provides code completion for keywords and cross-references by increasing the usability of the language and decreasing the learning curve.

Textual notation definition. Our goal was to introduce a textual notation (and related editor) to the already existing UML-RT profile supported by RTist. A possible alternative was to use the underlying metamodel consumed by the RTist's graphical editor as an input for Xtext to automatically generate a textual editor. However, although easy to implement, this solution generates erroneous and unintuitive grammar, too far from the expectations of RTist's architects and customers. Manually editing this generated grammar would have been a tedious and potentially error-prone process. Therefore, we decided to design a textual notation in terms of an Xtext grammar, from scratch. Starting from a wish-list of RTist's customers and architects, and using the UML-RT metamodel portion describing state-machines as blueprint, we manually defined our UML-RT textual notation for state-machines in Xtext. The steps needed for the definition of the grammar were the following.

① *Identify reserved keywords:* When defining a DSML, it is crucial to identify the reserved keywords used to typify the core concepts of the language. The importance of these keywords lies in improved readability, higher language familiarity, and efficient parsing as they serve as directives for specific concepts. The chosen keywords for the textual syntax for UML-RT state-machines are the following: *capsule, statemachine, state, initial, junction, choice, entry, exit, entrypoint, exitpoint, history, transition, when, on* and *inherits*. A more detailed description of the concepts represented by each keyword can be found in the official documentation [9] of UML-RT by HCL.

② *Elements' ordering strategy:* Even though it is not mandatory for our language to have a fixed order of elements, this approach enhances readability and navigation of the textual syntax, as well as increased predictability on where the elements created by using the graphical notation will be placed in the textual syntax. Our grammar is based on the vertical distance approach where elements that affect each other's understandability and are closely related [12], are grouped together and have a low vertical distance. Furthermore, being that this grammar prohibits cross-references before element declaration, we take the aforementioned statement into consideration and make sure that elements that need to be cross-referenced will be declared before the cross-reference occurs.

③ *A spoonful of syntactic sugar:* The majority of programming languages, including C++, which is used as action code for behavioral state-machines, makes use of statement terminators in the form of semi-colons. Being that one of the main goals when introducing this textual syntax is for developers to use it jointly with the C++ action code, we introduced consistent use of semi-colons for indicating statement termination to make the grammar more conforming to C++ and to increase readability. For the same readability reasons and developers'

[9] https://rtist.hcldoc.com/help/topic/com.ibm.xtools.rsarte.webdoc/pdf/RTist%20\Concepts.pdf

preferences, we also introduce colons after transition names. Furthermore, to make the grammar more compact, we allow the declaration of multiple objects of the same type in one single line of code. Due to the combination of the textual syntax with action code, we need to handle C++ code blocks so we can “isolate” them and make them distinguishable from the rest of the grammar. For this reason, we include back-ticks in order to enclose code snippets and to make the lexer aware of where the code block begins and ends. The overall goal during this process was to keep a fixed concrete syntax while simultaneously enhancing the abstract syntax, even though frequently we had to trade-off between ease of expression in the concrete syntax and extra complexity in the abstract syntax.

Enforcing UML-RT’s modularity. Scoping in Xtext is concerned with the visibility of elements; therefore, the scope provider computation returns the target candidates that are visible in the current context and by a given reference. In order to enforce the UML-RT’s modularity, it is necessary to specify a custom scope provider. The default behavior of Xtext allows establishing a cross-reference to all the elements of a particular type that are located inside the same Eclipse resource (i.e., project). By customizing the scope provider, we restrict this behaviour, and only allow cross-references for elements declared in the same model file. The rationale behind this decision lies in the fact that multiple model files containing different capsules can be located inside the same resource, and a particular capsule should not be able to cross-reference the elements of other capsules. However, a key concept in which UML-RT relies on to reuse and extend parts of existing state-machines is the inheritance mechanism. When capsule A inherits capsule B, the state-machine of capsule A implicitly inherits the state-machine of capsule B. Therefore, to support inheritance, we need to customize the scope provider so that it allows cross-references for elements not only from the capsule itself, but also from the inherited capsule, in case there is one. Another default behavior of Xtext consists in allowing cross-references for all elements of a particular type declared in the same model file, regardless of their level of nesting. This contradicts an important UML-RT concept; compound transitions. Since transitions in UML-RT state-machines can not cross state boundaries, the concept of compound transitions is applied, consisting of multiple segments that are connected by means of pseudo-states.

However, with the default behaviour of Xtext, a transition can cross state boundaries. Therefore, the scope provider is customized to restrict that and provide the desired behavior in conformance with UML-RT concepts by allowing transitions to only cross-reference pseudo states and states that are on the same level of nesting as the transition, or their immediate entry and exit points.

Advanced textual editing features. The aforementioned steps provide a solid platform for developing a sophisticated editor for the specification of textual state-machines. Besides the editing features provided out-of-the-box by Xtext for textual languages, we incorporated formatting features like text indentation and syntax highlighting in the textual editor to simplify the specification of these textual models. Furthermore, the support to associate both single and multiline comments within textual specifications is provided too.

9.3.2 Synchronization transformations

Model transformation language. For model transformations, we chose QVTo, which is an implementation of the Operational Mapping Language defined by Object Management Group's (OMG's) Meta-Object Facility (MOF) 2.0 Query/View/Transformation (QVT[10]). The reasons behind this choice were first of all the fact that QVT is a MOF standard, and since our focus is on MOF languages, a transformation language also based on MOF is preferred. In addition, QVTo brings together benefits from both declarative and imperative QVT and it is very well-suited for both exogenous and endogenous transformations, also in-place.

Transformation structure. The transformations are executed in Eclipse QVTo, the only actively maintained QVTo implementation, adhering to its default structure composed of the modeltype declarations, the transformation declaration, main function and mapping operations. In the following, we detail the QVTo structure.

- *Modeltype declaration*: The modeltype declaration in QVTo serves as a reference to the metamodels that will be used for the transformation. When declaring the modeltype, it is obligatory to define the name and

[10]https://www.omg.org/spec/QVT/1.3/About-QVT/

reference. The latter can be specified either by using the package `nsURI` or file location `URI`. In our use case, we reference the metamodels via the `nsURI` which is resolved via the global package registry in the running platform instance. Optionally, the modeltype definition can include the conformance kind (i.e., strict or effective) and a set of constraint expressions (i.e., OCL expressions). In our case we do not define the conformance kind, thus by default it assumes an effective conformance. The rationale behind this decision is to allow the transformations to be applied to similar metamodels. Moreover, we do not define any constraint expressions as we have no additional restrictions over the set of the involved models. As an example, the modeltype definition that references MM$_T$, is detailed in the following.

modeltype *MM$_T$* ***uses*** *MM$_T$_Package_Name (MM$_T$_Package_nsURI)*

- *Transformation declaration*: The transformation declaration defines the name of the unidirectional transformation and specifies the involved metamodels. Additionally it details the direction kind of the transformation via the following values; `in`, `out`, and `inout`. As an example, the MM$_T$2MM$_G$ transformation declaration has the following structure that details the name of the transformation, the involved metamodels, and the direction of the transformation.

  transformation *MM$_T$2MM$_G$ (**in** source:MM$_T$, **out** target:MM$_G$)*;

- *Main function*: The main function is also referred to as the entry point of the transformation as it initiates the execution of the transformation by executing the operations defined in the body of the function. As an example, for the MM$_T$2MM$_G$ transformation, the defined operation selects the root metalelements (i.e., metaelements at the highest level) of MM$_T$, and filters out the `StateMachine` metaelement. Additionally, it invokes the `SM2SM()` top-level mapping rule that maps the `StateMachine` metaelement of MM$_T$ to the `StateMachine` metaelement of MM$_G$.

```
main() {
src.rootObjects()[MMT::StateMachine] -> map SM2SM();
}
```

- *Mappings:* The transformations in QVTo are executed by means of mapping operations. Each mapping operation consists of a signature, an optional guard (i.e., *when* clause), a mapping body, and an optional post condition (i.e., *where* clause). The signature of the mapping operation minimally includes the following elements:
  Mapping Type: A mapping operation can either be an abstract mapping or a concrete mapping (non-abstract). An abstract mapping operation is distinguished by the `abstract` keyword which indicates that the mapping can not be invoked in isolation. Such mapping operations are common when the target metaelement is abstract and are usually inherited by other mappings with concrete target types.
  Metaelements: QVTo does not strictly require the fully qualified name of the metaelements that are to be mapped (i.e, metamodelName :: metaelementName), but in the presence of source and target metamodels that contain similar concepts, the fully qualified name is used to resolve possible ambiguities.
  Mapping Name: Serves to identify the mapping and it is always unique.
  As an example of a mapping signature, in the following we detail an abstract mapping between the concrete `State` metaelement of MMT and the abstract `State` metaelement of MMG, where we use the fully qualified name to separate them from one another.

  ```
  abstract mapping MMT::State::State2State() : MMG::State { ... }
  ```

  Moreover, a mapping declaration can include mapping guards described with OCL expressions and distinguished by the `when` keyword. If the guard evaluates to true, it restricts the execution of the mapping operation only to a subset of elements; alternatively, the mapping operation is not invoked.

Finally, the body of the mapping operation is populated by assigning EReferences and EAttributes of the source metaelement to corresponding EReferences and EAttributes of the target metaelement. For EReferences, a type-dependent mapping operation is invoked by using the `map` keyword. When invoking the mapping on a single element, the element is followed by a dot which precedes the `map keyword`. Alternatively, when invoking the mapping on a collection of elements (i.e., Set, Bag, Sequence, or OrderedSet), the latter is followed by an arrow which precedes the `map keyword`. Moreover, the `self` and `result` variables, refer to the source and target metaelements, respectively. As an example, in the following we detail a regular mapping between `State` and `CompositeState` that is extended with a mapping condition.

```
mapping MMT::State::State2CMPState() : MMG::CompositeState
when {not(self.states -> isEmpty() ...)}
{
result.choicePoints := self.choice.map Choice2Choice();
result.name := self.name; }
```

To conclude the definition of the synchronization transformations, we detail two additional QVTo concepts that we used for this purpose: `inherits` and `disjuncts`. Inheritance enables the reuse of other mapping operations with the condition that the signature of the mapping which is inheriting must conform to the signature of the mapping that is being inherited. In short, the source and target metaelements of the inheriting mapping operation must either be the same or subtypes of the source and target metaelements of the inherited mapping, respectively. This feature allows for a more compact code and increased readability as the operations are defined once in the inherited mapping and reused in each inheriting mapping. In the following, we provide an example of these mapping operations. Mapping operation `State2SimpleState` inherits mapping operation `State2State`. The signatures are conformant as the source metaclasses are the same, while the target metaclass of the inheriting mapping (i.e., `State2SimpleState`), is a subtype of the output metaclass of the inherited mapping (i.e., `State2State`). By inheriting this mapping, in the `State2SimpleState` mapping operation, we do not need to rewrite what is already defined in the `State2State` mapping operation, as it is automatically invoked when the `State2SimpleState` mapping operation is executed.

mapping *MMT::State::State2SimpleState() : MMG::SimpleState*
inherits *MMT::State::State2State { ... }*

Moreover, mapping operations can be defined as disjunctions of multiple other mappings, which are then extended with distinct guards. When invoking such operation, the guards of the mapping operation alternatives specified after the `disjuncts` keyword are sequentially checked. When the first guard evaluates to `true`, the corresponding mapping operation is invoked; alternatively if they all evaluate to `false` it returns `null`. The body of such mapping operations is always empty, because that part of the code is unreachable. This concept is primarily applied to operations transforming abstract types that are extended by multiple subtypes. In this case, the alternative mapping specified after the `disjunct` keyword, consists of subtypes of the original mapping.

mapping *MMT::State::StateDisjunct() : MMG::State*
disjuncts *MMT::State::State2SimpleState,*
MMT::State::State2CompositeState { }

9.4 Implementation

In this section we present the implementation details of the proposed solution and show examples both of the textual syntax and model instances after applying the model transformations for synchronization.

9.4.1 Textual language and editor for UML-RT

Based on the aforementioned approach (see Section 9.3.1), in this section we detail the implementation specifics of the textual language and editor in Eclipse Xtext. We focus particularly on the customization of the scope provider in Xtext to enable inheritance and compound transitions in our textual UML-RT, as well as on the customization of the Xtext's formatter for advanced textual editing and formatting features.

Customization of the scope provider.

Listing 9.1 provides a snippet of the customized scope provider in Xtext for supporting the concept of inheritance. The conditional `if` statement in Line 1 checks whether the current capsule inherits another capsule. If this condition evaluates to true, the `EObject` is down casted to a `Capsule` object in Line 3. Depending on the instance type of the context's container, the desired elements of the inherited capsule are added to the list of eligible candidates that the scope provider will return as detailed in Lines 4-6.

```
1 if (rootCapsule.getSuperclass() != null) {
2   parentinheritance = rootCapsule.getSuperclass();
3   Capsule inheritedCapsule = (Capsule) parentinheritance;
4   if (context.eContainer() instanceof StateMachine) {
5   transitionfrom.addAll(inheritedCapsule.getStatemachines().getStates()
      );
6   ...
7   }
8 }
```

Listing 9.1. Inherited capsule scope provider

Listing 9.2 provides instead a snippet of the customized scope provider in Xtext for supporting the concept of compound transitions. The `eContainer()` method in Line 2 is used to return the containing object of the context object. The list of objects `T_F` in Line 3, is initialized to be used for storing all the eligible candidates that can be cross-referenced. The block of code to be executed if the specified condition of the `if` statement in Line 4 evaluates to true, down casts the `currentParent` EObject into a `StateMachine` object and uses the `addAll()` method to add all elements of a specific type that are contained in the state-machine as the context element, to the list.

```
1 else if (reference == HclScopingPackage.Literals.TRANSITION__FROM) {
2   EObject currentParent = context.eContainer();
3   List<EObject> T_F = new ArrayList<>();
4   if (currentParent instanceof StateMachine) {
5     StateMachine s = (StateMachine) currentParent;
```

```
    T_F.addAll(s.getStates());
    ....
    for (State states : rootCapsule.getStatemachines().getStates()) {
          transitionfrom.addAll(states.getEntrypoint());
          transitionfrom.addAll(states.getExitpoint());
    }
  }
return Scopes.scopeFor(T_F, N_C, IScope.NULLSCOPE);
```

Listing 9.2. Compound transitions scope provider

Customization of the textual editor

Parsing and serialization are two major concepts in Xtext associated with the textual model and Abstract Syntax Tree (AST), respectively. The instance of a grammar in the editor, technically referred to as XtextResource, is represented through a textual model. The equivalent AST is generated from the textual model through the parser. On the other hand, the serializer converts the AST into the equivalent textual model. The conversions between textual model to AST and vice versa are very frequent and, therefore, Xtext supports the exploitation of built-in APIs to customize certain functionalities that may be required before or after the conversions from one to the other. We exploited the built-in APIs for customizing our textual editor.

The synchronizations targeted in our solution between the textual and graphical models lead to frequent changes in the AST related to the textual model, like deletion or addition of textual elements. In this case, the line numbers and other hidden region elements like comments need to be updated in the textual editor. Furthermore, the formatting of the text needs to be preserved in the textual editor after synchronizations since it brings along semantic information in most cases. To achieve this, we utilized Xtext's formatting infrastructure. In particular, we extended the `AbstractFormatter2` class to implement a customized state-machine formatter, composed of two core functions (i.e., *Lines* and *Hidden Regions*). The *Lines* function updates the sequence of lines in the textual editor according to the synchronized changes to the AST. *Hidden Regions* instead preserves the place of hidden regions that group all hidden tokens (e.g., whitespace, newlines, tabs and comments) between two semantic tokens upon changes to the AST.

9.4.2 Synchronization

Based on the aforementioned approach (see Section 9.3.2), in this section, we detail the implementation specifics of the synchronization model transformations in Eclipse QVTo. Synchronization mechanisms are provided in terms of two unidirectional M2M transformations; MMT2MMG and MMG2MMT. The majority of metaelements between the two metamodels require a one-to-one mapping, thus the mapping rules are rather straightforward. In the following, for each model transformation, we discuss the mapping operations that highlight a few interesting and less simple cases.

Textual to graphical synchonization – MMT2MMG

- The `StateMachine` metaelement behaves as a root element both in MMT and MMG. Nevertheless, there is a notable difference between the two. In MMT, `StateMachine` has multiple children and its containment of elements `State` has a zero-to-many (0..*) cardinality. Instead, in MMG, `StateMachine` has a one-to-one (1..1) cardinality to `CompositeState`. In short, whilst in MMT `StateMachine` can contain many `States` as immediate children, in MMG the `StateMachine` can only contain one `CompositeState` as its immediate child, and in turn `CompositeState` would contain the other elements. Consequently, when transforming a `StateMachine` in MMT to a `State Machine` in MMG, two possible narratives need to be taken into account. First, if we consider a model instance of MMT (i.e., MT) and the `StateMachine` of this model instance contains only one `State` and no other immediate children, `State` is transformed to a `CompositeState` in MG (i.e., model instance of MMG) as detailed in Lines 4-5 in Listing 9.3. Otherwise, if the `StateMachine` in MT contains more than one immediate state, when transforming to a `StateMachine` in MG, a `CompositeState` object is created (Lines 9-10) and the immediate children of the `StateMachine` in MT are assigned as immediate children (Line 11) of the `CompositeState` in MG.

```
1 mapping text::StateMachine::SM2SM() : graph::StateMachine{
2 result.name:=self.name;
```

```

if (self.states -> size() = 1 and self.initialtransition ->
    isEmpty() and self.transition -> isEmpty() and self.junction
    -> isEmpty() and self.choice -> isEmpty()) {
  result.top := self.states -> first().map State2CMPState();
}

else {
var cs := object graph::CompositeState{};
top :=cs;
cs.substates := self.states.map toState();
```

Listing 9.3. `StateMachine` to `StateMachine`

- With respect to states, MMT considers only `State` metaclass, while MMG makes a distinction between `SimpleState` and `CompositeState`, which extend the `State` metaclass. Thus, when transforming a `State`, the mapping operations in Lines 1-9 in Listing 9.4 need to be extended with mapping guards that determine if the `State` metaclass will be transformed to a `SimpleState` or `CompositeState`. Moreover, an additional mapping operation is defined in Lines 13-14, which is a disjunction of the aforementioned mapping operations and is invoked in Line 10. Upon its execution, the guards of `State2SimpleState` and `State2CMPState` are checked in a sequential order. For a `State` to be transformed to a `SimpleState`, the `State` should have no children. To evaluate that we use the OCL expression `isEmpty()`, which evaluates whether the collection is empty or not, in Line 3. Alternatively, a `CompositeState` has children, thus in Line 8 we use the OCL expression `notEmpty()`.

```
1 mapping text::State::State2SimpleState() : graph:: SimpleState
2 inherits text::State::State2State
3 when {self.states -> isEmpty() and self.entrypoint -> isEmpty()
      and self.exitpoint -> isEmpty()  and self.junction -> isEmpty
      () and self.choice -> isEmpty() }
4 {  }
5
6 mapping text::State::State2CMPState() : graph:: CompositeState
7 inherits text::State::State2State
8 when {self.states -> notEmpty() or self.entrypoint -> notEmpty()
      or self.exitpoint -> notEmpty()  or self.junction -> notEmpty
      () or self.choice -> notEmpty()}
```

```
{
substates := self.states.map State2StateDisjunct();
...
}
mapping text::State::State2StateDisjunct() : graph::State
disjuncts text::State::State2SimpleState, text::State::State2CMPState
{}
```

Listing 9.4. `State` to `SimpleState` and `CompositeState`

Graphical to textual synchronization – $\mathrm{MM_G 2 MM_T}$

- `SimpleState` and `CompositeState` metaelements in $\mathrm{MM_G}$ both have to be transformed to `State` in $\mathrm{MM_T}$. Hence, two corresponding mapping operations are defined in Line 1 and Line 4 in Listing 9.5. In this particular situation, a disjunctive mapping operation (i.e., `State2StateDisjunct()`) is introduced in Line 10. Contrary to the first two mapping operations where a mapping body is defined, this operation specifies a list of mapping operations (i.e., `SimpleState2State` and `CMPState2State`) which are evaluated when the mapping operation is invoked. The invocation of the operation occurs in Line 7 when trying to map `substates` to `states` as the EType for `substates` is the abstract metaclass `State` which is extended by `SimpleState` and `CompositeState`.

```
1  mapping graph::SimpleState::SimpleState2State() : text::State
2  inherits graph::State::State2State {}
3
4  mapping graph::CompositeState::CMPState2State() : text::State
5  inherits graph::State::State2State
6  {
7      result.states := self.substates -> map State2StateDisjunct();
8      ...
9  }
10 mapping graph::State::State2StateDisjunct() : text::State
11 disjuncts graph::CompositeState::CMPState2State,
12           graph::SimpleState::SimpleState2State{}
```

Listing 9.5. `SimpleState` and `CompositeState` to `State`

- `Transition` metaclass in MMG can be transformed to either `History Transition`, `InitialTransition`, `InternalTransition` or `Transition` in MMT, all extending the `Transitions` metaclass, depending on the source and/or target vertices it connects. First, the `T2Ts()` mapping operation is defined, which details the creation of the `TransitionBody` and the assignment operations for its children. This is then inherited by all other mapping operations that in their signature include `Transition` as source and a subtype of `Transitions` as target. Inheritance eliminates the need to rewrite the operations that are already defined in `T2Ts()` such as the creation of the `TransitionBody` and the corresponding assignments. In addition, all mapping operations that map `Transition` to subtypes of `Transitions` include a mapping guard (i.e., `when` clause) that specifies the type of vertices that the `Transition` must connect, to be transformed to a specific subtype of `Transitions`. The mapping body of these mapping operations details the assignments of the `sourceVertex` and `targetVertex` properties of the source metaelement, to `from` and `to` properties of the target metaelement, respectively as shown in Lines 18-19 in Listing 9.6. The mapping operation that is invoked on them is a disjunction of other mapping operations in which the source and target metaelements are subtypes of the source and target metaelements of the original mapping (i.e., `Vertex2VertexDisjunct`). This is because the types of the source and target metaelements of the invoked mapping (i.e., `Vertex2VertexDisjuncts`) must conform to the types of properties `from` and `to`.

```
1 mapping graph::Vertex::Vertex2VertexDisjunct() : text::Vertex
2 disjuncts graph::CompositeState::CMPState2State, graph::
      SimpleState::SimpleState2State, graph::EntryPoint::
      EntryPoint2EntryPoint, graph::ExitPoint::ExitPoint2ExitPoint,
      graph::ChoicePoint::ChoicePoint2Choice, graph::JunctionPoint
      ::JunctionPoint2Junction{}
3
4 mapping graph::Transition::T2Ts() : text::Transitions
5 {
6     var TransitionBodyObject := object text::TransitionBody{
        transitionguard := self.guard.map Guard2TransitionGuard();
7         ....
8     };
```

```
    transitionbody := TransitionBodyObject;
    result.name := self.name;
}
mapping graph::Transition::T2T() : text::Transition
inherits graph::Transition::T2Ts
when {not(self.sourceVertex.oclIsTypeOf(InitialPoint) or
     self.targetVertex.oclIsTypeOf(DeepHistory) or
     self.sourceVertex = null or self.targetVertex = null)}
{
    result._from := self.sourceVertex.map Vertex2VertexDisjunct();
    result.to := self.targetVertex.map Vertex2VertexDisjunct();
}

mapping graph::Transition::T2HT() : text::HistoryTransition
inherits graph::Transition::T2Ts
when {self.targetVertex.oclIsTypeOf(DeepHistory)} { ...}

mapping graph::Transition::T2INI_T() : text::InitialTransition
inherits graph::Transition::T2Ts
when {self.sourceVertex.oclIsTypeOf(InitialPoint)} {...}

mapping graph::Transition::T2INT_T() : text::InternalTransition
when {self.sourceVertex = null and self.targetVertex = null} {...}
```

Listing 9.6. `Transition` to `Transitions`

- `Trigger` metaclass in MMG can be transformed to either `Trigger`, `PortEvent Trigger`, or `MethodParameterTrigger` in MMT depending on which of the mapping guards (i.e., `when` clause) evaluates to true. Hence, three corresponding mapping operations are defined. The `PortEventTrigger` consists of a `Port` and `Event` separated by a dot (e.g., port.event), thus in order for a `Trigger` in MMG to be transformed to a `PortEventTrigger` in MMT it should match the pattern defined in Line 7 in Listing 9.7. Moreover, the `Port` and `Event` metaclasses in MMT have no correspondence in MMG therefore they should be created as new elements. Lines 9-17 detail the creation of `Port` and `Event` metaclasses and the assignment of the name attribute for each of them. The same procedure is applied to transform `Trigger` in MMG to `MethodParameterTrigger` in MMT and to create the `Method` and `Parameter` metaelements. The `MethodParameterTrigger` consists of a `Method` followed by parentheses, which may or not contain a `Parameter` (e.g., method(parameter)). In this case, the mapping condition specifies that the name of the `Trigger` in MMG should match the pattern specified in Line 19.

```
 1 mapping graph::Trigger::Trigger2Trigger() : text::Trigger
 2 when {not(self.name.matches(".*\\(.*") or self.name.matches("
       .*\\..*"))}
 3 {
 4     result.name := self.name;
 5 }
 6 mapping graph::Trigger::Trigger2PETrigger() : text::
       PortEventTrigger
 7 when {self.name.matches(".*\\..*")}
 8 {
 9     var PortObject := object text::Port{
10         name := self.name.substringBefore(".");
11     };
12     port := PortObject;
13     var EventObject := object text::Event{
14         name := self.name.substringAfter(".");
15     };
16     event := EventObject;
17 }
18 mapping graph::Trigger::Trigger2MPTrigger() : text::
       MethodParameterTrigger
19 when {self.name.matches(".*\\(.*")}
20 { ... }
```

Listing 9.7. `Trigger` to `Trigger`, `PortEventTrigger`, and `MethodParameterTrigger`

9.5 Validation and discussion

The M2M transformations detailed in this paper make possible the synchronization between multiple notations. As such, the correctness of the unidirectional transformations and the consistency between them is crucial.

The validation is conducted for RTist and Papyrus-RT by applying the model transformations to instances (i.e., models) of MMG and MMT. The representation of the graphical instance MG is detailed in Fig. 9.4, while the representation of the textual instance MT is detailed in Fig. 9.5. Fig. 9.4a details the graphical editor of MG, whereas Fig. 9.4b details the Exeed editor (an extended version of the built-in tree-based reflective editor provided by EMF) of MG. Alternatively,

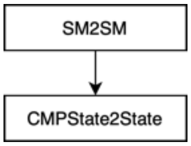


Figure 9.2. Decomposition diagram portion

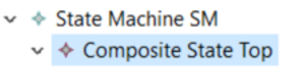


Figure 9.3. Input

Fig. 9.5a details the Xtext editor of MT, whereas Fig. 9.5b details the Exeed editor of MT. We have included the Exeed editor for both instances, as it conveys additional structural information that is not glaring in the other editors. In the following we describe the execution of the validation process.

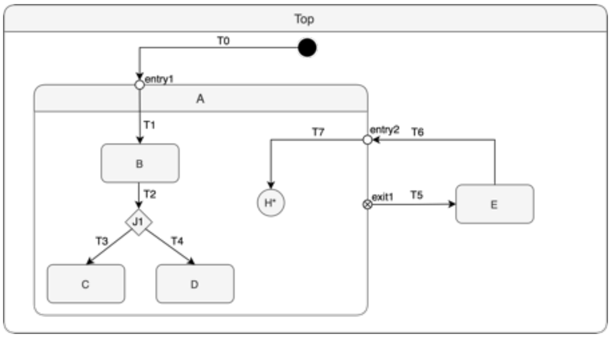
① The correctness of the model transformations is validated by carrying out testing of the MMG2MMT and MMT2MMG model transformations at the unit level [13]. To carry out this procedure, we have defined a functional decomposition diagram for each model transformation. Such diagram, can facilitate the identification of the test *subjects*, as the nodes of the diagram (i.e., mapping operations) will represent the *subjects*. Additional consideration has been given to guarantee that the test cases cover all the mapping operations of a given model transformation. The metaelements of the input metamodels will be considered as *test inputs* whereas the *expected output* will be represented by a regular expression. For a test case to *pass*, the *expected output* of a given mapping operation must match *the actual output*. The latter is the result that we get after the execution of the mapping operation. In the following we use the `CMPState2State` mapping operation defined in the MMG2MMT model transformation to exemplify our manual testing process. In Fig. 9.2 we detail only a portion of the decomposition diagram to highlight `CMPState2State`. Then we extract the input of the `CMPState2State` mapping operation, which is the `CompositeState`. In Fig. 9.3 we detailed an instance of MMG, which is used to test the `CMPState2State` mapping operation.

The *expected output* after the execution of the `CMPState2State` on the instance of MMG detailed in Fig. 9.3 is as follows. The `CompositeState` **Top** included in the `StateMachine` **SM** in MG must be transformed to a `CompositeState` **Top** included in the `StateMachine` **SM** in MT. Lastly we execute the transformations and check whether the *actual output* is same as the *expected output*.

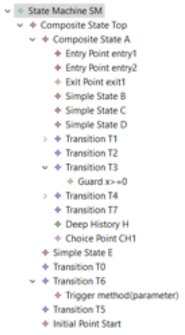
```
<hcl:StateMachine name=SM>
<states name=Top>
</hcl:StateMachine>
```

② The second step involves validating the consistency between the two model transformations. For achieving this, we apply the MMG2MMT model transformation to MG detailed in Fig. 9.4. The output is MT detailed in Fig. 9.5. We then apply the MMT2MMG model transformation to the output of the MMG2MMT model transformation. The output of the MMT2MMG model transformation must be identical to the instance of MG detailed in Fig. 9.4 for the QVTo transformation to be considered consistent. The model transformations are revised until full consistency is achieved. In addition to the testing results, architects at HCL have assessed both the usability and usefulness of the textual notation, as well as the synchronization mechanisms between notations.

On another note, by inspecting the instances of MMG and MMT detailed in Fig. 9.4 and Fig. 9.5 respectively, we can highlight the most significant differences in terms of semantics and structure. For instance, `CompositeState A` and `SimpleState B` in MMG are transformed into `State A` and `State B` in MMT, respectively, where `State A` is the parent of `State B`. Although the structure is identical, there exist semantic differences between `SimpleState B` and `State B`, because `SimpleState B` cannot contain other elements, whilst `State B` can. For transitions, if we consider the same transformation, it is the opposite. For instance, `Transition T0` in MMG is transformed into `InitialTransition T0` in MMT and `Transition T7` in MMG is transformed into `HistoryTransition T7` in MMT. Alternatively, structural differences are prominent when creating new metaelements instead of transforming them (because of the lack of a corresponding source metaelement). This is the case of `TransitionBody`, `Port`, `Event`, `Method`, and `Parameter` metaelements in MMT. For instance, `Transition T1` in Fig. 9.4 has two children, `Trigger` and `ActionChain`, whilst `Transition T1` in Fig. 9.5 has only one child, `TransitionBody`, which contains the `PortEventTrigger` and `TransitionOperation`. Another interesting difference is that whilst in Fig. 9.4 `Transition T7` and `Deep History H` reside on the same



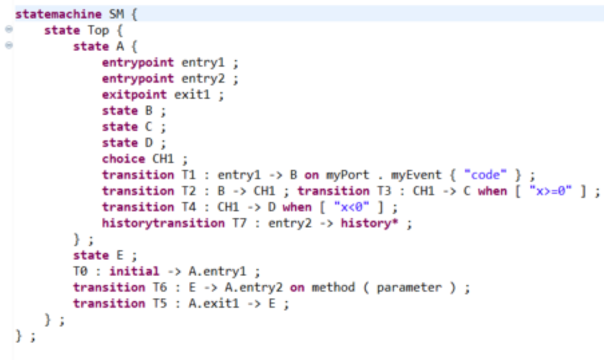
(a) Graphical editor



```
State Machine SM
  Composite State Top
    Composite State A
      Entry Point entry1
      Entry Point entry2
      Exit Point exit1
      Simple State B
      Simple State C
      Simple State D
      Transition T1
      Transition T2
      Transition T3
        Guard x>=0
      Transition T4
      Transition T7
      Deep History H
      Choice Point CH1
    Simple State E
    Transition T0
    Transition T6
      Trigger method(parameter)
    Transition T5
    Initial Point Start
```

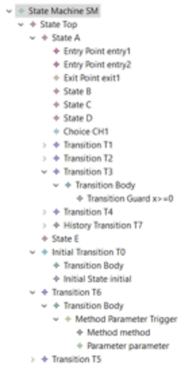
(b) Exeed Editor.

Figure 9.4. Graphical model



```
statemachine SM {
    state Top {
        state A {
            entrypoint entry1 ;
            entrypoint entry2 ;
            exitpoint exit1 ;
            state B ;
            state C ;
            state D ;
            choice CH1 ;
            transition T1 : entry1 -> B on myPort . myEvent { "code" } ;
            transition T2 : B -> CH1 ; transition T3 : CH1 -> C when [ "x>=0" ] ;
            transition T4 : CH1 -> D when [ "x<0" ] ;
            historytransition T7 : entry2 -> history* ;
        } ;
        state E ;
        T0 : initial -> A.entry1 ;
        transition T6 : E -> A.entry2 on method ( parameter ) ;
        transition T5 : A.exit1 -> E ;
    } ;
} ;
```

(a) Xtext editor



```
State Machine SM
  State Top
    State A
      Entry Point entry1
      Entry Point entry2
      Exit Point exit1
      State B
      State C
      State D
      Choice CH1
      Transition T1
      Transition T2
      Transition T3
        Transition Body
          Transition Guard x>=0
      Transition T4
      History Transition T7
    State E
    Initial Transition T0
      Transition Body
      Initial State initial
    Transition T6
      Transition Body
        Method Parameter Trigger
          Method method
          Parameter parameter
    Transition T5
```

(b) Exeed Editor

Figure 9.5. Textual model

level (i.e., they are siblings), in Fig. 9.5 `Deep History H` is a child of `Transition T7`. The reason for this is to allow the user to initialize `DeepHistory` when writing `Transition T7`, and not before initializing it, and then reference it in `Transition T7`.

9.6 Related work

The Action Language for Foundational UML (Alf) [14] is a textual language standardized by the Object Management Group (OMG) for representing UML models. Since its underlying semantics are indicated by a limited subset of UML, named Foundational UML (fUML), the Alf syntax is restricted within its bounds and does not support state-machines as they are not available in the fUML subset. tUML is a textual language for a limited subset of the standard UML metamodel targeted at real-time embedded systems that consists of class diagrams, composite structure diagrams, and state diagrams. The implementation of tUML has been carried out to have a very close proximity to the UML metamodel. Consideration has been given to propose tUML to OMG as an extension of Alf, being that the latter lacks support for state-machines [15]. There also exists a plethora of tools and modeling languages that support textual notations for UML models. Earl Grey [16] is a textual modeling language that supports the creation of UML class and state models. MetaUML [17] is a MetaPost library for creating UML diagrams using textual notations, and it supports class diagrams, package diagrams, component diagrams, use case diagrams, activity diagrams, and state diagrams. The textual notation is not only used to define the elements and their relationships but also their layout properties. PlantUML [11] is an open-source tool that supports the generation of both UML and non-UML diagrams from a textual language. Among the most important UML diagrams they support are sequence diagrams, class diagrams, activity diagrams, state diagrams, and more. Umple [18] is an open-source modeling tool that can be used to add UML abstractions to programming languages (i.e., Java, C++, PHP, and Ruby) and create UML class and state diagrams from a textual notation. The generated graphical view for class diagrams can be edited, while for state-machines, it is read-only. Textual, executable, and translatable UML (txtUML) [19] is an open-source project that supports the creation of models from a textual notation and generates the corresponding graphical visualization. TextUML Toolkit [12] is an open-source IDE that allows the creation of UML2 models from a textual notation. This toolkit is available on Cloudfier, as a

[11] http://plantuml.com/guide
[12] ttp://abstratt.github.io/textuml/

plug-in for Eclipse IDE and as a standalone command-line tool.

There have been a handful of attempts at providing textual syntax for UML-RT and we have been involved in some of them. Calur [20] provides a textual syntax only for UML-RT's action language, not state-machines. Unlike our approach, both eTrice[13] and Papyrus-RT[14] provide a kind of all-or-nothing approach. They both provide syntax for both structure and behaviour, but the entire model is described as either textual or graphical, whereas in our approach the user can select only parts of the model to be represented textually. This allows the user to retain the ability to use existing RTist tooling for graphical modelling. Note also that our textual notation for UML-RT state-machines has been designed and implemented to maximise user experience of architects and engineers, as their throughput thanks to the possibility of blended modelling.

9.7 Outlook

In this work, we have reported on our work to provide a seamless blended graphical and textual modelling environment for UML-RT state-machines. Our proposed solution involves the provision of (i) a textual notation as complement to the existing graphical notation for UML-RT state-machines and (ii) ad hoc synchronization mechanisms between the metamodels underlying the two notations. The synchronization mechanisms have been designed as model-to-model transformations and implemented using the operational implementation of the QVT language in Eclipse. With regards to the limitations of this approach, we argue that the solution is language-agnostic (i.e., applicable to UML-RT state machines only). For any other language, the related editors and transformations have to be re-done from scratch. On a similar note, if the metamodels evolve, the model transformations would have to be manually updated.

For that reason, future work involves the definition of a mapping language that allows the definition of explicit mappings between arbitrary metamodels (MMs), and automatic generation of synchronization transformations via Higher Order Transformations (HOTs). The HOTs are transformations that take as input and/or generate as output other model transformations [21]. Given two

[13] `https://www.eclipse.org/etrice/`

[14] `https://www.eclipse.org/papyrus-rt/`

MMs defined, a mapping model, conforming to the mapping language, would conceive the mapping rules for synchronizing models conforming to the two MMs. The mapping model together with the two MMs would be given in input to a set of HOTs that we are currently designing. The outputs of the HOTs are synchronization model transformations, as the ones defined manually in Operational QVT for the solution presented in this paper. The type of generated transformations (i.e., endogenous, exogenous, out-of-place, in-place) depends on the nature of the two MMs. In fact, this architecture and the HOTs in it would entail multiple usage scenarios, as follows. In case the MMs are two entirely disjoint (but somehow connected/dependent) languages, the generated transformations provide synchronization across different languages (either same or different notations). In case the MMs represent two notations of the same language, the generated transformations provide synchronization across different notations of the language. In addition, in case the target MM represents an evolution of the source MM, the generated transformations provide co-evolution mechanisms for models conforming to MM.

This work is run in the context of an international consortium across 4 countries within the ITEA3 BUMBLE project[15]. In that context, we will run more extensive controlled experiments and industrial case-studies too. An important element of the dissemination plan consists in leveraging the different opportunities provided in the Eclipse community, including Eclipse conferences (e.g., EclipseCon Europe) and marketing. We will also collaborate with the Eclipse Working Groups, Papyrus and Capella Industry Consortia to reach out to industrial MDE tool users. We plan to disseminate results via research forums (conferences, workshops), corporate presentations, participation to industrial events like expos, on-line community forums for Eclipse, social media, fact sheets, and wikis.

[15] `https://blended-modeling.github.io/`

Bibliography

[1] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied software architecture*. Addison-Wesley Professional, 2000.

[2] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

[3] Bran Selic. The pragmatics of model-driven development. *IEEE software*, 20(5):19–25, 2003.

[4] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, 2012.

[5] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 471–480, 2011.

[6] Bran Selic. Real-time object-oriented modeling. *IFAC Proceedings Volumes*, 29(5):1–6, 1996.

[7] Lorenzo Addazi and Federico Ciccozzi. Blended graphical and textual modeling for UML profiles: A proof-of-concept implementation and experiment. *Journal of Systems and Software*, 175:110912, 2021.

[8] Malvina Latifaj, Federico Ciccozzi, Mattias Mohlin, and Ernesto Posse. Towards automated support for blended modeling of UML-RT embedded software architectures. In *15th European Conference on Software*

Architecture ECSA 2021, 13 Sep 2021, Virtual (originally Växjö), Sweden, 2021.

[9] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended modelling – what, why and how. In *MPM4CPS workshop*, September 2019.

[10] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. Multi-view approaches for software and system modelling: A systematic literature review. *Software & Systems Modeling*, 2019.

[11] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic notes in theoretical computer science*, 152:125–142, 2006.

[12] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[13] Alessandro Tiso, Gianna Reggio, and Maurizio Leotta. Unit Testing of Model to Text Transformations. In *AMT 2014–Analysis of Model Transformations Workshop Proceedings*, page 14, 2014.

[14] Object Management Group (OMG). Action Language for Foundational UML (Alf), Version 1.1. OMG Document Number formal/2017-07-04. http://www.omg.org/spec/ALF/1.1, 2017.

[15] Frédéric Jouault and Jérôme Delatour. Towards fixing sketchy UML models by leveraging textual notations: Application to real-time embedded systems. In *OCL@ MoDELS*, pages 73–82, 2014.

[16] Martin Mazanec and Ondrej Macek. On general-purpose textual modeling languages. In *Dateso*, volume 12, pages 1–12. Citeseer, 2012.

[17] Ovidiu Gheorghies. MetaUML: Tutorial, reference and test suite. https://profs.info.uaic.ro/~ogh/files/main.pdf, 2005. Retrieved: 02/05/2021.

[18] Timothy C Lethbridge, Vahdat Abdelzad, Mahmoud Husseini Orabi, Ahmed Husseini Orabi, and Opeyemi Adesina. Merging modeling and

programming using Umple. In *International Symposium on Leveraging Applications of Formal Methods*, pages 187–197. Springer, 2016.

[19] Gergely Dévai, Gábor Ferenc Kovács, and Ádám An. Textual, executable, translatable UML. In *OCL@ MoDELS*, pages 3–12. Citeseer, 2014.

[20] Nicolas Hili, Ernesto Posse, and Juergen Dingel. Calur: an action language for UML-RT. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

[21] Massimo Tisi, Frédéric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bézivin. On the use of higher-order model transformations. In *5th European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA 2009)*, pages 18–33. Springer, 2009.

Chapter 10

Paper D: Automatic generation of synchronization mechanisms for blended modeling

Malvina Latifaj, Federico Ciccozzi, Mattias Mohlin.
Published in the Frontiers of Computer Science journal (Frontiers 2023).

Abstract

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. In this paper, we propose an automated solution for generating synchronization transformations in an industrial setting. 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. 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. We validated the proposed approach in two use cases.

10.1 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 [1], 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 [2]).

10.1.1 Blended modeling

We have defined the notion of blended modeling in a previous work [3] 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 [4].

At first sight, the notion of blended modeling may seem similar to or overlapping with multi-view modeling [5] (and even multi-paradigm modeling) that is based on the paradigm of viewpoint/ view/ model as formalised 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" [6]) that are materialized through views, which are composed of one or more models. In blended modeling, the focus is *not*

[1] `https://www.iso.org/standard/50508.html`

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 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.

10.1.2 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) [7] and DSMLs described using EMF's meta-metamodel, Ecore.

10.1.3 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 10.2. We discuss work related to the problem domain in Section 10.3. The developed approach and implementation are described in detail in Section 10.4. Experiences from validating the approach are included in Section 10.5. A discussion is provided in Section 10.6 and the paper is concluded in Section 10.7.

10.2 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.

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

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 [8] and [9]. More specifically, in [8] we describe the effort of designing, implementing and integrating a textual notation for UML-RT state machines in RTist. In [9] 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 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.

10.3 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 [10], or generation of model transformations for the exchange of models between meta-modeling tools such as [11], 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.

10.3.1 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 [12], Umple by [13], Excalibur by [14], 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

[3] `http://lightuml.sourceforge.net`
[4] `https://github.com/ogheorghies/MetaUML`
[5] `https://plantuml.com`
[6] `https://jankoehnlein.github.io/FXDiagram/`

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, [15] 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. [16] propose a blended modeling framework, but the solution is specific to UML-based DSMLs. [17] 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. [18] 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 [19] 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 [20]. 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.

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

10.3.2 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) [21]. Several publications in the literature have been devoted to efforts to automate the generation of model transformations by means of weaving models. [22] 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. [23] 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, [24] 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. [25] 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. [26] 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.

10.4 Proposed solution

Consider the model of our solution depicted in Figure 10.1. Depending on what the involved pair of DSMLs represents, we focus on two different scenarios:

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

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

[9] https://www.eclipse.org/modeling/emf/
[10] https://www.eclipse.org/xtend/
[11] https://wiki.eclipse.org/QVTo

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]).

10.4.1 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 [22]. 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

[12] `https://github.com/MLJworkspace/BlendedModellingSolution`

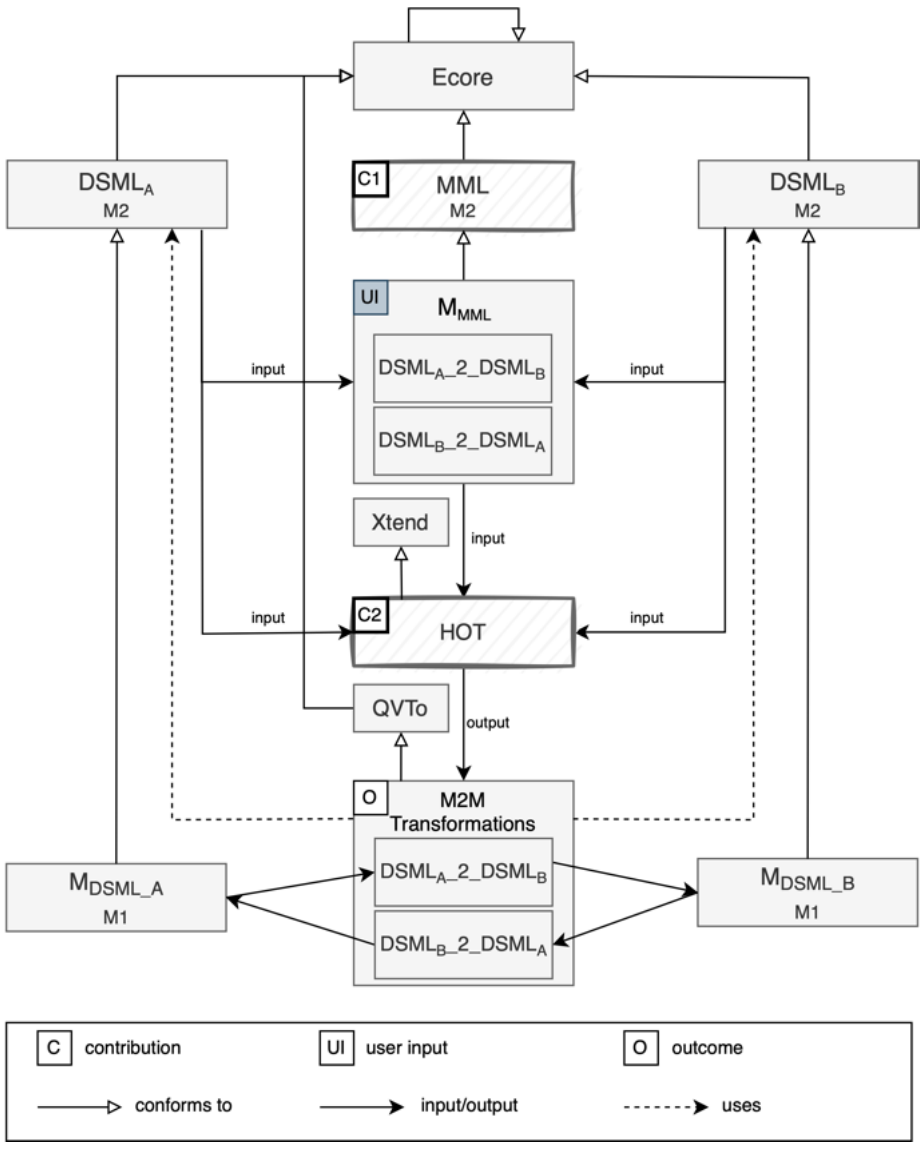


Figure 10.1. Semi-automatic synchronization infrastructure

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 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 10.2 we describe the MML metamodel defined as an Ecore model, and in the following we detail the metaconcepts of MML.

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

`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

[13] `https://wiki.eclipse.org/OCL`
[14] `https://www.eclipse.org/Xtext/`

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

SourceMetamodel and **TargetMetamodel** represent the DSMLs that will be involved in the 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.

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.

10.4.2 Higher-order transformation

The automatic generation of model transformations for synchronization purposes is achieved by means of HOTs. According to their definition [27], HOTs are 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 10.1, 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.

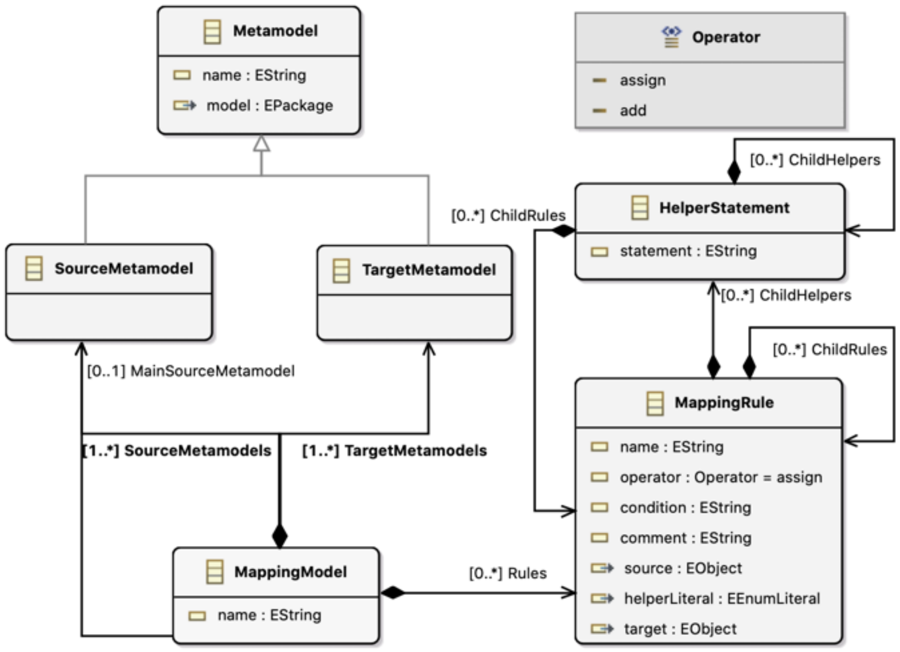


Figure 10.2. MML metamodel in Ecore

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 artefacts. 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. `MappingRules` 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. `MappingRules` are contained in other `MappingRules` or `HelperStatements` 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.

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.

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 10.3, 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 `Organization-Organization2Company` as detailed in Figure 10.4. 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.manager.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

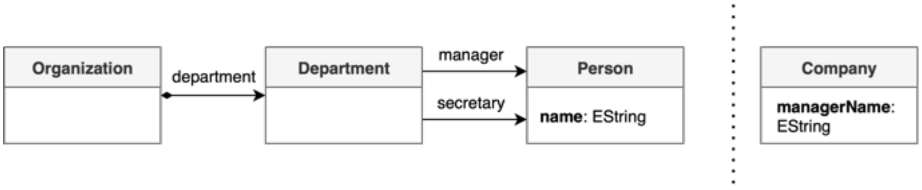


Figure 10.3. Ambiguous navigation

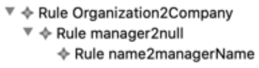


Figure 10.4. Ambiguous navigation mapping rules

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`.

10.5 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 10.5.1 we provide details on the two use cases in isolation and then conduct a comparison between the two, while in Section 10.5.2 we provide the details of the model-to-text test cases. Lastly, in Section 10.5.3 we provide an example of our industrial use case.

10.5.1 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.

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.

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.

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.

Use case comparison

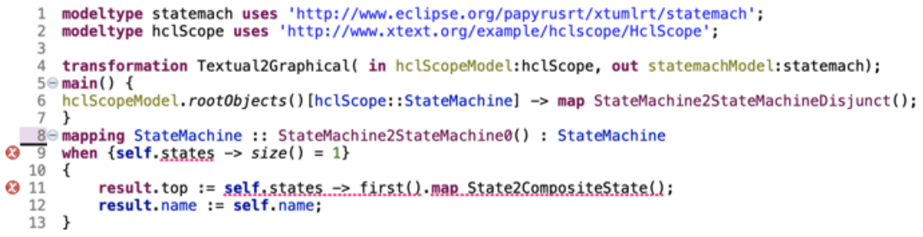
Comparing the distribution of the mapping rules between the three scenarios, in the first use case, the number of mapping rules that fall under SC2 and SC3 is mainly due to the specialization of concepts, while in the second use case it is due to semantic and syntactical differences. Despite the fact 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.

10.5.2 Model-to-text testing

Validation was performed based on model-to-text tests classified by [28] 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.

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



```
 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 10.5. Example of failed conformance test

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 10.5. 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 10.10. 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.

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 10.3. 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 10.3, navigation between EClass `Department` and EClass `Person` can be accomplished through `manager` or `secretary` references.

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 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

[15] `https://extendsclass.com/xml-diff.html`

(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.

10.5.3 Example

In Figure 10.7 we provide an example of a mapping model between the excerpts of $DSML_A$ and $DSML_B$ illustrated in Figure 10.6 and Figure 10.8. In Figure 10.9 we illustrate the properties view for three mapping rules defined in Figure 10.7, 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 10.9. 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 10.10 we present a more extensive excerpt of the `Textual2Graphical` mapping model for UML-RT state 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

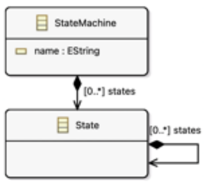
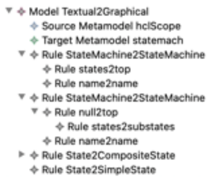


Figure 10.6. $DSML_A$



```
▼ Model Textual2Graphical
    Source Metamodel hclScope
    Target Metamodel statemach
  ▼ Rule StateMachine2StateMachine
      Rule states2top
      Rule name2name
  ▼ Rule StateMachine2StateMachine
    ▼ Rule null2top
        Rule states2substates
      Rule name2name
  ▶ Rule State2CompositeState
    Rule State2SimpleState
```

Figure 10.7. Mapping model

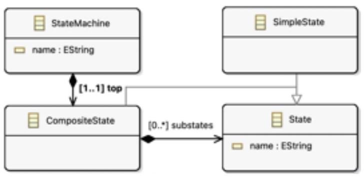
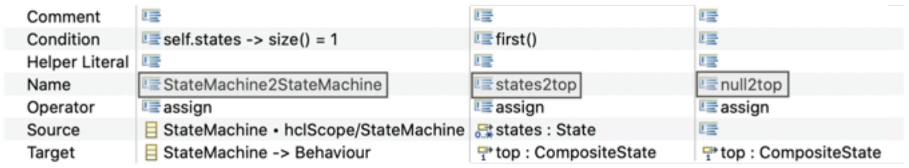


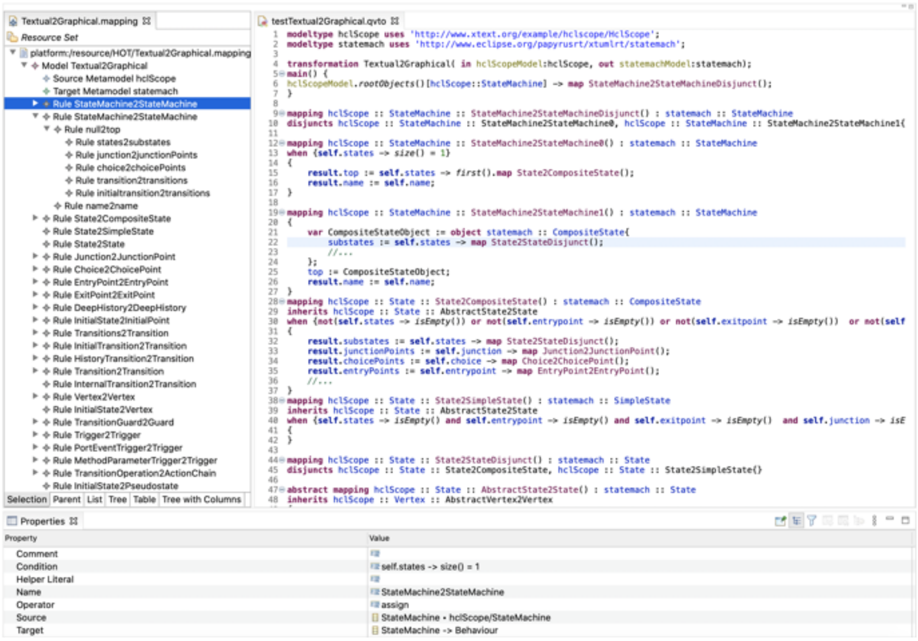
Figure 10.8. $DSML_B$



Comment			
Condition	self.states -> size() = 1	first()	
Helper Literal			
Name	StateMachine2StateMachine	states2top	null2top
Operator	assign	assign	assign
Source	StateMachine • hclScope/StateMachine	states : State	
Target	StateMachine -> Behaviour	top : CompositeState	top : CompositeState

Figure 10.9. Properties view for the mapping rules defined in Figure 10.7

disjuncts the other two (Lines 12 and 19). The disjunctive mapping operation is invoked on Line 6 and the first matching candidate (`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 to 25 detail the generation of mappings from child mapping rules according to



Textual2Graphical.mapping

Resource Set

```
▼ platform:/resource/HOT/Textual2Graphical.mapping
  ▼ Model Textual2Graphical
      Source Metamodel hclScope
      Target Metamodel statemach
    ▶ Rule StateMachine2StateMachine
    ▼ Rule StateMachine2StateMachine
      ▼ Rule null2top
          Rule states2substates
          Rule junction2junctionPoints
          Rule choice2choicePoints
          Rule transition2transitions
          Rule initialtransition2transitions
        Rule name2name
    ▶ Rule State2CompositeState
      Rule State2SimpleState
      Rule State2State
    ▶ Rule Junction2JunctionPoint
    ▶ Rule Choice2ChoicePoint
    ▶ Rule EntryPoint2EntryPoint
    ▶ Rule ExitPoint2ExitPoint
    ▶ Rule DeepHistory2DeepHistory
    ▶ Rule InitialState2InitialPoint
    ▶ Rule Transitions2Transition
    ▶ Rule InitialTransition2Transition
    ▶ Rule HistoryTransition2Transition
    ▶ Rule Transition2Transition
      Rule InternalTransition2Transition
    ▶ Rule Vertex2Vertex
      Rule InitialState2Vertex
    ▶ Rule TransitionGuard2Guard
    ▶ Rule Trigger2Trigger
    ▶ Rule PortEventTrigger2Trigger
    ▶ Rule MethodParameterTrigger2Trigger
    ▶ Rule TransitionOperation2ActionChain
      Rule InitialState2Pseudostate
```

testTextual2Graphical.qvto

```
1  modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
2  modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';
3
4  transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
5  main() {
6  hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachineDisjunct();
7  }
8
9  mapping hclScope :: StateMachine :: StateMachine2StateMachineDisjunct() : statemach :: StateMachine
10 disjuncts hclScope :: StateMachine :: StateMachine2StateMachine0, hclScope :: StateMachine :: StateMachine2StateMachine1{
11
12 mapping hclScope :: StateMachine :: StateMachine2StateMachine0() : statemach :: StateMachine
13 when {self.states -> size() = 1}
14 {
15     result.top := self.states -> first().map State2CompositeState();
16     result.name := self.name;
17 }
18
19 mapping hclScope :: StateMachine :: StateMachine2StateMachine1() : statemach :: StateMachine
20 {
21     var CompositeStateObject := object statemach :: CompositeState{
22         substates := self.states -> map State2StateDisjunct();
23         //...
24     };
25     top := CompositeStateObject;
26     result.name := self.name;
27 }
28 mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
29 inherits hclScope :: State :: AbstractState2State
30 when {not(self.states -> isEmpty()) or not(self.entrypoint -> isEmpty()) or not(self.exitpoint -> isEmpty())  or not(self
31 {
32     result.substates := self.states -> map State2StateDisjunct();
33     result.junctionPoints := self.junction -> map Junction2JunctionPoint();
34     result.choicePoints := self.choice -> map Choice2ChoicePoint();
35     result.entryPoints := self.entrypoint -> map EntryPoint2EntryPoint();
36     //...
37 }
38 mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
39 inherits hclScope :: State :: AbstractState2State
40 when {self.states -> isEmpty() and self.entrypoint -> isEmpty() and self.exitpoint -> isEmpty()  and self.junction -> isE
41 {
42 }
43
44 mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
45 disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}
46
47 abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
48 inherits hclScope :: Vertex :: AbstractVertex2Vertex
```

Properties

Property	Value
Comment	
Condition	self.states -> size() = 1
Helper Literal	
Name	StateMachine2StateMachine
Operator	assign
Source	StateMachine • hclScope/StateMachine
Target	StateMachine -> Behaviour

Figure 10.10. Textual to graphical mapping model and generated QVTo transformations for UML-RT state machines

SC2 described in Section 10.4.2, 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.

10.6 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.

10.6.1 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.

10.6.2 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.

10.6.3 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 [29]. 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 [30]. 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.

10.7 Conclusions

In this paper, we presented our effort towards 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.

Bibliography

[1] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 471–480, 2011.

[2] Johan Lilius and I Porres Paltor. vUML: A tool for verifying UML models. In *14th IEEE International Conference on Automated Software Engineering*, pages 255–258. IEEE, 1999.

[3] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended modelling – what, why and how. In *MPM4CPS workshop*, September 2019.

[4] Magnus Persson, Martin Törngren, Ahsan Qamar, Jonas Westman, Matthias Biehl, Stavros Tripakis, Hans Vangheluwe, and Joachim Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *13th International Conference on Embedded Software (EMSOFT 2013)*, pages 1–10. IEEE, 2013.

[5] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. Multi-view approaches for software and system modelling: A systematic literature review. *Software & Systems Modeling*, 2019.

[6] David Emery and Rich Hilliard. Every architecture description needs a framework: Expressing architecture frameworks using ISO/IEC 42010. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pages 31–40. IEEE, 2009.

[7] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: Eclipse Modeling Framework.* Pearson Education, 2008.

[8] Malvina Latifaj, Federico Ciccozzi, Mattias Mohlin, and Ernesto Posse. Towards automated support for blended modeling of UML-RT embedded software architectures. In *15th European Conference on Software Architecture ECSA 2021, 13 Sep 2021, Virtual (originally Växjö), Sweden*, 2021.

[9] Malvina Latifaj, Federico Ciccozzi, Muhammad Waseem Anwar, and Mattias Mohlin. Blended graphical and textual modeling of UML-RT state-machines: An industrial experience. In *European Conference on Software Architecture*, pages 22–44. Springer, 2021.

[10] István Ráth, András Ökrös, and Dániel Varró. Synchronization of abstract and concrete syntax in domain-specific modeling languages. *Software & Systems Modeling*, 9(4):453–471, 2010.

[11] Heiko Kern, Fred Stefan, Vladimir Dimitrieski, and Milan Čeliković. Mapping-based exchange of models between meta-modeling tools. In *Proceedings of the 14th Workshop on Domain-Specific Modeling*, pages 29–34, 2014.

[12] Anis Charfi, Artur Schmidt, and Axel Spriestersbach. A hybrid graphical and textual notation and editor for UML actions. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 237–252. Springer, 2009.

[13] Timothy C Lethbridge, Vahdat Abdelzad, Mahmoud Husseini Orabi, Ahmed Husseini Orabi, and Opeyemi Adesina. Merging modeling and programming using Umple. In *International Symposium on Leveraging Applications of Formal Methods*, pages 187–197. Springer, 2016.

[14] Benoît Ries, Alfredo Capozucca, and Nicolas Guelfi. 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*, pages 103–107. ACM, 2018.

[15] Salome Maro, Jan-Philipp Steghöfer, Anthony Anjorin, Matthias Tichy, and Lars Gelin. 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*, pages 1–12, 2015.

[16] Lorenzo Addazi and Federico Ciccozzi. Blended graphical and textual modeling for UML profiles: A proof-of-concept implementation and experiment. *Journal of Systems and Software*, 175:110912, 2021.

[17] Codruţ-Lucian Lazăr. Integrating Alf editor with Eclipse UML editors. *Studia Universitatis Babes-Bolyai, Informatica*, 56(3), 2011.

[18] Markus Scheidgen. Textual modeling embedded into graphical modeling. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 153–168. Springer, 2008.

[19] Colin Atkinson and Ralph Gerbig. Flexible deep modeling with MelanEE. *Modellierung 2016-Workshopband*, 2016.

[20] Istvan David, Malvina Latifaj, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer, and Regina Hebig. Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. *Software and Systems Modeling*, 22(1):415–447, 2023.

[21] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2):171–188, 2005.

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

[23] Marcos Didonet Del Fabro and Patrick Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Software and Systems Modeling*, 8(3):305–324, 2009.

[24] Zinovy Diskin, Abel Gómez, and Jordi Cabot. Traceability mappings as a fundamental instrument in model transformations. In *International Conference on Fundamental Approaches to Software Engineering*, pages 247–263. Springer, 2017.

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

[26] Guillaume Hillairet, Frédéric Bertrand, and Jean Yves Lafaye. Bridging EMF applications and RDF data sources. In *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering, SWESE*, 2008.

[27] Massimo Tisi, Frédéric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bézivin. On the use of higher-order model transformations. In *5th European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA 2009)*, pages 18–33. Springer, 2009.

[28] Alessandro Tiso, Gianna Reggio, and Maurizio Leotta. A method for testing model to text transformations. In *AMT@ MoDELS*, 2013.

[29] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software & Systems Modeling*, 9(1):7–20, 2010.

[30] Frédéric Jouault and Ivan Kurtev. On the interoperability of model-to-model transformation languages. *Science of Computer Programming*, 68(3):114–137, 2007.

Chapter 11

Paper E: Role-based access control for collaborative modeling environments

Malvina Latifaj, Federico Ciccozzi, Antonio Cicchetti.
Conditionally accepted at the Journal of Object Technology (JOT) at the time of writing.

Abstract

Collaborative model-driven software engineering fosters efficient cooperation among stakeholders who collaborate on shared models. Yet, the involvement of multiple parties brings forth valid concerns about the confidentiality and integrity of shared information. Unrestricted access to such information, especially when not pertinent to individual responsibilities, poses significant risks, including unauthorized information exposure and potential harm to information integrity. This work proposes a dual-layered solution implemented as an open-source Eclipse plugin that leverages the role-based access control policy to ensure the confidentiality and integrity of model information in collaborative modeling environments. The first layer limits stakeholders' access to the shared model based on their specific roles, while the second layer refines this access by restricting manipulations to individual model elements. By ensuring that stakeholders access only the information pertinent to their roles and are authorized to manipulate such information in accordance with their expertise and responsibilities, this approach ensures the confidentiality and integrity of shared model information. Furthermore, it alleviates information overload for stakeholders by enabling them to focus only on the model information relevant to their specific roles, thereby enhancing the collaborative efforts.

11.1 Introduction

Model-Driven Software Engineering (MDSE) [1] has redirected the focus of software engineering towards prioritizing models as the fundamental artifacts in the development of complex software systems. MDSE aims to increase the level of abstraction and reduce the accidental complexity associated with the tools and methods used during development [2]. Despite its benefits, the inherent complexity and ongoing development demands of such systems render MDSE an endeavor that cannot always be managed single-handedly. Collaborative MDSE emerges as an integration of collaborative software engineering principles [3, 4] with the abstraction and automation advantages provided by MDSE [5, 6]. This approach fosters the development and maintenance of models collaboratively, enhancing the efficiency and quality of the software development process [7]. While collaborative modeling practices offer considerable advantages, they also introduce notable challenges, particularly in safeguarding the confidentiality and integrity of sensitive information carried by the shared models. Such an environment encompasses a wide range of stakeholders, including developers, domain experts, and managers, each bringing their distinct expertise and responsibilities to the table. Their collaboration on a single base model exposes far more information than necessary to each participant, significantly increasing the risk of confidentiality breaches and information integrity compromise. This issue is especially alarming considering that models can incorporate proprietary algorithms, business logic, and personal data, making privacy and security paramount. Insights from industrial practices highlight the essential need for trustworthy collaborative modeling environments to feature comprehensive access control mechanisms [8], which are pivotal in safeguarding the confidentiality, integrity, and availability – collectively known as the CIA triad – of information [9]. Access control mechanisms should be tailored to meet practitioners' needs, increasing the likelihood of their adoption in collaborative modeling environments. They should support the definition of fine-grained access permissions and facilitate the management of these permissions to adapt to evolving project needs. In addition, beyond the mere definition and management of access permissions, they must ensure the consistent and accurate enforcement of access permissions through automated processes.

Existing access control approaches hinge on transient, virtual view models that lack independence from their base models [10]. This limitation renders the models unsuitable for contexts requiring persistent views such as offline collaboration scenarios. Others enforce access permissions via bidirectional transformations, potentially leading to user frustration due to delayed feedback on unauthorized actions [11]. Additionally, most current methodologies focus primarily on basic read and write permissions. Such gaps highlight the need for approaches that support persistent views and provide immediate, granular access control feedback to enhance user experience and efficiency.

In this article, we propose a dual-layered approach that leverages the role-based access control (RBAC) policy [12] to ensure the confidentiality and integrity of model information in collaborative modeling environments based on the Eclipse Modeling Framework (EMF) [13]. The first layer limits access to the base model by employing multi-view modeling techniques [14] to create materialized view models, which are essentially subsets of the base model containing only the elements essential for specific user roles. Users can only access and interact with the view models designated to their roles, effectively preventing access to the entire base model. The second layer further refines access down to the individual elements within view models, establishing fine-tuned access permissions. These permissions are enforced by model editors that dictate the extent to which a specific user role can interact with and manipulate each element of the view model. Unlike the trial and error methods, this approach proactively prevents restricted operations.

The remainder of this paper is structured as follows. Section 11.2 provides background information on the key concepts. Section 11.3 illustrates a running example used throughout the paper. Section 11.4 presents the proposed approach, while Section 11.5 describes the application of the approach on the illustrative running example. Section 11.6 describes the related work to this research. Section 11.7 provides a discussion on the benefits and limitations of the approach. Section 11.8 concludes the paper and describes future research directions.

11.2 Background

This section describes the key concepts relevant to our study. Section 11.2.1 outlines the modeling framework for our proposed solution. Sections 11.2.2 and 11.2.3 discuss multi-view modeling and access control, respectively.

11.2.1 Eclipse modeling framework and Ecore

Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for building tools and other applications based on a structured data model [13]. It utilizes XMI-based model specifications to generate a suite of Java classes, complemented by adapter classes that enable viewing and command-based editing of the model, and a basic editor. EMF consists of three fundamental parts. EMF's core framework encompasses Ecore[1], a metamodel for defining models, and provides runtime support including change notifications, default XMI serialization for persistence, and a reflective API for efficient manipulation of EMF objects. The EMF.Edit framework provides generic reusable classes for building editors for EMF models. Lastly, the EMF.Codegen, a code generation facility, is designed to generate all necessary components for a complete EMF model editor. This includes a GUI for setting generation options and initiating generators.

11.2.2 Multi-view modeling

Multi-view modeling delivers customized views designed to cater to the unique needs, expertise, and goals of various stakeholders, ensuring alignment and relevance to their specific contributions. The core of multi-view modeling lies in the viewpoint/view/model paradigm, as formalized by the ISO/IEC 42010:2011 standard [15]. According to this standard, a viewpoint represents a specific abstraction using a chosen set of constructs and rules, addressing specific concerns within a system. Consequently, it determines the conventions, like notations, languages, and model types, for crafting a specific kind of view. A view is the resulting instance of applying a viewpoint to a particular system of interest and

[1] https://download.eclipse.org/modeling/emf/emf/javadoc/2.11/org/eclipse/emf/ecore/package-summary.html

is composed of one or more models. In the context of multi-view modeling, domain-specific languages (DSLs) are leveraged to address certain system concerns. The models conforming to each DSL then provide a distinct view of the system. Multi-view modeling approaches are classified into *projective* and *synthetic* [16]. Projective methods involve defining viewpoints by selectively abstracting concepts from an existing base language. This approach is notable for facilitating automatic synchronization through centralized manipulations in a single model. However, it requires a well-defined semantics of the base language and may restrict customizability due to the static nature and predefined views of the base language. On the other hand, synthetic methods establish viewpoints as independent metamodels. In this approach, synchronization is achieved by defining interactions among different viewpoints or views. As the number of views increases, this method becomes increasingly complex, making synchronization progressively more challenging. In prior work [16], we have developed a hybrid methodology that combines the best elements of both projective and synthetic approaches. This method allows for the creation of views based on a base metamodel, similar to the projective approach, yet these views emerge as separate metamodels, similar to the synthetic approach. This ensures inherent synchronization during view definition, alongside the flexibility of introducing views at any development stage.

11.2.3 Access control

Access control refers to a security mechanism, pivotal in safeguarding shared resources against unauthorized access, thereby ensuring information security [17]. It acts as a defensive barrier, blocking unauthorized individuals from accessing or altering sensitive data, including proprietary algorithms and strategic business information. This mechanism not only preserves the confidentiality of information but also safeguards its integrity from malicious tampering. Furthermore, access control policies are key in mitigating unintentional alterations by individuals who might not possess the required knowledge or expertise, such as novice engineers. Access control operates fundamentally through two key processes being *authentication* and *authorization* [18]. Authentication entails the verification of a user's credentials, assuring that users are who they claim to be. It necessitates users to provide valid credentials, which are cross-verified against a pre-established database, thereby ensuring that only authorized individuals can access specific information within an organization. On the other hand, authorization determines the extent of access and actions permissible to authenticated users. Authorization operates through access control policies, incorporating rules that dictate the allowable levels of access to various data resources. Two main types of access control policies are recognized [19]:

- Discretionary Access Control (DAC): a user-centric approach that grants users the autonomy to assign access permissions. For instance, platforms like Google Drive [2] allow owners to share files or folders, granting specific access levels to other users, such as edit or view permissions, which can be modified or revoked as necessary.

- Non-Discretionary Access Control (NDAC): here, the determination of access permissions is centralized and administered by system authorities or administrators, rather than the resource owner. This model is particularly apt for scenarios necessitating rigorous security and hierarchical access controls, such as in Amazon Web Services (AWS) Identity and Access Management (IAM) [3], where access is managed centrally using predefined policies and roles.

Of the many access control policies encompassed within NDAC—like multi-level security (MLS), attribute-based access control (ABAC), and Separation of Duty (SoD), we employ Role-Based Access Control (RBAC) [18]. The selection of RBAC stems from a deliberate consideration of organizational structures and the principles of effective access management. RBAC mirrors the hierarchical and role-oriented nature of organizational structures, making it a more intuitive choice for access control in contexts where users are entrusted with permissions based on their designated roles, responsibilities, and hierarchical positions within the organization. Furthermore, RBAC offers a more streamlined and scalable approach to access management, as it simplifies the process of assigning and managing permissions by associating them with roles rather than individual users.

[2] https://www.google.com/drive/
[3] https://aws.amazon.com/iam/

11.3 Running example

This section introduces a simplified version of a university metamodel, serving as an illustrative example throughout the paper to elucidate the proposed solution and its workflow. Figure 11.1 illustrates the Ecore-based metamodel of the structure of the *university* consisting of multiple *departments*. Each *department* is tasked with the administration of various academic *programs*. Each *program* is comprised of a series of individual *courses*. Department *employees*, are responsible for delivering course content. Additionally, each department undertakes a variety of research *projects*. *Projects*, represent collaborative research initiatives that may involve a combination of department *employees* and external *partners*, which may include individuals from *industrial* sectors or other *academic* institutions.

11.3.1 User roles

In this example, we outline three main user roles engaging with the university model.

- *Project Managers:* represent a group of individuals hired by the university to coordinate the research projects for each department, and oversee the assignment of both internal employees and external partners to various projects.

- *Program Coordinators:* represent a group of individuals hired by the university to coordinate the development and management of academic programs, and guarantee that both programs and courses remain current and relevant.

- *Administration:* their role is related to the coordination of administrative matters related to both research and academic programs. As such, they maintain an overview of the entire model.

User roles and users are defined by an authorized individual, which throughout the rest of this paper we refer to as *admin*. The admin is also in charge of assigning users to user roles.

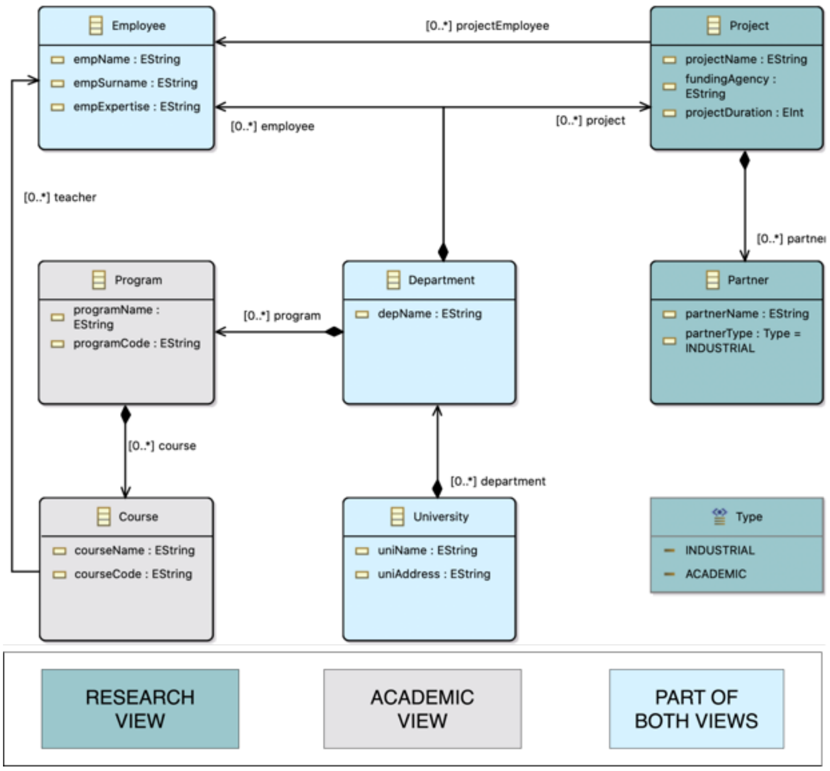


Figure 11.1. University metamodel

11.3.2 Views

Each of the outlined user roles has unique responsibilities and areas of expertise, necessitating interaction with only certain parts of the university model that are relevant to their needs. Defining views that contain only a few aspects of the overall base university model allows them to concentrate on the elements that are most crucial to their roles and goals. These views act as a primary access control layer by excluding non-essential elements, thereby minimizing the risk of exposure to potential security vulnerabilities. For this example, the following views are required:

- *Teaching View:* tailored to the needs and concerns of program coordinators, while also accessible to the administration. It includes the following elements: University, Department, Program, Course, and Employee.
- *Research view:* tailored to the needs and concerns of project managers, while also accessible to the administration. It includes the following elements: University, Department, Employee, Project, Partner, and Type.

Figure 11.1 employs color coding to visually distinguish the elements present in each view. The admin is responsible for defining the properties and meta-elements of a view and allocating access to this view to specific user roles.

11.3.3 Access permissions

In this example, multiple roles share access to a single view. However, their access to view information must be further customized to suit their distinct responsibilities. For instance, both project managers and administrators can access the research view. Nevertheless, administrators should be limited to performing actions such as creating, reading, updating, or deleting employee data, with read-only access to other elements of the view. Conversely, project managers should be granted the ability to create, read, update, or delete project and partner information, while maintaining read-only access to all other elements of the view. The admin should define the access permissions for every role that has access to the view.

11.4 Proposed approach

The proposed approach is designed to ensure the confidentiality and integrity of model information in collaborative modeling environments through a dual-layered strategy. Essentially, the first layer filters access by defining view models on top of the base model – in a multi-view modeling fashion – and allocating roles to these views, thereby allowing interactions solely to users with the designated roles. The second layer refines this access, allowing for precise control over what each role can do within the view model through fine-grained access permissions. The solution is designed specifically for EMF and as such it is tailored to work with Ecore-based metamodels using EMF tree-based model editors. Before delving into the details and technical aspects of our approach, we clarify some core terminology. A *base metamodel* refers to a metamodel defined in terms of Ecore and representing the foundation for the views. A *base model* is a model that conforms to the base metamodel. A *view metamodel* represents a selection of elements from the base metamodel, also defined in Ecore. A *view model* conforms to a view metamodel.

Figure 11.2 provides a high-level overview of our proposed solution. For the first layer, an admin (not shown in the figure) establishes a group of *users* and *roles*, and then *assigns* users to these roles. At the same time, an admin can also define a series of *view metamodels*, which may have overlapping elements. This process results also in the generation of *view models*, subsets of the base model, and conforming to the defined *view metamodels*. A *synchronization infrastructure* is generated and maintained between view (meta)models and base (meta)model. The established roles are granted access to these views. For the second layer, the admin sets specific *permissions*, defining the extent of access each role has over a particular view. To enforce these permissions, each role uses a dedicated *model editor* for each view it can access. The total number of model editors required is the sum of those needed by each role. These model editors enforce the specified permissions, allowing users with a given role to *manipulate* the view models within their permitted scope. For instance, Sara, a program coordinator, uses model editor PC_A to manipulate the academic view model. This editor is tailored to enforce the permissions associated with the program coordinator's role when manipulating the academic view model. Any modifications that Sara makes to the academic view model are then systematically reflected in the base university model. If there are elements that overlap between the academic and research view models and Sara has altered these in the academic view, these changes are automatically propagated from the university model to the research view model.

Section 11.4.1 describes the design and technical execution of the first layer, highlighting the development of the multi-view modeling environment. Sections 11.4.2 and 11.4.3 delve into the design and technical execution of the second layer, focusing on the definition and enforcement of access permissions.

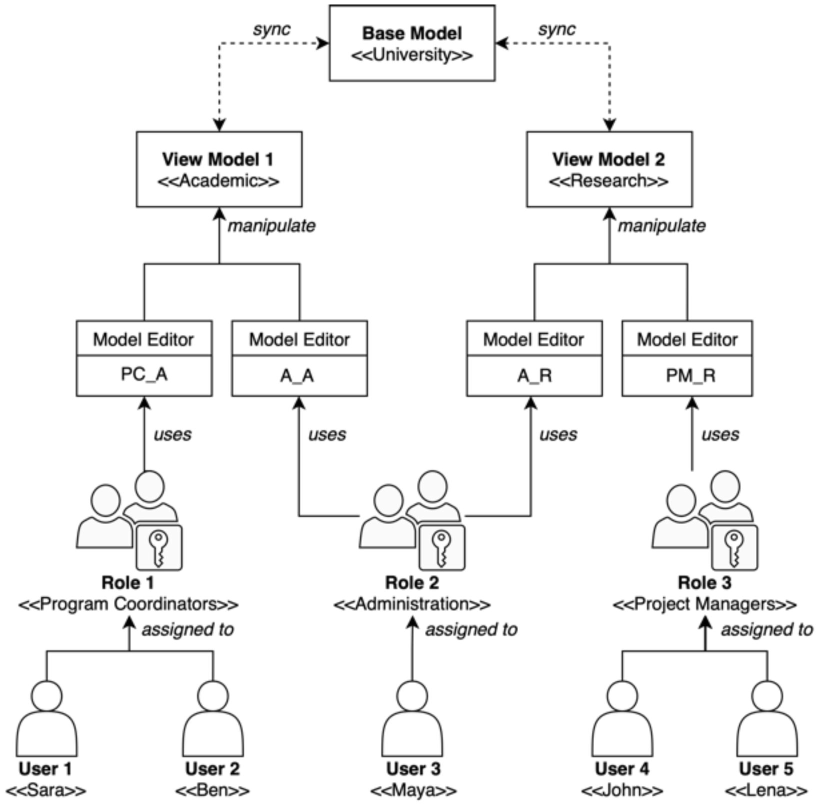


Figure 11.2. Workflow of the proposed approach

11.4.1 Setting up the multi-view modeling environment

The first step of our approach deals with determining the users and their roles. These roles will be granted access to the views, utilizing the defined wizards. The second step involves the development of the multi-view modeling environment, including the definition and generation of views, and the setup of a synchronization framework among these views.

Role and user management

The definition of users and roles is carried out by the admin via specialized Java SWT [4] wizards. The *role wizard* streamlines role management, allowing for the creation, alteration, and removal of roles. Each new role shall have a name and a description that outlines its specific functions. The *user wizard*, facilitates the management of user accounts, allowing for their creation, modification, and deletion. It necessitates details such as the user's first name, last name, username, email, password, and the roles allocated to each user. The solution supports the assignment of multiple roles to a single user, too.

View metamodel definition

The view metamodel's definition is administered through a specialized *view wizard*, comprised of several pages, each with a specific purpose. On the *details page*, the admin inputs essential parameters such as viewName, viewNSUri, and viewPrefix. In addition, it assigns the roles authorized to access the view models and loads the base model. The *selection page* displays the meta elements from the base metamodel. Here, the admin chooses the meta elements to be included in the view metamodel. Such selection follows a set of rules described in our prior work [20] for the views and the base metamodel to be consistent and their respective models to be synchronizable. In addition, for each EClass element, the admin selects a sub-element that serves as a unique identifier for matching elements between the base and view model.

View metamodel generation

View definition is followed by the generation of the view metamodel. The generation process operates by traversing the elements of the base metamodel, organized in a tree structure. For each element, it evaluates whether it has been selected by the admin. Selected elements are included in the view metamodel, while the rest are omitted. Once all elements have been processed, an Ecore Modeling Project is automatically generated. This project is configured with the necessary structure and properties. The generated view metamodel, containing

[4] `https://www.eclipse.org/swt/`

only the user-selected elements, is then saved within this project as an Ecore file.

Synchronization infrastructure

Multi-view modeling environments have two fundamental aspects: the former involves defining and creating views; the latter involves developing a synchronization infrastructure that propagates changes between the base model and view models. Considering the broad applicability of our solution across various Ecore metamodels and the potential for users to define numerous views, the synchronization infrastructure in this work is automatically generated. The generation process builds upon a previously introduced solution, where we contributed with a mapping modeling language for specifying relationships between elements within two Ecore-based metamodels, along with higher-order transformations (HOTs) that facilitate the generation of model-to-model (M2M) transformations based on these mapping models [21]. Figure 11.3 depicts the generation of the synchronization infrastructure, combining our previously proposed solution with necessary customizations for its adaptation to the current scenario, and is referenced repeatedly in this section for a detailed exploration of the synchronization infrastructure. This solution builds upon our previous work, but it sets itself apart in two main aspects.

- In the current approach, we have streamlined the process by automating the generation of mapping models. This is a significant improvement over our previous method, which requires manual user-definition of these mapping models. This enhancement is a direct outcome of the narrower and more defined context in this work, where the view metamodel forms a precise subset of the base metamodel, and every meta element in the view metamodel is identical to its counterpart in the base metamodel, having been derived directly from the latter.
- The previous approach was designed for scenarios where there is a complete mapping between the source and target metamodels, and the developed HOTs are intended for such scenarios. In our current approach, however, the view metamodel is just a part of the base metamodel, hence, not every element of the base has a corresponding element in the view.

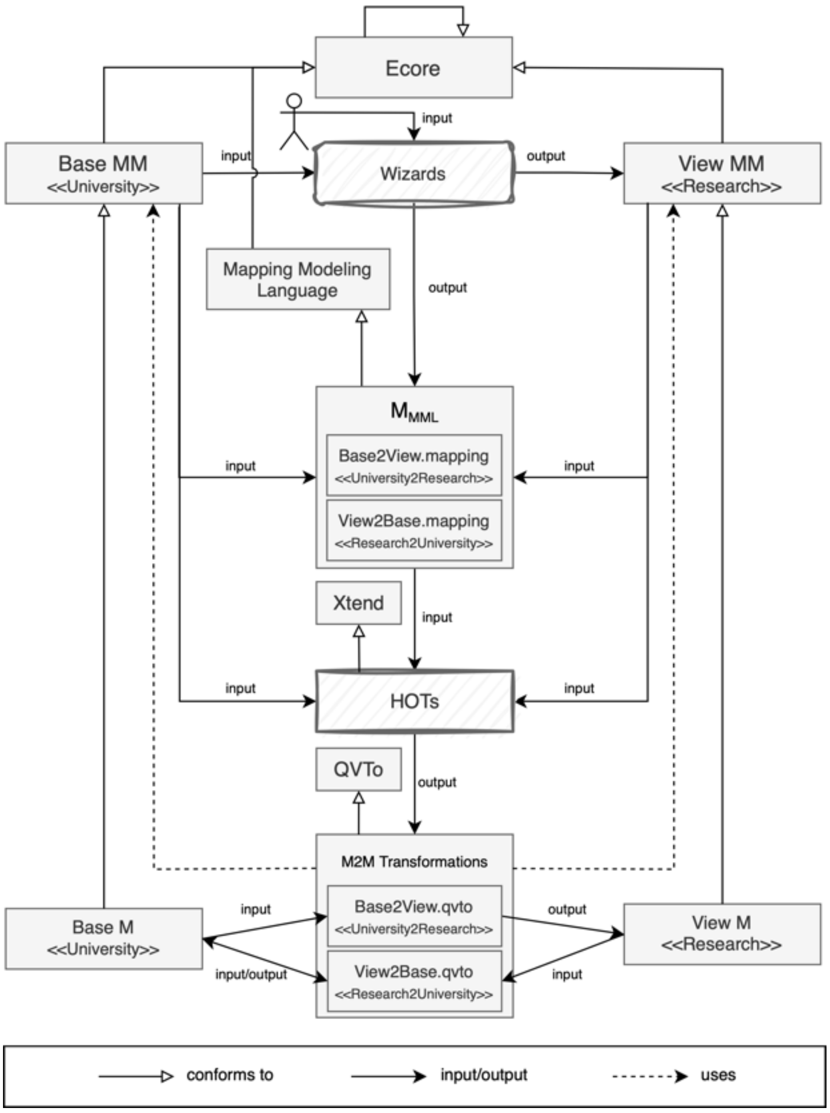


Figure 11.3. Setup of the synchronization infrastructure between base and view model

> As a consequence, employing the previously defined HOTs could lead to the generation of model transformations, erroneously modifying elements of the base model not pertaining to the view. To overcome this challenge, we have designed and implemented new HOTs specifically aimed at ensuring these transformations effectively retain all the unique elements and information of the base model.

The following paragraphs provide details on the generation of mapping models and the HOTs employed for the generation of the synchronization infrastructure (i.e., model transformations).

Generation of mapping models. The selection of elements for a particular view triggers not only the construction of an Ecore metamodel representing the view, but also the generation of two mapping models (i.e., `Base2View.mapping` and `View2Base.mapping`), which contain the relationships between elements of the base and view metamodels. These mapping models conform to our mapping modeling language, introduced in previous works [21], and encapsulate the links between the corresponding elements of the two metamodels. Given that the view metamodel represents an exact subset of the base metamodel, the process inherently ensures that each element in the view metamodel has an unambiguous correspondence with a counterpart in the base metamodel and vice versa.

Higher-order transformations. HOTs are a type of model transformation where the input and/or output are transformation models themselves [22]. They are employed to leverage the capabilities of transformations, treating them as objects. In prior research [21], we have employed HOTs that, driven by user-defined mapping models, generate M2M transformations conforming to the Query/View/Transformation Operational (QVTo)[5] language. These transformations rebuild the target model based on information from the source model. In the given context, employing these HOTs has proven effective for initially generating and subsequently updating the view model (i.e., target) from the base model (i.e., source). This is because the generated model transformation

[5] `https://wiki.eclipse.org/QVTo`

(i.e., `Base2View.qvto`) can account for every element in the view model by referencing its counterpart in the base model. However, challenges arise when attempting to propagate changes from the view model back to the base model. The M2M transformation generated from the previously defined HOTs regenerates the base model (i.e., target) to match the view model, but cannot account for all its elements since not all of them have a counterpart in the view model (i.e., source). As a result, information associated with these unmatched elements is lost during the transformation process. In response to this challenge, we developed HOTs that use the generated mapping model defining correspondences from the view model to the base model (i.e., `View2Base.mapping`) as input for generating a model transformation (i.e., `View2Base.qvto`) for propagating changes from the view model to the base model. This model transformation operates directly on the base model, updating elements that have a correspondence in the view model, all while preserving those elements that are unique to the base model. This can also be seen in Figure 11.3. In the case of `Base2View.qvto`, it takes a base model as input and produces a view model as output. On the other hand, `View2Base.qvto` takes both the view model and base model as input and directly modifies the base model to reflect changes made in the view model. In the following, we describe the generated `View2Base.qvto` model transformation.

Input/Output specifications: the model transformation uses the base and view metamodels as input and base metamodel as output. During the execution of this transformation, it accepts a base and view model as input and produces an updated base model that accurately reflects the applied changes.

Element matching: the model transformation requires matching elements between the base and view model by comparing the elements' unique identifiers defined at view definition phase. Using identifiers allows to correctly find the two corresponding elements between base and view models.

Handling of containment EReferences: can be updated, added, or deleted (i.e., their target EClass can be updated, added, or deleted). The model transformations use the information from the element matching process for differentiating between updates, additions, and deletions of elements.

- Update: an element in the view model with a match in the base model, implies that the element has neither been added or deleted, but may have been updated. Hence, an update rule is invoked to synchronize the possible changes.
- Addition: an element in the view model with no match in the base model implies the addition of the element in the view model. Thus, this element is also added to the base model.
- Deletion: an element in the base model without a match in the view model implies the deletion of the element from the view model. Thus, this element is also deleted from the base model.

Handling of non-containment EReferences: can be added or removed (i.e., their target EClass can be added or removed from the list of referenced EClasses). The model transformations use the information from the element matching process to differentiate between the two.

- Addition: a non-containment EReference in the view model, pointing to a target element with no match among the target elements of the same non-containment EReference in the base model, implies an addition. Thus, this element is also added to the list of target elements of the non-containment EReference in the base model.
- Removal: a non-containment EReference in the base model, pointing to a target element with no match among the target elements of the same non-containment EReference in the view model, implies a removal. Thus, this element is also removed from the list of target elements of the non-containment EReference in the base model.

Handling of EAttributes and EEnumLiterals: For EAttributes and EEnumLiterals, which by design cannot be added or deleted, a standard update transformation is applied. This step ensures that they are consistently updated between the view and base model.

After outlining the structure and logic of the model transformation, we leveraged Xtend [6] – a high-level, Java-based programming language noted for its efficacy in code generation tasks – to generate the model transformation from a mapping model. The generated model transformation propagates changes from the view to the base model while simultaneously ensuring that elements in the base model, which are not impacted by the transformation, remain intact. The completion of the synchronization infrastructure, achieved through the generation of two unidirectional model transformations, marks the establishment of the initial access layer. This layer permits users to access and alter only designated areas of the base model, referred to as view models, according to their allocated roles.

11.4.2 Access permissions definition

The generation of the multi-view modeling environment is followed by the definition of access permissions for each user role to interact with a given view. Definition of access permissions is carried out by the admin using CRUD (Create, Read, Update, Delete) operations [23]. The admin selects the allowed operations for each role and in relation to each meta element in the view metamodel. The selection follows a set of predefined rules outlined in the following, ensuring the consistency of access permissions.

Wizard for access permissions definition

The definition of access permissions for a set of roles on a given view is carried out using the *view wizard* introduced in Section 11.4.1. The *permissions* page is populated with the roles with access to the view and the view's meta elements. Each meta element is associated with four checkboxes, representing CRUD operations, as shown in Figure 11.4. Depending on the specific kind of EObject – be it an EClass, a containment or non-containment EReference, EAttribute, EENum, or EENumLiteral – only certain checkboxes are active and selectable. Other checkboxes are inactive since the operations that they correspond to do

[6] `https://eclipse.dev/Xtext/xtend/documentation/index.html`

not apply to the EObjects they are associated with. An overview of checkbox status for each EObject type is provided in Table 11.1.

	C	R	U	D
EClass	Active	Active	Inactive*	Active
ERef (containment)	Active	Active	Inactive	Active
ERef (non-containment)	Inactive	Active	Active	Inactive*
EAttribute	Inactive	Active	Active	Inactive*
EENum	Inactive	Active	Inactive	Inactive
EENumLiteral	Inactive	Active	Inactive	Inactive

Table 11.1. Checkbox status per type of EObject

To maintain a consistent layout that helps users navigate the wizard interface more intuitively, we have retained the inactive checkboxes in the wizard. Although not directly selectable, they serve to provide a clear and coherent structure. Additionally, checkboxes marked with an asterisk (*) are designed to be automatically activated in response to the selection of certain related checkboxes, even though they remain inactive for direct user interaction. For instance, when a user selects the Create (C), Update (U), or Delete (D) checkboxes for any EAttribute or EReference within an EClass, the solution triggers the selection of the Update (U) checkbox for that EClass. Similarly, selecting the Delete (D) checkbox for an EClass triggers the automatic selection of the Delete (D) checkbox for all contained EReferences and EAttributes.

Consistency rules

The mechanism for automatically managing checkbox states goes beyond just handling inactive ones. We established a comprehensive set of rules to ensure uniform behavior across all checkboxes. For instance, if an EClass is removed, it naturally entails the removal of its associated EAttributes, EReferences, and any EClasses linked through containment EReferences. This reasoning is embedded in the wizard to avoid potential errors in the enforcement of permissions. The wizard updates checkbox statuses (either selecting or deselecting them) on-the-fly. This real-time mechanism gives users a clear understanding of how their choices affect the permissions of related meta-elements. To define the consistency rules, we started with overarching principles. These principles served as the foundation for defining the rules. The latter are presented in Table 11.2 and their interpretation is facilitated by the legend provided in Table 11.3. Each rule is directly linked to the underlying principles, which are outlined below.

P1: Permission to perform a create (C), update (U), or delete (D) operation on an object is conditional to having the read (R) permission on the object – see R1, R5 in Table 11.2.

P2: Permission to perform any CRUD operation on a nested object is conditional to having the read (R) permission on the container object. In addition, any CUD operation on a nested object is conditional to having the update (U) permission on the container object – see R2, R3, R6, R7.

P3: Permission to perform a delete (D) operation on a container object is conditional to having the delete (D) permission on all nested objects – see R4.

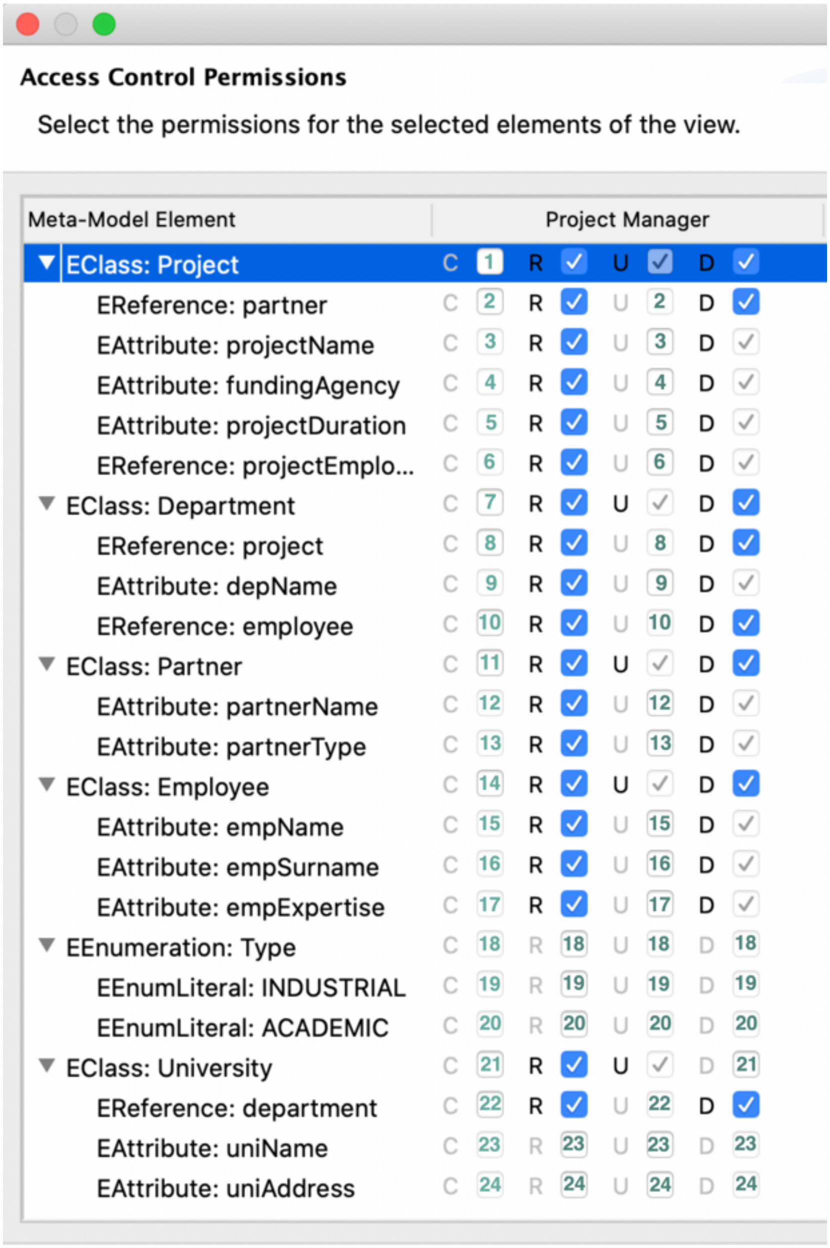
P4: Permission to perform any CRUD operation on an EClass is conditional to having those permissions on its incoming containment EReference and the container of that EReference (the latter is based on P2) – see R8 to R11, R8 toR11.

P5: Permission to perform any CRUD operation on an EReference is conditional to having those permissions on its source and target EClass – see R12 to R17, R12 to R15.

P6: Permission to perform a read (R) or update (U) operation on an EAttribute is conditional to having the read (R) permission on the container EClass – see R18 to R21.

P7: Permission to perform a read (R) operation on an EEnumLiteral is conditional to having the read (R) permission on the container EEnum – see R22, R23.

In alignment with the established principles, we formulated the set of consistency rules delineated in Table 11.2. The first column lists the identifiers of the rules, the second outlines the user actions in the wizard interface, the third describes the effects of these actions on other checkboxes, and the fourth provides concrete examples for each rule, based on the wizard shown in Figure 11.4. The user actions and the examples have been structured to ensure a precise understanding of the effects. For clarity, we tried out all examples involving the selection of a specific checkbox with the wizard in a baseline state, with all checkboxes initially deselected. Similarly, for the deselection of checkboxes, we consistently used the same checkbox that was previously selected. This methodological consistency allowed us to isolate the effects of each action without interference. It is important to note that the user's interaction with a checkbox (either through selection or deselection) can initiate a ripple effect, where each affected checkbox might further alter the state of others. This chain reaction continues until all affected checkboxes are appropriately adjusted. To illustrate, assume all checkboxes in Figure 11.4 are initially deselected. If a user selects checkbox R(9) – which corresponds to the read (R) operation for `EAttribute:depName` at index 9 – this action triggers the automatic selection of the read (R) operation checkbox for `EClass:Department`, by principle P2. Following this, selecting R(7) would lead to the selection of R(22), as dictated by principle P4. Subsequently, selecting R(22) would result in the selection of R(21), again following principle P2. This example represents a three-level chain reaction within the checkbox interactions (i.e., R(9) $\xrightarrow{\text{L1}}$ R(7) $\xrightarrow{\text{L2}}$ R(22) $\xrightarrow{\text{L3}}$ R(21)). For the sake of readability and conciseness, in Table 11.2 we limited the illustration of chain reactions to just the second level. This applies to both the examples shown and their effects, to avoid the complexity of longer chains that could span multiple levels.



Access Control Permissions

Select the permissions for the selected elements of the view.

Meta-Model Element	Project Manager							
EClass: Project	C	1	R	☑	U	✓	D	☑
EReference: partner	C	2	R	☑	U	2	D	☑
EAttribute: projectName	C	3	R	☑	U	3	D	✓
EAttribute: fundingAgency	C	4	R	☑	U	4	D	✓
EAttribute: projectDuration	C	5	R	☑	U	5	D	✓
EReference: projectEmplo...	C	6	R	☑	U	6	D	✓
EClass: Department	C	7	R	☑	U	✓	D	☑
EReference: project	C	8	R	☑	U	8	D	☑
EAttribute: depName	C	9	R	☑	U	9	D	✓
EReference: employee	C	10	R	☑	U	10	D	☑
EClass: Partner	C	11	R	☑	U	✓	D	☑
EAttribute: partnerName	C	12	R	☑	U	12	D	✓
EAttribute: partnerType	C	13	R	☑	U	13	D	✓
EClass: Employee	C	14	R	☑	U	✓	D	☑
EAttribute: empName	C	15	R	☑	U	15	D	✓
EAttribute: empSurname	C	16	R	☑	U	16	D	✓
EAttribute: empExpertise	C	17	R	☑	U	17	D	✓
EEnumeration: Type	C	18	R	18	U	18	D	18
EEnumLiteral: INDUSTRIAL	C	19	R	19	U	19	D	19
EEnumLiteral: ACADEMIC	C	20	R	20	U	20	D	20
EClass: University	C	21	R	☑	U	✓	D	21
EReference: department	C	22	R	☑	U	22	D	☑
EAttribute: uniName	C	23	R	23	U	23	D	23
EAttribute: uniAddress	C	24	R	24	U	24	D	24

Figure 11.4. Permissions triggered by selection of D(22)

Rule	Action	Effect	Example
General rules			
R1	$P_{C\|U\|D}(O)$	$P_R(O)$	$C(21) \Rightarrow R(21)$
R2	$P_R(N)$	$P_R(C)$	$R(23) \Rightarrow R(21)$
R3	$P_{C\|U\|D}(N)$	$R1 \wedge P_{RU}(C)$	$U(23) \Rightarrow R(23) \wedge RU(21)$
R4	$P_D(C)$	$P_D\forall(N) \rightarrow R3\forall(N)$	$D7 \Rightarrow D(8-10) \rightarrow [R(7-10) \wedge U(7)]$
R5	$P_R(O)$	$P_{CUD}(O)$	$R(21) \Rightarrow$ CUD(21)
R6	$P_R(C)$	$R5 \wedge P_{CRUD}\forall(N)$	$R(21) \Rightarrow$ CUD(21) $\wedge CRUD(22-24)$
R7	$P_{CUD}\forall(N)$	$P_U(C)$	$CUD(22-24) \Rightarrow U(21)$
EClass (Class) specific consistency rules (not the root)			
R8	$P_{C\|R\|D}(Class)$	$P_{C\|R\|D}(CR)$	$R(7) \Rightarrow R(22)$
R8	$P_{C\|R\|D}(Class)$	$P_{C\|R\|D}(CR)$	$R(7) \Rightarrow R(22)$
R9	$P_C(Class)$	$R1 \wedge [P_C(inCR) \rightarrow R3(inCR)]$	$C(7) \Rightarrow R(7) \wedge [CR(22) \rightarrow RU(21)]$
R9	$P_C(Class)$	$P_C(inCR) \rightarrow R7(inCR)$	$C(7) \Rightarrow C(22) \rightarrow U(21)$
R10	$P_R(Class)$	$P_R(inCR) \rightarrow R2(inCR)$	$R(7) \Rightarrow R(22) \rightarrow R(21)$
R10	$P_R(Class)$	$P_R(inCR)$	$R(7) \Rightarrow R(22)$
R11	$P_D(Class)$	$[R4 \rightarrow R13(CR)] \wedge$ $[P_D(inCR) \rightarrow R15(inCR)]$	$D7 \Rightarrow$ (see Figure 11.4)
R11	$P_D(Class)$	$R8 \rightarrow R7$	$D(7) \Rightarrow D(22) \wedge U(21)$
Containment EReference (CR) specific consistency rules			
R12	$P_{C\|R\|D}(CR)$	$P_{C\|R\|D}(tClass)$	$R(22) \Rightarrow R(7)$
R12	$P_{C\|R\|D}(CR)$	$P_{C\|R\|D}(tClass)$	$C(22) \Rightarrow C(7)$
R13	$P_C(CR)$	$R3 \wedge [R12 \rightarrow R1(tClass)]$	$C(22) \Rightarrow R(22) \wedge RU(21) \wedge [C(7) \rightarrow [R(7)]$
R13	$P_C(CR)$	$R7 \wedge R12$	$C(22) \Rightarrow U(21) \wedge C(7)$
R14	$R(CR)$	$R2 \wedge R12$	$R(22) \Rightarrow R(21) \wedge R(7)$
R14	$P_R(CR)$	$R12$	$R(22) \Rightarrow R(7)$
R15	$P_D(CR)$	$R3 \wedge [R12 \rightarrow R11(tClass)]$	$D(22) \Rightarrow$ (see Figure 11.4)
R15	$P_D(CR)$	$R7 \wedge R12$	$D(22) \Rightarrow U(21) \wedge D(7)$
Non-Containment EReference (NCR) specific consistency rules			
R16	$P_R(NCRef)$	$[R2 \rightarrow R10(sClass)] \wedge$ $[P_R(tClass) \rightarrow R10(tClass)]$	$R(6) \Rightarrow [R(1) \rightarrow [R(8) \rightarrow R(7)]] \wedge [R(14) \rightarrow [R(10) \rightarrow R(7)]]$
R17	$P_U(NCRef)$	$R3 \rightarrow R16$	$U(6) \Rightarrow [R(6) \rightarrow [R(1) \rightarrow [R(8) \rightarrow R(7)]]] \wedge [R(14) \rightarrow [R(10) \rightarrow R(7)]]$
EAttribute (Att) specific consistency rules (R20 and R21: Attribute EType = ENum)			
R18	$P_R(Att)$	$R2 \rightarrow R10(C)$	$R(12) \Rightarrow R(11) \rightarrow R(2-1)$
R19	$P_U(Att)$	$R3 \rightarrow R10(C)$	$U(12) \Rightarrow [R(12) \wedge RU(11)] \rightarrow R(2-1)$
R20	$P_R(Att)$	$R18 \wedge [P_R(EN) \rightarrow R22(EN)]$	$R(13) \Rightarrow [R(11) \rightarrow R(2-1)] \wedge [R(18) \rightarrow R(19-20)]$
R21	$P_U(Att)$	$R19 \wedge [P_R(EN) \rightarrow R22(EN)]$	$U(13) \Rightarrow [R13 \wedge [R(11) \rightarrow R(2-1)]] \wedge [R(18) \rightarrow R(19-20)]$
ENum (EN) and ENumLiteral (ENL)specific consistency rules			
R22	$P_R(ENL)$	$P_R(EN)$	$R(19) \Rightarrow R(18)$
R23	$P_R(EN)$	$P_R\forall(ENL)$	$R(18) \Rightarrow R(19-20)$

Table 11.2. Set of rules for consistency enforcement between access permissions

Legend	
The acronyms employed in Table 11.2 are as follows: object (O), container object (C), nested object (N), containment reference (CR), non-containment reference (NCR), incoming containment reference (inCR), source class (sClass), target class (tClass), attribute (Att), enumeration (EN), enumeration literal (ENL).	
$P_X(O)$ and $P_X(O)$	Selection and deselection of a permission X ={C,R,U,D} on an object O
$P_X\forall(O)$ and $P_X\forall(O)$	Selection and deselection of a permission X ={C,R,U,D} on all objects O
$P_{C\|U\|D}(O)$ and $P_{C\|U\|D}(O)$	Selection and deselection of at least one of the specified permissions
$P_{CUD}(O)$ and $P_{CUD}(O)$	Selection and deselection of all the specified permissions

Table 11.3. Legend for reading Table 11.2

11.4.3 Access permissions enforcement

The enforcement of the defined access permissions is achieved through EMF tree-based model editors. These editors provide a graphical user interface that allows users to visualize and modify models through a hierarchical tree structure. While they come with essential functionalities, their design allows for customization to meet specific needs. We leverage this flexibility to enforce the defined access permissions. The following paragraphs detail EMF's standard approach to creating these tree-based model editors and the customization methods we have employed for generating model editors that enforce the defined access permissions.

Generation of EMF tree-based model editors

EMF tree-based model editors are generated through an automated process supported by the framework itself. Starting from an Ecore model, the process first involves the creation of a generator model (GenModel). The GenModel is a configuration model that dictates the generation of the following plugins.

- *Model plugin:* hosts the Java classes that represent the Ecore model. These classes are derived from the Ecore model, conforming to the GenModel's specifications.

- *Edit plugin:* provides infrastructure for structured interaction with the model. The key components of this plugin are the Item Providers. Item Providers are Java classes that support viewing and editing objects within the EMF tree-based editor. They specify how model elements are presented and manipulated in the editor.

- *Editor plugin:* comprises the graphical user interface for the EMF editor. This includes the tree-based interface for model interaction and additional UI components like wizards that enhance user engagement with the model.

The generation process of these plugins involves the use of Java Emitter Templates (JET)[7]. EMF comes with a suite of JET templates that are written in a (Java Server Pages) JSP-like syntax and express EMF code patterns. These templates undergo processing by the JET engine, which converts each template into the source of a Java class. Subsequently, the JET engine compiles, dynamically loads, and utilizes these classes to produce the specified output. By default, EMF uses *static templates* which are converted into template classes and compiled ahead of time to speed up the generation process [13].

Customization of EMF tree-based model editors

To enforce the user-defined access permissions in EMF tree-based model editors, we focus on customizing Item Providers, which manage the viewing and editing of elements. While one could customize each generated Item Provider separately, this approach is time-consuming and inefficient. Our methodology, therefore, adopts a streamlined and automated process where we initially customize the JET template utilized for generating Item Providers to consider the defined access permissions. Then, we guide the generator to use the customized JET template, which is also referred to as *dynamic* because, unlike static templates, it has not been pre-compiled. This process is described in Figure 11.5. When the admin finalizes the *view wizard*, it initiates the generation of *view.ecore*, *view.genmodel*, and *view.rbac*. The *view.rbac* file represents a model conforming to a custom-designed language (detailed in in the following) which encapsulates the defined access permissions. The *view.ecore* file encapsulates the specifics of the established classes, while the *view.genmodel* file contains essential information required for code generation. When generating the *view.genmodel*, we specified particular properties to instruct the generator to utilize the customized JET template. Specifically, we enabled the `dynamic templates` property by setting its value to `true`, guiding the system to skip the use of default templates from `org.eclipse.emf.codegen.ecore.genmodel`

[7] `https://projects.eclipse.org/projects/modeling.m2t.jet`

and instead choose to translate and compile the JET templates we provided. Moreover, we configured the `template directory` to point to the location of the dynamic templates in our project. The combination of these sources of information enables the EMF code generator to generate a tree-based model editor enforcing the defined access permissions.

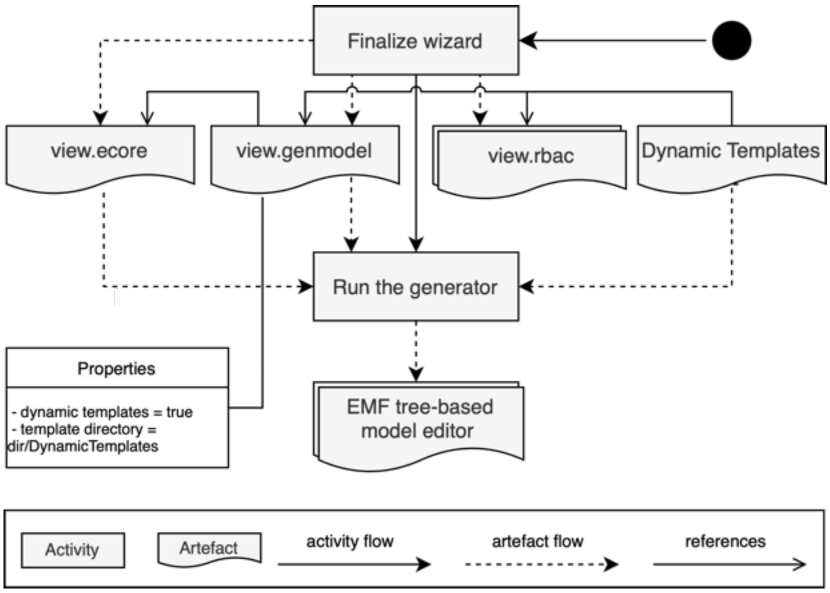


Figure 11.5. Generation workflow of tree-based model editors with RBAC

The advantages of this approach are multiple. Firstly, we automate the customization process of Item Providers, avoiding the need for individual manual modifications. Secondly, by embedding the access permissions directly into the generation process of the Item Providers, we ensure a consistent implementation of these permissions across the model editor. Finally, we offer flexibility and adaptability, allowing for straightforward updates to access permissions without extensive manual reworking of the underlying Item Provider code. The following paragraphs describe the role-based access control domain-specific modeling language (DSML) and the customizations made to the JET template responsible for the generation of Item Providers.

Role-based access control DSML

To encapsulate the user-defined access permissions, we defined a role-based access control DSML. For each role that has access to a particular view, a unique RBAC model is generated. These RBAC models encapsulate the access permissions each role has over each element within the view. We intentionally structured the DSML to imitate the structure of an Ecore metamodel for seamless integration as input to the JET template and possible reverse engineering needed for future work. As depicted in Figure 11.6, the root element is the *AccessControlModel* characterized by attributes *name* – reflecting the name of the view metamodel – and *role*, denoting the user role to which the permissions apply. The elements *EClass* and *EEnum* are compositionally linked to the root element. In a similar compositional manner, *EAttribute* and *EReference* are linked to *EClass*, while *EEnumLiteral* to *EEnum*. Each of these elements possesses a set of features that they inherit from the *ElementPermission* abstract class. These features include a *name* attribute for the EObject they are associated with, and a reference to the EObject *element*. Moreover, they feature a list of *permissions* that define the CRUD operations, as detailed in the *Permissions* enumeration.

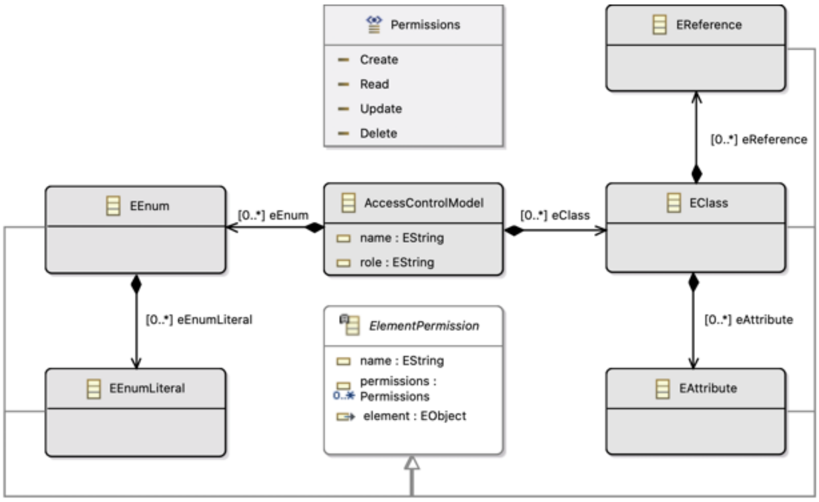


Figure 11.6. Role-based access control DSML

Dynamic templates

EMF utilizes a collection of JET templates to automatically generate tree-based model editors. To refine this generation process to enforce access permissions within the model editor, we adapted the JET template responsible for Item Provider generation. This adaptation involves using a generated RBAC model as input, directing the generation of Item Providers in a way that enforces the specified permissions. Initially, we duplicated the JET templates into our project and we modified the `ItemProvider.javajet` template accordingly. Since the tree-based editor's default setup allows for unrestricted CRUD operations on all model elements, we intervened to implement constraints where the out-of-the-box functionality permits more than what the RBAC model dictates. Specifically, our custom logic in the template evaluates the RBAC model's permissions for each model element and adjusts the generation of Item Providers to enforce these restrictions accordingly. For each element, depending on which operations are restricted, the template selectively generates or omits code segments in the Item Providers. For instance, if the RBAC model restricts the creation of certain EClasses for specific roles, the template is designed to omit the generation of code segments in Item Providers that would otherwise enable such creation capabilities. This principle is similarly applied for all the remaining operations. The mechanisms described so far represent the realization of the second security layer which permits users to manipulate the elements of the accessible view models according to the access permissions defined for each element. The interested reader can find further details on the required adjustments to the Item Providers for restricting each operation type in our GitHub repository [8].

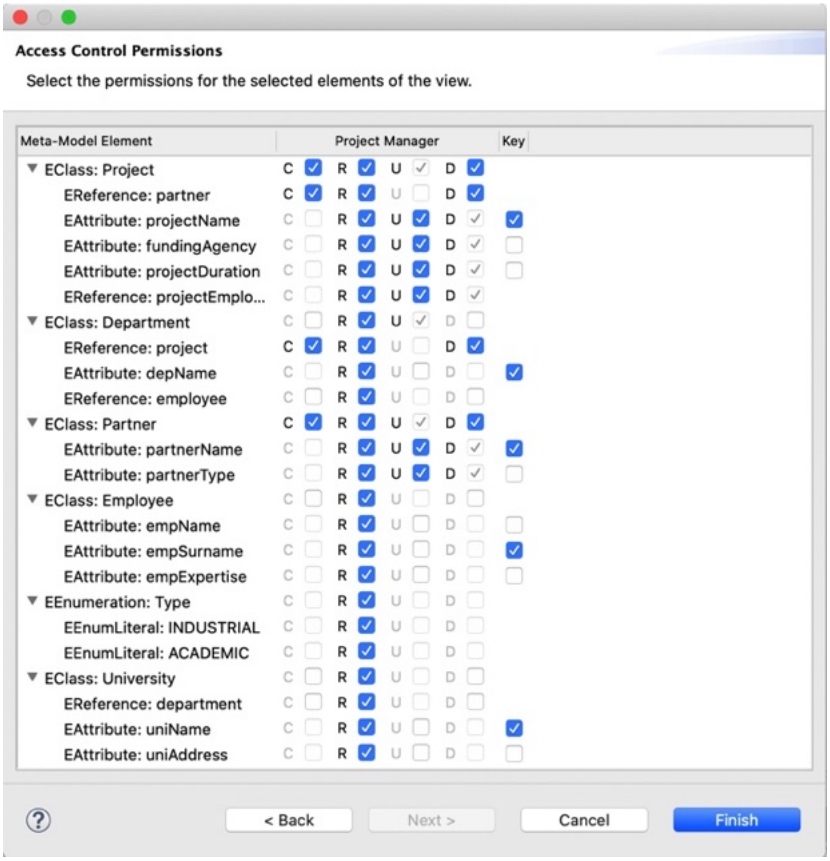
11.5 Illustrative example

The approach has been applied to the running example described in Section 11.3 and a demo has been made available in our GitHub repository [8]. Figure 11.7 illustrates the *permissions'* page of the view wizard. In this example, we define the *project manager's* access permissions on the *research view*. The first column of the wizard displays the meta model elements that comprise the *research view*,

[8] `https://github.com/MLJworkspace/RoleBasedAccessControl`

selected in prior step. The second column features four checkboxes for each view element, representing the complete set of CRUD operations. The final column includes a checkbox for each EAttribute object, enabling the user to select a unique identifier for each EClass. The wizard ensures real-time consistency in access permissions through a dynamic check activated each time a checkbox is selected. This process may prompt the selection of additional checkboxes to maintain consistency. For instance, selection of delete (D) operation on `EClass:Project` leads to the automatic selection of read (R) operation on `EClass:Project` , selection of delete (D) permission on all its children, and selection of delete (D) operation on `EReference:project`. This cascade of automatic selections continues, ensuring all relevant checkboxes are selected to preserve the consistency of access permissions. Finalization of the wizard results in the creation of several artefacts, including the view metamodel and the corresponding view model, where the latter is a subset of the base model. The base model illustrated in Figure 11.8a is an instantiation of the university (base) metamodel illustrated in Figure 11.1, while the view model illustrated in Figure 11.8b, is an instantiation of the research (view) metamodel comprised of the view elements illustrated in Figure 11.7. Model transformations, generated upon finalizing the wizard, ensure that modifications in the base and view models are accurately propagated among one another, and guarantee that information is preserved without loss during the propagation of changes. For example, when the project is renamed from BUMBLE to Orpheus in the view model, this change is propagated in the base model. Simultaneously, the program and course elements, which exist in the base model but not in the view model (nor are they part of the view metamodel), remain intact.

In the context of enforcing access permissions, the models illustrated in Figure 11.8 are accessed through their corresponding customized EMF tree-based model editors. The comparison between the two reveals distinct permission settings. In the base model, it is possible to delete a *department* instance and create new *projects*, *employees*, and *programs*. Conversely, the view model restricts these capabilities. Here, deleting a *department* instance is disabled, and the creation of *employees* and *programs* is prohibited. These limitations align with the access permission established in Figure 11.7, which explicitly restricts project managers from performing these operations. Access permission



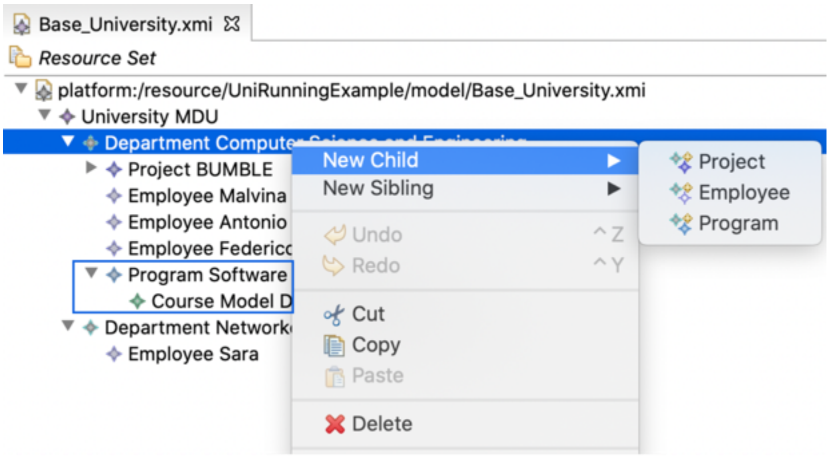
Access Control Permissions
Select the permissions for the selected elements of the view.
< Back
Next >
Cancel
Finish

Meta-Model Element	Project Manager				Key
	C	R	U	D	
▼ EClass: Project	✓	✓	✓	✓	
EReference: partner	✓	✓	☐	✓	
EAttribute: projectName	☐	✓	✓	✓	✓
EAttribute: fundingAgency	☐	✓	✓	✓	☐
EAttribute: projectDuration	☐	✓	✓	✓	☐
EReference: projectEmplo...	☐	✓	✓	✓	
▼ EClass: Department	☐	✓	✓	☐	
EReference: project	✓	✓	☐	✓	
EAttribute: depName	☐	✓	☐	☐	✓
EReference: employee	☐	✓	☐	☐	
▼ EClass: Partner	✓	✓	✓	✓	
EAttribute: partnerName	☐	✓	✓	✓	✓
EAttribute: partnerType	☐	✓	✓	✓	☐
▼ EClass: Employee	☐	✓	☐	☐	
EAttribute: empName	☐	✓	☐	☐	☐
EAttribute: empSurname	☐	✓	☐	☐	✓
EAttribute: empExpertise	☐	✓	☐	☐	☐
▼ EEnumeration: Type	☐	✓	☐	☐	
EEnumLiteral: INDUSTRIAL	☐	✓	☐	☐	
EEnumLiteral: ACADEMIC	☐	✓	☐	☐	
▼ EClass: University	☐	✓	☐	☐	
EReference: department	☐	✓	☐	☐	
EAttribute: uniName	☐	✓	☐	☐	✓
EAttribute: uniAddress	☐	✓	☐	☐	☐

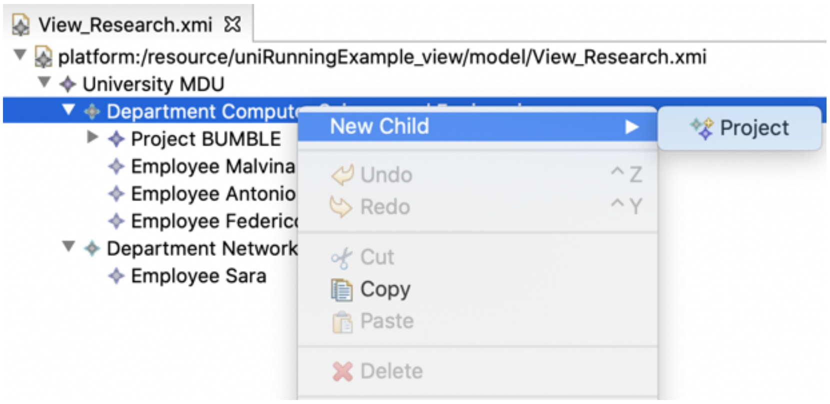
Figure 11.7. Defined access permissions for llustrative example

enforcement is achieved by customizing the generation process of the Item Providers – responsible for viewing and editing elements in the model editor – which in the customized version, considers the permissions defined in the wizard in Figure 11.7 and generates the Item Providers accordingly.

Listing 11.1 presents a code snippet that illustrates the createRemoveCommand() method as implemented in the University ItemProvider. This method is not generated by the standard generation process. Instead, it is specifically generated due to the project manager's lack of delete (D) permission for the



(a) Editor providing full access to the base model



(b) Editor providing limited access to the view model

Figure 11.8. Tree-based editors for base and view models

`EClass:Department`. As a result, this method disables the Delete option when right-clicking on a Department element, a functionality highlighted in Figure 11.8b.

```
public Command createRemoveCommand (EditingDomain editingDomain,
    EObject owner, EStructuralFeature feature, Collection<?> collection
    ) {
```

```
    if (owner instanceof University) {
        for (Object object : collection) {
            if (object instanceof Department) {
                return UnexecutableCommand.INSTANCE;
            }
        }
    }
    return RemoveCommand. create (editingDomain, owner, feature,
    collection);
}
```

Listing 11.1. createRemoveCommand() method for Department in University ItemProvider

11.6 Related work

Considerable research has been dedicated to exploring access control mechanisms and their application across various domains. In previous works [20, 16], we propose a hybrid strategy for multi-view modeling that leverages an arbitrary number of views built atop a base modeling language. However, we only entailed the specification of basic read-only and editable permissions. Martinez et al. [10] introduce a method that leverages views as mechanisms for enforcing access control, utilizing EMFViews [24] to dynamically generate views through live queries on the base model. The elements within these virtual models serve merely as proxies for the actual elements in the base model; hence, view models remain transient and can neither be viewed nor edited independently in other modeling contexts, since they are tightly linked to the base model. By having self-persistent views our approach supports offline collaboration, but also benefits from the versatility of a typical metamodel, such as linking any desired concrete syntax or be modified if necessary, including extensions or additional abstraction layers. Other works [25, 26, 27, 11] propose a secure collaborative modeling framework utilizing lenses to generate and maintain synchronized secure views with the underlying base model. Lenses support bidirectional model transformation mechanisms called GET and PUTBACK. The GET function controls read access by filtering the gold (base) model into a front (view) model based on read permissions, while the PUTBACK function validates front model alterations against write permissions before potentially updating the gold model. Hence, the enforcement of access permissions is done by the bidirectional transformations, whereas in our work, this enforcement is managed by the model editor, and is not dependent on the underlying synchronization infrastructure. In addition, this approach to change propagation informs users about their editing permissions through a trial-and-error process, a method the authors themselves acknowledge could lead to user frustration and inefficiency due to delayed feedback on unauthorized modifications. Connected Data Objects (CDO)[9] model repository by Eclipse follows a similar approach in which users are only informed of restrictions on write operations at the time of commit. Our approach adopts a preventive model by integrating a model editor designed to enforce editing permissions. This ensures that users can only execute changes they are explicitly authorized to make. Additionally, while the existing approaches are focused on read and write operations, our solution extends the functionality to CRUD operations, thus providing a more fine-grained control over model interactions. Another relevant study [28] explores the definition of finely-grained role-based access control, but it targets mobile collaborative modeling with active DSLs with a primary focus on mobility aspects. An alternative path of research in access control management proposes implementing security policies directly at the file level. A notable example is Apache Subversion [10], which provides administrators the capability to enforce path-based authorization, thereby controlling user access to specific segments of the repository. Within a collaborative MDSE environment, this approach necessitates the division of models into distinct files. Such fragmentation may obstruct seamless collaboration and is limited to enforcing access control policies with relatively coarse granularity. As can be observed, our discussion in this section is intentionally directed towards strategies that predominantly address access control rather than the broader scope of synchronization in multi-view modeling. This decision stems from our research's use of multi-view modeling primarily as a tool for achieving finely-grained access control efficiently. However, we acknowledge the existence of other approaches for providing synchronization in multi-view modeling, such as view triple graph grammars (VTGGs) [29, 30] which similar to EMFViews, provide non-materialized views, and lenses [31]. Since the en-

[9] `https://wiki.eclipse.org/CDO/Security_Manager`

[10] `https://subversion.apache.org`

forcement of permissions is separate from the synchronization infrastructure, and the consistency rules applied during the definition of views and permissions ensure the well-formedness of the view model, any appropriate synchronization transformation approach can be utilized.

11.7 Discussion

Benefits of the proposed solution. The proposed solution enhances both the confidentiality and integrity of model information in collaborative modeling environments. Simultaneously, it streamlines the individual modeling experience by minimizing the information overload that results from interacting with a comprehensive base model. The latter is achieved by offering customized view (meta)models, allowing users to engage with only the relevant aspects of the base (meta)model. The solution employs the RBAC policy and supports the definition of fine-grained access permissions using CRUD operations. Central to the solution is the implementation of automatic consistency rules for access permissions, alongside the automatic enforcement of the latter by the model editor. This automation plays a crucial role in mitigating the risk of human errors that could compromise the access permissions' consistency and their enforcement. The solution allows for immediate adjustments to access permissions, with changes being automatically reflected in the model editor. This flexibility ensures that the system can swiftly respond to alterations in project requirements and team composition. Incorporating these capabilities, the solution effectively meets the requirement for a fine-grained, consistent, and flexible access control system in collaborative modeling settings. It not only safeguards shared model information, mitigating the risks of confidentiality breaches and ensuring the integrity of model information, but it also enhances the collaborative process, fostering a more efficient and effective environment for teamwork. This is achieved by supporting users to perform efficiently in their designated sections of a shared model, thus enhancing both individual and collective productivity.

Limitations of the proposed solution. The solution is designed for Ecore-based metamodels as part of the Eclipse Modeling Framework (EMF). Ecore, a central component of EMF, is widely used in diverse modeling scenarios.

However, the focus on Ecore-based metamodels restricts the application of our solution to meta-metamodeling languages other than Ecore. Additionally, the access permissions are currently implemented solely in EMF tree-based editors since they are relatively straightforward for users to generate out-of-the-box and are essential for manipulating instances of Ecore-based metamodels. This choice, while pragmatic, does not support the diversity of user needs for different notations and model editors. Hence, we consider our initial implementation a stepping stone, with plans to broaden our access control solutions to encompass both textual and graphical editors in future enhancements. Moreover, our current focus is on the authorization aspect of access control. While this approach effectively allows users to define roles and user profiles, it does not yet incorporate an authentication mechanism. The implication of this is that, as of now, the system does not restrict user access to model editors based on their assigned roles upon login, marking a clear path for future enhancement. In terms of synchronization infrastructure, our approach relies on user-selected unique identifiers for element matching, which poses a risk of inconsistencies. Additionally, the synchronization process is manually triggered, highlighting the need for automated change detection and conflict resolution strategies. Overall, these limitations, mostly of implementative nature, reflect our choice to focus on access control related aspects in the scope of this initial solution, laying the groundwork for future developments that will address these initial constraints and broaden the solution's applicability and functionality.

11.8 Conclusions and future work

This article proposes a dual-layered approach that provides role-based access control to ensure the confidentiality and integrity of model information in collaborative modeling environments. The first layer focuses on the creation of view (meta)models on top of a base (meta)model, along with a seamless synchronization mechanism between them. This efficiently restricts access to the base model, limiting users to interact solely with the view models assigned to their respective roles. The second layer fine-tunes this access by establishing fine-grained access permissions describing the permissible operations over each element type within the view models. Together, the two layers contribute to enhancing the confidentiality and integrity of shared model information in collaborative modeling. For future work, we plan to extend the solution to apply to blended editors [32]. Moreover, we aim to extend this research by focusing on establishing permissions at the instance level. This will allow assigning permissions to each instance, rather than applying a one-size-fits-all rule to all instances of a given meta element. Future work will also combine the RBAC policy with the attribute-based access control (ABAC) policy, adding restrictions based on attributes, since the current approach can be easily adapted to include additional access control policies without altering the underlying framework for defining and enforcing permissions. For example, in ABAC, for attributes that are static and manually defined this can be achieved by: (i) establishing a method to define and reference attributes during permission specification, and (ii) implementing a method to extract attribute values and enforce permissions accordingly. Attributes that are dynamic, like location, necessitate distinct mechanisms for detection and value extraction. The methodologies proposed by [28] offer valuable insights into addressing the latter scenario. In addition, in future work, the adaptability of the theoretical approach is intended to be explored across various modeling frameworks beyond EMF. Our analysis into the dependencies among model elements and CRUD operations has yielded reusable consistency rules applicable across various technological stacks, potentially requiring only minimal adjustments. Moreover, the strategy for enforcing permissions by utilizing predefined permissions as inputs in the model editor generation process can be extended to other modeling platforms. However, a more profound examination of existing modeling platforms is essential to comprehend both the similarities and differences with our implementation. Finally, upcoming efforts will be dedicated to assessing the proposed solution in industrial contexts to analyze scalability aspects.

Bibliography

[1] Douglas C Schmidt. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.

[2] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE'07)*, pages 37–54. IEEE, 2007.

[3] Jim Whitehead. Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*, pages 214–225. IEEE, 2007.

[4] Ivan Mistrík, John Grundy, Andre Van der Hoek, and Jim Whitehead. *Collaborative software engineering: challenges and prospects*. Springer, 2010.

[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

[6] Henry Muccini, Jan Bosch, and André van der Hoek. Collaborative modeling in software engineering. *IEEE Software*, 35(6):20–24, 2018.

[7] Davide Di Ruscio, Mirco Franzago, Ivano Malavolta, and Henry Muccini. Envisioning the future of collaborative model-driven software engineering. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 219–221. IEEE, 2017.

[8] Istvan David, Kousar Aslam, Ivano Malavolta, and Patricia Lago. Collaborative model-driven software engineering—a systematic survey of practices and needs in industry. *Journal of Systems and Software*, 199:111626, 2023.

[9] Michael Nieles, Kelley Dempsey, and Victoria Yan Pillitteri. An introduction to information security. *NIST special publication*, 800(12):101, 2017.

[10] Salvador Martínez, Alexis Fouche, Sébastien Gérard, and Jordi Cabot. Automatic generation of security compliant (virtual) model views. In *Conceptual Modeling: 37th International Conference, ER 2018, Xi'an, China, October 22–25, 2018, Proceedings 37*, pages 109–117. Springer, 2018.

[11] Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró. Enforcing fine-grained access control for secure collaborative modelling using bidirectional transformations. *Software & Systems Modeling*, 18:1737–1769, 2019.

[12] Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998.

[13] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.

[14] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. Multi-view approaches for software and system modelling: a systematic literature review. *Software and Systems Modeling*, 18(6):3207–3233, 2019.

[15] ISO/IEC/IEEE. Systems and software engineering–architecture description. *ISO/IEC/IEEE 42010: 2011 (E)(Revision of ISO/IEC 42010: 2007 and IEEE Std 1471-2000)*, 2011:1–46, 2011.

[16] Antonio Cicchetti, Federico Ciccozzi, and Thomas Leveque. Supporting incremental synchronization in hybrid multi-view modelling. In *Workshops and Symposia on Models in Software Engineering at International Conference on Model Driven Engineering Languages and Systems (MODELS 2011)*, pages 89–103. Springer, 2012.

[17] Henk CA van Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security*, volume 1. Springer Science & Business Media, 2011.

[18] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: Towards a unified standard. In *ACM workshop on Role-Based Access Control*, volume 10, 2000.

[19] Vincent C Hu, Rick Kuhn, and Dylan Yaga. Verification and test methods for access control policies/models. *NIST Special Publication*, 800:192, 2017.

[20] Antonio Cicchetti, Federico Ciccozzi, and Thomas Leveque. A hybrid approach for multi-view modeling. *Electronic Communications of the EASST*, 50, 2012.

[21] Malvina Latifaj, Federico Ciccozzi, and Mattias Mohlin. Higher-order transformations for the generation of synchronization infrastructures in blended modeling. *Frontiers in Computer Science*, 4:1008062, 2023.

[22] Massimo Tisi, Frédéric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bézivin. On the use of higher-order model transformations. In *5th European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA 2009)*, pages 18–33. Springer, 2009.

[23] James Martin. *Managing the data base environment*. Prentice Hall PTR, 1983.

[24] Hugo Bruneliere, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. EMF views: A view mechanism for integrating heterogeneous models. In *Conceptual Modeling. ER 2015. Lecture Notes in Computer Science*, volume 9381, pages 317–325. Springer, 2015.

[25] Gábor Bergmann, Csaba Debreceni, István Ráth, and Dániel Varró. Query-based access control for secure collaborative modeling using bidirectional transformations. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 351–361, 2016.

[26] Csaba Debreceni, Gábor Bergmann, Márton Búr, István Ráth, and Dániel Varró. The MONDO collaboration framework: secure collaborative modeling over existing version control systems. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 984–988, 2017.

[27] Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró. Secure views for collaborative modeling. *IEEE Software*, 35(6):32–38, 2018.

[28] Léa Brunschwig, Esther Guerra, and Juan de Lara. Towards access control for collaborative modelling apps. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–10, 2020.

[29] Johannes Jakob, Alexander Königs, and Andy Schürr. Non-materialized model view specification with triple graph grammars. In *Graph Transformations: Third International Conference, ICGT 2006 Natal, Rio Grande do Norte, Brazil, September 17-23, 2006 Proceedings 3*, pages 321–335. Springer, 2006.

[30] Anthony Anjorin, Sebastian Rose, Frederik Deckwerth, and Andy Schürr. Efficient model synchronization with view triple graph grammars. In *Modelling Foundations and Applications: 10th European Conference, ECMFA 2014, Held as Part of STAF 2014, York, UK, July 21-25, 2014. Proceedings 10*, pages 1–17. Springer, 2014.

[31] J Nathan Foster, Michael B Greenwald, Jonathan T Moore, Benjamin C Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3):17–es, 2007.

[32] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended Modelling - What, Why and How. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019*, pages 425–430. IEEE, 2019.