Software and Systems Modeling
https://doi.org/10.1007/s10270-024-01246-1

EXPERT VOICE

An elucidation of blended modeling from an industrial perspective

Jörg Holtmann[1] · Federico Ciccozzi[2] · Wim Bast[3] · Joost van Pinxten[4]

Received: 5 November 2024 / Accepted: 9 November 2024

Abstract
Model-Driven Engineering (MDE) has been widely adopted across various industrial sectors due to its ability to manage the complexity of modern engineering products. However, traditional modeling languages and tools are often limited to a single, specific concrete syntax, which poses challenges for the diverse stakeholders involved in the modeling process.. To address these limitations, the emerging field of blended modeling introduces the use of multiple concrete syntaxes, and in some cases, even multiple abstract syntaxes, for representing the same information. In this expert perspective, we present generalized, technology-agnostic concepts developed within a European research and development project focused on blended modeling. Specifically, we contribute a standardized terminology and ontology for blended modeling, along with a methodology for creating blended modeling environments. These concepts were developed through collaboration between academic and industrial partners, who aligned on the motivations and benefits of this approach. The insights gained from this project are not only relevant to blended MDE but also can be applied to traditional MDE practices.

Keywords Model-Driven Engineering · Blended modeling · Terminologies · Ontologies · Methodologies

1 Introduction

Model-Driven Engineering (MDE) has been widely adopted in industry as an effective approach to manage the complexity of systems across various sectors, as demonstrated by empirical studies (e.g., [21, 25]). By using abstractions formalized in modeling languages, MDE allows domain experts to describe complex functions in a more abstract and human-centric way than traditional programming languages or specification documents in systems engineering.

Modeling languages formalize the communication between engineers at the level of general-purpose or domain-specific concepts. The application of such languages requires significant customization of tools, often involving extensions or combinations of modeling languages, to align with the specific requirements of development domains and contexts [30]. Moreover, modeling tools must support various means of representation (e.g., textual and graphical) to meet the needs of different development phases, stakeholder roles, and application domains.

However, traditional modeling tools typically focus on a single editing notation—either graphical or textual—though they may provide multiple notations for visualization purposes [14]. This approach restricts communication, particularly across disciplines, as a notation that is well understood in one field may not be easily grasped by experts in another. Additionally, stakeholders within the same or different disciplines may have different preferences for notation, and failing to support multiple notations can hinder collaboration and reduce efficiency. Relying on a single notation also limits the availability of tools for model manipulation. For example, graphical notations restrict the use of text-based tools like diff/merge, which are essential for collaborative work. Conversely, using only textual notations can impair model comprehension and communication.

For larger systems with heterogeneous components, involving multiple disciplines and stakeholders, the restriction to one type of notation undermines many of the benefits MDE can offer. Efficient collaboration in large-scale industrial projects requires tools that enable stakeholders to work on shared models using different notations.

✉ Federico Ciccozzi
federico.ciccozzi@mdu.se

1 Independent Researcher, Paderborn, Germany

2 Malärdalen University, Västerås, Sweden

3 Modeling Value Group B.V., Amersfoort, The Netherlands

4 Canon Production Printing Netherlands B.V., Venlo, The Netherlands

Published online: 06 December 2024

Blended modeling, as defined in [11], addresses these challenges by allowing seamless interaction with one or more models (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes). This approach aims to (1) reduce the cognitive load required to maintain effective and efficient modeling workflows, (2) offer stakeholders diverse formalisms and notations for collaborative development, and (3) support computer-aided mechanisms essential for handling the increasing complexity of modern systems [19].

From November 2019 to June 2023, an international consortium of 22 partners—including 18 industrial companies—across four European countries conducted an industrial research and development (R&D) project called 'Blended Modeling for Enhanced Software and Systems Engineering' (*BUMBLE*)[5] under the Eureka Cluster for software innovation[17] (ITEA4). The primary goal of BUMBLE was to provide reference solutions for creating and managing comprehensive blended modeling environments to support the development of complex, heterogeneous systems through seamless textual and graphical collaborative modeling.

The BUMBLE consortium developed a shared terminology and ontology to guide the planning, execution, and validation of the R&D work on blended modeling, as documented in a project deliverable [6]. Additionally, the project produced a methodology for realizing blended modeling environments, which are technologically more complex than those focusing on a single notation.

In this expert voice paper, we offer a retrospective on these efforts, presenting both the results and the reasoning behind the development of the ontology and methodology for blended modeling environments. We also discuss the implications of these contributions for future R&D activities. While the ontology and methodology were developed specifically for use within the BUMBLE project, we share them with the broader MDE community to foster further discussion and consensus building.

The remainder of the paper is organized as follows. In the subsequent section, we characterize blended modeling compared with multi-view modeling. Section 3 presents related work. In Sect. 4, we summarize and characterize technology use cases that were realized in the BUMBLE R&D project. Subsequently, we present the terminology and ontology (Sect. 6) as well as the methodology for realizing blended modeling environments (Sect. 7). Finally, we provide a discussion of these aspects in Sect. 8 and summarize in Sect. 9.

2 Blended modeling and multi-view modeling

Blended modeling, as introduced by Ciccozzi et al. [11], emphasizes the use of multiple concrete syntaxes to represent a shared set of abstract syntax concepts. This approach allows different notations to be applied based on specific modeling needs. One of the main advantages of blended modeling is the ability to seamlessly switch between graphical and textual notations during the modeling process. This flexibility empowers stakeholders to choose the most appropriate notation according to their expertise and the task at hand. By integrating both graphical and textual notations, blended modeling enables simultaneous visualization and editing of information across synchronized notations, enhancing the separation of concerns, improving interdisciplinary communication, and accelerating the overall modeling process.

In the MDE landscape, there are also several multi-view modeling approaches that focus on creating viewpoints and views, with mechanisms to manage consistency across them [10, 40]. However, blended modeling addresses a different challenge: providing multiple concrete syntaxes for a common set of abstract syntax concepts, regardless of whether the underlying modeling approach is multi-view or not. Instead of defining viewpoints and views, blended modeling emphasizes offering diverse notations for editing and visualizing the same concepts.

Blended modeling and multi-view modeling are two complementary yet distinct approaches within the MDE domain. While multi-view modeling focuses on managing multiple perspectives through viewpoints and views, blended modeling enhances interaction by offering multiple notations for the same set of concepts. Together, these approaches provide a more comprehensive solution to the challenges of modern modeling.

3 Related work

In this section, we discuss related work on MDE terminology and terminology organization schemes as well as outline related work on approaches for the provisioning of blended modeling.

3.1 MDE terminology and organization schemes

Due to the way the Model-Driven Engineering (MDE) community has grown There is a wide range of definitions for various MDE concepts. Many of these definitions, particularly for the term 'model,' tend to be highly conceptual, often overlooking important technical aspects. One of the earliest conceptual definitions comes from Stachowiak [49], who argues that a model must homomorphically reflect reality, abstract from it by reducing information, and serve a specific purpose. This definition has influenced others, such as those by Ludewig [36] and Kühne [33]. Other conceptual definitions have been provided by Bézivin and Gerbé [3] and Seidewitz [46].

However, as highlighted in our survey on model-based traceability terminology [23], such conceptual definitions do not help clarify what constitutes a *machine-processable* model. Favre [18] brings attention to this issue by humorously pointing out that, according to these conceptual definitions, even his paper on MDE could be considered a model. In response, Favre advocates for a more technical definition, such as the one proposed by Kleppe et al. [31], which defines a model as a description of a system (or part of a system) written in a well-defined language with clear syntax and semantics. However, this definition introduces aspects like syntax and semantics, which we prefer to address separately.

Paige et al. [44] similarly observe the lack of a standardized MDE terminology and offer mathematical definitions of models, metamodels, and their relationships. However, further MDE concepts like syntax, views, etc. as well as blended modeling concepts are missing.

Standards in systems and software engineering also provide definitions related to MDE. For example, the ISO/IEC/IEEE vocabulary for systems and software engineering [27] offers a definition of 'model,' but it merely compiles a range of conflicting definitions from other standards. Similarly, the ISO/IEC/IEEE standard on MDE methods and tools [28] defines 'model' and 'metamodel.' Whereas its definition of 'metamodel' is adequate, the broad and somewhat weakened definition of 'model' is further diluted by numerous supplementary notes.

Where standards lack detail, Bodies of Knowledge, such as the SWEBOK [50] and SEBOK [26], attempt to complement them. However, as noted in a community survey by Burgueño et al. [7], these Bodies of Knowledge do not sufficiently address foundational modeling concepts, particularly MDE terminology, and naturally do not cover newer concepts like blended modeling. To address this gap, Burgueño et al. propose the development of a dedicated Body of Knowledge for MDE.

Favre [18] also attempts to formalize the relationships between MDE components through a 'megamodel' based on set theory and language theory. However, this model is limited in scope, heavily centered on the outdated Model-Driven Architecture (MDA) [41], and does not account for recent advancements such as blended modeling.

Czarnecki and Helsen [13], along with Mens and Van Gorp [38], developed partially overlapping taxonomies of model transformations, which emerged from discussions at a Dagstuhl seminar on Language Engineering for Model-Driven Software Development. However, these taxonomies focus exclusively on model transformations and overlook other important aspects of MDE, including blended modeling.

In summary, despite the many seminal papers that have laid the groundwork for MDE, the community has yet to converge on a unified terminology. This is especially evident when considering organizational schemes that define the relationships between core MDE concepts.

3.2 Blended modelling and synchronization

Several methods and tools offer various forms of blended modeling capabilities. However, many of these tools, such as FXDiagram [1], MetaUML [2], and PlantUML [3], primarily rely on textual notation for model interaction, using graphical notation only for visualization. Some tools, like QuickDBD [4] and DBDiagrams [5], offer limited editing capabilities within their graphical editors. While a few tools support interactions through both textual and graphical notations, they are typically restricted to specific modeling languages, often tailored for UML and its extensions [1, 9, 37].

To facilitate the transition between graphical and textual syntaxes, two approaches have been proposed for transforming models that combine both elements. The first approach, Grammarware [51], exports a mixed model as text, while the second, Modelware [51], transforms such models into fully graphical representations. These transformations are performed on demand rather than in real time, and neither approach supports concurrent editing. Mixed textual and graphical modeling is also achievable with Qt [45], which embeds textual editors within a graphical environment; however, this process typically requires manual implementation by engineers of domain-specific modeling languages (DSMLs). JetBrains MPS [29] supports mixed notations and allows seamless switching between them for non-UML DSMLs, using projectional editing as its underlying principle.

A comprehensive analysis of these and other tools can be found in our systematic literature review on blended modeling within commercial and open-source model-driven software engineering tools [14].

In BUMBLE, we explored and developed solutions based on both projective approaches, such as those in JetBrains MPS, and transformative approaches. The latter involved higher-order transformations for the automatic generation of synchronization, migration, and co-evolution transformations to support collaborative blended modeling.

[1] https://jankoehnlein.github.io/FXDiagram/

[2] https://github.com/ogheorghies/MetaUML

[3] https://plantuml.com

[4] https://www.quickdatabasediagrams.com/

[5] https://dbdiagram.io/d

4 Application of blended modeling in different technology use cases

In the BUMBLE R&D project, the consortium planned and implemented 13 technology use cases [4]. These use cases applied different components of the overall BUMBLE technology, primarily in industrial settings. "Primarily industrial" refers to the fact that 12 of the use cases were initiated by industrial partners, with the final use case initiated by an academic partner. The industrial use cases were either executed solely by the initiating partner or collaboratively with one or more academic partners.

Beyond the naturally diverse development processes of the participating industrial partners, the use cases spanned several heterogeneous dimensions, covering a broad range of MDE-related aspects:

Industry sectors. The industrial partners who initiated use cases were active in various sectors, including automotive, printing, telecommunications, space and innovation, and application software.
MDE frameworks. The use cases were built on two primary MDE frameworks: the Eclipse Modeling Framework (EMF)[16] and JetBrains Meta Programming System (MPS)[29], which the consortium referred to as "technology spaces" following the terminology in [2, 32].
Purposes and MDE applications. The use cases served different purposes, resulting in a variety of MDE applications. Some aimed to demonstrate blended modeling with state machine modeling languages or automotive architecture description languages. Others focused on integrating cross-disciplinary models or modeling workbenches, while some explored analytical aspects such as consistency checking and design rule verification.
Modeling languages. Depending on the context and purpose, the use cases involved a wide range of modeling languages. This included numerous domain-specific modeling languages (some of which were company- or department-specific) as well as established standard languages like SysML [42], EAST-ADL [15], and UML-RT state machines [47, 48].

In summary, the BUMBLE project did not produce a single unified technology that could be applied universally. This is largely due to the diverse nature of the use cases and their underlying dimensions. However, this diversity enabled the project to cover a wide array of MDE aspects. In the following sections, we will generalize and document these aspects using a generic example, as well as the ontology and methodology developed throughout the project.

5 Generic example

In this section, we present the results of the BUMBLE R&D project and the implemented technology use cases using a generic example. This approach provides easier access to our subsequent discussion on the project's ontology and methodology. We have intentionally chosen a generic example rather than one based on a specific use case, as BUMBLE did not produce a single unified technology. Instead, it addressed various aspects of blended modeling through different use cases.

Blended modeling can be achieved by applying general modeling and transformation principles in specific ways. To maintain consistency with model-driven practices, we adhere to the de facto standard terminology and approaches as much as possible. In this paper, we distinguish between the following:

- The definition of modeling languages (metamodel level M2), created by the modeling language engineer.
- The definition of models (metamodel level M1), created by the modeling language user, which conform to the M2-level modeling languages.

We recognize that the concept of meta-levels is relative, and that language specifications are themselves models. Therefore, an M(x)-level model conforms to an M(x+1)-level model. However, for the sake of clarity in this paper, we focus on distinguishing between the M2 and M1 levels. Moving forward, when we refer to a "model" we mean an M1-level model. In other cases, we will use more specific terms such as "metamodel" or "transformation specification" to clarify the purpose of M2- or M3-level models.

Figure 1 shows the functional parts that blend and synchronize models at level M1. Figure 2 shows the functional parts that define the languages and their transformations at level M2. The figures show how models (M1) relate to the languages and transformations (M2). Beyond that, Fig. 2 also shows that blending and synchronization functionality (M1) is derived from the language and transformation specifications (M2) by means of generators. The examples are presented as graphs, with nodes and edges belonging to a limited set of types. These types align with the terminology defined in the following section.

Figure 1 illustrates two client environments participating in a collaborative session, where the modeling languages (and potentially the MDSE technologies) differ between the environments. In both, blended modeling is employed, using different concrete syntaxes for the same underlying model. The synchronization happens in two ways: immediate synchronization through a model persistency service, and deferred synchronization via a version control system.

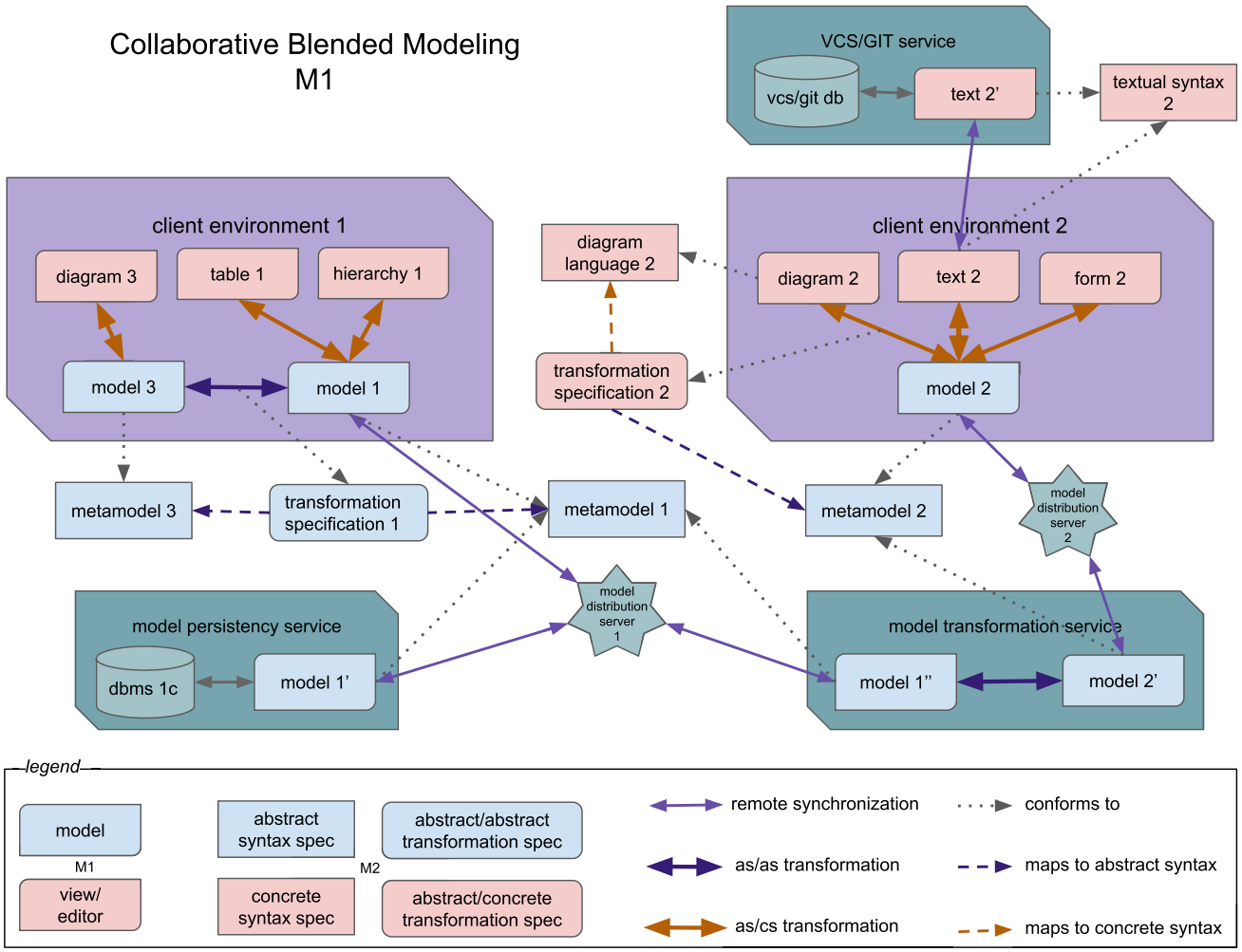

Fig. 1 M1 example

This example combines various elements of blending and collaboration across multiple syntaxes. It is intentionally complex and not intended to represent a typical modeling scenario. Instead, it demonstrates how the proposed building blocks can be used to create any desired modeling solution, whether simple or intricate.

Several transformations are running across both client environments and services. Remote synchronization between the two environments is managed by model distribution servers, which exchange changes for synchronized models. Note that remote synchronization only occurs between models that conform to the same metamodel, meaning they are instances of the same language.

The example highlights that all models and transformations adhere to their respective metamodels and transformation specifications, while the views/editors conform to their concrete syntax specifications. A transformation can occur between two models or between a model and a view/editor. Although transformations between two views/editors are possible, it is preferable to define two separate transformation specifications that each map their distinct concrete syntax to a shared abstract syntax.

The goal of the BUMBLE project was to develop solutions for blending across different MDE technologies, ensuring compatibility with all possible modeling languages. Whereas MDE technologies already offer generic functionality for defining modeling languages, we needed to extend this functionality to support transformations between them. The specific approach for realizing this depends heavily on the MDE technology involved.

Figure 2 demonstrates how the blending of different syntaxes is achieved within a modeling environment that is partially generated from language and transformation specifications. These M2-level models are also developed in a modeling environment (shown in yellow in Fig. 2), and a generator (represented by the yellow arrow) produces parts of the M1 modeling environment (shown in purple). The methods and solutions required to define and execute these transfor-

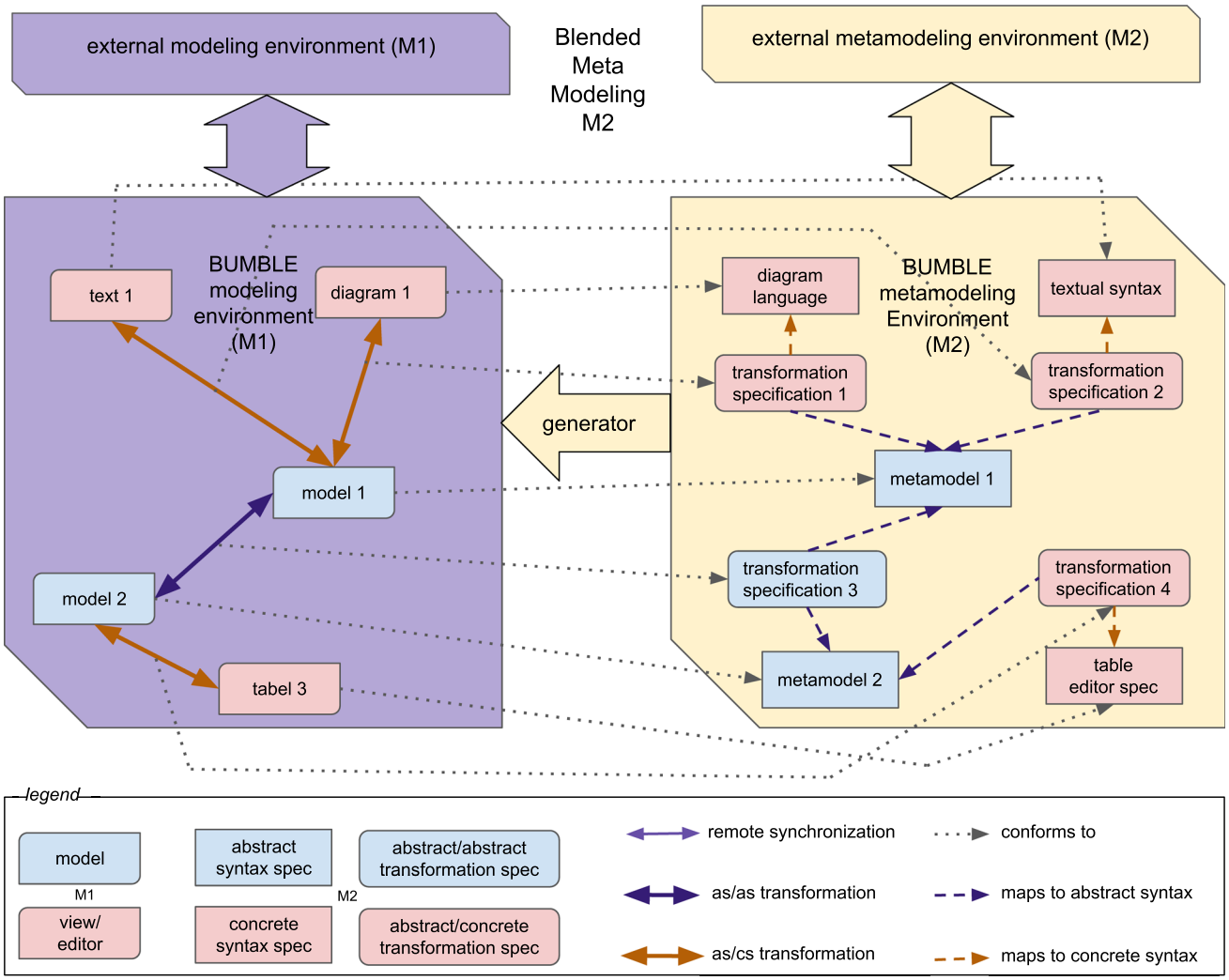

Fig. 2 M2 and M1 relation example

mations within the BUMBLE project depend significantly on the MDE technology used, such as the JetBrains Meta Programming System (MPS) [29] or the Eclipse Modeling Framework (EMF) [16].

6 A terminology and ontology for blended modeling

During the BUMBLE project, our consortium encountered communication challenges, particularly when discussing the implementation of the various technology use cases (cf. Sect. 4). Cabré [8] notes that ambiguous terminology can hinder communication among specialists, which we also observed in our project. To address this, both our BUMBLE consortium and the broader scientific community need a shared, clear terminology to facilitate effective communication and research collaboration.

Introducing approaches like blended modeling into the MDE community requires expanding the existing terminology. In response, the BUMBLE consortium developed, discussed, and aligned on a terminology that extends the established MDE concepts (cf. Sect. 3.1) to incorporate elements specific to blended modeling. To formalize and visualize the relationships between these concepts, we enhanced the terminology with an ontology, represented as a UML class model [43].

Figure 3 illustrates this ontology, and we define the individual terms and relationships in the following sections.

Model A model represents an aspect of a system under development captured in a specific instance of a machine-processable **modeling language**, where the model serves a purpose within the development lifecycle (based on [23]). For example, Fig. 1 depicts the models model 1, model 2, and model 3.

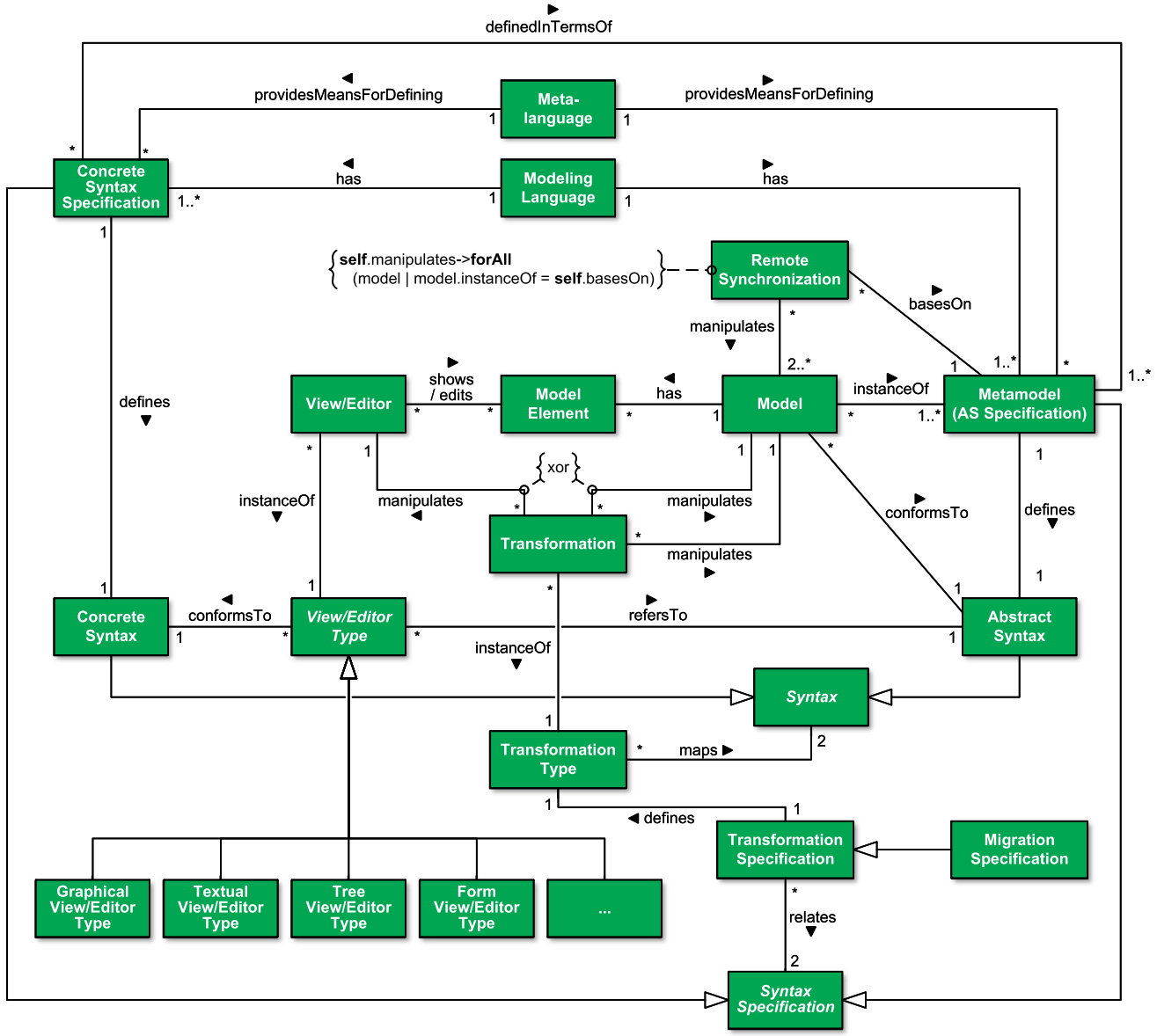

Fig. 3 Ontology for blended modeling

Modeling language A modeling language defines the **syntax** and well-formedness rules of **model**s. The syntax defines the concepts and rules to be used and conformed to, for any **model** to be a well-formed instance of that **modeling language**.

Syntax The syntax defines how to read and write (either by humans or by computers) **model**s of a specific **modeling language**. The syntax is defined by and has to conform to a **syntax specification**, which is part of a **modeling language**. A syntax can be an **abstract syntax** or a **concrete syntax**.

Syntax specification A syntax specification defines a **syntax** for a **modeling language**.

Abstract syntax The abstract syntax defines the concepts and rules by which **model**s shall be constructed to conform to a specific **modeling language**. The abstract syntax is mainly useful for addressing the static semantic aspects of **model**s. The abstract syntax is defined by and has to conform to an **abstract syntax specification** (i.e., a **metamodel**), which is part of a **modeling language**.

Metamodel (abstract syntax specification) A metamodel defines the **abstract syntax** of a **modeling language**. For example, Fig. 1 depicts the metamodels metamodel 1, metamodel 2, and metamodel 3, with their corresponding models conforming to them and therefore being instances of modeling languages.

Concrete syntax The concrete syntax defines how humans and computers interact with **models** of a specific **modeling language**. The concrete syntax is defined by and has to conform to a **concrete syntax specification**.
Concrete syntax specification A concrete syntax specification defines a **concrete syntax** for a **modeling language**. For example, Fig. 1 depicts diagram language 2 (with diagram 2 conforming to it) and textual syntax 2 (with text 2 and text 2′ conforming to it).
Metalanguage A metalanguage is a language that provides means for defining an aspect (**metamodel**, **concrete syntax specification**, static semantics, etc.) of a **modeling language**.
Remote synchronization A synchronization mechanism that keeps two or more **model**s of the same **modeling language** edited by different **view**s/**editor**s the same across a network. Changes in one **model** are propagated to equivalent changes in the other **model**. For example, Fig. 1 depicts model 1 in client environment 1, model 1′ in the model persistency service, and model 1″ in the model transformation service. All three models are synchronized via the model distribution server 1 and conform to metamodel 1.
Transformation A transformation is a manipulation of a pair of one **model** and another **model** or a **view** that preserves the relation between them according to a **transformation specification** for the two syntaxes involved.
Transformation specification A transformation specification is a specification of a **transformation type**.
Transformation types map two syntaxes; therefore, transformation specifications relate two **syntax specifications**. For example, Fig. 1 depicts the transformation specification 1 (mapping two distinct abstract syntax specifications) and the transformation specification 2 (mapping an abstract and a concrete syntax specification).
Transformation type A transformation type is a meaning-preserving relation between two **syntax**es. A **syntax** can be an **abstract syntax** or a **concrete syntax**. The transformation type is defined by and has to conform to a **transformation specification**. We distinguish a transformation type from its specification. A transformation type is an artifact that executes the transformations. A **transformation specification** is only the definition of a transformation type. A transformation type can be automatically or manually derived from its specification.
Migration specification A migration specification is a **transformation specification** that specifies a **transformation type** mapping a **metamodel** to an evolved **metamodel**, which is a new version of the first **metamodel**.
View/editor type A view/editor type defines rules according to which **views/editors** of the respective type are created based on the **concrete syntax** and thereby its **concrete syntax specification** of a **modeling language**. It defines the set of metaclasses whose instances a view[/editor] can display and be edited. A view/editor type can be graphical, textual, tree-based, form-based, etc. (extended from definition in [20]).
View/editor A view/editor is the actual set of objects and their relations (i.e., the elements of a **model**) displayed using a certain representation and layout and providing the allowed editing commands. A view/editor resembles the application of a view/editor type on the [...] **model**s. A view/editor can therefore be considered an instance of a **view/editor type** (extended from definition in [20]). For example, Fig. 1 depicts the particular views/editors as part of client environment 1 and client environment 2.

Highlights

- Defining the terminology and building the ontology required addressing potential ambiguity, polysemy, deciding on granularity and abstraction, and mitigating subjectivity of the researchers involved.
- Our terminology and ontology brought clarity, improved communication, supported automation, and played a crucial role in improving decision-making in the consortium, driving innovation, and ensuring consistency across domains and collaborators.

7 Methodology on creating/evolving blended modeling environments

Similar to the reasoning behind developing the BUMBLE ontology, our project consortium eventually realized that we were following similar procedures for implementing various technology use cases (cf. Sect. 4). However, in the early phases of the project, we found that these procedures lacked systematization, and the modeling language engineers expressed a need for more structured guidance and lessons learned in the later stages.

To address this, the consortium reverse-engineered the procedures used in earlier technology use cases and generalized them into the methodology described below. This methodology was continuously refined as we gained new insights from subsequent use cases.

In this section, we focus on the creation of blended modeling environments for domain-specific modeling languages (DSMLs). Unlike general-purpose modeling languages (e.g., UML and SysML), which have fixed abstract and concrete syntaxes defined by standardization bodies like the Object Management Group, there are numerous proprietary tools that already implement these syntaxes. In contrast, most of

BUMBLE's technology use cases involved DSMLs, which required creating new modeling environments or extending existing ones to support blended modeling. For these DSML environments, we went through the entire process of defining both the abstract and concrete syntaxes.

Figure 4 illustrates our idealized workflow for initially creating or evolving a Blended Modeling Environment, independent of specific technology platforms like EMF or MPS. The workflow primarily reflects the perspective of a DSML Engineer. We have intentionally left the DSML User workflow unspecified, as this process depends heavily on the particular DSML in question, which we abstract from here. In the remainder of this section, we detail the specific activities within the workflow.

Create/Evolve Abstract Syntax Specification Like for a conventional DSML, the basis for a blended DSML is an abstract syntax specification that defines the DSML concepts, their relationships, and (parts of) its static semantics (i.e., the DSML's abstract syntax). As defined and explained in Sect. 6, such an abstract syntax specification is typically defined by a metamodel (e.g., an Ecore metamodel or an MPS structure model). In this workflow activity, the DSML Engineer creates a new Abstract Syntax Specification or evolves an existing one. An abstract syntax specification can evolve from version A to version B, where a model that is an instance of version A already exists. If we want to continue to evolve that model in version B, and the model does not conform to version B, we need to transform that model from A to B. In order to do so, we need a (uni-directional) transformation specification that relates A and B, which we call migration specification in this evolution case (cf. Section 6). Thus, the DSML Engineer creates a new Migration Specification or evolves an existing one for this purpose.

Create/Evolve Concrete Syntax Specifications Like for a conventional DSML and as defined in Sect. 6, the concrete syntax specification defines the representation of and interaction possibilities with the DSML abstract syntax elements. For blended DSMLs however, a DSML engineer typically defines multiple concrete syntax specifications due to typically multiple notations. Thus, based on the abstract syntax specification, the DSML Engineer creates new Concrete Syntax Specifications or evolves existing ones. A concrete syntax specification relates to an abstract syntax specification. Therefore, we need a transformation specification to define that relationship. As defined in Sect. 6, a transformation specification can relate an abstract syntax to a concrete syntax.

Create/Evolve Transformation Specifications The creation of transformation specifications is in the heart of realizing a blended modeling environment. The blending of languages (abstract and/or concrete) is defined in transformation specifications. Transformation specifications can also define the migrations from one version of an abstract syntax to another version of that abstract syntax.

Realize/Adapt Transformation Types The transformation types are the artifacts that will drive the actual transformations of models and views. The DSML Engineer realizes (i.e., automatically generates, manually handcrafts, or mixture) Transformation Types based on the Transformation Specifications.

Realize/Adapt View/Editor Types Like for conventional DSMLs and as defined in Sect. 6, a user interacts with the DSML through view/editors whose type defines their look and feel. For blended DSMLs however, the DSML engineer has to create multiple views/editor types due to typically multiple notations. Thus, based on the concrete syntax specifications, the DSML Engineer realizes (i.e., automatically generates, manually handcrafts, or mixture) new View/Editor Types or adapts existing ones.

Realize/Adapt Blended Modeling Environment Finally, the DSML Engineer realizes a new blended modeling environment or adapts an existing one based on the Transformation Specifications and the View/Editor Types. That is, the DSML engineer basically configures these inputs and glues them together. The environment is then input to the user of the blended DSML.

Several semi-automatic approaches developed by the BUMBLE consortium for creating and evolving blended modeling environments are instantiations of this generic workflow. One approach began with generating and evolving modeling environments based on documentation metamodels for the EAST-ADL modeling language [22], which aligns with the first step of the workflow shown in Fig. 4. We later extended these methods to generate and evolve textual concrete syntaxes and corresponding editors for EAST-ADL, eventually generalizing the approach to support any EMF-based metamodel [24, 52, 53], covering the remaining steps of the workflow.

Another approach focused on automating synchronization mechanisms across multiple notations, regardless of whether the abstract syntaxes represent the same or different languages [34]. This approach initially automated synchronization transformations for multiple notations of the same EMF-based language, but its broader applicability allows it to handle synchronization across notations of different languages as well as co-evolution and migration between language versions.

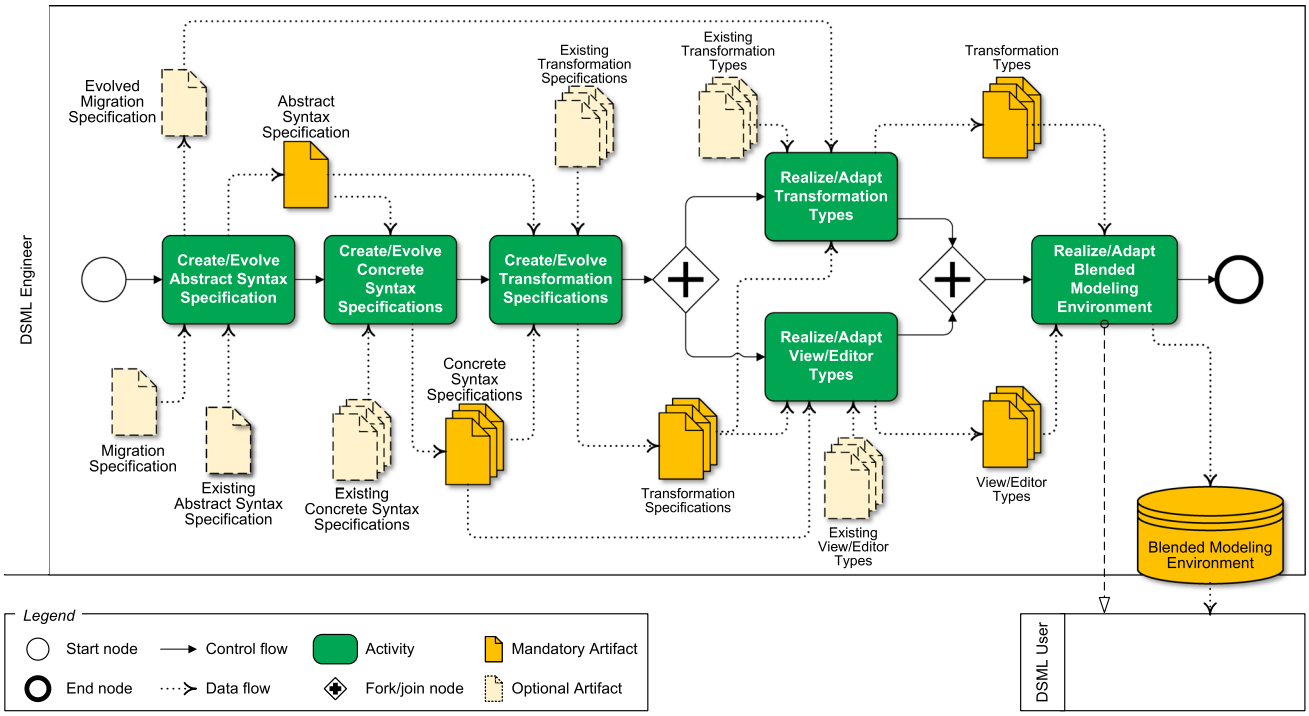

Fig. 4 Workflow for realizing/adapting a blended modeling environment

Highlights

- Creating a common workflow was considered a priority in combination with the definition of the ontology for the consortium.
- The definition of the workflow was challenging, especially in terms of aligning stakeholders, understanding our processes, and choosing the right tools.
- The workflow brought several benefits, including enhanced efficiency, better communication, consistency, and ultimately improved decision-making and customer satisfaction.

8 Lessons learned and discussion

Ontology as a communication enabler. In the BUMBLE project consortium, establishing a unified terminology significantly improved communication. Formalizing this terminology as an ontology further clarified the relationships between terms by providing an explicit semantic context. This ontology is not only beneficial for blended MDE but also for non-blended MDE, as stakeholders uninterested in blended aspects can simply disregard them. For instance, the first author of this paper is currently using a simplified version of the ontology in a company training on modeling fundamentals. We believe this ontology will be valuable to the broader MDE community, enhancing communication and alignment across various modeling efforts.

Methodology for systematization and guidance. The explicit definition of the methodology as a workflow provided the consortium with greater guidance and a more systematic approach when implementing later technology use cases. Moreover, this formalized methodology led to new insights that enhanced parts of the ontology. The methodology can also be adapted for non-blended modeling environments, with the DSML engineer skipping specific steps or artifacts as needed. Like the ontology, we aim to share this methodology with the MDE community to promote a structured approach for developing both blended and non-blended modeling environments.

Emphasis on model transformations. Our ontology and methodology place a strong focus on model transformations, which we also apply to map between abstract and concrete syntaxes. This transformation-centric approach makes the relationships between various artifacts explicit. While the initial effort required to develop model transformations is higher, our experience shows that this is outweighed by the long-term benefits in terms of maintainability. This aligns with the findings of Mohagheghi and Dehlen [39], who emphasize the importance of focusing on model transformations in MDE-based development processes.

Model transformations for multiple purposes. In our project, model transformations served multiple purposes, including relating abstract and concrete syntaxes. Specifically, we defined higher-order transformations (HOT) that, based on a manually defined mapping model [34] and two Ecore-based

metamodels, could automatically generate model transformations. These transformations can be used for synchronization between notations with disjoint abstract syntaxes or for migration between different versions of the same abstract syntax. Furthermore, we engineered transformations to bridge between languages and technologies, such as between EMF and MPS [35].

Extensive use of DSMLs by industrial partners The majority of BUMBLE's industrial partners used domain-specific modeling languages (DSMLs) rather than general-purpose modeling languages (cf. Sect. 4). This is likely due to these partners joining the consortium to leverage their expertise in developing DSML environments (as described in Sect. 7), which naturally led to a greater focus on DSMLs.

An interesting observation is that some of the DSMLs in the BUMBLE consortium were just as complex as general-purpose modeling languages, despite the common perception that DSMLs are simpler due to their domain-specific abstractions. In fact, several DSMLs were highly specialized, tailored to specific companies or even departments. This demonstrates the significant effort these companies invested to develop modeling environments that are custom-tailored to their stakeholders, countering the notion that DSMLs are inherently simpler than general-purpose modeling languages.

9 Conclusion

In this expert's voice paper, we present key findings from the European R&D project BUMBLE. Unlike previous BUMBLE publications that focus on technical, solution-specific aspects, this paper discusses broader, technology-agnostic concepts related to (blended) Model-Driven Engineering (MDE). Specifically, we introduce a terminology and ontology for blended modeling, along with a generic methodology for creating blended modeling environments. Additionally, we explore the rationale behind these contributions and the benefits they bring to the project and the wider MDE community.

Throughout the BUMBLE project, as we developed these initial contributions and further explored related work while drafting this paper, we identified a general consensus on MDE terminology and methodologies for realizing (blended) modeling environments. This aligns with recent reflections on MDE conceptual foundations, such as the SoSyM editorial by Combemale et al.[12], which revisits Stachowiak's definitions [49]. However, we found a notable gap: Whereas MDE terminology and methodologies are widely discussed, they are not yet condensed or standardized across the MDE community (as highlighted by the ongoing discourse on the MBEBOK [7]).

With this paper, we aim to provide a stepping stone to fill this gap by offering insights from our R&D project, and we hope to spark further efforts toward developing a cohesive and aligned Body of Knowledge for (blended) MDE.

Funding Open access funding provided by Mälardalen University.

References

1. Addazi, L., Ciccozzi, F.: Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment. Journal of Systems and Software (JSS) **175**, 110912 (2021). https://doi.org/10.1016/j.jss.2021.110912
2. Bézivin, J.: Model driven engineering: An emerging technical space. In: Revised Papers of the Intl. Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE), *LNCS*, 4143, 36–64. Springer (2006). https://doi.org/10.1007/11877028_2
3. Bézivin, J., Gerbé, O.: Towards a precise definition of the OMG/MDA framework. In: 16th IEEE Intl. Conf. on Automated Software Engineering (ASE), p. 273. IEEE (2001).https://doi.org/10.1109/ASE.2001.989813
4. BUMBLE Consortium: BUMBLE Deliverable D2.1 (V1.1) — Use Cases for Blended Modeling (2022). https://itea4.org/project/workpackage/document/download/8387/BUMBLE_D2.1_v1.1.pdf. Last accessed November 2024
5. BUMBLE Consortium: BUMBLE — blended modelling for enhanced software and systems engineering (2023). https://itea4.org/project/bumble.html. Last accessed November 2024
6. BUMBLE Consortium: BUMBLE Deliverable D3.3 (V3) — BUMBLE Methodology (2023). https://itea4.org/project/workpackage/document/download/8704/BUMBLE_D3.3v3_Final.pdf. Last accessed November 2024
7. Burgueño, L., Ciccozzi, F., Famelis, M., Kappel, G., Lambers, L., Mosser, S., Paige, R.F., Pierantonio, A., Rensink10, A., Salay, R., Taentzer, G., Vallecillo, A., Wimmer, M.: Contents for a model-based software engineering body of knowledge. Software & Systems Modeling (SoSyM) **18**, 3193–3205 (2019). https://doi.org/10.1007/s10270-019-00746-9
8. Cabré, M.T.: Terminology and standardization. In: Terminology: Theory, methods, and applications, *Terminology and lexicography research and practice*, vol. 1, chap. 6. John Benjamins Publishing (1999)
9. Charfi, A., Schmidt, A., Spriestersbach, A.: A hybrid graphical and textual notation and editor for UML actions. In: European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA), *LNCS*, vol. 5562, pp. 237–252. Springer (2009). https://doi.org/10.1007/978-3-642-02674-4_17
10. Cicchetti, A., Ciccozzi, F., Pierantonio, A.: Multi-view approaches for software and system modelling: a systematic literature review.

Software & Systems Modeling (SoSyM) **18**, 3207–3233 (2019). https://doi.org/10.1007/s10270-018-00713-w
11. Ciccozzi, F., Tichy, M., Vangheluwe, H., Weyns, D.: Blended modelling: What, why and how. In: 1st Intl. Workshop on Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS). IEEE Press (2021). https://doi.org/10.1109/MODELS-C.2019.00068
12. Combemale, B., Gray, J., Jézéquel, J.M., Rumpe, B.: How does your model represent the system? a note on model fidelity, underspecification, and uncertainty. Software & Systems Modeling (SoSyM) (2024). https://doi.org/10.1007/s10270-024-01210-z
13. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal **45**(3), 621–645 (2006). https://doi.org/10.1147/sj.453.0621
14. David, I., Latifaj, M., Pietron, J., Zhang, W., Ciccozzi, F., Malavolta, I., Raschke, A., Steghöfer, J.P., Hebig, R.: Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. Software and Systems Modeling (SoSyM) **22**(1), 415–447 (2023). https://doi.org/10.1007/s10270-022-01010-3
15. EAST-ADL Association: EAST-ADL. https://www.east-adl.info/Specification.html. Last accessed November 2024
16. Eclipse Foundation: Eclipse Modeling Framework (EMF) (2024). https://eclipse.dev/modeling/emf/. Last accessed: June 2024
17. Eureka Association: Eureka Clusters (2024). https://eurekanetwork.org/programmes/clusters/. Last accessed November 2024
18. Favre, J.M.: Towards a basic theory to model model driven engineering. In: 3rd UML Workshop in Software Model Engineering (WiSME) (2004)
19. Galster, M., Zdun, U., Weyns, D., Rabiser, R., Zhang, B., Goedicke, M., Perrouin, G.: Variability and complexity in software design: Towards a research agenda. SIGSOFT Softw. Eng. Notes **41**(6), 27–30 (2017). https://doi.org/10.1145/3011286.3011291
20. Goldschmidt, T., Becker, S., Burger, E.: Towards a tool-oriented taxonomy of view-based modelling. In: Modellierung 2012, *Lecture Notes in Informatics*, vol. P201, pp. 59–74 (2012)
21. Holtmann, J., Liebel, G., Steghöfer, J.P.: Processes, methods, and tools in model-based engineering — a qualitative multiple-case study. Journal of Systems and Software (JSS) **210**, 111943 (2024). https://doi.org/10.1016/j.jss.2023.111943
22. Holtmann, J., Steghöfer, J.P., Lönn, H.: Migrating from proprietary tools to open-source software for EAST-ADL metamodel generation and evolution. In: 25th International Conference on Model Driven Engineering Languages and Systems (MODELS): Tools & Demonstrations, pp. 7–11. ACM (2022). https://doi.org/10.1145/3550356.3559084
23. Holtmann, J., Steghöfer, J.P., Rath, M., Schmelter, D.: Cutting through the jungle: Disambiguating model-based traceability terminology. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), pp. 8–19 (2020). https://doi.org/10.1109/RE48521.2020.00014
24. Holtmann, J., Steghöfer, J.P., Zhang, W.: Exploiting meta-model structures in the generation of Xtext editors. In: 11th International Conference on Model-Based Software and Systems Engineering (MODELSWARD), pp. 218–225. SciTePress (2023). https://doi.org/10.5220/0011745900003402
25. Hutchinson, J., Whittle, J., Rouncefield, M.: Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Science of Computer Programming **89**, 144–161 (2014). https://doi.org/10.1016/j.scico.2013.03.017. (**Special issue on Success Stories in Model Driven Engineering**)
26. Hutchison, N., Hoffman, C. (eds.): Guide to the Systems Engineering Body of Knowledge (SEBOK), Version 2.10. IEEE (2024). https://www.sebokwiki.org. Last accessed November 2024
27. ISO/IEC/IEEE: ISO/IEC/IEEE 24765:2017(E) - IEEE/ISO/IEC Intl. Standard — Systems and software engineering – Vocabulary (2017). https://doi.org/10.1109/IEEESTD.2017.8016712
28. ISO/IEC/IEEE: ISO/IEC/IEEE 24765:2023(E) - IEEE/ISO/IEC Intl. Standard — Systems and software engineering – Methods and tools for model-based systems and software engineering (2023). https://doi.org/10.1109/IEEESTD.2023.10123376
29. JetBrains s.r.o.: JetBrains Meta Programming System (MPS) (2024). https://www.jetbrains.com/mps/. Last accessed November 2024
30. Kärnä, J., Tolvanen, J.P., Kelly, S.: Evaluating the use of domain-specific modeling in practice. In: Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling (2009)
31. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained — The Model Driven Architecture: Practice and Promise. Addison-Wesley (2003)
32. Kurtev, I., Bézivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: 4th International Symposium on Distributed Objects and Applications (DOA), pp. 1–6 (2002)
33. Kühne, T.: Matters of (meta-) modeling. Software & Systems Modeling (SoSyM) **5**, 369–385 (2006). https://doi.org/10.1007/s10270-006-0017-9
34. Latifaj, M., Ciccozzi, F., Mohlin, M.: Higher-order transformations for the generation of synchronization infrastructures in blended modeling. Frontiers in Computer Science **4**, 1008062 (2023). https://doi.org/10.3389/fcomp.2022.1008062
35. Latifaj, M., Taha, H., Ciccozzi, F., Cicchetti, A.: Cross-platform blended modelling with JetBrains MPS and Eclipse Modeling Framework. In: ITNG 2022 19th International Conference on Information Technology-New Generations, *Advances in Intelligent Systems and Computing*, vol. 1421, pp. 3–10. Springer (2022). https://doi.org/10.1007/978-3-030-97652-1_1
36. Ludewig, J.: Models in software engineering – an introduction. Software & Systems Modeling (SoSyM) **2**, 5–14 (2003). https://doi.org/10.1007/s10270-003-0020-3
37. Maro, S., Steghöfer, J.P., Anjorin, A., Tichy, M., Gelin, L.: 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 (SLE), pp. 1–12. ACM (2015). https://doi.org/10.1145/2814251.2814253
38. Mens, T., Van Gorp, P.: A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science **152**, 125–142 (2006). https://doi.org/10.1016/j.entcs.2005.10.021. Proc. of the Intl. Workshop on Graph and Model Transformation (GraMoT 2005)
39. Mohagheghi, P., Dehlen, V.: Where is the proof? — A review of experiences from applying MDE in industry. In: European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA), *LNCS*, vol. 5095, pp. 432–443. Springer (2008). https://doi.org/10.1007/978-3-540-69100-6_31
40. N. Boucké et al.: Characterizing Relations between Architectural Views. Springer (2008)
41. Object Management Group: MDA — The Architecture of Choice for a Changing World. https://www.omg.org/mda/. Last accessed November 2024
42. Object Management Group: OMG Systems Modeling Language (SysML). https://www.omg.org/spec/SysML/. Last accessed November 2024
43. Object Management Group: OMG Unified Modeling Language (UML). https://www.omg.org/spec/UML/. Last accessed November 2024
44. Paige, R.F., Kolovos, D.S., Polack, F.A.: A tutorial on metamodelling for grammar researchers. Science of Computer Programming **96**, 396–416 (2014). https://doi.org/10.1016/j.scico.2014.05.007.

(**Selected Papers from the 5th Intl. Conf. on Software Language Engineering (SLE 2012)**)
45. The Qt Company. Qt. https://www.qt.io/. Last accessed November 2024
46. Seidewitz, E.: What models mean. IEEE Software **20**(5), 26–32 (2003). https://doi.org/10.1109/MS.2003.1231147
47. Selic, B.: Real-time object-oriented modeling. IFAC Proceedings Volumes **29**(5), 1–6 (1996). https://doi.org/10.1016/S1474-6670(17)46346-4
48. Selic, B.: Using UML for modeling complex real-time systems. In: Languages, Compilers, and Tools for Embedded Systems, *LNCS*, vol. 1474, pp. 250–260. Springer (1998). https://doi.org/10.1007/BFb0057795
49. Stachowiak, H.: Allgemeine Modelltheorie. Springer (1973)
50. Washizaki, H. (ed.): Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0. IEEE (2024). https://www.computer.org/education/bodies-of-knowledge/software-engineering. Accessed September 2024
51. Wimmer, M., Kramler, G.: Bridging Grammarware and Modelware. In: International Conference on Model Driven Engineering Languages and Systems (MODELS), *LNCS*, vol. 3844, pp. 159–168. Springer (2005). https://doi.org/10.1007/11663430_17
52. Zhang, W., Hebig, R., Steghöfer, J.P., Holtmann, J.: Creating Python-style domain specific languages: A semi-automated approach and intermediate results. In: 11th International Conference on Model-Based Software and Systems Engineering (MODELSWARD), pp. 210–217 (2023).https://doi.org/10.5220/0000170800003402
53. Zhang, W., Holtmann, J., Strüber, D., Hebig, R., Steghöfer, J.P.: Supporting meta-model-based language evolution and rapid prototyping with automated grammar transformation. Journal of Systems and Software (JSS) **214**, 112069 (2024). https://doi.org/10.1016/j.jss.2024.112069

Jörg Holtmann independently conducts research while working in the German railway industry. Formerly, he was a PostDoc at Chalmers University of Technology | University of Gothenburg in Sweden, after being a senior expert at the Fraunhofer IEM in Paderborn, Germany. His research interests center around model -based requirements/systems/soft ware engineering for software-inte nsive systems.

Federico Ciccozzi is a Full Professor in Computer Science at Mälardalen University (Västerås, Sweden), head of research education in computer science and in electronics, and leader of the automated software language and software engineering research group. His research focuses on several aspects of model-driven engineering and automated software engineering for the development of complex embedded safety-critical cyber-physical systems based on domain-specific modeling languages (DSMLs). Federico has (co-)authored over 130 publications in international journals, international conferences, and workshop proceedings.

Wim Bast is a senior meta automation consultant and one of the founders of the Modeling Value Group. Wim has more than 25 years experience in model-driven software development and helped to make several model-driven projects with high impact a success. Wim was involved in the development of modeling standards at the OMG. Wim is coauthor of the book "MDA Explained". Wim is the driving force behind DClare, an open source framework and engine for purely declarative languages.

Joost van Pinxten is an architect at Canon Production Printing. He has been applying model-driven development for over 10 years in an industrial context. His research and development interests include (re-entrant) machine scheduling, modeling of timing constraints, modeling system behavior, (interactive) visualization of system behavior. He is also combining domain specific models and simulations with field data to improve products' diagnostics processes.