Blended Modelling – What, Why and How

Federico Ciccozzi
Mälardalen University
Västerås, Sweden
federico.ciccozzi@mdh.se

Matthias Tichy
Ulm University
Ulm, Germany
matthias.tichy@uni-ulm.de

Hans Vangheluwe
Univ. of Antwerp - Flanders Make
Antwerp, Belgium
hans.vangheluwe@uantwerpen.be

Danny Weyns
KU Leuven and Linneaus Univ.
Leuven, Belgium
danny.weyns@kuleuven.be

***Abstract*—Empirical studies indicate that user experience can significantly be improved in model-driven engineering. Blended modelling aims at mitigating this by enabling users to interact with a single model through different notations. Blended modelling contributes to various modelling qualities, including comprehensibility, analysability, and acceptability. In this paper, we define the notion of blended modelling and propose a set of dimensions that characterise blended modelling. The dimensions are grouped in two classes: user-oriented dimensions and realisation-oriented dimensions. Each dimension describes a facet that is relevant to blended modelling together with its domain (i.e., the range of values for that dimension). The dimensions offer a basic vocabulary to support tool developers with making well-informed design decisions as well as users to select appropriate tools and configure them according to the needs at hand. We illustrate how the dimensions apply to different cases relying on our experience with blended modelling. We discuss the impact of blended modelling on usability and user experience and sketch metrics to measure it. Finally, we outline a number of core research directions in this increasingly important modelling area.**

***Index Terms*—modelling, user experience, blended modelling, abstract syntax, concrete syntax, notations, tools.**

I. Introduction

Modelling is a common activity in most technical and engineering domains that aims at representing a complex problem in a simpler, more abstract, more understandable, or more focused manner, for a specific purpose. Modelling yields models that can be used for simplifying communication among different stakeholders, serve as a blue-print for performing engineering tasks, as sources (or targets) of specific analysis or verification and validation (V&V) activities. Models together with model manipulations are the core artefacts of Model-Driven Engineering (MDE) [1]. MDE is an engineering approach that aims at raising the level of abstraction at which engineers operate, moving from a problem-/reality-specific complex and detailed world, to a less detailed, more flexible and more comprehensive level. For example, in software engineering, modelling allows moving from a detailed machine-oriented code-centric level to an architecture level, allowing engineers to focus on the coarse-grained structures and behaviours of a system, supporting reasoning about qualities of the system, such as maintainability, reliability and security.

Modelling languages are commonly divided into two categories: general-purpose and domain-specific. General-purpose languages are broadly applicable across domains (examples are UML and XML); however these languages lack features specific for a particular domain of interest. In contrast, domain-specific modelling languages (DSMLs) offer engineers domain-specific modelling concepts and features that formalise their *communication* for the domain at hand. The heterogeneity and variability within a domain, the modelling needs across development phases, and the diversity of stakeholders and their roles, require DSMLs that offer a flexible set of *notations*[1] by which stakeholders can interact with models.

Nevertheless, DSMLs usually focus on one reference notation (e.g., graphical or textual), restricting stakeholder communication within a domain and across different disciplines. A notation that is convenient for and well-understood by one stakeholder may not be as convenient for another stakeholder. Moreover, engineers may have different notation preferences, possible for different modelling tasks. Hence, supporting only a single notation may negatively affect the throughput of engineers. The diversity of stakeholders calls for a variety of domain-specific editing facilities, which can be graphical, table-based, form-based, and also textual (e.g., formal specification [2]). Besides hampering communication and cooperation, restricting modelling to one notation has the drawback of limiting the pool of available tools to develop and manipulate models. For example, choosing a diagrammatic representation (without a corresponding textual representation) may limit the usability of text manipulation tools, such as text-based diff/merge, which is important for team collaboration. When adopting MDE in large-scale projects, efficient modelling support for a team is crucial. Therefore, DSMLs and related tools are expected to enable different stakeholders to work on overlapping parts of models using different notations. In summary, modelling based on one single notation is often too restrictive and undermines many MDE benefits.

In this paper we formalise the use of multiple modelling notations as *blended modelling* and define it as follows:

> *Blended modelling 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 modelling entails the ability to seamlessly interleave modelling and change operations at arbitrary times and across multiple users working collaboratively. Moreover, blended modelling can (and shall) provide variable degrees of non-conformance of edited models to both the underlying language

[1] We use the term "notation" as a synonym for concrete syntax of a DSML.

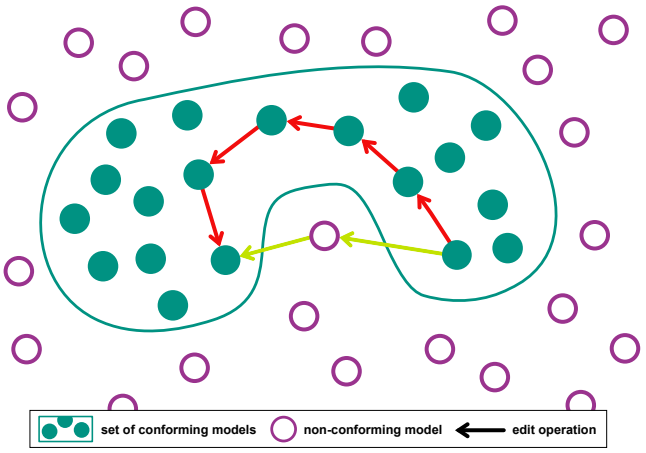


Fig. 1. Illustrating the benefits of temporary non-conforming models

and the provided notations, to maximise modelling flexibility and, consequently, productivity for the engineers. Figure 1 illustrates the benefits of temporary non-conforming models as they allow, e.g., to "take shortcuts" during modelling activities.

In the next section, we relate blended modelling to other paradigms in MDE and provide a motivation for a targeted research effort on blended modelling. Section III provides a snapshot of the state of the art and current practice in blended modelling. In Section IV, we provide a discussion of different dimensions of blended modelling grouped into user-oriented and realisation-oriented dimensions. Section V discusses the expected positive impact of blended modelling on user experience. Finally, Section VI concludes the paper with a set of core research directions.

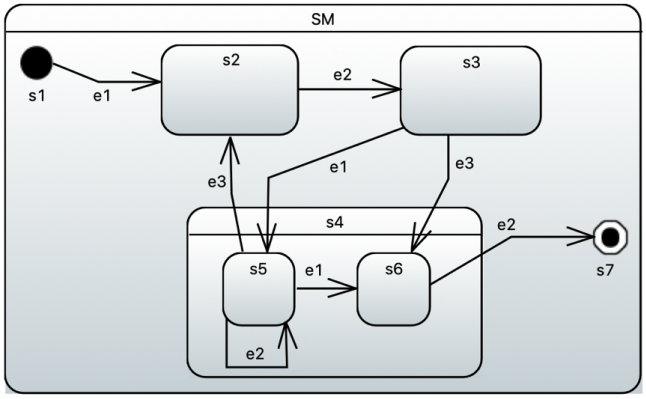
II. Contextualising Blended Modelling

At first sight, the notion of blended modelling may seem similar or overlapping with multi-view modelling [3] that is based on the paradigm of viewpoint/view/model as formalised in the ISO/IEC 42010 standard[2].

Multi-view modelling is commonly based on viewpoints (i.e. "conventions for the construction, interpretation and use of architecture views to frame specific system concerns" [4]) that are materialised through views that are composed of one or more models. In blended modelling, the focus is *not* on identifying viewpoints and related views, but rather on providing multiple blended editing and visualising notations to interact with a set of concepts. In short, blended modelling could be seen as orthogonal to multi-view modelling. While multi-view modelling aims at defining viewpoints/views, blended modelling aims at providing a powerful multi-notation characterisation that may be used to define viewpoints/views. Multi-view modelling approaches focus on the creation of viewpoints/views and mechanisms for consistency management across them [3], [5]. Blended modelling focuses on the specific problems related to the provision of multiple concrete syntaxes

[2]https://www.iso.org/standard/50508.html

a)



b)

state \ event	e1	e2	e3
s1	s2	-	-
s2	-	s3	-
s3	s5	-	s6
s4	-	-	-
s5	s6	s5	s2
s6	-	s7	-
s7	-	-	-

c)

```
stateMachine SM{
  region Region1{
   initialState s1;
   state s2;
   ..
   compositeState s4{
     state s5;
     state s6;
     transition s5 to s6 with e1;
     ...}}}
```

Fig. 2. Model of a state-machine in three partially overlapping concrete syntaxes: a) diagrammatic, b) tabular, c) textual

for a set of abstract syntactical concepts, independently of whether the base modelling approach is multi-view or not.

User experience (UX) is a major concern in MDE, especially in relation to the way users can use modelling tools to interact with models [6], [7]. Blended modelling aims at improving UX by providing multiple notations suitable for different aspects of design, development and stakeholder communication in an MDE process. Consider the example in Fig. 2 that shows how three partially overlapping concrete syntaxes can be used to model a state-machine. While the diagrammatic notation (Fig. 2.a) is most appropriate for designing the state-machine and grasping the model behaviour at a glance, a tabular notation (Fig. 2.b) is more convenient for identifying specific properties, e.g. check whether a specific state (s5) covers all entailed events (e1, e2, e3). The textual notation (Fig. 2.c) on the other hand is the most convenient for complex restructuring of the state-machine, e.g., moving internal states and transitions from one composite state to another (e.g., from s4 to s3), which can be done with a single copy & paste action. The same would be particularly difficult using the tabular

notation since it does not carry information about composite states, while it would require a set of editing actions in the diagrammatic notation (at least in state of the art tools [8]).

In a preliminary work [8], we proposed a prototype framework for blended modelling and experimented with the potential usefulness of it. We focused on mixed textual and graphical notations, in particular for UML profiles. By seamlessly combining graphical and textual modelling, the framework showed potential to mitigate the drawbacks of both notations and to exploit the synergy of combining their benefits. The experiments highlighted several concrete benefits, such as enhanced flexibility for separating concerns, improved features for multi-view modelling, increase in convenience of text-based editing operations, and increase in modelling efficiency.

III. Related Work

Some of the issues related to blended modelling, especially from the point of view of consistency management across models described in different concrete syntaxes, can be similarly found in multi-view modelling. In this paper we focus on a first classification and characterisation of blended modelling solutions, thus leaving out other aspects of multi-view modelling approaches. For the interested reader, an empirical classification and characterisation of existing multi-view modelling approaches can be found in [3].

Several solutions have been proposed to intermix different concrete syntaxes (mostly textual and graphical); in the following, we present them highlighting their strengths and weaknesses in relation to our classification.

Umple [9] merges programming and modelling concepts by adding modelling abstractions into programming languages and offering features to actively perform model edits on both textual and graphical concrete syntaxes. Some edit operations are specific to one concrete syntax and not customisable.

A plethora of other tools such as FXDiagram [10], Eclipse Sprotty [11], LightUML [12], TextUML [13], MetaUML [14], PlantUML [15] focuses on textual concrete syntax for actively editing the modelling artefacts, while providing a graphical notation for visualisation purposes only. FXDiagram is based on JavaFX 2 and provides on-the-fly graphical visualisation of changes in the textual concrete syntax including change propagation; the focus is on EMF models. Eclipse Sprotty, as FXDiagram, focuses on the visualisation of textual models, but it also provides a limited degree of editing of the models. LightUML focuses more on reverse engineering by generating a class diagram representation of existing Java classes and packages. TextUML allows modellers to leverage a textual notation for defining UML models and providing textual comparison, live graphical visualisation of the model in terms of class diagrams, syntax highlighting and instant validation. MetaUML is a MetaPost library for creating UML diagrams through a textual concrete syntax and it supports a few read-only diagrams. Similarly, PlantUML allows the modelling of UML diagrams by using a textual notation; graphical visualisations are read-only.

JetBrains MPS [16] is a meta-modelling environment for the development of DSML tools, which support synchronised editing in multiple (customisable) concrete syntaxes for non-UML DSMLs, but lacks out-of-the-box support for UML profiles and has limited automated support for integration with domain-specific environments. Furthermore, the "textual" view of a model in MPS is not actual text but a form-based representation with a fixed format. This implies, not only that these form-based editors restrict the user, but more critically, standard text-based tools such as regex search/replace, or diff/merge cannot be used on the model.

Similarly to JetBrains MPS, MelanEE [17] exploits the projectional approach too. With projectional editing, the user edits the model through a syntax-specific view or editor, which itself updates the underlying abstract syntax model and these changes are automatically reflected in other views or editors for alternative concrete syntaxes. The main advantage is that the model can be projected in various concrete syntaxes depending on what the user prefers. However, it adds a considerable overhead for the DSML developer as the user actions (i.e., keyboard, trackpad and mouse events) have to be translated into change actions on the abstract syntax tree. For parser-based textual DSMLs (e.g., Xtext), a text editor in combination with a lexer/parser combination can be used.

A textual editor for the Action Language for Foundational UML (Alf) has been developed based on Xtext [18]. In [19], the authors provide an approach for defining combined textual and graphical DSMLs based on the AToM3 tool. Starting from a metamodel definition, different diagram types can be assigned to different parts of the metamodel. A graphical concrete syntax is assigned by default, while a textual one can be given by providing triple graph grammar rules to map it to the specific metamodel portion. This approach targets specific DSMLs defined through AToM3 and is not applicable to, e.g., UML. Charfi et al. [20] explore the possibilities to define a single concrete syntax supporting both graphical and textual notations specifically for UML actions only.

In [21], the authors provide the steps needed for embedding generated EMF-based textual model editors into graphical editors defined in terms of GMF. That approach provides pop-up boxes to textually edit elements of graphical models rather than allowing seamless editing of the entire model using a chosen syntax. The change propagation mechanisms are on-demand triggered by a modeller's commit.

Related to the switching between graphical and textual syntaxes, two approaches are proposed to ease transformations of models containing both graphical and textual elements. The first is Grammarware [22], by which a mixed model is exported as text. The second is Modelware [22], by which a model containing graphical and textual content is transformed into a fully graphical model. Transformation from mixed models to either text or graphics is on demand rather than on-the-fly and the approach does not allow concurrent editing.

Maro et al. [23] provide a solution blended modelling of a specific UML profile, based on Ecore as pivot-language between graphical and concrete syntaxes and higher-order trans-

formations for on-demand synchronisation purposes. Addazi et al. [8] propose a solution for blended modelling of UML and profiles, where both graphical and textual editing is done on a common persistent model resource, thus reducing the need for synchronisation among the two concrete syntaxes (more flexible but less performant than projectional approaches); the authors report on a small-scale experiment showing the potential benefits of blended modelling too.

Although there are several works on the topics related to blended modelling, a reference classification and characterisation of it has not been proposed yet.

IV. Classification

We now discuss all relevant dimensions for blended modelling. While many of them are applicable for general modelling, they are specifically needed for blended modelling. We give a domain for each dimension, a description as well as an example. We distinguish between dimensions relevant for a user of a blended modelling tool and technical dimensions relevant for a developer of a blended modelling tool.

A. User-Oriented Dimensions

Table I shows the user-oriented dimensions. We highly value the flexibility provided by blended modelling to increase productivity as discussed in the introduction. Specifically, we distinguish two dimensions of flexibility: *language flexibility* and *notation flexibility*. The first dimension describes how much the abstract syntax of a model is allowed to deviate from the language specification. For example, a state machine model might temporarily contain unconnected transitions, which is useful for re-factoring operations. The second dimension describes how much the notation of the model might deviate from the notation defined for the language. For example, the states of a state machine describing a traffic light might be shown as pictographs of the traffic light in its states to improve comprehensibility of the state machine.

While these dimensions are applicable to single notations, the dimension *degree of overlap* between the elements shown in multiple notations and the dimensions related to inconsistency focus on multiple notations. Similarly to above's discussion on flexibility, bearing with inconsistencies is a vital ingredient of blended modelling to increase productivity. This includes the ability to support temporary inconsistencies between the different notations of a model while being fully able to edit it as well as propagating changes in local, consistent parts of the model. This requires different variants of *inconsistency detection* and *inconsistency resolutions*.

B. Realisation-Oriented Dimensions

Table II shows some technical dimensions of blended modelling. The *Mapping cardinality* refers to how the multiple notations can be technically realised. The easiest case is when one abstract syntax ($N{=}1$) can be shown in multiple ($M{>}1$) different notations. However, often technical constraints of the used modelling frameworks require also multiple ($N{>}1$) abstract syntaxes. E.g., some frameworks require for each notation element a corresponding object in the abstract syntax even though sometimes a link between objects in the abstract syntax would be enough to capture the same information.

The *mapping technology* refers to the different implementation variants of how the mapping between abstract and concrete syntax is kept consistent. Here, we see two extremes that have a notable impact on the other blended modelling dimensions: *parsing-based* and *projection-based*. Parsing is mostly used for textual notations whereas projections are mainly used for diagrammatic and other notations. Those technologies differ mostly in usability and inconsistency handling. While parsing-based approaches offer a broad variety of user interactions, independently of the language constraints, projection-based approaches follow the correct-by-construction paradigm, thus leading to easier realisation of blended modelling, but suffering from lower usability (cf. [24], [25]).

Finally, the *change propagation* dimension covers how changes between the different notations and the abstract syntax are propagated. Here, we distinguish between sequential and concurrent change propagation, buffered or not, as well as whether and how multiple changes are merged.

V. Impact on usability and user experience

In the following, we discuss those usability attributes, based on the ISO 9241-110 standard[3], which we believe blended modelling will positively impact, leading to a better user experience.

- **Understandability:** Understandability is the ability to be understood. Blended modelling supports understandability as different concrete syntaxes support understanding activities differently. In the case of a state-machine, a diagrammatic concrete syntax gives a good overview of the different states and their transitions, while a tabular concrete syntax easily shows whether a state reacts to any event in the state-machine.
- **Learnability:** Learnability is the capability of a software artefact to enable the user to learn how to use it [26]. Diagrammatic concrete syntaxes and corresponding editors provide a better learnability for domain experts as they can show the model using the visual shapes of domain elements, which a textual concrete syntax cannot.
- **Changeability:** Changeability is the ability to change or be changed. Different changes might be easier or harder to perform in different concrete syntaxes. Blended modelling allows users to choose, for each type of change, the concrete syntax where a specific change type is easier to perform. For example, creating a transition between states might be easier in a state-machine diagram since the states are easily identifiable compared to a textual syntax, where one would need to refer to the state name that might be defined in a different location in the text. However, moving a set of states into a composite state might be much easier to do in a textual syntax as one could simply cut&paste the text representing the states

[3]https://www.iso.org/standard/38009.html

TABLE I
USER-ORIENTED BLENDED MODELLING DIMENSIONS

Dimension	Dimension scope	Description	Example
Number of concrete syntaxes	From 2 to N	Blended modelling requires at least two different concrete syntaxes.	A combination of diagram, text, and table. For example, a state machine could be shown in a diagrammatic way to give an overview, a textual way to easily re-factor, and a table to easily analyse for completeness
Degree of language flexibility	No deviation from abstract syntax, deviation only from well-formedness rules, deviation from the metamodel	Switchable degree of flexibility to deviate temporarily from the rules of the language	a correct-by-construction editor does not allow the deviation from the abstract syntax, a textual editor may allow deviations from the base metamodel/grammar
Degree of notation flexibility	No deviation from the shapes of the notation, deviation from the shapes of the notation	Switchable degree of flexibility to deviate temporarily from the rules of the notation	A graphical editor for state machines might allow to use different shapes for states
Degree of overlap	From partial to complete	The percentage of language constructs expressible in multiple concrete syntaxes	A diagrammatic language shows a state machine fully whereas a table only shows which states have transitions to which states
Inconsistency detection	On-the-fly, on-demand, time-triggered, condition-triggered	Inconsistency detection between the multiple concrete syntax representations	Each time a user changes a diagrammatic representation of a state machine, its consistency with the table is automatically checked
Inconsistency resolution	From manual to automatic	How the system supports resolution of inconsistencies	If a transition between two states is added, a corresponding table entry is changed

TABLE II
REALISATION-ORIENTED BLENDED MODELLING DIMENSIONS

Dimension	Dimension scope	Description	Example
Number of abstract syntaxes required for multiple concrete syntax	$\#CS > 1,1 <= \#AS <= \#CS$	Cardinality of the technical mapping between abstract and concrete syntaxes	Different abstract syntaxes have to be used for diagram and table notations of hierarchical state machines due to technical constraints by the underlying frameworks
Mapping process between abstract and concrete syntaxes	Projectional, parsing (text, diagrammatic, graph)	This dimension covers how elements of the abstract syntax are technically kept consistent with the representational elements	While text is usually parsed against a, e.g., context-free grammar, a diagrammatic tabular notation is usually a projection of a specific part of the underlying model
Change propagation	Sequential, concurrent	How changes are propagated between concrete syntax and abstract syntax as well as when and how multiple concurrent changes are merged	A change of the name of a state in a concrete syntax is propagated first to the abstract syntax and then to all other overlapping concrete syntaxes

into the composite state. In a diagrammatic concrete syntax, this change requires a specific re-factoring operation to be performed swiftly.

- **Analysability:** Analysability is the ability of something to be analysed. A tabular concrete syntax makes it easy to check whether a state has transitions for each potential event, which is often a requirement in state-machines representing safety-critical behaviours.
- **Acceptability:** Acceptability is the quality of being accepted. A lack in user acceptance hinders the adoption and fruitful use of MDE. Our experience from several industrial projects shows that different developers prefer different concrete syntaxes, e.g., software developers often prefer textual syntax, while electrical engineers are more used and prefer diagrammatic notations.

These quality attributes can be measured through different metrics, design complexity, like number of modelling steps, fault density, number of concrete syntax elements, steepness of the learning curve. Eventually, improvement of these attributes will improve productivity and, thus, reduce costs [27].

VI. CORE RESEARCH DIRECTIONS

To realise the vision of blended modelling, we believe that research efforts should concentrate on the following directions.

a) Diagrammatic parsing: in Section IV we sketched the idea of leveraging parsing approaches instead of projectional approaches for graphical editing. More precisely, starting from a set of basic graphical elements, e.g., rectangles for shapes and arrows for transitions, and a set of (grammar) rules how diagrams are constructed, a model is generated. Grammar rules could be for example, if two shapes, previously recognised as states, are connected via an arrow, the arrow is recognised as a transition connecting the two shapes. Similarly, if a state shape is graphically fully contained in another state shape, the state is a sub-state of the other state.

b) Architectures for blended modelling user interfaces: complementary to the previous point, the architecture of graphical modelling frameworks is based on the model-view-controller paradigm [28], which works well for projectional approaches. However, if a diagram parsing approach is adopted, the architecture may need to be adapted to accommodate a unidirectional data flow from diagrammatic notation to model, particularly, at times when the diagrammatic notation is not parseable, e.g., if two states overlap. Notwithstanding, changes in the model need also to be performed and reflected back to the diagrammatic notation, e.g., if a re-factoring like moving states into a hierarchical state is carried out. In this case, underlying model and its diagrammatic representation must conform to each other.

c) Flexibility of blended modelling frameworks: one of the major advantages that blended modelling can bring is flexibility in modelling activities. A flexible modelling approach enables the language/framework engineer to customise both the modelling language and related modelling notations independently. Mechanisms for automatically generating ad-hoc blended editors and synchronisation facilities from a given language, as well as for gracefully co-evolving the blended modelling framework (i.e., editors and synchronisation) as consequence of, e.g., the manual customisation of the context-free grammar representing a textual notation, are needed. Particularly designed higher-order transformations and mapping metamodels may be helpful for this.

d) User studies on benefits of blended modelling: while we strongly believe that blended modelling will improve user experience and productivity, see also preliminary studies [8], this needs to be thoroughly investigated. For example, we need to identify the classes of use cases that would benefit the most from blended modelling. Furthermore, we would like to know when and why people switch between notations. With respect to temporary non-conformance, it would be interesting to investigate possible patterns regarding its use by developers (when, how often, and why). Finally, while the aforementioned aspects can be addressed with qualitative studies, we also need to perform additional and more extensive quantitative studies to capture how much usability and productivity are actually improvable through blended modelling.

REFERENCES

[1] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of MDE in industry," in *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 2011, pp. 471–480.

[2] J. Lilius and I. P. Paltor, "Formalising uml state machines for model checking," in *International Conference on the Unified Modeling Language*. Springer, 1999, pp. 430–444.

[3] A. Cicchetti, F. Ciccozzi, and A. Pierantonio, "Multi-view approaches for software and system modelling: a systematic literature review," *Software & Systems Modeling*, 2019.

[4] D. Emery and R. 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*, 2009.

[5] N. Boucke, D. Weyns, R. Hilliard, T. Holvoet, and A. Helleboogh, "Characterizing relations between architectural views," *European Conference on Software Architecture*, 2008.

[6] S. Abrahão, F. Bourdeleau, B. H. C. Cheng, S. Kokaly, R. F. Paige, H. Störrle, and J. Whittle, "User experience for model-driven engineering: Challenges and future directions," in *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2017, pp. 229–236.

[7] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software and System Modeling*, vol. 17, no. 1, pp. 91–113, 2018.

[8] L. Addazi, F. Ciccozzi, P. Langer, and E. Posse, "Towards Seamless Hybrid Graphical–Textual Modelling for UML and Profiles," in *Procs of ECMFA*. Springer, 2017, pp. 20–33.

[9] "Umple," http://cruise.eecs.uottawa.ca/umple/, latest access: 2019-02-06.

[10] "FXDiagram," http://jankoehnlein.github.io/FXDiagram/, latest access: 2019-02-06.

[11] "Eclipse Sprotty," https://projects.eclipse.org/proposals/eclipse-sprotty, latest access: 2019-02-06.

[12] "LightUML," http://lightuml.sourceforge.net/, latest access: 2019-02-06.

[13] "TextUML," http://abstratt.github.io/textuml/, latest access: 2019-02-06.

[14] "MetaUML," https://github.com/ogheorghies/MetaUML, latest access: 2019-02-06.

[15] "PlantUML," http://plantuml.com/, latest access: 2019-02-06.

[16] "Jetbrains MPS," https://www.jetbrains.com/mps/, latest access: 2019-02-06.

[17] C. Atkinson and R. Gerbig, "Harmonizing textual and graphical visualizations of domain specific models," in *2nd Workshop on Graphical Modeling Language Development*. ACM, 2013, pp. 32–41.

[18] C.-L. Lazăr, "Integrating Alf editor with Eclipse UML editors," *Studia Universitatis Babes-Bolyai, Informatica*, vol. 56, no. 3, 2011.

[19] F. P. Andrés, J. De Lara, and E. Guerra, "Domain specific languages with graphical and textual views," in *International Symposium on Applications of Graph Transformations with Industrial Relevance*, 2007.

[20] A. Charfi, A. Schmidt, and A. Spriestersbach, "A hybrid graphical and textual notation and editor for UML actions," in *European Conference on Model Driven Architecture-Foundations and Applications*, 2009.

[21] M. Scheidgen, "Textual modelling embedded into graphical modelling," in *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 2008, pp. 153–168.

[22] M. Wimmer and G. Kramler, "Bridging grammarware and modelware," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2005, pp. 159–168.

[23] S. Maro, J.-P. Steghöfer, A. Anjorin, M. Tichy, and L. Gelin, "On Integrating Graphical and Textual Editors for a UML Profile Based Domain Specific Language: An Industrial Experience," in *ACM SIGPLAN International Conference on Software Language Engineering*, 2015.

[24] C. H. Damm, K. M. Hansen, and M. Thomsen, "Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard," in *Conference on Human factors in computing systems.* ACM, 2000, pp. 518–525.

[25] Qi Chen, J. Grundy, and J. Hosking, "An e-whiteboard application to support early design-stage sketching of UML diagrams," in *IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003*, Oct 2003, pp. 219–226.

[26] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, "Usability meanings and interpretations in iso standards," *Software quality journal*, vol. 11, no. 4, pp. 325–338, 2003.

[27] A. Holzinger, "Usability engineering methods for software developers," *Communications of the ACM*, vol. 48, no. 1, pp. 71–74, 2005.

[28] A. Syromiatnikov and D. Weyns, "A journey through the land of model-view-design patterns," in *2014 IEEE/IFIP Conference on Software Architecture*, April 2014, pp. 21–30.