[GJRR22a] R. Gupta, N. Jansen, N. Regnat, B. Rumpe:
Design Guidelines for Improving User Experience in Industrial Domain-Specific Modelling Languages.
In: Proceedings of the 25th International Conference on Model Driven Engineering Languages
and Systems: Companion Proceedings, Association for Computing Machinery, Oktober 2022.
www.se-rwth.de/publications/

Design Guidelines for Improving User Experience in Industrial Domain-Specific Modelling Languages

Rohit Gupta
Siemens AG
Munich, Germany
rg.gupta@siemens.com

Nico Jansen
Software Engineering
RWTH Aachen University
Aachen, Germany
jansen@se-rwth.de

Nikolaus Regnat
Siemens AG
Munich, Germany
nikolaus.regnat@siemens.com

Bernhard Rumpe
Software Engineering
RWTH Aachen University
Aachen, Germany
rumpe@se-rwth.de

ABSTRACT

Domain-specific modelling languages (DSMLs) help practitioners solve modelling challenges specific to various domains. As domains grow more complex and heterogeneous in nature, industrial practitioners often face challenges in the usability of graphical DSMLs. There is still a lack of guidelines that industrial language engineers should consider for improving the user experience (UX) of these practitioners. The overall topic of UX is vast and subjective, and general guidelines and definitions of UX are often overly generic or tied to specific technological spaces. To solve this challenge, we leverage existing design principles and standards of human-centred design and UX in general and propose definitions and guidelines for UX and user experience design (UXD) aspects in graphical DSMLs. In this paper, we categorize the key UXD aspects, primarily based on our experience in developing industrial DSMLs, that language engineers should consider during graphical DSML development. Ultimately, these UXD guidelines help to improve the general usability of industrial DSMLs and support language engineers in developing better DSMLs that are independent of graphical modelling tools and more widely accepted by their users.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**; **Model-driven software engineering**.

KEYWORDS

Domain-Specific Languages, Model-Based Systems Engineering, Industrial Domain-Specific Modelling Languages, Industrial Language Engineering, User Experience

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10.
https://doi.org/10.1145/3550356.3561595

ACM Reference Format:
Rohit Gupta, Nico Jansen, Nikolaus Regnat, and Bernhard Rumpe. 2022. Design Guidelines for Improving User Experience in Industrial Domain-Specific Modelling Languages. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3550356.3561595

1 INTRODUCTION

With the advancement of various systems engineering domains, we are observing a notable shift in the way modelling is being introduced in projects early on. The engineering process in modelling involves models instead of documents for complex heterogeneous systems. Consequently, model-based development and model-based systems engineering (MBSE) techniques are constantly evolving [21]. General-Purpose Languages (GPLs) often provide difficulties in system modelling [44], most notably by ignoring domain experts from contributing to solutions. Instead, Domain-Specific Languages (DSLs) aim to reduce the gap in a particular domain by supporting domain-specific abstractions [14, 25]. The technological spaces for DSLs are heterogeneous, either textual, graphical, or projectional [4, 17, 48]. Regardless of their technological spaces, various domain-specific modelling languages (DSMLs) have been developed to support modelling in their respective domains. Naturally, as the complexity of these DSMLs increases, so does the complexity in its engineering, as does its consequence on users. This often leads to users, who may or may not be modelling experts, struggling to use DSMLs effectively.

Modelling involves key decision making in systems engineering. To assist end users in achieving their modelling goals, it is important for the DSML to convey all relevant aspects of their domain. The entire lifecycle involved in the modelling process should not only involve an elaborate modelling language, but also a solid methodological foundation and an appropriate tooling mechanism [25] (section 3). Only with the combination of these aspects would modelling be effective with novice and advanced users and with small and medium enterprises [45]. Therefore, as domains become more heterogeneous, the complexity of the syntax and the semantics of the language increases [13, 14], and the need for a good user experience (UX) also grows. Aspects of UX are often overlooked by language engineers because of the false notion that

users are always modelling experts in their domains and can understand every construct of a language easily. Especially in industrial DSMLs, novice users who are introduced to graphical modelling tools, often in the latter stages of the project lifecycle, need guidance because of their lack of know-how in modelling. Challenges of UX in using graphical DSMLs include, but are not limited to, improper visual representation of domain aspects, difficult to find and use language elements, burdening users with unnecessary tool functionalities [50], unavailability of predefined templates of models, and unclear documentation for the modelling language constructs.

Various definitions of UX and usability heuristics have been proposed in the literature to solve these challenges [27, 38]. However, these are overly generic and often tied to specific technological spaces. Further, UX is a subjective topic, the preference of users vastly differs across domains and is influenced by recent trends from the ubiquitous software domain (section 2). UX and business logic is often intertwined in projects, and while considerations are made by involving key stakeholders early on, they are often lost in translation as the project complexity grows. Naturally, no cookie cutter solutions exist, but there are industry standards, design guidelines [15, 35, 49], and user experience design (UXD) aspects that language engineers should consider when developing graphical DSMLs. Improving UX for users is key to achieving modelling goals. To this end, we aim to provide guidance based on our experiences in developing industrial DSMLs combining aspects of UX for a more holistic modelling experience for users of various domains.

Given our years of experience in building industrial DSMLs for a variety of users in various domains using graphical modelling tools such as Enterprise Architect [18], Rational Rhapsody [29], and MagicDraw [37], we put the choice on a firmer basis in this paper:

- We define the terms *user experience (UX)* and *user experience design (UXD)* for graphical DSMLs in general modelling tools on the example of MagicDraw (section 4).
- We discuss human-centred design approaches and define the important UXD aspects: visual design, information architecture, interaction design, and usability heuristics that language engineers should consider during graphical DSML development.
- We illustrate the UXD aspects using an example of a feature model DSML with product lines and products (section 5).
- Finally, we clarify the scope of UX in our modelling approaches, discuss the benefits of our approach (section 6), related work (section 7), and conclude the paper (section 8).

2 BACKGROUND

The engineering of DSLs and DSMLs comes with the usual challenges faced in software engineering, such as maintenance or evolution, as they are software themselves [19]. Generally speaking, a software language consists of [13, 14]: (1) an abstract syntax that defines the structure of its models, e.g., in form of context-free grammars [28]; (2) a concrete syntax that defines how the models are presented, e.g., graphical [17], textual [4], or projectional [12], (3) semantics, in the sense of meaning [26]; and (4) context conditions, to check the well-formedness of a language. Methodologies for developing graphical DSMLs in the industry are often tied to specific departments, where language engineers must be trained to represent the domain in consideration effectively. The development of such graphical DSMLs must also include all functional and non-functional aspects of the language. Only with the combination of a modelling language, a methodology, and a modelling tool can this challenge be fully addressed. Central research units such as Siemens Technology, therefore, provide guidelines to their language engineers in designing DSMLs that improve the overall UX for both novice and advanced modellers across various domains.

Several definitions of UX [27] and best practices for DSLs and model-driven development [49] exist in general. Consequently, a number of notions, such as feelings, experiences, and insights, are subsumed under the definitions of UX. This could be attributed to the fact that UX is a vastly subjective topic covering a multitude of domains, one that cannot be possibly represented by a single definition. ISO 9241-210 [31] defines UX as "person's perceptions and responses resulting from the use and/or anticipated use of a product, system or service". While such a definition can be reused in the context of graphical modelling, it is overly generic, needs more clarity, and does not discuss domain-specific aspects. There is also a lack of consensus as to which UX and usability definitions (e.g., ISO 9241-11 [6], ISO 13407 [33], Nielsen's [41]) are best suited for graphical DSMLs. Aspects of Human-Computer Interaction (HCI) discusses how people interface with computers (e.g., ISO 9241-161 [30]) and how it can be leveraged to obtain practical results related to user interfaces [43]. The proposed definitions of UX, usability, and HCI are often overlooked by language engineers [1], who are often neither UX experts nor domain experts in various domains. Thus, developing graphical DSMLs requires focusing on specific aspects of UX, and that a one-size-fits-all approach is probably not suited.

A good UX is key to modellers effectively reaching their modelling goals. Benefits of a good UX in modelling ameliorate the challenges between the users and the constructs of the language and reduce unnecessary burdens of the modelling tool functionalities. In our experience of developing a diverse set of industrial graphical DSMLs, we have not come across a standard set of UX and usability guidelines that language engineers should consider during graphical DSMLs development. We believe there is a need to start the discussion towards defining specific UX guidelines that benefit novice and advanced users and is ultimately independent of a specific implementation or a graphical modelling tool.

3 MODELLING IN MAGICDRAW

The methodologies for the development of graphical DSMLs are generally tied to specific departments in a large organization such as Siemens AG. This introduces challenges in the way the combination of a modelling language is used with a methodology and using an appropriate graphical modelling tool. We have focussed on MagicDraw as a choice of modelling tool in developing graphical DSMLs, as it is based on the Unified Modelling Language (UML), comes with comprehensive extensions such as the Systems Modelling Language (SysML) plugin, and provides a wide range of customization possibilities that are used to capture most, if not all, issues of domain-specific language engineering. A systematic engineering process of developing industrial DSLs [20] using modular reusable DSL Building Blocks in MagicDraw is described in [25]. Each building block conceptually consists of language components, a method and a user experience design (UXD) part. The re-usable

language components define the language [47], wholly or in part. The method part describes a suitable methodology for the language to help users achieve their modelling goals. This can be in the form of training material, including methodical steps, that ultimately provide a comprehensive guidance to DSML users. Finally, the UXD part describes the design decisions language engineers must consider to improve the overall usability and UX of a DSL or DSML. The heterogeneous building blocks are finally composed together to create the DSL. The combination of the modelling language, method, and design decisions makes modelling effective and simpler for all kinds of users, novice and advanced alike. To this end, we consider UX aspects in a DSL as important as the modelling language itself.

The development process is iterative and involves all key stakeholders in the project. A feedback loop is established between modellers, domain experts, and the language engineers to ensure aspects of the domain are continuously integrated in the DSML. Further, UX experts are involved to ensure aspects of user experience and usability are effectively captured in graphical DSMLs. The customization capabilities of MagicDraw allows creation of a language profile consisting of language component artefacts. An example of an artefact is a language element, referred to as a *stereotype* in MagicDraw. In addition, *customization* elements allow the definition of rules for the MagicDraw DSL customization engine. Figure 1 shows an example of the configuration of stereotypes for a feature model [11] along with its relationships and customizations in MagicDraw. These customizations define what model elements or diagrams can be created and where, how an element shortcut menu should look like, what properties of the elements are presented to the user, and so on. MagicDraw also provides a specific Java based plugin mechanism that supports the integration of automation and creation of custom functionalities. Additionally, model templates allow a predefined model structure to be automatically instantiated during modelling and perspectives help configure the visible number of functionalities of MagicDraw for different kinds of users.

Over the years, we have used this methodology to build industrial DSMLs in MagicDraw for a variety of consumers in various domains. In the case of Siemens Healthineers, we developed DSMLs that focussed on the modelling of medical devices, imaging for radiation therapy, and X-ray equipment. For Siemens IT, we developed a DSML that models the concerned IT workflow and inventory management processing. For Siemens Energy AG, we developed DSMLs that allow for the modelling of complex industrial appliances such as turbines along with their workflows. In the case of Siemens Digital Industry, we developed DSMLs to model the hardware and software aspects of the SINAMICS[1] frequency converter that allows for variable frequency drives and control systems, whereas the SIMOTICS[2] electric motors allow the modelling of synchronous as well as asynchronous industrial motors. In a recently ongoing public funded SpesML project [46], we built modular reusable DSL Building Blocks to separate the concerns of the models created for a system under development as different heterogeneous viewpoints. In all of these projects, feedback from users regarding the usability and UX of the DSMLs, to invoke positive feelings during the interaction with the DSMLs, have been of paramount importance.

[1]https://new.siemens.com/global/en/products/drives/sinamics.html
[2]https://new.siemens.com/global/en/products/drives/electric-motors.html

4 USER EXPERIENCE IN MAGICDRAW

Definition 1 (User Experience in MagicDraw). *We define user experience (UX) for graphical DSMLs in MagicDraw as an instantaneous intuitive feeling (positive or negative) of a user (modeller) while interacting with the defined constructs of the graphical modelling language and the graphical modelling tool, MagicDraw.*

In other words, UX is primarily an intuitive feeling for users during modelling with the aim that a good UX satisfies their modelling expectations in easy, simple terms while also minimising the number of interactions required between them and the modelling tool. These interactions are the abilities of systems and users to influence each other in order to reach a goal. We consider any positive feeling during the interaction with the modelling language and the modelling tool as a good UX. A bad UX generally tends to invoke negative feelings that not only leaves DSML users dissatisfied, but also introduces incomprehensibility between the different stakeholders in their modelling.

Definition 2 (User Experience Design in MagicDraw). *We define user experience design (UXD) as any design decision taken by a language engineer during the development of a graphical DSML in MagicDraw, that ultimately fosters a good UX for a user.*

The design decisions are realized and implemented by language engineers in consultation with practitioners during the graphical DSML development process. Any design decision should follow the principles of human-centred design as defined in ISO 9241-210. Language engineers must take into account the people who use the graphical modelling language as well as other stakeholders who are involved in the project. The following categorization of UXD aspects in MagicDraw originate from experiences found in the literature [23]. These UXD aspects are non-exhaustive but provide a general foundation for graphical DSMLs, independent of their modelling tools. Each of these design decisions is elaborated with a rationale that provides a reasoning as to why the design decision is required and consequently what its benefits are. These rationales are also based on feedback from users and domain experts in various projects (section 3). We also assign an identifier (in brackets) to distinguish each design decisions based on their categories.

4.1 Categorization of UXD

4.1.1 Visual Design. Visual designs represent the aesthetics or the look and feel of models and model elements and how they are presented to users [38]. Model elements can be configured in the form of various icons, colours, appearance, dialogs along with their properties such as shape, size, and opacity. In other words, the graphical concrete syntax of the DSML is enhanced using the following visual designs to effectively represent the various heterogeneous domains involved in modelling:

- **Icon (V-1)**: an extra graphical element that is displayed upon selection for a specific model element.
  Rationale: Icons help to distinguish between different model elements and convey real-world representations of a specific entity with some meaning for a user.
- **Colour (V-2)**: enhances the appearance of a model element through a specific colour.

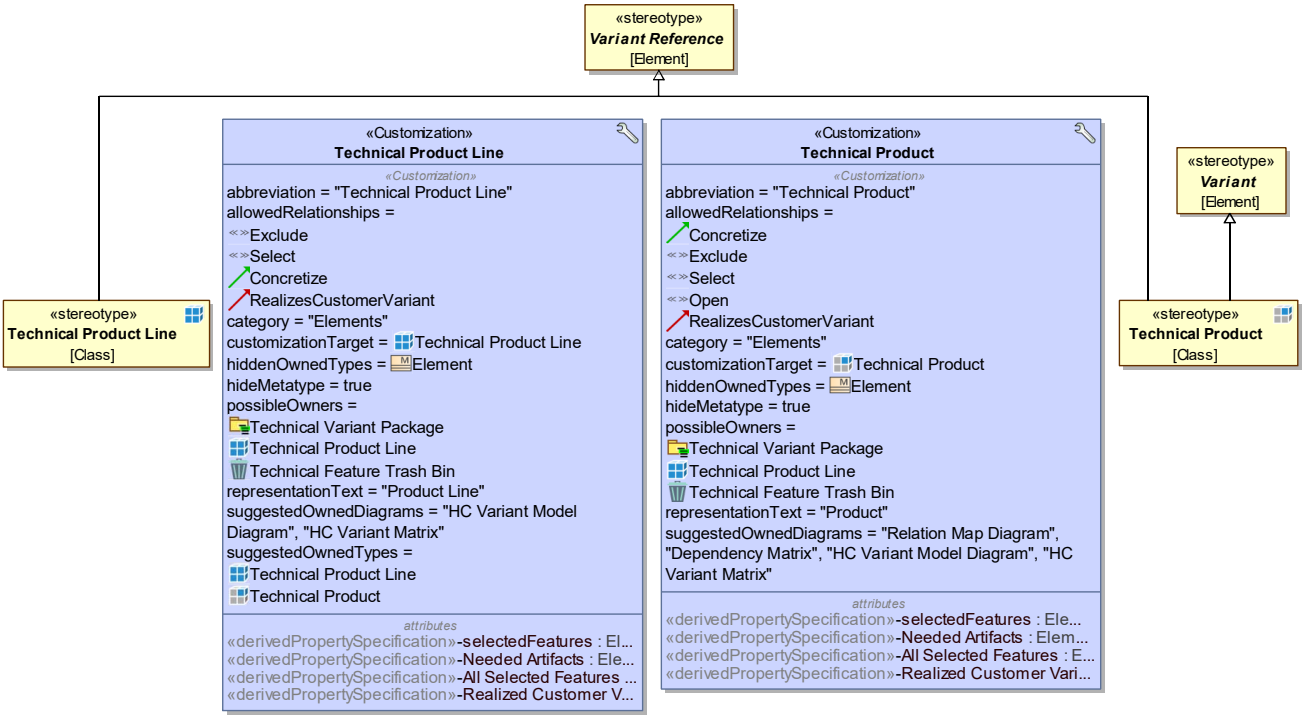

Figure 1: An example of the configuration of the product line and product language elements for a feature model DSL using the customization capabilities of MagicDraw.

Rationale: Colouring schemes such as red for errors or warnings provoke an increased user attention [10] and subsequently aids in differentiating model elements.

- **Modal Dialog (V-3)**: a graphical control element showing information to users on making relevant modelling decisions.
  Rationale: Modal dialogs direct important information towards users and therefore require a user's immediate attention. Although it is useful in preventing or correcting vital errors, a drawback of a modal dialog is that it often interrupts a user's workflow.
- **Custom View (V-4)**: a visual representation of the textual information of model elements in the form of matrices, tables, UML or free-form diagrams.
  Rationale: Custom views represent textual information visually using specific diagrams that contains information in a particular format. As each diagram serves a specific purpose, language engineers must consider the suitability for such diagrams and implement them in the language definition.
- **Dynamic View Plugin (V-5)**: a GPL (e.g., Java) based plugin for enabling dynamic filtering and/or displaying specific model information (e.g., legends, annotations) on existing UML diagrams or custom views.
  Rationale: Dynamic view plugins help users focus on specific model information, for example, supporting the enabling or disabling of a power supply unit in a complex power system. Other model information such as legends or annotations from other users can also be toggled using a dynamic view plugin, allowing quick display or removal of information on different views based on certain filtering conditions.

4.1.2 Information Architecture. Information architecture is the practice of structuring and organizing the constructs of the graphical DSML in a way that they are easy to find and use [16]. As heterogeneous systems grow in complexity, so does the complexity in modelling. With a growing number of domain concepts and functionalities that a graphical modelling tool provides, there is a need to alleviate the concerns of users, novice or advanced, for presenting the DSML and the functionalities of MagicDraw in a structured and organized manner. We consider the following information architecture designs important in graphical modelling:

- **Layout (IA-1)**: determination of the position of model elements on custom diagrams based on the context of use.
  Rationale: The placement of model elements on specific areas of custom diagrams helps provide structure in complex modelling scenarios. An example is the positioning of incoming ports, generally to the left of a model element, or outgoing ports, which are positioned to the right of a model element.
- **Model Browser (IA-2)**: a visual representation of the hierarchy of model elements. It is a hierarchical navigation tool for managing the model data.
  Rationale: Users often struggle to quickly navigate, find, or arrange model elements, especially in complex DSMLs. A

model browser is designed to provide a sound hierarchical structure and organization of such model elements.

- **Perspective (IA-3)**: displaying a fixed set of modelling language constructs or tool functionalities based on the kind of user, novice or advanced.
  Rationale: Often, novice users do not need additional tooling functionalities or language constructs to start their modelling. As the complexity of models increases, users tend to use more advanced concepts that require the enabling of advanced functionalities.
- **Creation View (IA-4)**: an additional pane or window that shows the logical grouping of different language elements, standard UML diagrams and custom views during model element creation on the model browser.
  Rationale: Language elements that belong to a certain logical grouping should not belong to other logical groups to avoid disorder and inconsistencies in structuring and organizing the constructs of the modelling language. As an example, model elements and custom views of functional and non-functional requirements should exclusively belong to a logical group that defines a requirements specification.

4.1.3 Interaction Design. Interaction designs in MagicDraw help users interact effectively with the constructs of the graphical DSML while focussing on the cognitive dimensions [7, 40]. Such designs subsumes auto-completion of model element names, automatic instantiation of model elements, and transformation of models into other formalisms [36]. Interaction designs focus mostly on the behaviour of the modelling language constructs. We consider the following interaction designs important in graphical modelling:

- **Project Template (ID-1)**: is a customized project pattern that serves as a starting point for creating a project in a predefined format.
  Rationale: When users start modelling for the first time, they often lack the know-how on creating a robust, well-organized model consisting of possible model elements. Project templates help such users with the automatic instantiation of a model in a predefined format. An example is the automatic creation of a basic traffic light model as states in a state machine [25], with only the triggers or actions that need to be additionally defined by the user.
- **Default Naming Scheme (ID-2)**: a naming scheme automatically assigning default names or numbers to model elements.
  Rationale: Users often forget to assign relevant names, labels, or numbering to model element to help them distinguish from other model elements and to avoid naming conflicts. In complex systems, this problem cascades drastically and adds unnecessary debug times. An example of a default naming scheme is to create a functional requirement as "Functional Requirement x", x being an automatic incremental number.
- **Model Transformation (ID-3)**: a transformation of a model into another formalism.
  Rationale: Model transformations are an effective way to dynamically transform models into other formalisms during design time. An example of a model transformation is to transform a model from an optional feature to a mandatory feature in a feature model during modelling. The user should be able to simply achieve such a transformation.
- **Custom User Interface (ID-4)**: a custom user interface (UI) programmed using frontend frameworks (e.g., Java Swing) to access and/or edit specific model and model elements based on the DSML requirements.
  Rationale: Some modelling tools may not provide out of the box functionalities to achieve a specific purpose. A custom user interface adds functionality to a modelling tool by providing enhanced capabilities. An example of a custom UI is to edit or configure model information that may not be possible using the modelling tool's default functionalities.

4.1.4 Usability Heuristics. Usability heuristics provide guidelines to language engineers in making decisions during graphical DSML development that are ultimately beneficial to users in achieving modelling with a greater sense of effectiveness and satisfaction [42, 43]. Usability heuristics, should therefore, complement the earlier proposed design decisions. Since most graphical modelling tools have an underlying Application Programming Interface (API) to support in the development of DSMLs, we consider the following usability taxonomy attributes, proposed and compared to UX studies in APIs [39], important in graphical modelling:

- **Knowability (Clarity) (U-1)**: constructs of the modelling language should be self-explanatory.
  Rationale: Readability is often a challenge users face during modelling of complex scenarios. A clarity in terms of names, types, structure, logical grouping of common model elements, easy to comprehend modelling constructs, which elements to use where and when, and the ability to design model elements that serve a particular function, is critical to improving the overall modelling experience.
- **Knowability (Helpfulness) (U-2)**: the graphical DSML should provide helpful annotations and documentation to users and also identify deprecated elements.
  Rationale: Documentation is a key aspect often missing when building DSMLs. Users are left to comprehend the meaning and usage of modelling language constructs by themselves, which is tedious and time consuming. Providing sufficient documentation with good usage examples reduces the effort needed to model complex elements and consequently limits introducing errors during modelling.
- **Operability (U-3)**: the graphical DSML should provide the necessary domain-specific functionalities in addition to being extensible for further language compositions.
  Rationale: DSML users should be presented with constructs that are relevant to their domain and performs the specified functionality. Model elements should be precise, universally recognized, and flexible to allow language compositions and extensibility to support effective model transformations.
- **Robustness (U-4)**: the graphical DSML should be well-formed and be free from bugs and vulnerabilities that could potentially expose flaws in the system.
  Rationale: Every DSML should be thoroughly tested to ensure that runtime errors are not encountered during modelling with a graphical modelling tool. The DSML constructs should be error free and checked for potential vulnerability leaks.

- **Safety (U-5)**: the graphical DSML should not compromise the confidentiality or the assets of a user.
  Rationale: Data belonging to the user should not be exposed to unauthorized entities at any cost. License of use, legality, and the personal information of users and their data should be protected to ensure the safety of users' assets.

4.2 Scope of UXD

The overall topic of UXD is very wide and can potentially cover a lot of scenarios. The design decisions we discuss are non-exhaustive, yet important in the development of any graphical DSML and leads to a good UX. Further, the combination of quality management standards such as ISO standards along with the ergonomics of human-system interaction provide best practices to improve graphical DSMLs for a seamless UX. Language engineers should not only focus on the aesthetics of modelling language constructs but also account for design decisions that improve the general usability of graphical modelling languages. It is also common for language engineers to focus more on the functional aspects of the modelling language. Constraints in project duration, resources, and budget introduces trade-offs in design decisions that language engineers must consider. To this end, we propose the following three questions that language engineers must answer during language development:

- **Q1**: Does a specific design decision fulfil a user's needs or a modelling goal?
- **Q2**: Is the design decision a cause for any potential conflict, either between the constructs of the modelling language or with the existing functionalities of the modelling tool?
- **Q3**: Is the design decision specific, non-subjective, and has relevance to the domain in consideration?

Answers to the above questions help language engineers define the scope of UXD. In principle, those design decisions that are not aligned with the modelling goals should not be considered. These must be examined with all the stakeholders in the project to avoid compromising the overall quality and UX of a graphical DSML.

5 CASE STUDY

To demonstrate the significance and implications of the presented UX guidelines, in this section, we show their effects using a DSML in MagicDraw. In Figure 1, we presented an example of the configuration of the language elements, product line, and product for a feature model [11] DSML that has been developed for Siemens Healthineers. Further, we reuse the feature model DSL building block, in other DSMLs such as the one we developed for Siemens Digital Industry (section 3). The customization capabilities in MagicDraw allow design decisions such as icon (V-1), colour (V-2), default name (ID-2), and creation views (IA-4) to be incorporated directly into the stereotype definitions and their corresponding customizations. Modal dialogs (V-3) are created using the MagicDraw Open Java API capabilities and custom views (V-4) are created using the customization capabilities offered by MagicDraw. The information architecture and interaction designs are also configurable using the MagicDraw configuration settings. Java based plugin mechanism allow various plugins creation, such as our dynamic view plugin (V-5). Plugins that enable model transformations (ID-3) and custom UI (ID-4) are also integrated directly into the feature model DSML.

Figure 2 shows an illustration of a feature model created by a user in MagicDraw and is based on the design decisions defined by the language engineers. The model browser (IA-2) is a functionality of MagicDraw that hierarchically organizes and structures the model constructs, such that a user can find and navigate quickly through model elements. A predefined feature model project template (ID-1) is created automatically inside the model browser upon the creation of a new feature model project. This template structures and organizes the various models configured internally as DSL building blocks (section 3). This ensures that all the relevant models and model elements are exclusive to their respective building blocks and eventually help users in quicker navigation and ease of finding models, thus addressing Q1. The predefined project template includes a default naming scheme (ID-2), with automatic naming and numbering conventions, such as "1 UseCase Model" and "2 Requirements Model", so that users do not forget to add relevant identifiers and are also guided to create models in a sequential manner (U-1). Each of the model elements is also configured with icons (V-1), and colouring (V-2) of features in the feature model. While MagicDraw, by default, provides the ability to set icons and colours, it is left to the language engineers to design the appropriate icons and colouring to their model elements that effectively represents the domain in consideration. In our example, we designed the product line icon to show four blue cubes whereas a product icon, say P1, is configured to show only one blue cube and three other white cubes, thus providing the distinction between a product line and a product. Similarly, design differences between the icons of different feature types, such as mandatory (with an exclamation mark) and an optional feature (with a question mark), provides the appropriate distinction between the different model elements and helps improve the aesthetics for users during modelling. The designs for the icons we built for our feature model are similar to the icons used by fully featured product line engineering tools like pure::variant [5]. We also note that novice users may not necessarily be aware of standard representations, and therefore proper documentation (U-2) for such constructs are also needed.

MagicDraw offers perspectives (IA-3), for enabling or disabling certain MagicDraw functionalities that maybe beneficial to advanced users. In addition, language engineers can use this functionality to restrict certain modelling constructs that is only visible to advanced users. In our example, the perspective (IA-3) is configured to a beginner (a novice user), showing only basic MagicDraw toolbar options that is a good starting point for novice users. A perspective for an advanced user allows additional functionalities such as collaboration on cloud servers for project migration, advanced context condition checking, and merge of different projects. Perspectives do not cause any potential conflicts between the feature model DSML constructs or the existing functionalities of MagicDraw, thus addressing Q2. The custom variant matrix (V-4) shows various product lines or products and their respective connections to the various features. This is also supported by specifying a layout in the matrix definition (IA-1) that combines the variants and the features. In our example, we show the product lines and products in the rows of the matrix, whereas the features are displayed on the columns for the matrix. The legend information, enabled using our dynamic view plugin (V-5), on the matrix shows the kinds of relationships available for the product lines, products, or features.

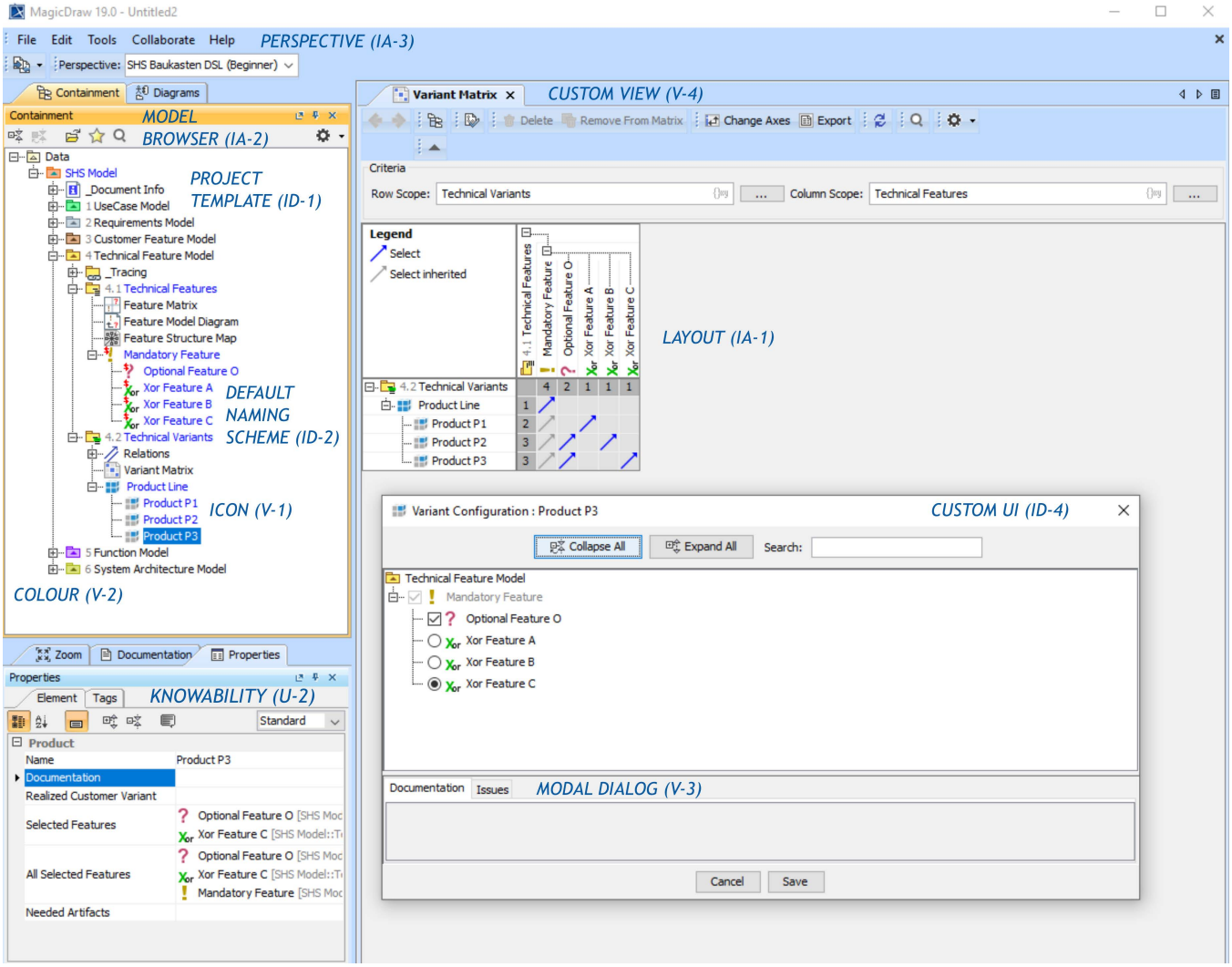

Figure 2: Annotations of design decisions in MagicDraw for a feature model consisting of (1) product lines and products, (2) mandatory and optional features, and (3) a UI for variant configuration. The figure shows enhanced aesthetics of models, the structuring and organization of model elements, the interactive aspects focussing on the cognitive dimensions, and the various usability aspects that complement the design decisions.

This would otherwise be not possible as matrices and tables, in contrast to class diagrams, composite structure diagram, and so on, do not generally allow legends information by default in MagicDraw. Additionally, the matrix is interactive, meaning users can directly right click on an empty cell and choose a relationship, directed from the variants to the features, thereby addressing Q3. Using the MagicDraw Open Java API a dedicated variant configuration user interface (ID-4) is shown to users, allowing for a convenient configuration of product lines or products. Based on the selection of users, we can automatically create or remove relationships between the model elements. Additional options, such as collapsing and expanding of the hierarchical structure of features, and also the inclusion of a search bar to effectively search for features in a complex or long list of items, enhances the configuration of each variant. A modal dialog (V-3), integrated with the whole variant configuration UI (ID-4), presents additional documentation and issues that could possibly occur during a variant configuration. In general, the usability heuristics are applied throughout the DSML. They ensure that modelling language constructs remain self-explanatory (U-1) and are supported with helpful documentation and annotations (U-2). Language engineers must also validate and verify the DSML and any external source code, such as GPL code to create user interfaces or dynamic view plugins, to ensure robustness (U-4) and safety (U-5) of the modelling language and the modelling tool.

Figure 3 shows an illustration of how a custom view (V-4) is integrated with the Java API based dynamic view plugin (V-5) in a feature model to enhance the UX in a graphical environment. While only the graphical modelling canvas is provided by MagicDraw, the

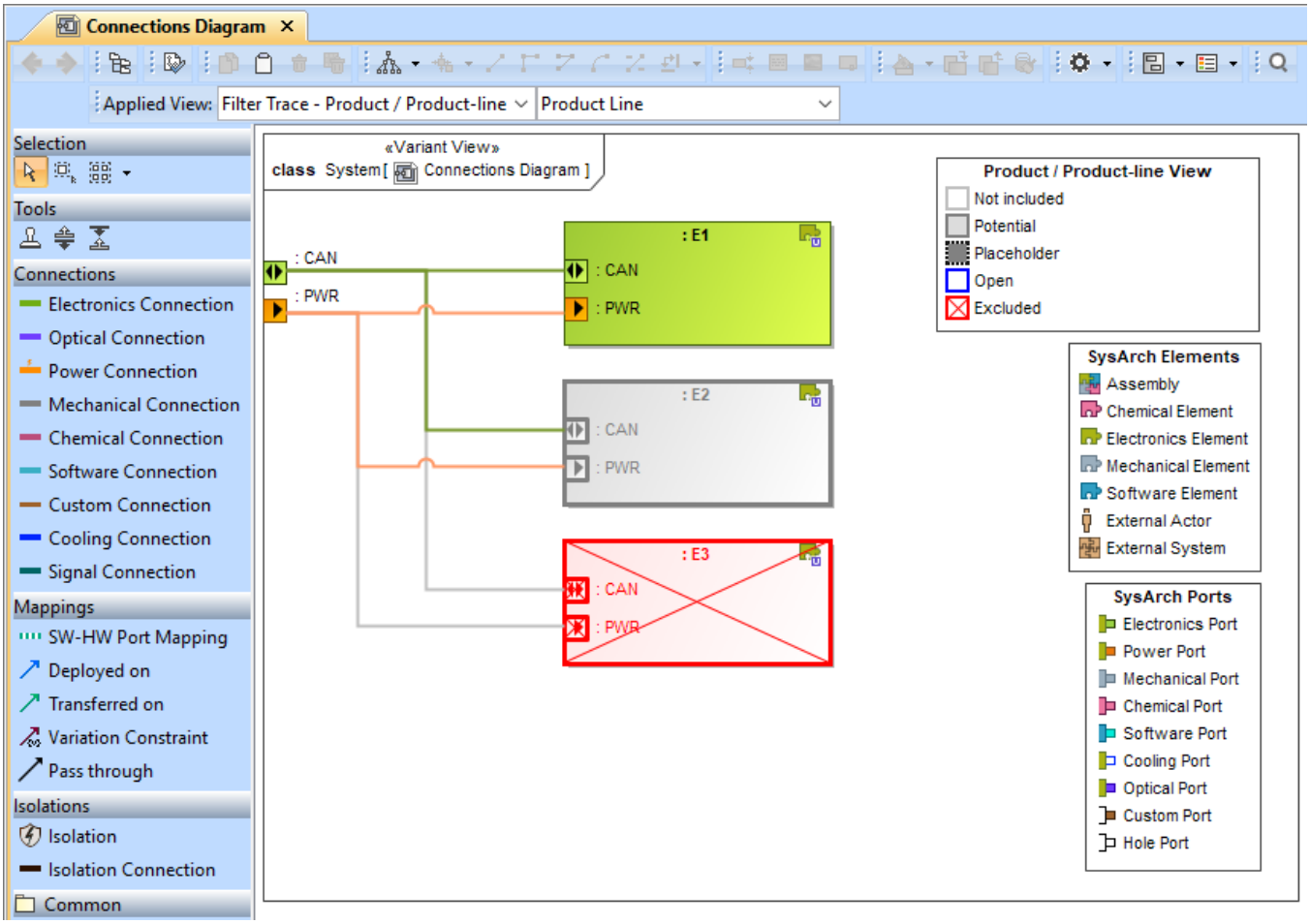

Figure 3: An example of a custom view (V-4) integrated with a dynamic view plugin (V-5), that filters diagram elements based on a configured product line or product. The element E2 is marked as potential, which means that the element might or might not be included in the products, but on the currently displayed product line it is still undecided. On the other hand, the element E3 is marked as excluded, hence the respective connections are greyed out and the element is marked with a red cross.

configuration and structuring (IA-1) of model elements inside the canvas is configured by the language engineers. During design time, users can reposition and restructure these model elements based on their needs, fostering flexibility in their modelling. The applied view can be dynamically changed to select either a product line or a product in a feature model tree. The selected configurations for this particular variant, a product line in this case, is dynamically updated and made visible to users in this connections diagram (V-4). In this example, the electronic element E1 is configured to be included (IA-1) using a custom user interface (ID-4). Therefore the electronics and power connections to E1 is also made visible to this product line on this custom view. The element E2 is marked as potential, meaning it is currently not included in the currently displayed product line, but could be included for a subsequent product in the product line. In this scenario, the respective connections are visible, although the element itself is greyed out for the product line. Finally, element E3 is marked for exclusion for this product line, meaning the features are excluded for subsequent products. Such a scenario can exist when building a low cost variant of a product line requiring less features. For E3, the connections are greyed out and the element is marked with a red cross (V-1, V-2).

Model transformations (ID-3) in the form of refactoring of models, is achieved by changing the stereotype configuration of a model element. A modal dialog (V-3) is created by language engineers using the MagicDraw customization capabilities to support such a kind of model transformation. Figure 4 shows an example of a model transformation (ID-3) for refactoring features in a feature model. An already defined optional feature in a feature model can be refactored to a mandatory feature. While such a transformation is allowed during design time to enhance the modelling experience and to introduce extensibility of the feature model language, it also includes the risk of losing certain incompatible properties, which is subsequently informed to users using a modal dialog (V-3).

Figure 5 illustrates an example of a creation view (IA-4), created by language engineers, that lists the creation of only feature elements: mandatory, optional, or, and Xor. This additional window

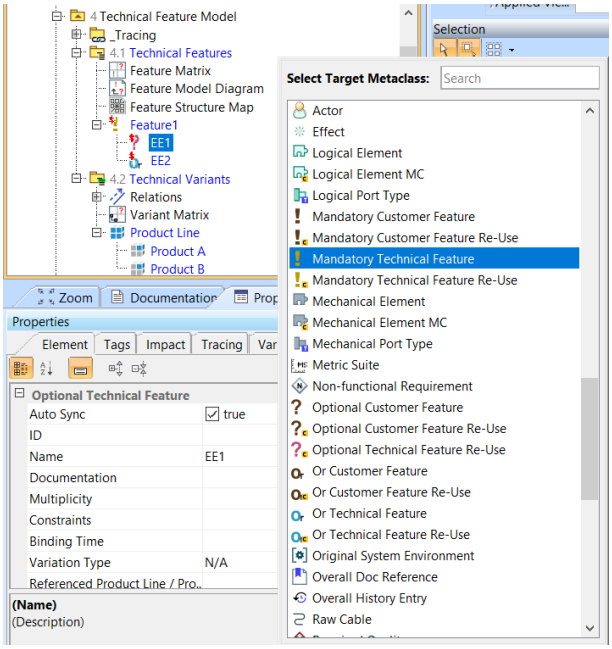

Figure 4: An example of a model transformation (ID-3) in the form of refactoring an optional feature to a mandatory feature. While the modal dialog for selecting the target metaclass lists all possible element conversion types, the risk of losing incompatible properties is informed to users.

showing the logical grouping of language elements ensures that users can only create the above mentioned four features under the "4.1 Technical Features" package. This avoids inconsistencies in model element creation, such as the creation of features under the "4.2 Technical Variants" package, which introduces disorder in logically structuring and grouping elements together. Further, a default naming scheme (ID-2) not only assigns relevant names and labels to features and variants, but also guides a user into first creating the features, within "4.1 Technical Features" package. Next, users create the variants, within "4.2 Technical Variants" package, needed for a product line or a product configuration.

6 DISCUSSION

In this paper, we present definitions for UX and UXD for graphical DSMLs in MagicDraw. Further, we discuss a categorized set of design decisions that help improve UX for all kinds of users to achieve their modelling. Many traditional approaches to developing graphical DSMLs exist, but they often lack good UX aspects that language engineers must consider during DSML development. The combination of a methodology along with a modelling tool elevates the experience of using graphical DSMLs, which can get complex and hard to use. In our years of experience in building graphical DSMLs for industrial projects, we have explored MagicDraw as a graphical modelling tool capable of providing the combination of a modelling language, methodology and good UX for both novice and advanced users. Relevant key stakeholders must be involved from the start of the project. An iterative feedback loop should be established

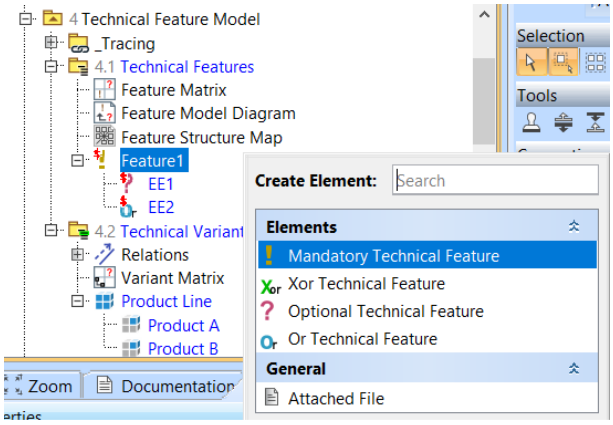

Figure 5: An example illustrating the creation view (IA-4) where the creation of a model element inside a Technical Features package allows only a mandatory, or, Xor, or an optional feature to be created.

between users (modellers), domain experts, and language engineers to understand the specifics of the domain in consideration.

Language engineers are certainly not UX or programming experts, therefore, they often need additional trainings. Consequently, design decisions taken by language engineers are aimed at improving the UX. Categorization of the design decisions achieves separation of concerns for language engineers possessing different skills. Visual designs improve the aesthetics of the model elements and the models. Models should be designed to look and feel similar to their real world abstractions for easy understandability. Icons, colours, modal dialogs, and custom views can be configured to model real world abstracts. Information architecture design decisions help organize and structure the constructs of the modelling language so that they are easy to find and use. Interaction designs help users interact effectively with the modelling language while focussing on the cognitive dimension. These designs alleviate problems of building models from scratch with little guidance. Finally, usability heuristics provide the necessary guidelines to language engineers in developing graphical DSMLs that improve the effectiveness and satisfaction of users. Therefore, we consider the taxonomy attributes of knowability, operability, robustness, and safety most important.

The customization capabilities of MagicDraw allow design decisions to be easily integrated into the DSML. Each language element can be configured as per the project requirements and language components consisting of such language elements, or stereotypes, can be easily created and bundled together with MagicDraw. Functionalities that are not provided by default can be achieved through Java based plugin mechanisms in MagicDraw. Finally, all the components are composed together into the final DSML that is used by modellers to realize their modelling. Leveraging the customization capabilities of MagicDraw allow language engineers to capture a greater variety of design decisions that ultimately are beneficial for all kinds of users. Further, building blocks from heterogeneous domains are composed together that ultimately fosters re-usability of language components and design decisions.

Generally, our design guidelines for improving the UX are tailored to industrial DSMLs. However, they naturally apply to all

graphical DSMLs (e.g., research). We specifically focus on industrial languages since they have a greater necessity for good UXD decisions. While DSMLs in research often represent proofs of concept that do not need to support all categories of visual design, information architecture, interaction design, or usability heuristics, this is essential for industrial applications as they are intended for practical use. Therefore, although the guidelines have a more general claim, they present particular challenges in the industrial context. Specific guidelines, such as defining a custom UI for generating abstract syntax trees from a textual grammar, could also be considered for non-graphical DSMLs but needs more research. Additionally, while the presented guidelines are primarily based on our experiences with the modelling tool MagicDraw, they generally apply to comparable frameworks as well, as the principle of graphical modelling remains unchanged.

Naturally, no single solution exists for improving the UX in graphical DSMLs. This is due to the fact that UX is a vastly subjective topic covering a wide range of domains and is a research area that transcends traditional usability heuristics. Often, language engineers are not UX experts and struggle to effectively convey the semantics of a modelling language. While some definitions of UX can be reused in the context of graphical DSMLs, it is often overly generic or tied to very specific technological spaces. We believe our definitions of UX and UXD along with the categorization of UXD aspects in MagicDraw will foster the discussion in improving UX for graphical DSMLs across various modelling tools. Language engineers should work cohesively with UX experts in integrating design decisions during industrial DSML development for improving the UX. Consideration should also be made for language engineers to be trained with relevant domain and UX knowledge. The aspects of UX described in our paper are not exhaustive, but in our experience of developing industrial DSMLs, ameliorates the challenges faced by a multitude of users in graphical modelling. While these design aspects may seem specific to MagicDraw as tooling functionalities, we observe that other graphical modelling tools also provide ample functionalities and customization capabilities for the effective application of these design decisions. We intend on exploring the application of the design aspects considered in this study to other modelling tools as well. In this way, both the risk to a vendor-locked scenario and the generalizability of the design aspects can be overcome. This would also help us shape, refine, and improve the design guidelines to be able to perform an elaborate guideline review. Further, the scope of UX and UXD must be defined by language engineers and trade-offs in design decisions must be carefully analysed according to project requirements, resources, and costs. One key takeaway in developing industrial DSMLs across various domains over the years is that these guidelines improve the overall DSML experience of users and domain experts in modelling effectively. To this end, we consider the proposed UX and UXD guidelines in this paper a good reference point for future discussions towards defining UX aspects for industrial DSMLs that is independent of graphical modelling tools.

7 RELATED WORK

Many studies exist on general design guidelines for DSLs and DSMLs [3, 22, 35]. While these guidelines apply generically to DSLs, our guidelines are tailored specific to graphical modelling tools and focusses on industrial DSMLs designed for practical use. Studies on usability driven development of DSLs [2, 8, 42] focus on usability evaluation during the DSL development process to address usability problems, but miss discussions on a variety of complex industrial domains. In contrast, our guidelines have been applied over the years on a wide variety of industrial domains (section 3). Often, experimental usability evaluations [34] do not consider all stages in the DSML development lifecycle. Our guidelines propose that the design decisions are taken early on in the projects with all the stakeholders involved. While crowdsourcing and collaborative techniques for shaping graphical notations of DSLs have emerged [9, 32], it allows for more subjectivity as it permits a wider spectrum of users to collaborate, validate, and promote DSL acceptance. For the description of interaction designs in our guidelines, the cognitive dimensions of notations framework (CDF) [7, 24] has been considered. This is beneficial in covering cognitive dimensions aspects of *hidden dependencies*, *diffuseness*, and *viscosity*. While human-centred design approaches for usability evaluation [43] have been reviewed, we consider the principles of human-centred design defined in ISO 9241-210 [31] to be generally applicable. In our work, we consider only certain DSL usability heuristics proposed by [39] important in the context of industrial DSMLs. Further, challenges and future direction for UX in model-driven engineering approaches is discussed in [1], whereas our work focusses on improving the general usability and UX in industrial graphical DSMLs.

8 CONCLUSIONS

As systems become complex and heterogeneous in nature, challenges in developing industrial DSMLs that ameliorate UX have emerged. Modelling tools such as MagicDraw aim to improve UX by introducing a variety of customization capabilities during graphical DSML development. While such tools provide sufficient functionalities for developing a modelling language, there still exists the challenge of providing UX and UXD guidelines for language engineers. To address this challenge, we leverage the standards of human-centred design and UX in general, and propose aspects of UX and UXD for graphical modelling languages in MagicDraw, which we hope is generalizable to other graphical modelling tools. We categorize design decisions by utilising the wide range of customization capabilities in MagicDraw: visual designs to improve the aesthetics of model and model elements, information architecture to organize and structure the constructs of the modelling language, interaction designs to help users interact seamlessly with the modelling language and providing usability heuristics language engineers should consider to help users model with effectiveness and satisfaction. These aspects of UX and UXD improve the overall experience of novice and advanced users in using graphical DSMLs. Further, these guidelines serve as a reference point for future discussions towards defining UX aspects for graphical DSMLs. Naturally, UX is a subjective topic and our proposed list of design decisions are non-exhaustive. We consider discussions on improving UX for graphical modelling for every kind of user important. We believe guidance for language engineers in developing industrial DSMLs that combines various UX aspects ultimately improves the usability and UX in graphical modelling that is independent of graphical modelling tools.

REFERENCES

[1] Silvia Abrahão, Francis Bourdeleau, Betty Cheng, Sahar Kokaly, Richard Paige, Harald Stöerrle, and Jon Whittle. 2017. User experience for model-driven engineering: Challenges and future directions. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 229–236.

[2] Ankica Barišić, Vasco Amaral, and Miguel Goulão. 2018. Usability driven DSL development with USE-ME. *Computer Languages, Systems & Structures* 51 (2018), 118–157.

[3] M. Becker and J.L. Diaz-Herrera. 1994. Creating domain specific libraries: a methodology and design guidelines. In *Proceedings of 1994 3rd International Conference on Software Reuse*. 158–168. https://doi.org/10.1109/ICSR.1994.365788

[4] Lorenzo Bettini. 2016. *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd.

[5] Danilo Beuche. 2008. Modeling and building software product lines with pure:: variants. In *Software Product Line Conference, International*. IEEE Computer Society, 358–358.

[6] Nigel Bevan, James Carter, and Susan Harker. 2015. ISO 9241-11 revised: What have we learnt about usability since 1998?. In *International conference on human-computer interaction*. Springer, 143–151.

[7] Alan F Blackwell, Carol Britton, Anna Cox, Thomas RG Green, Corin Gurr, Gada Kadoda, Maria S Kutar, Martin Loomes, Chrystopher L Nehaniv, Marian Petre, et al. 2001. Cognitive dimensions of notations: Design tools for cognitive technology. In *International conference on cognitive technology*. Springer, 325–341.

[8] Holger Stadel Borum, Henning Niss, and Peter Sestoft. 2021. On Designing Applied DSLs for Non-programming Experts in Evolving Domains. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 227–238.

[9] Marco Brambilla, Jordi Cabot, Javier Luis Cánovas Izquierdo, and Andrea Mauri. 2017. Better call the crowd: using crowdsourcing to shape the notation of domain-specific languages. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. 129–138.

[10] Scott Brave and Cliff Nass. 2007. Emotion in human-computer interaction. In *The human-computer interaction handbook*. CRC Press, 103–118.

[11] Arvid Butting, Robert Eikermann, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. 2019. Systematic Composition of Independent Language Features. *Journal of Systems and Software* 152 (June 2019), 50–69. https://doi.org/10.1016/j.jss.2019.02.026

[12] Fabien Campagne. 2014. *The MPS language workbench: volume I*. Vol. 1. Fabien Campagne.

[13] María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. 2009. Variability within Modeling Language Definitions. In *Conference on Model Driven Engineering Languages and Systems (MODELS'09) (LNCS 5795)*. Springer, 670–684. http://www.se-rwth.de/publications/Variability-within-Modeling-Language-Definitions.pdf

[14] Tony Clark, Mark van den Brand, Benoit Combemale, and Bernhard Rumpe. 2015. Conceptual Model of the Globalization for Domain-Specific Languages. In *Globalizing Domain-Specific Languages (LNCS 9400)*. Springer, 7–20. http://www.se-rwth.de/publications/Conceptual-Model-of-the-Globalization-for-Domain-Specific-Languages.pdf

[15] Gerald Czech, Michael Moser, and Josef Pichler. 2018. Best practices for domain-specific modeling. A systematic mapping study. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 137–145.

[16] André de Lima Salgado, Fabrício Horácio Sales Pereira, and André Pimenta Freire. 2016. User-Centred Design and Evaluation of Information Architecture for Information Systems. In *Handbook of Research on Information Architecture and Management in Modern Organizations*. IGI Global, 219–236.

[17] Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais, and Jean-Marc Jézéquel. 2015. Melange: A Meta-language for Modular and Reusable Development of DSLs. In *8th International Conference on Software Language Engineering (SLE)*. Pittsburgh, United States.

[18] Enterprise Architect 2022. . Retrieved May 10, 2022 from https://sparxsystems.com/

[19] Jean-Marie Favre, Dragan Gasevic, Ralf Lämmel, and Ekaterina Pek. 2010. Empirical language analysis in software linguistics. In *International Conference on Software Language Engineering*. Springer, 316–326.

[20] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.

[21] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering (FOSE '07)* (May 2007), 37–54. http://www.se-rwth.de/publications/Model-driven-Development-of-Complex-Software-A-Research-Roadmap.pdf

[22] Ulrich Frank. 2013. Domain-specific modeling languages: requirements analysis and design guidelines. In *Domain engineering*. Springer, 133–157.

[23] Jesse James Garrett. 2010. *The elements of user experience: user-centered design for the web and beyond*. Pearson Education.

[24] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.

[25] Rohit Gupta, Sieglinde Kranz, Nikolaus Regnat, Bernhard Rumpe, and Andreas Wortmann. 2021. Towards a Systematic Engineering of Industrial Domain-Specific Languages. In *2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SE&IP)*. IEEE, 49–56. http://www.se-rwth.de/publications/Towards-a-Systematic-Engineering-of-Industrial-Domain-Specific-Languages.pdf

[26] David Harel and Bernhard Rumpe. 2004. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer* 37, 10 (October 2004), 64–72. http://www.se-rwth.de/~rumpe/publications20042008/Meaningful-Modeling-Whats-the-Semantics-of-Semantics.pdf

[27] Marc Hassenzahl. 2008. User Experience (UX): Towards an Experiential Perspective on Product Quality. In *Proceedings of the 20th Conference on l'Interaction Homme-Machine* (Metz, France) *(IHM '08)*. Association for Computing Machinery, New York, NY, USA, 11–15.

[28] Katrin Hölldobler, Oliver Kautz, and Bernhard Rumpe. 2021. *MontiCore Language Workbench and Library Handbook: Edition 2021*. Shaker Verlag. http://www.monticore.de/handbook.pdf

[29] IBM Rhapsody 2022. . Retrieved May 10, 2022 from https://www.ibm.com/products/systems-design-rhapsody/

[30] ISO 9241-161:2016(E) 2016. *Ergonomics of human-system interaction — Part 161: Guidance on visual user-interface elements*. Standard. International Organization for Standardization, Geneva, Switzerland.

[31] ISO 9241-210:2010(E) 2010. *Ergonomics of human system interaction - part 210: Human-centred design for interactive systems*. Standard. International Organization for Standardization, Geneva, Switzerland.

[32] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2016. Collaboro: a collaborative (meta) modeling tool. *PeerJ Computer Science* 2 (2016), e84.

[33] Timo Jokela, Netta Iivari, Juha Matero, and Minna Karukka. 2003. The Standard of User-Centered Design and the Standard Definition of Usability: Analyzing ISO 13407 against ISO 9241-11. In *Proceedings of the Latin American Conference on Human-Computer Interaction* (Rio de Janeiro, Brazil) *(CLIHC '03)*. Association for Computing Machinery, New York, NY, USA, 53–60.

[34] Juha Kärnä, Juha-Pekka Tolvanen, and Steven Kelly. 2009. Evaluating the use of domain-specific modeling in practice. In *Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling*.

[35] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2009. Design Guidelines for Domain Specific Languages. In *Domain-Specific Modeling Workshop (DSM'09) (Techreport B-108)*. Helsinki School of Economics, 7–13. http://www.se-rwth.de/publications/Design-Guidelines-for-Domain-Specific-Languages.pdf

[36] Thomas Kühne, Gergely Mezei, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. 2009. Systematic transformation development. *Electronic Communications of the EASST* 21 (2009).

[37] MagicDraw Enterprise 2022. . Retrieved May 10, 2022 from https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/

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

[39] Eduardo Mosqueira-Rey and David Alonso-Ríos. 2020. Usability heuristics for domain-specific languages (DSLs). In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 1340–1343.

[40] Jakob Nielsen. 1994. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 152–158.

[41] Jakob Nielsen. 2000. Designing web usability. (2000).

[42] Ildevana Poltronieri, Avelino Francisco Zorzo, Maicon Bernardino, and Marcia de Borba Campos. 2018. Usa-dsl: usability evaluation framework for domain-specific languages. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2013–2021.

[43] Ildevana Poltronieri Rodrigues, Márcia de Borba Campos, and Avelino F Zorzo. 2017. Usability evaluation of domain-specific languages: a systematic literature review. In *International Conference on Human-Computer Interaction*. Springer, 522–534.

[44] Henderik A. Proper and Marija Bjekovic. 2019. Fundamental challenges in systems modelling. *EMISA Forum* 39, 1 (2019), 13–28.

[45] Nikolaus Regnat. 2018. Why SysML does often fail - and possible solutions. In *Modellierung 2018, 21.-23. Februar 2018, Braunschweig, Germany*. 17–20.

[46] Nikolaus Regnat, Rohit Gupta, Nico Jansen, and Bernhard Rumpe. 2022. Implementation of the SpesML Workbench in MagicDraw. In *Modellierung 2022 Satellite Events*. Gesellschaft für Informatik e.V., Bonn, 61–76. https://doi.org/10.18420/modellierung2022ws-008

[47] Bernhard Rumpe. 2016. *Modeling with UML: Language, Concepts, Methods*. Springer International. http://www.se-rwth.de/mbse/

[48] Juha-Pekka Tolvanen. 2006. MetaEdit+ integrated modeling and metamodeling environment for domain-specific languages. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 690–691.

[49] Markus Voelter. 2009. Best Practices for DSLs and Model-Driven Development. *Journal of Object Technology* 8, 6 (2009), 79–102.
[50] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. 2013. Industrial adoption of model-driven engineering: Are the tools really the problem?. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 1–17.