[GJRR22] R. Gupta, N. Jansen, N. Regnat, B. Rumpe:
Implementation of the SpesML Workbench in MagicDraw.
In: Modellierung 2022 Satellite Events, pp. 61-76, Gesellschaft für Informatik e.V., Juni 2022.
se-rwth.github.io/publications/

Implementation of the SpesML Workbench in MagicDraw[1]

Nikolaus Regnat[2] Rohit Gupta[3] Nico Jansen[4] Bernhard Rumpe[5]

Abstract:

Model-Based Systems Engineering (MBSE) is a formalized methodology that focuses on creating models at the centre of system design, rather than traditional document-based approaches. While general purpose modelling languages such as the Systems Modelling Language (SysML) and their corresponding methodological approaches such as the Software Platform Embedded Systems (SPES) framework are available, the combination of a modelling language, method, and tooling is still lacking. Typically, industrial language engineers simply provide guidelines for users on how to use a particular modelling language as there is no methodical support for using the same with a well-defined method and a well-suited graphical modelling tool. This puts all the burden on the user, often resulting in a failure of the whole approach. To solve this challenge, we introduce a coherent and systematic approach for the efficient development of a SysML workbench that combines SysML and the SPES methodology using a modelling tool, MagicDraw. In this paper, we showcase the construction of a comprehensive methodical workbench by integrating key aspects of the modelling language, the SPES methodology and MagicDraw. Ultimately, the resulting SysML workbench for SPES serves as a reference point for future MBSE implementations, relieves users from the many burdens of traditional approaches, and helps mastering the complexity of creating collaborative model-based systems with efficient methods and tools.

Keywords: Model-Based Systems Engineering; Domain-Specific Languages; Industrial Language Engineering

1 Introduction

With the advancement in digitization of various systems engineering domains, there is a notable shift in the way modelling is introduced using a model-based systems engineering (MBSE) [Se03] approach into an organization. However, there still exists a conceptual gap between the fundamentals of system engineering and the combination of a relevant methodology along with modelling tools [FR07, Re18] in effectively describing and using a modelling language. The modelling of systems, therefore, requires a methodologically sound approach to software and system development. To this end, in the German Federal Ministry of Education and Research (BMBF), funded projects such as **"Software Platform**

[1] This work was supported by the German Ministry for Education and Research (BMBF) in the SpesML project (https://spesml.github.io/index.html/; grant number 01IS20092D and 01IS20092B).

[2] Siemens AG, Munich, Germany nikolaus.regnat@siemens.com
[3] Siemens AG, Munich, Germany rg.gupta@siemens.com
[4] RWTH Aachen, Aachen, Germany jansen@se-rwth.de
[5] RWTH Aachen, Aachen, Germany rumpe@se-rwth.de

Embedded Systems" (SPES) [Po12] and its follow-up project **"Software Platform Embedded Systems Extended" (SPES_XT)** [Po16] have been developed to provide foundations for a comprehensive methodological toolkit in model-based development. The complexity in developing embedded systems is addressed using the methods described in these projects. However, there still exists the challenge of combining such a methodology with a modelling tool effectively which often leads to laborious efforts for the end user in adopting such methodologies.

SPES and SPES_XT have further advanced the development of automated embedded systems with a solid methodology based on MBSE. The SPES methodology has been developed to provide guidelines for the specification, design and models of complex systems. SPES is based on a number of specific scientific modelling theories, with a special focus on consistency and semantic coherence called *FOCUS* [BS12, Br10]. SPES is comprised of four different system *viewpoints*, which are collections of model elements addressing a set of concerns of various stakeholders. Each of these viewpoints contribute to the description and understanding of the system, while still supporting the separation of different aspects into different models. SPES is able to achieve system decomposition into finer, less complex granular levels using these separation of concerns. Further, the BMBF project "Collaborative Embedded Systems" (CrESt) extends the SPES framework to networks of collaborative embedded systems (CESs), thereby addressing challenges of complex networks of CESs.

The SPES methodology provides a relevant direction to modelling experts in the model-based development of software intensive cyber physical systems [PB20]. Modelling languages such as SysML [FMS14], were built *by* modelling experts *for* modelling experts, ignoring the fundamental fact that most end users are not and probably never will be modelling experts. Any attempt at building a comprehensive SPES workbench introduces challenges in extensibility, to what extent systems can be decomposed, and how independently viewpoints are developed. These challenges can be alleviated by providing users with a complete tooling framework that bundles the language and the method. Such a tooling framework provides language engineers with scope for customizations to realize a successful SPES methodology and reduces laborious manual efforts for users in their modelling. A successful adoption of the SPES methodology covers extensibility, both in implementation and architecturally, for future SPES projects. Bringing these aspects together is a challenge, as small and medium enterprises do not have the resources to build a comprehensive solution that combines the aspects of a modelling language, methods provided by SPES, and a modelling tool into a cohesive whole (Section 2). Modelling tools must be capable of providing editing capabilities at a greater level of flexibility for language engineers as well as for end users. These include support for a complete language definition [CGR09, Cl15, HRW18], templates for assisting users in quickly designing models, choosing tool features based on their modelling needs, and other extensions that add to the default functionalities of the tool. Only with the combined use of a modelling language, a method, and a modelling tool can efficient and semantically sound modelling be achieved.

Given the basis of SPES as a methodology foundation, the project SpesML provides a direction to these challenges. In this paper we put the choice on a firmer basis:

- We explore the creation of a SpesML workbench using a graphical modelling tool, MagicDraw, and integrating the SPES methodology, described in Section 3, with it.
- We show the editing capabilities that MagicDraw offers to define custom plugins consisting of the SPES language profile, a predefined template for creation of models, application of features based on the modelling experience of a user and additional extensions written in Java, that help realize the complete language definition.
- In Section 4, we take a detailed look into how the SPES methodology is configured in MagicDraw, by showcasing the internals of the composition of the language. These include the creation of different viewpoints as building blocks of a language, individual customization on language elements to enhance the aesthetics [Mo09, Ni00], validation rules and the interoperability between the different viewpoints.
- We detail the separation of concerns used in MagicDraw, and show how individual SPES language components are composed at different levels of granularity and are structured to the users modelling needs.
- Ultimately, in Section 5 we discuss how adopting SPES as a methodology can be beneficial to users in mastering the complexities of modelling in a practical environment such as MagicDraw and in Section 6 we conclude the paper.

2 Background

Models are an abstraction of the original system, which aims to reduce the gap between the domain problem and its implementation [Ad20]. Model-based approaches intend to revolutionize the tools for software engineering as well as the process of their definitions from classical documents to models [Ru16]. Model-based development is not only about drawing or setting up models, but also about the inclusion of a comprehensive modelling methodology. Typically, just introducing models is not sufficient. Modelling experts must also think about how systems can be broken down into smaller parts, that can be better designed and later synthesized back into a whole [Gu21]. Engineering of such systems require principles, concepts and methods, which form the basis of MBSE. MBSE is a formal methodology used to support requirements, design, analysis, and a number of other modelling concepts to capture system properties precisely [Bö21]. The three main aspects of MBSE that must be considered independently and also in good coordination with others are: (i) modelling language, (ii) methodology; and (iii) modelling tools. Only when these aspects are considered as a coherent whole, can the realization of an MBSE approach work cohesively. MBSE is applied on a number of complex heterogeneous systems such as Cyber-Physical Systems (CPSs) and Collaborative Embedded Systems (CESs) [Ru19].

CPSs are software controlled, interconnected physical machines that typically sense their environment and are able to interact with their contexts in some way. CESs, on the other hand, are a result of the transition from traditional embedded systems to a network of CESs working with other systems to achieving a common goal.

The drastic increase in the scope of variant diversity across domains [TK05] and the complexity of systems and system networks, presents engineering of embedded systems with new and challenging problems. In the funded projects, SPES and the follow-up project SPES_XT, the basis for a comprehensive methodical construction kit for the consistent model-based development of embedded systems has been developed. The SPES framework consists of methods and tools based on specific modelling theories, that help master the complexity of embedded systems in an efficient, controllable and verifiable manner. By using separation of concerns, SPES ensures the central principles of consistency, assessability and tool support are all taken into account while solving the different engineering challenges. The project CrESt, on the other hand, aims to create a comprehensive framework for the development of collaborative embedded systems that addresses the new challenges in the development of collaborative embedded systems in dynamic system networks, by leveraging the SPES methodology.

Despite the progress made in these projects to create a solid methodology for MBSE development, the realization of the methodology is left to the end users. This leads to challenges for modelling experts in adopting the methodology using relevant modelling tools. Few possible explanations for this are that organizations often struggle in incorporating modelling tools whose integration with a modelling language and a methodology is laborious, the modelling tool may be just too expensive or it does not support handling of different variants of their heterogeneous complex systems. In the new funded project, SpesML, a SysML workbench for the SPES method has been developed aiming to provide a custom tailored SysML profile that integrates the SPES method using the tool MagicDraw.

3 Methodology

3.1 SPES Methodology

The SPES methodology is based on a solid scientific foundation of consistency and semantic coherence called FOCUS [BS12]. It is based on three important principles [Bö21]: (1) The design process must consider interfaces consistently; (2) Decomposition of the interface behaviour and description of systems via subsystems and components at different levels of granularity; and (3) Definition of models for a variety of cross-sectional topics and analysis options. SPES defines a system model as a conceptual model for describing systems and their properties. System models define the components of systems, the structure, essential properties, and other aspects that have to be considered during development. SPES defines an MBSE artefact model based on the concepts in the standard ISO-42010 that

assumes a System-under-Development (SuD) has an architecture and provides functions for determining the functional properties of the system. During the development process, different *viewpoints* separate the concerns of different stakeholders and allows for managing different artefacts while completely describing a system. SPES predefines four basic viewpoints: requirements viewpoint, functional viewpoint, logical viewpoint and technical viewpoint are described in Figure 1 along different layers of granularity with the topmost layer denoting the models of the SuD. The requirements viewpoint constitutes the system requirement engineering activities. The functional viewpoint describes a set of system functionalities. The logical viewpoint describes the decomposition of the system functions in terms of logical components. Finally, the technical viewpoint combines software and hardware components related to the SuD. The views for the different viewpoints are parts of the overall model that describes a system.

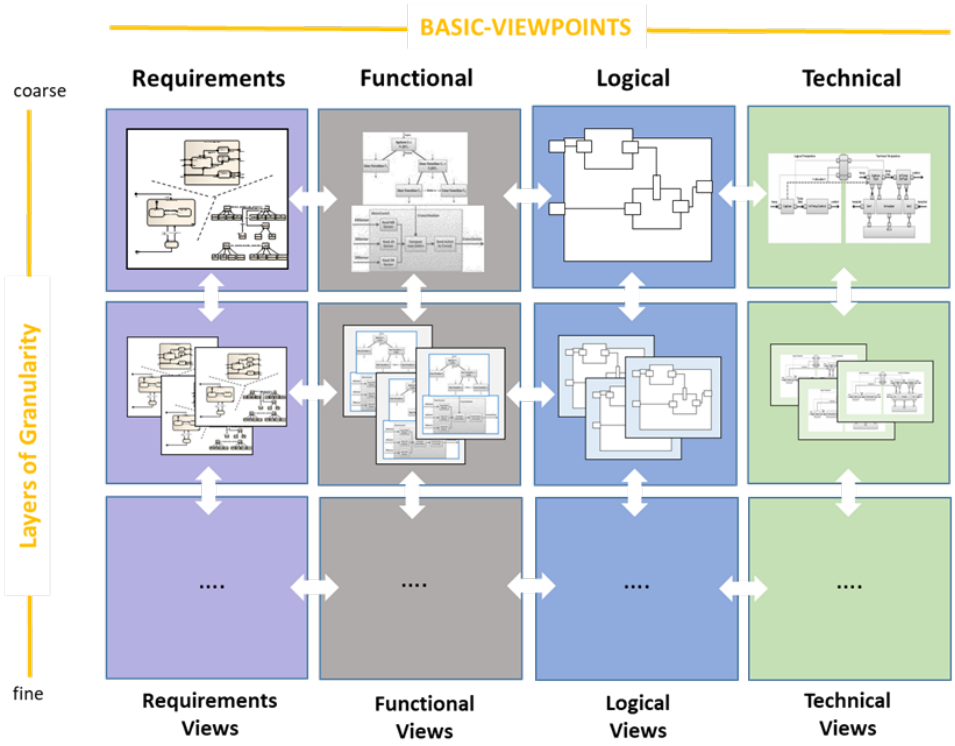

Fig. 1: An overview of the four basic SPES viewpoints: Requirements, Functional, Logical and Technical viewpoints described in [Bö21] across different layers of granularity.

3.2 MagicDraw: Modelling Tool

A good user experience is key to successfully introducing MBSE into any organization [Re18]. At the beginning of the SpesML project, all involved partners decided to use MagicDraw as the tool of choice for the prototypical implementation of SPES. MagicDraw is a modelling tool based on the Unified Modelling Language (UML). It comes with comprehensive extensions (such as a SysML plugin) and provides a wide range of customization possibilities that can be used to enhance the user experience, thus bringing the modelling language, method, and tool closer together. Gupta et al. [Gu21] describe a systematic engineering process of developing industrial domain-specific languages (DSLs) [Fo10] using modular reusable DSL Building Blocks in MagicDraw. Note that we consider SpesML a DSL consisting of the different viewpoints, or DSL Building Blocks, even if we only build upon the SysML and UML. Figure 2 describes the parts of this development approach conceptually by separating the concerns of industrial engineering and deployment of DSLs on the following levels: (1) Concept level: in this level, language engineers define the following parts: (i) the re-usable language components that, wholly or in part, defines the language [Ru16]; (ii) the method, describing a suitable methodology for the language to help users achieve their intended modelling goals; and (iii) the user experience design, where standards and usability heuristics related to user experience are described; (2) Tool-specific implementation level: in this level, language engineers realize the viewpoint aspects described in the concept level using MagicDraw; (3) Usage level: the level where end users model using MagicDraw. Ultimately, DSL Building Blocks, here SPES viewpoints, are composed together to create the SpesML DSL, consisting of heterogeneous domain constructs that the SpesML workbench leverages as different viewpoints.

MagicDraw provides the capability to define custom plugins that can be installed for the end user. Such a plugin typically consists of a MagicDraw project containing the profile, a template for new models, MagicDraw perspective definitions, and additional Java extensions to support dynamic customizations including defining context conditions. Standard modelling languages such as SysML are similarly bundled together as plugins and any MagicDraw user can install the plugin with few simple clicks. These functionalities of MagicDraw as a tooling environment makes it a good fit for realizing the SPES methodology.

4 SpesML Workbench

4.1 SpesML Profile

When implementing the SpesML Workbench in MagicDraw, we started with the definition of the SpesML *Language Components*. We created a dedicated MagicDraw profile to define all needed parts of the SpesML modelling language. A profile in MagicDraw does not only consist of stereotypes and tag definitions but also allows defining *customization* elements. These MagicDraw-specific elements allow to define additional rules or context conditions

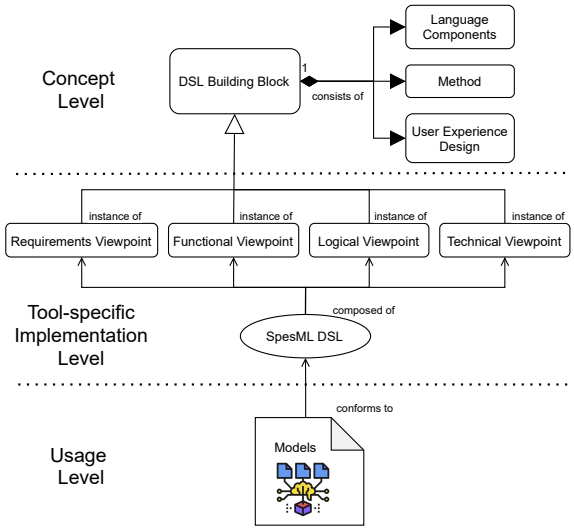

Fig. 2: A conceptual model for the development and usage of a graphical DSL describing the different levels in the engineering process that includes defining the DSL Building Blocks, or SPES viewpoints, each consisting of the method, language components and a user experience design part. The resulting SpesML DSL is composed of the different viewpoints.

for each stereotype. This also allows us to not only embed parts of the SpesML *Method* but also enables us to directly influence the *User Experience Design*.

The first step was to create dedicated stereotypes that represent the chosen SpesML model structure. Instead of using standard UML Package elements, we created individual stereotypes for each structural level. When defining the customizations for these stereotypes we allowed only certain other elements to be created. For example, under the *Logical Viewpoint* (package) we only allow specific elements or diagrams that are part of the *Logical Viewpoint* to be created. This step already provides instructive guidance to the user, as it combines the modelling language, method and tool in an easy to use way. This is less error prone compared to the typical approach of simply giving users access to every UML/SysML element without a defined scope and later providing guidelines on what to do [Re18]. It also ensures that all SpesML models are much more uniform, improving not only readability but overall model quality. Subsequently, it allows for easier integration of automation techniques, be it a document generation or any other form of accessing the model data using the API.

Fig. 3 shows an exemplary stereotype and its corresponding configuration from the SpesML profile. The stereotype «SpesML Logical Viewpoint» is defined with the metatype *Package* and comes with a distinct icon, setting it apart from normal packages. The corresponding *customization* element defines additional aspects for this stereotype. The *abbreviation* defines the default name of the element when the user creates a new element of that type. The

category influences the user interface: when a user right-clicks on the viewpoint this element will show up in a category called *SpesML Packages*. The *disallowedRelationships* is setup so that all possible relationships on this element are forbidden. The *hiddenOwnedDiagrams* and *hiddenOwnedTypes* configurations hide all normal UML and SysML diagrams and elements from the user; these will not show up in the context menu of this element. The *suggestedOwnedTypes* setting references only those SpesML stereotypes that we want the user to be able to create under this element. Due to this configuration, users do not have to create generic SysML elements and manually apply stereotypes, but can directly create the SpesML elements.

The «SpesML Logical Component» stereotype is defined with the metatype *Class* as MagicDraw does not allow the usage of SysML stereotypes as metatypes. However, we can inherit from the *SysML Block* stereotype. This way our new stereotypes works like a SysML block from the users' perspective. Most model elements, including the different viewpoints, that are used in the SpesML profile come with their individual stereotypes and customizations, similar to these examples.

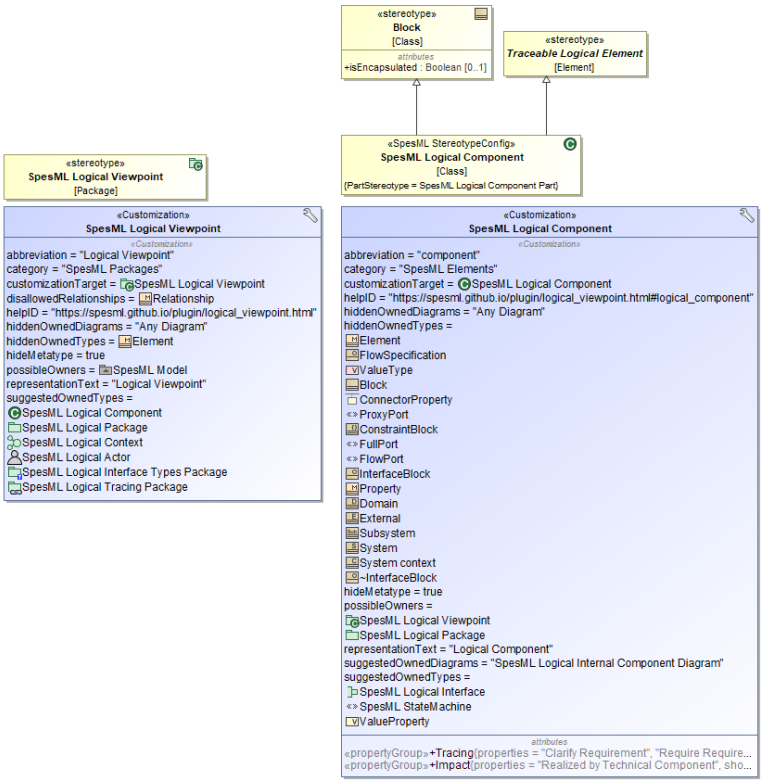

Fig. 3: An example of a SpesML profile stereotype and customization that allows the configuration of custom language elements in MagicDraw.

Fig. 4 shows the result of the customization configuration. During modelling, when the user clicks on *Create Element* on the «SpesML Logical Viewpoint» element, the upcoming dialog box will no longer show all UML/SysML elements, instead only the defined SpesML elements that are appropriately scoped are shown. Elements are also grouped into dedicated categories (such as *SpesML Packages*, *SpesML Elements*, and so on) making it easier for the user to work with the SpesML profile.

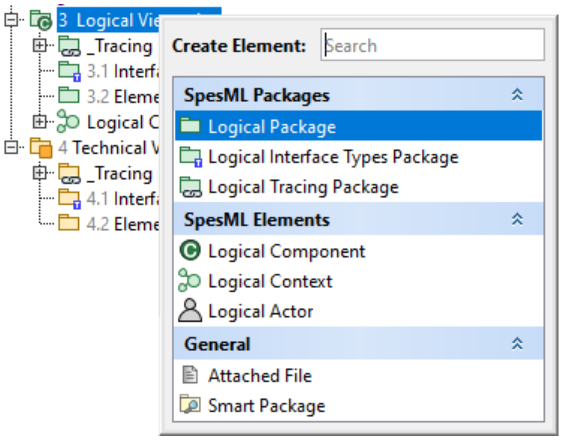

Fig. 4: An example of a dialog box to create a Logical Viewpoint element that shows the categorization and grouping of different elements.

4.2 Custom SpesML Diagrams

In the next step we created dedicated SpesML diagrams. MagicDraw allows the creation of custom diagram definitions based on UML diagrams. Note that we could not create customized SysML based diagrams due to limitations in MagicDraw. These custom SpesML diagrams allowed us to provide customized toolbars that only show those elements that are needed for SpesML. We can also directly reference our dedicated SpesML elements in these toolbars, enabling users again to create these elements without having to first create SysML elements and manually applying stereotypes. Apart from custom UML based diagrams MagicDraw also allows language engineers to define custom *Matrices*, *Tables* and *Relation Maps*.

These special diagrams are used for dedicated purposes and are pre-configured to show only those elements, their attributes or relationships that are required for a certain purpose. For example, in the SpesML Requirements Viewpoint, we have defined a *SpesML Requirements Table* that shows all requirements in a hierarchical table, also allowing users to directly create and modify requirements and their attributes directly on the table.

Fig. 5 shows an example diagram from the SpesML Logical Viewpoint. It is based on an UML Composite Structure Diagram and comes with a reduced set of diagram toolbar

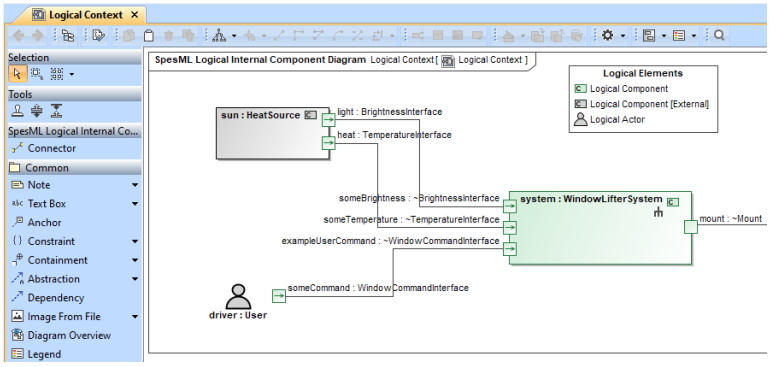

Fig. 5: An example model for a window lifter system using the Logical Viewpoint Diagram.

elements, allowing users only to create *connectors* between ports. This forces users to create all the *Logical Components* in the appropriate packages first, before using them on the diagrams. The diagram also comes with a pre-configured legend that describes all the relevant SpesML elements.

4.3 Embedding Textual Languages

MagicDraw predominantly fosters graphical modelling. However, some specific model elements have proven to be better written in textual form. Examples are guards and actions in state machine diagrams. While the state machine itself remains in its graphical representation, guards and actions on transitions are expressions in text form. MagicDraw comes with a predefined set of (semi-)textual languages such as structured expressions (coded in XML), groovy, or OCL. However, these languages are hardly extensible and did not integrate well into our SpesML workbench, where a modeller should be able to reference arbitrary model elements in the containment tree. Simple expressions for validating or assigning values are not supported.

Thus, we developed a method to integrate custom textual languages into MagicDraw and evaluate their models in our custom validation suite (cf. Section 4.6). While a modeller enters the expression as a String in the default text box for defining guards or actions, the expression is forwarded to an integrated parser. We developed this parser based on an existing library of textual language components [Bu20] using the MontiCore language workbench [HKR21]. If the input text cannot be parsed, our plugin reports the parsing errors back to the user. Otherwise, the plugin continues to construct a so-called symbol table of the containment tree. It is used for resolving references in the expressions to access various attributes of the existing model elements. Thus, it can also be checked whether the referenced elements are accessible from the current scope.

4.4 Model Template

Once we have created our dedicated SpesML elements and diagrams we can setup the SpesML model template. This template is used whenever a user creates a new SpesML project. It consists not only of a dedicated icon and description but a predefined package structure based on our dedicated SpesML stereotypes. While simple to setup, a model template not only makes things much easier for the user but also directly guides the user to follow the SpesML approach in a specific order. Fig. 6 shows the SpesML template as it is presented to the user. It does not only contain a predefined numbered package structure but can also be configured to contain certain other model elements (for example a *Logical Context* element), custom diagrams, matrices, tables and relation maps.

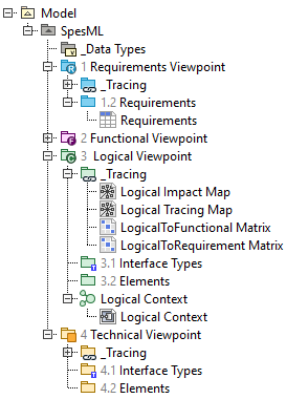

Fig. 6: A model template for the SpesML project that allows quicker and consistent creation of model elements in MagicDraw.

4.5 Perspectives and Help

In the next step, we defined the MagicDraw *perspectives*. These perspectives further customize the user interface of MagicDraw by removing toolbar menu entries, context menu actions, and a wide variety of MagicDraw functionalities. MagicDraw and similar modelling tools do typically offer many features, that are often overwhelming to new and inexperienced users. Reducing the user interface to a minimum will help these users to focus on the relevant functionalities of the tool, while still allowing more experienced users to make use of the full potential by choosing a perspective with more functionalities. In the context of SpesML, we have defined the *SpesML (Novice)* and *SpesML (Expert)* perspectives to support this approach.

Another aspect users often struggle with is finding the appropriate help or documentation when working with modelling tools. MagicDraw does provide the possibility to add dedicated hyperlinks to all stereotypes and custom diagram definitions. To this end, we

created dedicated SpesML web pages [6] and linked the complete SpesML DSL with the implementation of all the elements and diagrams to these pages. This allows a SpesML user to click on the *Help* button of any element or diagram and the SpesML specific help is shown to the user. In addition, we have added an API plugin that adds a new entry in the MagicDraw Help menu, also allowing direct access of the dedicated SpesML web pages.

4.6 API Extensions

As a last step we have created and integrated additional API-based extensions to overcome some MagicDraw limitations and to enhance the SpesML Workbench implementation. MagicDraw provides extensive APIs and allows the enhancement of the tool by creating Java extensions for a variety of purposes. We started by implementing the *SpesML Stereotype Plugin* that allows us to automatically apply our SpesML stereotypes to model elements under certain conditions. For example, when a user drags and drops a *SpesML Logical Component* to a *SpesML Logical Internal Component Diagram* we want to apply a the «SpesML Logical Component Part» stereotype to the instance of the part element that is created by MagicDraw under the diagram. This has the benefit that we have full control on this element: we can show the part element with a defined icon and colour and can also define what properties is visible to the user. The result is a more consistent user experience as practically all model elements the users interacts with, have a common look and feel.

We have also added a plugin to enhance the visualization possibilities of MagicDraw. In SpesML, we defined that we want to model external elements (for example an external Logical Component) not by using dedicated stereotypes but instead by allowing the user to define on a part level if an element is external or not. This approach has the benefit that, depending on the development subject we can change whether a certain part is considered external or not. MagicDraw provides the capability to change an elements icon based on an enumeration and this is perfect for the *Containment Tree*. However, a change for an icon is not very distinct on diagrams. With our *SpesML Visualization Plugin* we are able to additionally change the colour of an element based on the enumeration.

MagicDraw also allows to create custom validation rules to check the accuracy, completeness, correctness and well-formedness of a model and in the process, marks invalid elements in the model. We have implemented the *SpesML Validation Plugin* that bundles all rules that have been defined in the SpesML methodology. For example, we want to ensure that certain SpesML elements have at least one port defined or that certain SpesML elements require the user to provide proper naming conventions. Simple rules can be expressed using Object Constraint Language (OCL) but more complex rules can be implemented using dedicated Java classes. In order for the user to easily access and execute these rules we have also created a dedicated *SpesML Validation Suite*.

[6] `https://spesml.github.io/plugin/overview.html/`

5 Discussion

In this paper, we present the SPES methodology and its implementation in MagicDraw in brief. The SPES framework provides a solid MBSE foundation in model-based development but does not necessarily specify the order in which different models should be created for the different viewpoints. This means the end users often struggle in realizing such a methodology effectively using modelling tools and eventually spend laborious manual efforts to achieve their modelling. Therefore, demonstrating the combination of a modelling language with a method using a modelling tool is imperative.

In the funded projects, SPES and SPES_XT, the basis for a comprehensive methodical construction kit for the consistent model-based development of embedded systems has been developed. The projects describe the concepts needed for a methodology-based on MBSE for CPSs, embedded systems and more recently CESs. Ultimately, it is the end user who should benefit from such solid methodologies that provide the relevant direction in modelling. SPES has been realized by different users in the past but so far a reference implementation with the modelling language and the method itself in a modelling tool is missing. The creation of a UML and SysML based workbench for SPES in MagicDraw, in the project SpesML, alleviates tooling shortcomings by providing focus on the end user. MagicDraw is a modelling tool having a range of custom functionalities, and is therefore a good fit for a reference implementation for covering broadly the aspects of SPES.

Industry grade modelling tools such as IBM Rational Rhapsody, Enterprise Architect, Arcadia Capella and MagicDraw offer integrated workbench capabilities and were investigated to create prototypes. In our experience, as it supports the Open Java API, MagicDraw enables many flexible extensions, as described in Section 4, directly into the tool. The MagicGrid reference implementation in MagicDraw also provides a suitable process for the methodology but lacks the composition of smaller SPES artefacts into a single whole. In contrast, our implementation allows decomposition of the SPES viewpoints that can be used independently. We used MagicDraw to create a profile that allows creation of stereotypes and tag definitions along with its customization properties that directly influence the user experience. Dedicated elements and diagrams were created to help realize the different viewpoints and their respective sub-elements, reducing the effort needed to create individual SPES elements. The availability of MagicDraw templates provide the end-users with a predefined model structure of SPES on the tool, eliminating the need to realize every single SPES aspect from scratch. MagicDraw also offers perspectives that influence how many functionalities of the tool a user sees, benefiting both novice practitioners and advanced modelling experts. Documentation can be added to individual elements in the form of hyperlinks to easily navigate to dedicated SpesML webpages. Limitations on the functionalities of MagicDraw are overcome using Java based API plugins bundled in the SpesML workbench, allowing additional functionalities to be introduced, such as dedicated context conditions or dynamic modification of the appearance of model elements. We believe using these customization functionalities of MagicDraw to implement the SPES methodology is more beneficial than creating a completely new method-specific tool because

it helps reuse existing tooling functionalities and provides ample scope for generalizing specific aspects of the methodology independent of other modelling tools. Furthermore, the SPES methodology is intentionally designed to be language-agnostic to be applicable to different DSLs. Developing the SpesML workbench, we explored its application on SysML.

As of today, we are unaware of the existence of any other reference implementation of SPES using a modelling tool that is capable of providing such a wide range of customizations for a modelling language and a method. The realization of the SPES methodology in MagicDraw therefore presents a reference implementation for novice and advanced users as well as for small and medium organizations wanting to integrate the SPES methodology into their projects. Bringing heterogeneous domains together and building a library of reusable units is important in achieving modularity in systems engineering. Further, a solid methodology needs to be supported by appropriate tooling mechanisms to lessen the burden on end users. Finally, good user experience design aspects must be considered in order to achieve effective modelling for the end user. We believe the combination of a modelling language, method and tool needs further discussion between researchers and practitioners to foster efficient MBSE development in the future. To this end, we consider the SpesML workbench, realized in MagicDraw, a good reference point for future SPES implementations.

6 Conclusions

As various system domains become complex, heterogeneous and digitized, approaches to engineer such systems need solid methodological foundations. Software Platform Embedded Systems (SPES) and its follow up extended project (SPES_XT) have been developed to provide for a comprehensive methodological toolkit in model-based development for engineering cyber-physical systems and collaborative embedded systems. While such projects based on MBSE have further advanced the development of automated embedded systems, there still exists the challenge of combining such a methodology with a modelling tool to reduce laborious manual modelling efforts. To address this challenge, we developed a reference implementation of SPES in MagicDraw, a modelling tool, as the SpesML workbench. We developed the SpesML workbench by utilising the wide range of functionalities in MagicDraw: creation of custom language profiles consisting of stereotypes and diagrams, predefined model templates for consistent and ordered model creation, availability of tool functionalities based on the level of modelling expertise of a user, specific API plugins and documentation, all of which help realize the SPES methodology. The SpesML workbench, is therefore, designed to relieve users from the burden of laborious complex system engineering activities and improves the overall user experience in their modelling. Further, the SpesML workbench leverages techniques of separation of concerns by modularising language components fostering re-usability. Naturally, the SpesML workbench does not solve all problems related to systems engineering. As part of the SpesML workbench, we have identified the integration of the modelling tool with a solid MBSE methodology a key aspect in effectively designing complex, heterogeneous systems. We believe the

implementation of SPES, as the SpesML workbench, in MagicDraw is beneficial for both novice and advanced modelling users and thus provides a good reference point for future SPES implementations.

Bibliography

[Ad20] Adam, Kai; Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In: 40 Years EMISA: Digital Ecosystems of the Future: Methodology, Techniques and Applications (EMISA'19). volume P-304 of LNI. Gesellschaft für Informatik e.V., pp. 59–66, May 2020.

[Bö21] Böhm, Wolfgang; Broy, Manfred; Klein, Cornel; Pohl, Klaus; Rumpe, Bernhard; Schröck, Sebastian, eds. Model-Based Engineering of Collaborative Embedded Systems. Springer, January 2021.

[Br10] Broy, Manfred: A logical basis for component-oriented software and systems engineering. The Computer Journal, 53(10):1758–1782, 2010.

[BS12] Broy, Manfred; Stølen, Ketil: Specification and development of interactive systems: focus on streams, interfaces, and refinement. Springer Science & Business Media, 2012.

[Bu20] Butting, Arvid; Eikermann, Robert; Hölldobler, Katrin; Jansen, Nico; Rumpe, Bernhard; Wortmann, Andreas: A Library of Literals, Expressions, Types, and Statements for Compositional Language Design. Special Issue dedicated to Martin Gogolla on his 65th Birthday, Journal of Object Technology, 19(3):3:1–16, October 2020. Special Issue dedicated to Martin Gogolla on his 65th Birthday.

[CGR09] Cengarle, María Victoria; Grönniger, Hans; Rumpe, Bernhard: Variability within Modeling Language Definitions. In: Conference on Model Driven Engineering Languages and Systems (MODELS'09). LNCS 5795. Springer, pp. 670–684, 2009.

[Cl15] Clark, Tony; Brand, Mark van den; Combemale, Benoit; Rumpe, Bernhard: Conceptual Model of the Globalization for Domain-Specific Languages. In: Globalizing Domain-Specific Languages. LNCS 9400. Springer, pp. 7–20, 2015.

[FMS14] Friedenthal, Sanford; Moore, Alan; Steiner, Rick: A practical guide to SysML: the systems modeling language. Morgan Kaufmann, 2014.

[Fo10] Fowler, Martin: Domain-specific languages. Pearson Education, 2010.

[FR07] France, Robert; Rumpe, Bernhard: Model-driven Development of Complex Software: A Research Roadmap. Future of Software Engineering (FOSE '07), pp. 37–54, May 2007.

[Gu21] Gupta, Rohit; Kranz, Sieglinde; Regnat, Nikolaus; Rumpe, Bernhard; Wortmann, Andreas: 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, pp. 49–56, May 2021.

[HKR21] Hölldobler, Katrin; Kautz, Oliver; Rumpe, Bernhard: MontiCore Language Workbench and Library Handbook: Edition 2021. Aachener Informatik-Berichte, Software Engineering, Band 48. Shaker Verlag, May 2021.

[HRW18] Hölldobler, Katrin; Rumpe, Bernhard; Wortmann, Andreas: Software Language Engineering in the Large: Towards Composing and Deriving Languages. Computer Languages, Systems & Structures, 54:386–405, 2018.

[Mo09] Moody, Daniel: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Trans. Softw. Eng., 35(6):756–779, November 2009.

[Ni00] Nielsen, Jakob: Designing web usability. 2000.

[PB20] Proper, Henderik A.; Bjeković, Marija: Fundamental challenges in systems modelling. In (Mayr, Heinrich C.; Rinderle-Ma, Stefanie; Strecker, Stefan, eds): 40 Years EMISA 2019. Gesellschaft für Informatik e.V., Bonn, pp. 13–28, 2020.

[Po12] Pohl, Klaus; Hönninger, Harald; Achatz, Reinhold; Broy, Manfred: Model-based engineering of embedded systems: The SPES 2020 methodology. Springer, 2012.

[Po16] Pohl, Klaus; Broy, Manfred; Daembkes, Heinrich; Hönninger, Harald: Advanced model-based engineering of embedded systems. In: Advanced Model-Based Engineering of Embedded Systems, pp. 3–9. Springer, 2016.

[Re18] Regnat, Nikolaus: Why SysML does often fail - and possible solutions. In (Schaefer, Ina; Karagiannis, Dimitris; Vogelsang, Andreas; Méndez, Daniel; Seidl, Christoph, eds): Modellierung 2018, 21.-23. Februar 2018, Braunschweig, Germany. volume P-280 of LNI. Gesellschaft für Informatik e.V., pp. 17–20, 2018.

[Ru16] Rumpe, Bernhard: Modeling with UML: Language, Concepts, Methods. Springer International, July 2016.

[Ru19] Rumpe, Bernhard; Schaefer, Ina; Schlingloff, Bernd-Holger; Vogelsang, Andreas: Special Issue on Engineering Collaborative Embedded Systems (Editorial). SICS Software-Intensive Cyber-Physical Systems, 34(4):173–175, December 2019.

[Se03] Selic, Bran: The pragmatics of model-driven development. IEEE software, 20(5):19–25, 2003.

[TK05] Tolvanen, Juha-Pekka; Kelly, Steven: Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences. In: Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings. pp. 198–209, 2005.