DISS. ETH NO. 24166

A LIBRARY-BASED CONCEPT DESIGN APPROACH FOR MULTI-DISCIPLINARY SYSTEMS IN SYSML

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

BENJAMIN KRUSE

Dipl.-Ing. Univ., Technische Universität München

born on 26.06.1985
citizen of Germany

accepted on the recommendation of

Prof. Dr. Kristina Shea
Prof. Dr. Martin Eigner

2017

Acknowledgements

First, I would like to thank Prof. Shea for her great job as my supervisor. Without her offering the opportunity and supporting me during my doctorate it would not have come true. Her guidance throughout countless discussions motivated and helped me in all the time of research to not only write this thesis but also to grow personally as well as a researcher. Further, I would like to thank Prof. Eigner for being an excellent co-examiner and Prof. Hora for acting as chair person at my defense.

My sincere thanks also go to Dr. Wölkl for getting me started with my research and Dr. Münzer, for his valuable support. Additional thanks goes of course to all other precious colleagues from both VPD in Munich and most importantly EDAC in Zurich. You made my time enjoyable by providing the right mixture of help and advice as well as inspiration, distraction and entertainment. I have truly enjoyed working with you as part of a great team. Last but not least, I would like to thank my family and friends for supporting me spiritually throughout writing this thesis and my life in general.

Jersey City, July 2017 Benjamin Kruse

Table of Contents

Abstract

Conceptual design of modern multi-disciplinary systems that combine mechatronics with services for the creation of product service systems (PSS) is a crucial phase of the product development process. Yet, existing design support is not sufficient despite fundamental developments such as model-based systems engineering (MBSE) and the modeling language SysML. SysML is a standardized, multi-purpose graphical modeling language for specifying, designing and analyzing complex multi-disciplinary systems. However, it is still not widely used in mechanical and mechatronic design.

The problem addressed in this work is how to better support multi-disciplinary concept design. The presented approach integrates formal generic design libraries that can be reused, modeling guidance and integrated simulation for concept evaluation. More specifically, it implements the Functional Basis (FB), elements from a commercial simulation tool and a service catalogue in SysML libraries to offer a foundation of proven design knowledge together with multi-solution patterns.

The first library is for functional modeling with the FB. It defines operator-flow formulations of functions for formal and solution-neutral functional decomposition. The second library provides elements from an adjunct multi-physics simulation tool to support behavior modeling as well as the planning of concept simulation in SysML. The third contributed library offers formalized service catalogue elements to support modeling of the service domain. These libraries, together with an additional structure library, provide generic elements for reuse to support concept design, while allowing inter-model traceability to link aspects from different disciplines and levels of abstraction. The additional multi-solution patterns formally capture multiple alternative concept solutions to solve recurring design problems. Since they are based on solution-neutral functions,

they can offer multiple potential solutions in one pattern. The solutions are partial models that cover various aspects, for example functions, behavior or structure. Their contribution is the enabled reuse of common coherent subsystems beyond single library elements.

The research is conducted using the Design Research Methodology (DRM) including initial descriptive studies in the form of a case study and a conducted user study to evaluate the created support. The case study demonstrates the concept design approach through the development of a 3D printer model that uses the libraries and solutions from patterns. The 3D printer model not only contains mechatronic aspects, but also complementary services and representations of two alternative kinematic designs that are simulated. Evaluation of the approach in the user study focuses on the function library and shows that the approach results in greater use of the FB and improved model quality.

The presented work contributes a new approach to formal modeling with reuse of SysML models for supported multi-disciplinary concept design. It also serves as a basis for future additional computational support, e.g. automated design synthesis using the generic and formal design libraries.

Zusammenfassung

Die Konzeptentwicklung moderner, multidisziplinärer Systeme, welche Mechatronik mit Dienstleistungen zu hybriden Leistungsbündeln verbinden, ist eine entscheidende Phase des Entwicklungsprozesses. Allerdings ist, trotz umfangreicher Entwicklungen wie der modellbasierten Systemtechnik (MBSE) und der Modellierungssprache SysML, die dafür bestehende Unterstützung noch ungenügend. SysML ist eine standardisierte, multifunktionale und grafische Modellierungssprache für die Spezifizierung, Konstruktion und Analyse komplexer, multidisziplinärer Systeme. Jedoch wird sie noch wenig für die Entwicklung mechanischer und mechatronischer Systeme verwendet.

Das hier angegangene Problem besteht darin, die multidisziplinäre Konzeptentwicklung weiter zu entwickeln um die Unterstützung zu verbessern. Die vorgestellte Vorgehen verbindet verschiedene formal-generische Bibliotheken für die Wiederverwendung und Modellsimulation zur Konzeptevaluierung. Konkret werden die Functional Basis (FB), Elemente eines kommerziellen Simulationstools und ein Dienstleistungskatalog als SysML Bibliotheken realisiert, um grundlegendes und bewährtes Design-Wissen zusammen mit Multilösungs-Entwurfsmustern zur Verfügung zu stellen.

Die erste Bibliothek dient der Funktionsmodellierung mit der FB. Sie definiert Operator-Flow-Formulierungen von Funktionen für die formale und lösungsneutrale funktionale Dekomposition. Die zweite Bibliothek stellt Elemente aus einem verknüpften multiphysikalischen Simulationstool zur Verfügung, um das Systemverhalten zu modellieren und die Konzeptsimulation in SysML zu planen. Die dritte Bibliothek enthält formalisierte Dienstleistungen zur Unterstützung der Modellierung der Dienstleistungsdomäne. Diese Bibliotheken stellen zusammen mit einer zusätzlichen Struktur-Bibliothek generische Elemente für die Wiederverwendung zur Verfügung. Die Konzeptentwicklung beinhaltet dabei auch die

mögliche Vernetzung der verschiedenen Disziplinen und Abstraktionsebenen innerhalb des Systemmodells. Die Multilösungs-Entwurfsmuster erfassen mehrere alternative Konzeptlösungen, um wiederkehrende Probleme zu lösen. Da sie auf lösungsneutralen Funktionen basieren, können sie mehrere mögliche Lösungen innerhalb eines Entwurfsmusters anbieten. Die angebotenen Lösungen sind Teilmodelle, die verschiedene Aspekte abdecken, wie zum Beispiel Funktionen, Verhalten und Struktur. Ihr Beitrag ist die Wiederverwendung kohärenter Subsystemmodelle zusätzlich zur der einzelner Bibliothekselemente.

Die vorgestellte Forschung orientiert sich an der Design Research Methodology (DRM) und beinhaltet erste deskriptive Studien in Form einer Fallstudie und eines durchgeführten Experimentes an Anwenders zur Bewertung der erstellten Modellierungsunterstützung. Die Fallstudie veranschaulicht das entwickelte Vorgehen zum Konzeptentwurf durch die Entwicklung eines 3D-Druckermodells, dass die Bibliotheken sowie Lösungen aus Entwurfsmustern verwendet. Das 3D-Druckermodell berücksichtigt nicht nur mechatronische Aspekte, sondern auch ergänzende Dienstleistungen. Es enthält außerdem entsprechende Darstellungen von zwei alternativen kinematischen Lösungen zur Simulation. Die Evaluierung des Modellierungsvorgehens durch die Anwenderstudie konzentriert sich auf die Funktionsbibliothek, die eine verstärkte Nutzung der FB sowie eine verbesserter Modellqualität zur Folge hat.

Die vorliegende Arbeit leistet einen Beitrag zum formellen Modellieren mit Wiederverwendung in und von SysML-Modellen für die unterstützte multidisziplinäre Konzeptentwicklung. Sie dient ebenfalls als Grundlage für zukünftige automatisierte Design-Synthese, unter Verwendung der formalen Design-Bibliotheken.

Contained Publications

The following publications are part of the work presented in this thesis:

[1] Kruse, B., and Shea, K. Design Library Solution Patterns in SysML for Concept Design and Simulation. 26th CIRP Design Conference. Stockholm, Sweden; 2016. p.620-625.

[2] Kruse, B., Gilz, T., Shea, K., and Eigner, M. Systematic Comparison of Functional Models in SysML for Design Library Evaluation. 24th CIRP Design Conference. Milano, Italy; 2014.

[3] Kruse, B., Münzer, C., Wölkl, S., Canedo, A., and Shea, K. A Model-Based Functional Modeling and Library Approach for Mechatronic Systems in SysML. ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. Chicago, IL, USA; 2012.

[4] Kruse, B., Münzer, C., Wölkl, S., Canedo, A., and Shea, K. Workflow and Modeling Conventions for Function and Product Structure Modeling of Mechatronic Systems in SysML Using Libraries. Mechatronics 2012. Linz, Austria; 2012.

List of Tables

List of Figures

Glossary

ABS	Acrylonitrile Butadiene Styrene
AC	Alternating Current
ACT	Activity Diagram
BDD	Block Definition Diagram
DC	Direct Current
DRM	Design Research Methodology
DSL	Domain-Specific Language
EFFBD	Enhanced Functional Flow Block Diagram
FB	Functional Basis
FBS	Function – Behavior – Structure
FDM	Fused Deposition Modeling
GPL	GNU Public License
IBD	Internal Block Diagram
INCOSE	International Council of Systems Engineering
MBSE	Model-Based Systems Engineering
OCL	Object Constraint Language
OMG	Object Management Group
PAR	Parametric Diagram
PKG	Package Diagram
PLA	Polylactic Acid
PSS	Product Service System
SD	Sequence Diagram
STM	State Machine Diagram
SysML	Systems Modeling Language
UC	Use Case Diagram
UML	Unified Modeling Language

1. Introduction

Current trends in product development show an increasing number of required functions [5] to be fulfilled through the cooperation of multiple different disciplines. Examples are mechatronic systems [6] and product service systems (PSS) [7], which both extend traditional mechanical and electrical systems through the integration of software and services, respectively. These multi-disciplinary systems consequently have an increased complexity that must be handled during product development. At the same time there is the constant goal of reducing development time and cost.

The highest impact on the development costs exist during **concept development** [8], where also the crucial interactions between the different involved disciplines are defined. Yet, despite its importance there is little computational support for concept design [9]. Developed concepts are solution proposals described by characteristics that illustrate their unique selling points compared with existing products [10].

One approach that focuses on the conceptual phase as well as the cooperation of large multi-disciplinary teams is **model-based systems engineering (MBSE)** with its standardized systems modeling language SysML [11]. MBSE aims to increase productivity through minimized manual transcription of concepts by using unified system models. Unified system models capture knowledge in a central and consistent repository by combining system knowledge from different disciplines, levels of abstraction and viewpoints. This includes system design, analysis and simulation models to support the development of successful systems [12]. The model-based representation enables additional reuse potentials, e.g. to create concept models in a

composable way, further enhancing development productivity through design libraries [5].

Based on previous work by Wölkl [13] this thesis presents **a library-based concept design approach for multi-disciplinary systems in SysML.** It includes the steps of decomposing the identified design problem into manageable functions and finding suitable conceptual solutions [14]. This is achieved by extending the graphical modeling language SysML with created design libraries and design patterns for reuse. SysML models can represent multi-disciplinary systems, including solution-neutral and comprehensive target specifications and interconnected discipline-specific information.

The **design libraries** contribute by formalizing established and proven design knowledge from the design research, namely the Functional Basis (FB) [15] for solution-neutral functional modeling, elements from a commercial simulation tool [16] for behavior modeling and to plan multi-physics concept simulation, and a service catalogue by Schmidt et al. [17] for service modeling.

Design patterns of object-oriented modeling capture expert knowledge in the form of reusable solutions to known problems within their specific context [18]. The role of the solution patterns in this approach is the formal documentation of multiple concept solutions for identical functionalities. They correlate library elements with other aspects to offer coherent subsystems in the form of partial models.

To investigate the **usability of the modeling support** there is a small scale user study conducted as well as a case study model created. The user study investigates the use of the function library for its impact on modeling and model quality. Its results indicate some benefits of reuse, including improved model quality, formality and a risen modeling workload.

The case study is a SysML 3D printer model. It demonstrates the reuse of elements from libraries and patterns for a multi-disciplinary system with complementary services. The case study captures functional, behavioral, structural and service knowledge in SysML, with the behavior model representing simulation models for concept evaluation. Having all this conceptual information within a unified SysML model allows traceability among system elements from different disciplines and levels of abstraction.

The following section presents the detailed motivation, research goals and a concluding overview of the thesis.

1.1. MOTIVATION

The conceptual phase of product development is a key phase for a successful product [8]. However it lacks in practical computational design support [9]. History shows a clear and rising growth for developed systems to have an increasing numbers of components, interactions and especially functions that are provided [5]. These indicators for system complexity are qualitatively illustrated in Figure 1. This rising complexity is further enhanced by the involvement of multiple disciplines [5], for example for mechatronics and PSS.

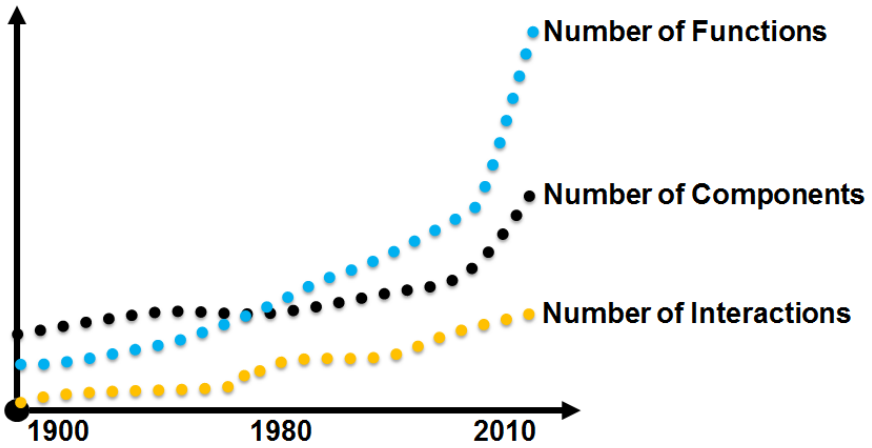


Figure 1: Historical trends of system complexity (adapted from [5])

Among the challenges of mechatronic design [6, 19-21] there is especially the need to connect system elements and knowledge from different domains together. For this, a domain-independent and solution-neutral basis, e.g. a **functional model**, is required to integrate the different viewpoints. Functional models are used for system decomposition [22-24] to handle the complexity of multi-disciplinary systems by breaking down the system into manageable elements to find partial solutions. Further traceability and reasoning is enabled through explicitly linking the properties of multi-disciplinary solutions to domain-independent system representations, e.g. functions [25]. These challenges are similar for PSS design [26-28]. Here especially the relations between service elements, the stakeholders and the physical system must be captured. The task of finding and allocating solutions to the decomposed functions is also comparable between mechatronics and PSS [29].

To support deciding among different alternative concept solutions, they must be **evaluated**. The support of this step is another challenge in the development of multi-disciplinary systems [19, 26]. One way to support it is the integration of early simulation, as envisioned for MBSE in [5] or focused on in simulation-based design [30]. For mechatronic or other multi-disciplinary systems the simulation must be capable of handling the multiple disciplines and their interactions, either by multi-physic simulation [16, 31] or co-simulation [32]. Simulating already during concept design also has the additional benefits of so called “front-loading”, as presented in [33] where it is defined as “shifting the identification and solving of [design] problems to earlier phases”.

Other challenges to develop these multi-disciplinary systems lie in a **lack of communication** between the involved disciplines [19, 21, 28]. One reason for this lack of communication is a missing common language for system

representation. Such a language needs a certain standardization and formality with defined semantics to avoid ambiguity between the disciplines.

Following these needs for a multi-purpose language there exists the **modeling language SysML**. SysML is part of MBSE and can serve as a computational system model for concept design [34] according to the VDI 2221 [14]. It is also suggested for mechatronic systems [20] and used for PSS design [35, 36]. To support the capabilities and use of SysML certain improvements are suggested [37, 38]. These improvements do not focus on SysML itself, but rather on its usability, e.g. lacking modeling methods, guidelines and the high effort to learn it. Modeling support, for example, is much more sought after from practitioners in industry than data exchange between tools [37]. An evolved SysML should include precise semantics to avoid ambiguity and integrate fully multi-disciplinary system representations, including analysis, simulation, and verification [38].

Other needed improvements include libraries that extend the current SysML notation. They are to support model construction by including the “ability to repeat common modeling patterns […] to increase modeling productivity and understanding” [38]. These **means of reuse** have potential in SysML [13, 39], analog to reuse in object-oriented modeling of software development [40, 41]. Knowledge reuse is also necessary to reach the goals of the systems engineering vision 2025 [5] of reducing lost knowledge between projects, which results in increased cost and risk. It further states that combining “formal models from a library of component, reference architecture, and other context models, different system alternatives can be quickly compared and probabilistically evaluated” [5]. This makes such composable design from design libraries a major key to productivity, similar to the practices in electrical engineering [42, 43]. Certain recommendations are given to improve reuse in engineering design:

First, to “leverage the expertise of third parties to improve design reuse”, second to “dedicate resources to prepare and verify designs for reuse” and third to “use direct modeling technologies to modify existing designs into new ones” [44].

The **raised formality** that comes from reusing clearly defined elements from libraries offers further advantages and therefore reasons for reuse. Informal design methods lack in systematic guidance, leading to domain experts often basing their work mostly on experiences, sometimes bias and not considering alternative solutions [45]. Informal design is also more likely in failing to abstract, document and represent the system for effective communication and reuse [46]. While avoiding these disadvantages formal design becomes increasingly more important, especially for handling increasingly complex design tasks [47]. Further opportunities of formal design include a forced “systemic thinking, which is often expressed as holistic and function-based thinking” [37] and more precise semantics to avoid ambiguity. Formally captured design knowledge enables further computational support, for example consistency and compatibility checking, a supported system evaluation [45, 48] or even automatic model generation through computational design synthesis [9, 49, 50]. Such automation also raises the chance of finding a novel and creative solution with more generated concepts [51] and it enables a significant speed up of the whole development process by again utilizing formal knowledge from libraries [52].

1.2. Research Goals

The **main goal** of the presented work is it to improve the support for concept modeling of multi-disciplinary systems by providing proven design knowledge for reuse as formal libraries in SysML. It builds on previous work by Wölkl [13] and extends it by validating the functional modeling library through a user study, adding additional libraries for behavior and services as well as multi-

solution patterns. It creates a direct link to simulation and has an extended multi-disciplinary focus on mechatronic systems and PSS.

A simplified **research impact model**, according to the Design Research Methodology (DRM) [53], is given in Figure 2. It describes the relations between influencing factors between success criteria and the design support. Its connecting directed edges indicate how the factors influence each other. The "+" and "-" signs describe for instance that one factor with a poor state (-) results in another factor having a strong state (+). Here it highlights the relations between the success criteria of improved concept design, shown on top, and the developed design support, shown on the bottom. Starting with an improved, formalized and standardized provision of knowledge for reuse as the design support, it enhances the key factors, which are identified as the abilities to reuse

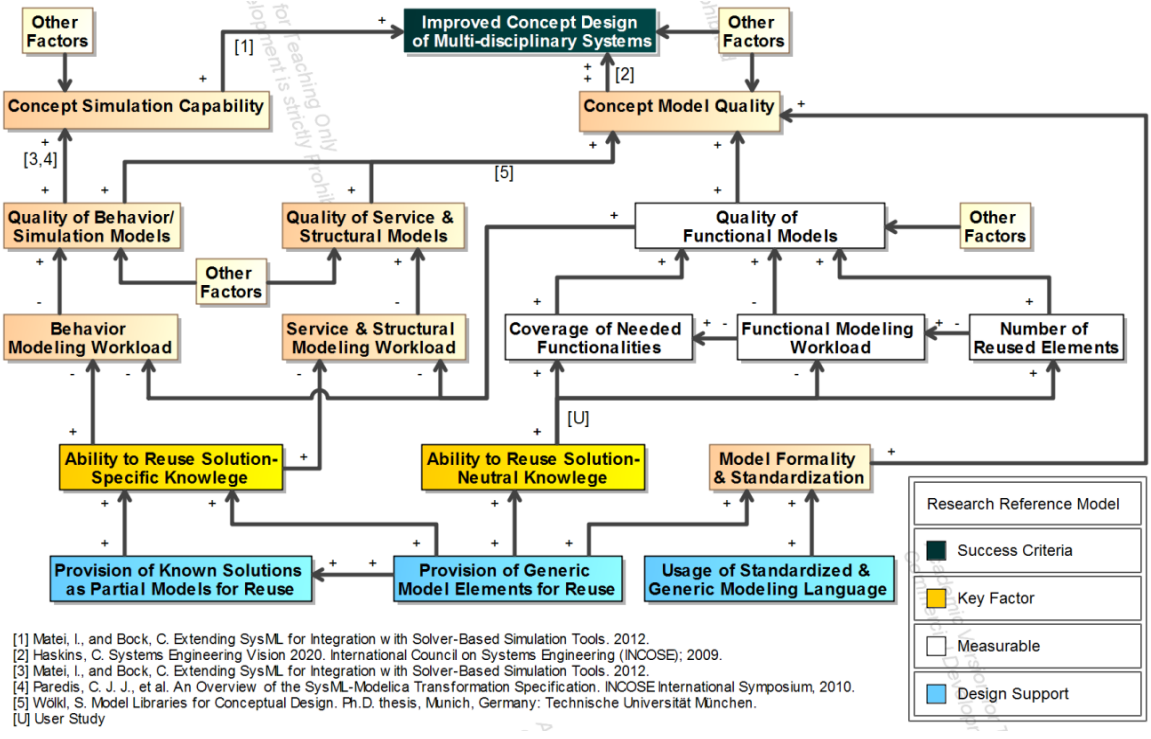


Figure 2: Simplified research impact model, modeled in UML

existing design knowledge. These abilities then reduce the necessary modeling workload, due to more reused elements. This in turn might lead to improved concept model quality, which, together with the included concept simulation capability, facilitates improved concept design. To validate the design support by the measurables of Figure 2 a user study is conducted for the functional modeling SysML library in Section 5. Other related factors exist, as partially indicated on the Figure.

The following **research questions** are identified to reach the research goals:

- How can we support multi-disciplinary concept design?
  - Which design knowledge should be integrated?
  - At what level of formality and detail should the knowledge be modeled?
  - How can we link the modeled knowledge from different domains and levels of abstraction?
  - How can we link the concept model to quantitative simulation models for evaluation of alternatives?
  - What workflow can be provided to use the developed approach?
- Can the developed approach be used to model mechatronic systems and PSS?
- How can the developed approach support the designer?

1.3. THESIS OVERVIEW

The research process follows the **Design Research Methodology (DRM) framework** [53] to realize the presented research goals and answer the research questions. This framework is shown in Figure 3. It has four different stages: (1) research clarification, (2) first descriptive study, (3) prescriptive study and (4) second descriptive study. During research clarification a worthwhile research

goal is defined through mainly literature studies. The first descriptive study includes more specific literature analysis as well as empirical data to elaborate understanding of the existing design situation and to identify major influencing factors, e.g. displayed on Figure 2. The prescriptive study is for developing support to improve the understood situation by addressing suitable factors to reach a more desired situation. This is based upon the previous descriptive study and uses mainly design experience and assumptions of the researcher. Finally, there is the second descriptive study to evaluate the developed design support through determining its impact on the current situation. It uses analysis, e.g. of the results of a case study, or experimentation to gain empirical data. Iterations between the stages and variations of this framework are necessary for its application.

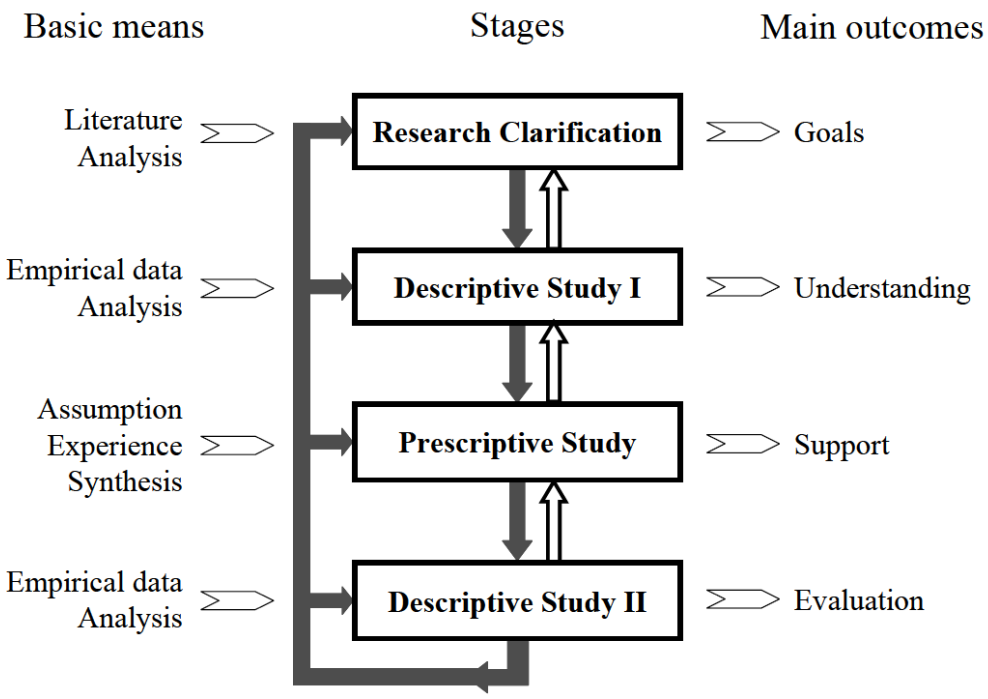


Figure 3: Design Research Methodology (DRM) framework [53]

The application of the DRM framework is shown in Figure 4 by the used methodological **research process**. The numbers on the arrows in Figure 4 indicate the order of the individual process steps. The research starts with clarifying and defining the research task together with investigating the two preceding libraries by Wölkl [13]. After an additional descriptive literature study the function library is iteratively improved, used for the case study and tested by the user study as part of a second descriptive study. Further prescriptive studies follow with the development of the behavior library, to also include simulation, the multi-solution patterns and the service library, of which all are used for an initial descriptive study, i.e. the case study.

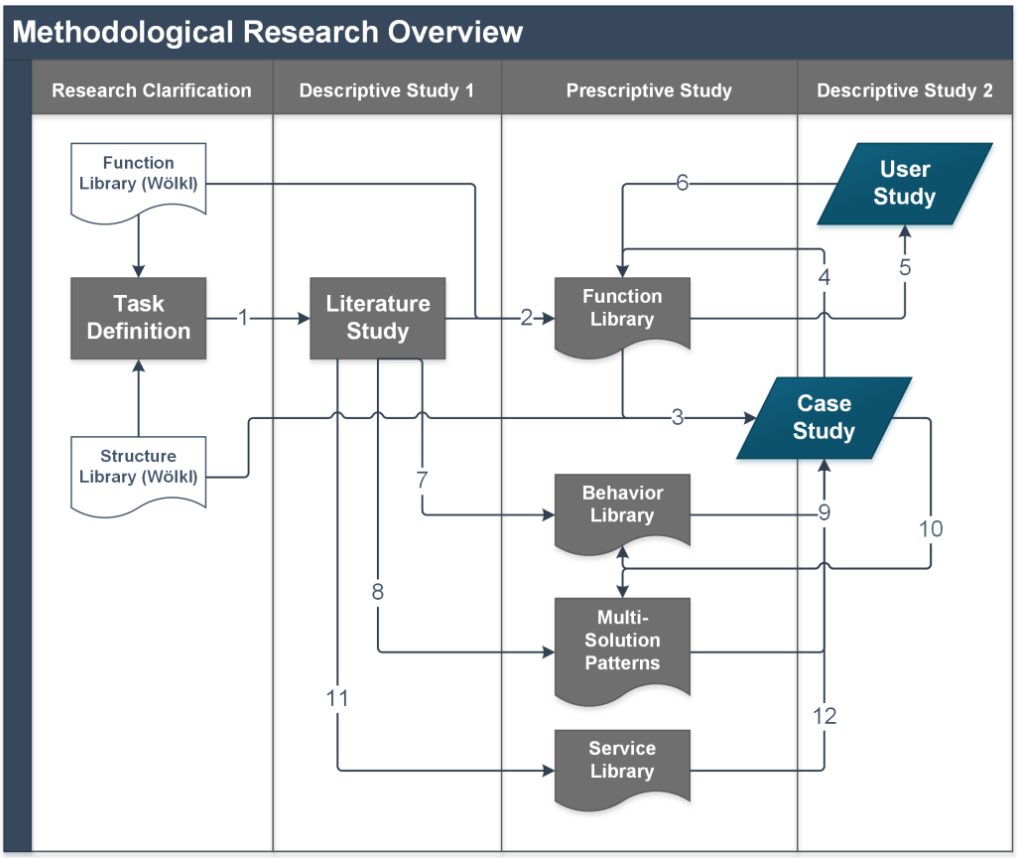


Figure 4: Overview of methodological research process

The presentation of the research in the remainder of the thesis follows this structure: In **Section 2** the background of the research is summarized. It starts with introducing systems engineering in Section 2.1, with the focus on MBSE and SysML. Section 2.2 follows with an overview of the development of multi-disciplinary systems, before Section 2.3 describes the Function – Behavior – Structure (FBS) [54, 55] framework for a design process with distinct design activities for decomposition and searching for solutions. To further elaborate on functional modeling, Section 2.4 describes the use of functional modeling, its definitions, controversies and introduces the Functional Basis (FB). Section 2.5 and 2.6 present libraries and patterns as means to formalize design knowledge for reuse, including the used knowledge for the libraries. At the end of Section 2 further related work is presented in Section 2.7 for approaches in MBSE that use SysML, FBS or include behavior simulation. They are used for the further identification of the research gap, addressed in this work.

Section 3 introduces the case study, which is a 3D printer model in SysML. In Section 3.1 it is presented generally before an overview over its model implementation follows in Section 3.2. It uses elements from the libraries and patterns. Hence, it is used in the following sections to illustrate their use.

The main **Section 4** presents the concept modeling approach in SysML. After an initial overview in Section 4.1, Section 4.2 describes the function library, Section 4.3 the behavior simulation library, Section 4.4 the service library and Section 4.5 the multi-solution patterns. Each of the library sections first defines its library before demonstrating the use within the case study. The multi-solution patterns section additionally gives an example pattern in between. The results of the library and pattern use for the case study conclude Section 4 in Section 4.6.

The following **Section 5** contains the functional modeling user study with its experimental set-up in Section 5.1 and results in Section 5.2. The results

include statistically significant relations, which indicate a good user acceptance of the design library in SysML, resulting in a greater use of the FB and improved model quality. Yet, at the same time the perceived workload increases, too.

The modeling approach is discussed in **Section 6**. The discussion is separated into modeling with library and pattern support in the Sections 6.1 and 6.2. For both parts there are identified advantages and disadvantages stated, resulting from the descriptive studies and comparison to related work. The thesis is summarized in **Section 7**, stating the main contributions, limitations and future extensions. **Section 8** concludes the work.

2. Background

The following section presents the background of the research with its most important concepts. It starts with systems engineering, to introduce MBSE and SysML, followed by aspects of the development of multi-disciplinary systems, the FBS framework, functional modeling, design libraries, design patterns and ends with related work in the form comparable approaches.

2.1. Systems Engineering

The presented work is heavily based on systems engineering principles, with their focus on the multi-disciplinary early development phases. **Systems engineering is defined** as “an interdisciplinary approach […] to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with system synthesis and system validation while considering the whole problem” [56]. The accompanying definition of system is: “An integrated set of elements, subsystems, or assemblies that accomplish a defined objective. These elements include products (hardware, software, firmware), processes, people, information, techniques, facilities, services and other support elements” [56].

2.1.1. Model-Based Systems Engineering (MBSE)

According to the International Council of Systems Engineering (INCOSE), **MBSE is defined as** “the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases” [12]. For the term model there exist many different definitions, which have in common that they refer to a model as a usually abstracted

representation of selected aspects of a system to promote selective understanding of the real system [57].

Generally MBSE is seen as an effective means for developing complex systems [58]. Compared to traditional document-based design, in model-based design it is implied that the models compose an integral set of representations. Such MBSE models are meant to be a "**single source of truth**" [59, 60], meaning that all system information is captured at a central repository. This includes formally captured design decisions and reasoning [61]. Within the model various elements are interconnected, to allow retrieving desired information through traceability [62], as well as automatic change propagation, consistency checking and error identification [63]. Through the included model verification and validation, MBSE enables an earlier aquisition of crucial information, leading again to benefits of "front-loading" [33]. By capturing design knowledge in a model-based way, MBSE allows the reuse of model elements [63], which is utilized in this work. Coming from reuse of object-oriented data in software development [40, 41] there are for example libraries or patterns to capture proven design knowledge for further development projects.

Another major benefit of MBSE is **improved communication** between stakeholders from various domains and disciplines. Systems engineering requires clear and unambiguous communication of the design problem, possible solutions and design reasoning [56]. By creating views from the unified system model different aspects of the model can be represented, fitting to needs and background of the user. This way MBSE provides an abstracted representation, suitable for all involved disciplines. The generic system model then builds the foundation for following discipline-specific detailed development activities [64].

To implement MBSE there is cultural change as well as a well-defined methodology required. This includes training in language, methods and tools

[37]. To support MBSE there are many **modeling tools** on the market. Some tools, e.g. ModelCenter [32], are meant to integrate simulation tools to create and automate simulation workflows and offer shared data from a repository. Other tools focus on system modeling, e.g. Magicdraw [65]. It enables modeling with the modeling languages UML and SysML, but needs for instance additional software to include simulation.

An overview of common **methodologies**, as related processes and methods in MBSE is provided by Estefan in [66]. It is noted that “most of the MBSE methodologies surveyed […], incorporate the UML and/or SysML into specific methods and artifacts produced as part of the methodology” [66]. The same is true for the approach presented here that uses SysML. Reasons for the lacking acceptance and application of MBSE in industry indicate that there is neither a broad agreement on systems engineering processes, nor on the proper use of tools to handle system complexity [67]. This refers back to the use of the selected modeling language SysML, which needs further improvements in respect to its usability [37, 38]. A brief presentation of SysML is now given.

2.1.2. The Systems Modeling Language SysML

The systems modeling language **SysML is defined as** “a multi-purpose graphical modeling language, for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities” [11]. Its current version is 1.4. SysML is developed under the Object Management Group (OMG), based on its unified modeling language UML [68, 69] for software development. SysML reuses and extends UML diagrams, as shown in Figure 5. This relationship to UML also serves as a natural link to software development, based on multi-disciplinary SysML models.

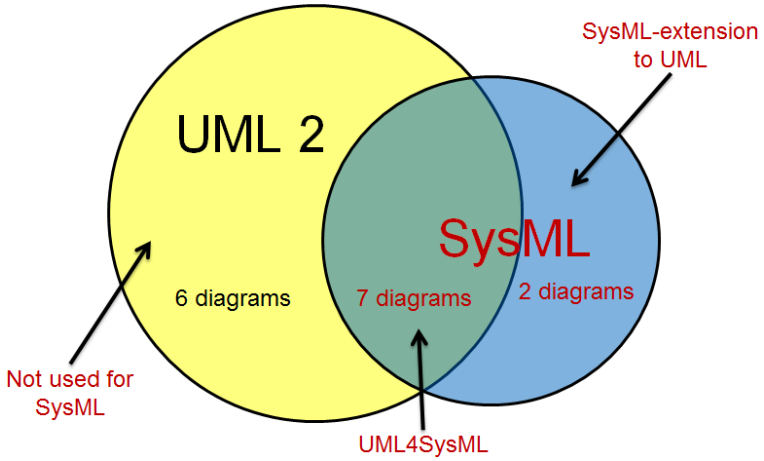


Figure 5: Overview of SysML/UML interrelationship (adapted from [11])

To improve the readability of the presented text there are certain **formatting conventions** applied regarding SysML. To highlight that certain terms come from SysML they are written in *italic* in the following descriptions. To extend and customize SysML there are *stereotypes* used. They are an extensibility mechanism of SysML to derive new types of modeling elements, e.g. for a domain specific language (DSL). Custom *stereotypes* of the presented approach are written in <<theses>> brackets, accordingly to their representation in SysML. Names of model elements of the case study are marked in “these” brackets, i.e. similar to citations from the SysML model.

The **nine diagrams of SysML** are displayed in Figure 6 [11]. There is the *requirement diagram* (REQ), which graphically depicts text-based *requirements*, their interrelations and other model elements that *satisfy* or *verify* them. The *package diagram* (PKG) serves quite flexibly to organize the model in *packages*.

The ***block definition diagram*** **(BDD)** represents structural elements as *blocks* with their interrelationships, e.g. by *associations* or *generalizations*. *Blocks* are defined in SysML [11] as modular units of the system description to provide a general-purpose capability to model systems as trees of modular components.

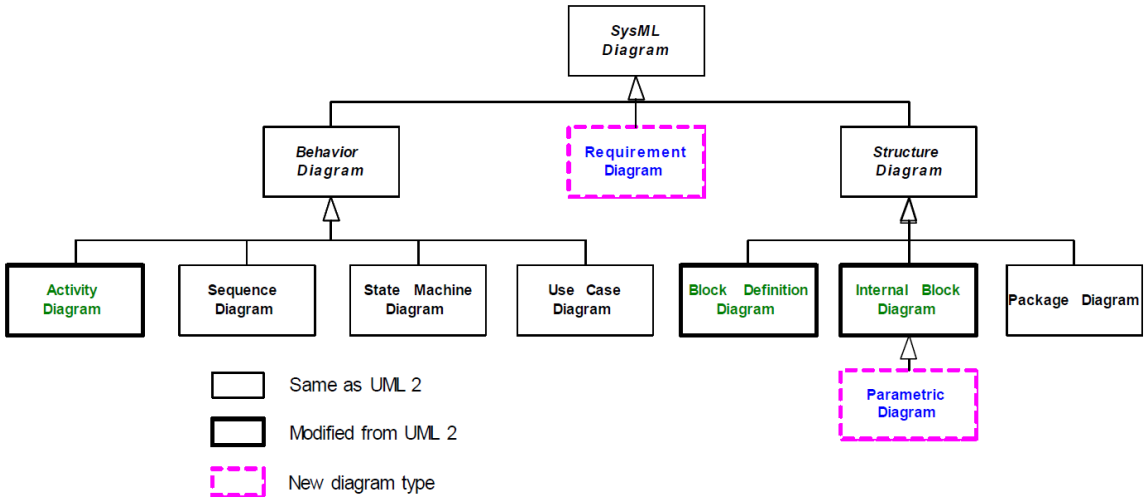


Figure 6: SysML diagram taxonomy [11]

They can for instance have properties to specify its values or parts by *value properties* and *part properties*. A property has a type that supplies its definition. A *part property* belonging to a *block*, for example, may be typed by another *block*. Example elements from a BDD are shown on the lower left side of Figure 7. There are for instance *generalizations* between a more general "Generic System" and its two more specific "System 1" and "System 2". With *generalizations* the specific element inherits the features of the more general element [69]. Here it is the *value property*, whose *default value* "DefaultValue" gets redefined into "Value 2". There are also two types of *associations* used: one general *association* and two *composition* relations between "System 1" and its thereby assigned *part properties* with the types "Part A" and "Part B". The relation to "Part A" has a *multiplicity* of "1..*" and not the standard *multiplicity* on one, meaning that "System 1" has one or more of "Part A".

The related ***internal block diagram*** **(IBD)** shows the internal structure of a particular *block* in terms of properties and *connectors* between them. To specify the involved interfaces, *ports* are used. "*Ports* represent interaction points

between a classifier and its environment. The interfaces associated with a *port* specify the nature of the interactions that may occur over a *port*" [69]. Extending the standard UML 2 *ports* there exist *flow ports* in SysML, which are deprecated in the current version 1.4, but still used here. "*Flow ports* are interaction points through which data, material, or energy can enter or leave the owning *block*" [11]. This way they specify the "input and output items that may flow between a *block* and its environment" [11]. Example elements from an IBD are shown on the top right corner of Figure 7. There are the two *part properties* of "System 1" with their interconnected *ports*. The *ports* have the *interface blocks* from the bottom of Figure 7 as types with matching input and output *flow properties*. The similar *parametric diagram* (PAR) displays equation systems as constraints on properties, to support analysis.

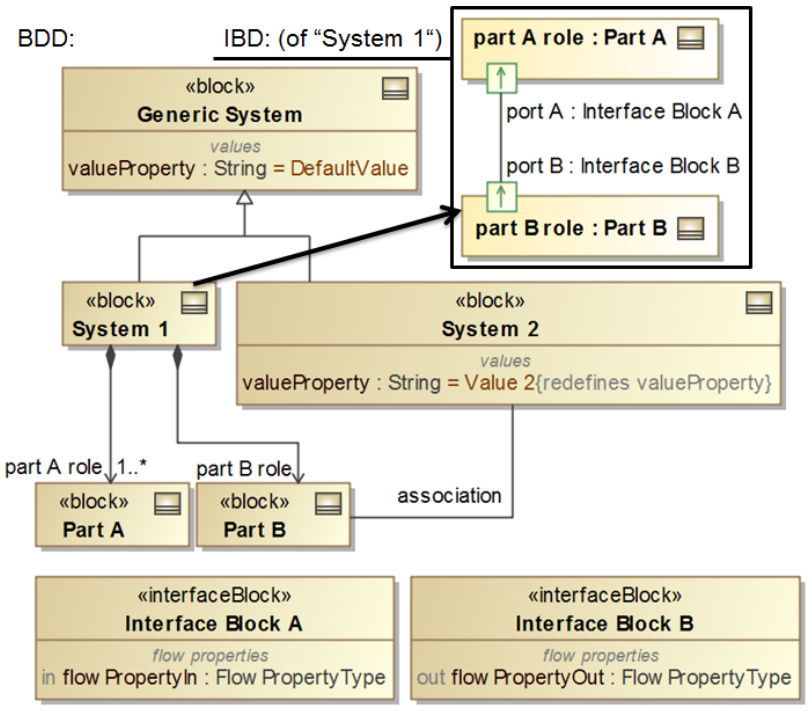


Figure 7: BDD and IBD example

The ***use case diagram*** **(UC)** describes how a system is used by its *actors* to accomplish its goals. *Actors* are defined to specify “a role played by a user or any other system that interacts” [69] with the system. *Use cases* are specified as s “set of actions performed by a system, which yields an observable result that is, typically, of value for one or more *actors* or other stakeholders of the system” [69].

The ***activity diagram*** **(ACT)** represents behavior in terms of *actions* based on their inputs, outputs, control and how the *actions* transform the inputs to outputs. This way they can show the complete flow of system operations. One ACT displays one particular *activity*, which represents behavior that is composed of individual elements, e.g. *activity nodes* such as *actions* [11]. The a*ctions* represent the single steps within an *activity*, which are not further decomposed within the *activity*. However, c*all behavior actions* reference an *activity* definition, in which case the execution of the *call behavior action* involves the execution of the referenced *activity* [69]. In Figure 8 there is the *call behavior action* “ActionName” with the called *activity* “CalledActivityName”. Its two *pins* are “typed elements and multiplicity elements that provide values to *actions* and accepts result values from them [69].

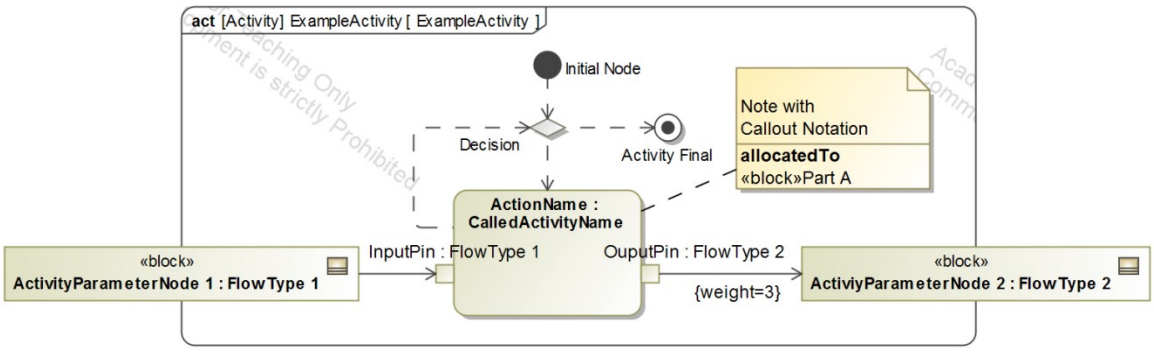


Figure 8: ACT example diagram

They correspond to the *parameter nodes* of the called *activity*, which have a direction and type defined. "*Activity parameter nodes* are object nodes at the beginning and end of flows that provide a means to accept inputs to an activity and provide outputs from the *activity*, through the *activity parameters*" [69]. In Figure 8 the types of the *activity parameters* for both *activities* are "FlowType 1" and "FlowType 2". The *activity parameter nodes* and *pins* are connected by *object flows,* which are activity edges that have objects or data passing along [69]. The outgoing *object flow* in Figure 8 has additionally a *weight* of "3" assigned to specify "the minimum number of tokens that must traverse the edge at the same time" [69]. Besides the *object flow* there exist the *control flow*, which is an edge that starts an *activity node* for sequencing their execution [69]. Figure 8 shows a *control flow*, that starts at an *initial node*, goes to a *decision node* for modeling a loop and ends the execution of the *activity* at an *activity final*.

The *sequence diagram* (SD) represents the system behavior in terms of a sequence of messages exchanged between elements. They include explicit duration and timing of these interactions. The *state machine diagram* (STM) contains transitions between states triggered by events. Examples for often used states are "On" and "Off".

With these diagrams **used for MBSE**, SysML allows formal modeling for its so called four pillars: for requirements, behavior, structure and parametrics. The integration to other engineering aspects, especially for analysis and more detailed discipline-specific design, is designated. An example framework is shown in Figure 9. To interconnect model elements from all diagram types cross-cutting relationships, e.g. *allocations*, are used. One example is given with the *callout notation* in Figure 8, showing the *allocation* from the *action* to the *block* "Part A" of Figure 7. Here a so called *allocation* from usage to definition is used.

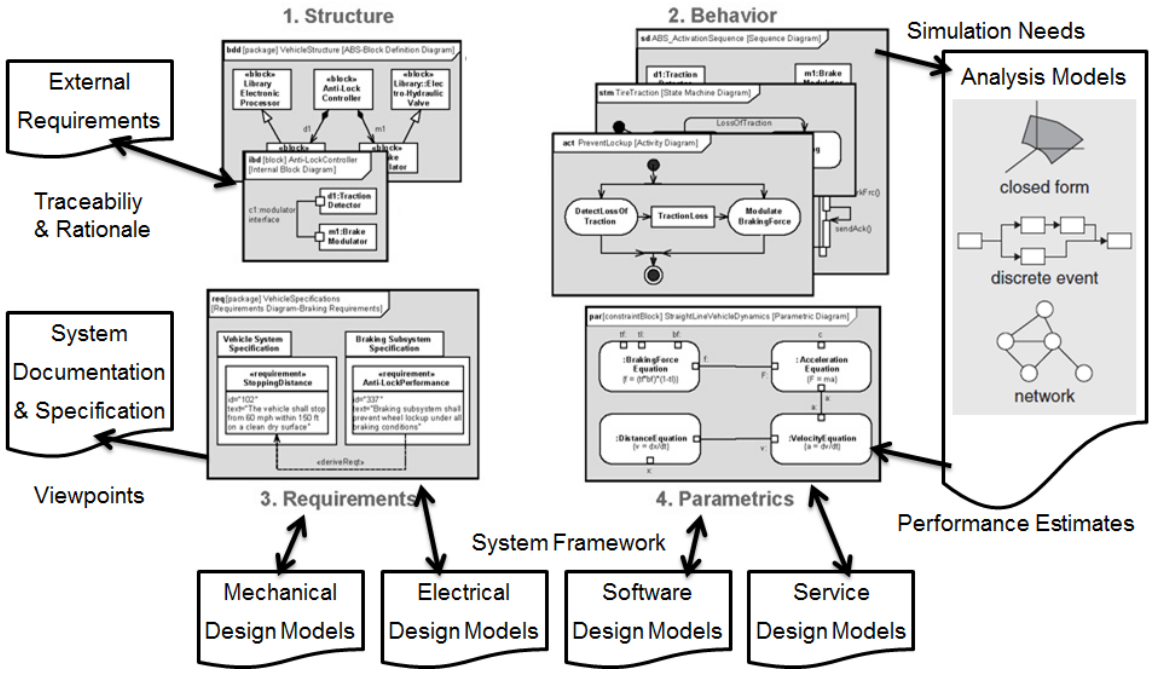


Figure 9: SysML system model as a framework for analysis and traceability (adapted from [70])

Allocations of usage apply when both ends of the relation are usage elements, e.g. *parts*, *actions* or *connectors*. *Allocations* of definition apply when both ends of the relation are elements of definition, e.g. *blocks*, *activities* or *use cases*. "When *allocating* definition, every usage of the defining element retains the *allocation* [whereas the *allocation* of usage] is only specific to that [usage], not to any other similar [occurrences], even if they are typed by the same *block*" [70].

As a modeling language SysML is defined by its **syntax and semantics**. For the syntax there is abstract and concrete syntax. For semantics there are static and dynamic semantics [71]. Abstract syntax, i.e. grammar, defines the syntactic elements, e.g. letters, and clarifies how they build up constructs, e.g. words. Concrete syntax defines the means of expression, e.g. the notation or form of presentation. Static semantics defines how constructs have to be combined to be meaningful, while the dynamic semantics describe which

meaning is contained [72]. Following these definitions, SysML provides a clear abstract syntax that has to be obeyed while modeling, together with a limited and adaptable concrete syntax. Semantically it also provides some static semantic rules, e.g. about specialized *dependencies*. Additional dynamic semantics are provided by the following design libraries, e.g. by the FB descriptions [15].

This provided formality of SysML reduces ambiguity and is together with its standardization an additional **reason for selecting SysML**. With its standardization it is well-known in industry and even envisioned to become the standard systems engineering language [70]. As such it manages complexity, improves communication and enhanced understanding [73] by offering general and multi-discipline applicability and adaptability.

2.2. Development of Multi-Disciplinary Systems

An overview over different methodologies and methods for the development of multi-disciplinary systems is given by Eigner in [74] additionally to MBSE methodologies by Estefan in [66] and PSS methodologies by Vasantha et al. in [28]. Relevant examples for multi-disciplinary systems, whose concept design is to be supported, are mechatronic systems and PSS. While **multi-disciplinary systems are defined** as those that involve elements from any different disciplines and domains, mechatronic systems are defined as a combination of mechanics, electronics and software. Their focus lies in the extension of mechanical systems through electronic sensors and controlled actuators [6]. PSS are defined as systems to enhance value “through the mutual provision of a product [and] service” [7]. Their services are defined “as a set of activities to deliver service contents from service providers to service receivers through service channels” [26] to “contribute to the realization of service goals” [26]. Taking up the challenges of the development of multi-disciplinary systems

from the motivation, one main aspect is the integration and cooperation of all different involved disciplines, including services [19-21, 26-28].

Several **example methodologies** are presented to gain an understanding of the development process in each separate domain and for multi-disciplinary systems. The VDI 2221 [14] for technical systems is an example from mechanical engineering. It provides the basic concept design tasks in Section 4.1. An example methodology from electrical engineering is the similar VDI/VDE 2422 [75]. A difference between mechanical engineering and electrical and software engineering is the role of behavior. While in mechanical engineering behavior is a result of the developed system, the other disciplines use a desired behavior as part of their requirement specification [74]. Such a behavior is then more similar to the functions of mechanical engineering, which are not used. An example is the Y-diagrams of Gajski [76] for embedded system design. Also for developing electronic embedded systems there exists refined automated design support [42, 43]. It utilizes defined functional, logical and physical elements for composable design and simulation. Yet, a direct translation of these automation capabilities into mechanical design cannot be expected [77].

From software engineering comes the object-oriented modeling language UML [68, 69], the basis of SysML of Section 2.1.2. An example for a software engineering methodology is the spiral model of Boehm [78], which is also the origin of the V-model of the VDI 2206 [79] for mechatronic systems. The V-model serves as a macro cycle within which the developed concept modeling approach here can be put into a broader context in Section 4.1. It combines discipline specific detailed design at the bottom of the V-model with unified development at beginning and end for concept design plus system testing and verification. For the development of multi-disciplinary PSS there exists, for instance, the service CAD approach by Komoto and Tomiyama [26], which focuses on service

formalization and systematic generation of PSS. A final current multi-disciplinary design framework for developing cyber-physical and mechatronic systems is from the research project mecPro² [80]. It is based on the VDI 2221 [14] and the similar SPES modeling framework [81] for embedded systems. Here it is considered for its levels of abstraction and consideration of SysML. Further insight into SysML and model-based systems engineering is given in Section 2.1.

2.3. Function – Behavior – Structure (FBS)

Function – Behavior – Structure is a framework developed by Gero [54] for engineering design. It describes design processes with distinct design activities for decomposition and the search for solutions. As such it is used for model-based knowledge representations. It uses solution-neutral functional models to capture the purpose of the design object. More details about functional modeling are given in the following section. Next comes the behavior, which is often modeled through a network of physical effects that fulfill functions by realizing them through working principles, e.g. by Helms and Shea in [82]. Similarly there exists Function – Behavior – State by Umeda et al. [55], who defines his behavior to cause state transitions, which are captured on the state level. In both ways, the behavior realizes the functions in a physical, but component independent way. This kind of expected behavior represents the system's expected interactions for guidance and evaluation of potential design solutions. It is often differentiated, e.g. by Kannengiesser and Gero [83] towards the so called structure behavior. The structure behavior includes properties of the system that are derived from the observation of a specific design solution and its interactions with the environment. It is used for the comparison with the expected behavior to evaluate design solutions. It requires the final structure level of FBS. This structure contains mostly physical components, i.e. it can also include software

or services, to provide the embodiment of the target functionality and respectively the physical effects of the expected behavior [3].

The Function – Behavior – Structure (FBS) ontology is an underlying **foundation of the presented approach**, even though the levels presented in Section 4.1 are not called as in FBS. With the provided libraries there is support for functional modeling, behavior modeling and final structural or service modeling, which reflects a FBS proceeding. One reason to not directly use FBS is to avoid limiting the modeling on using only provided behavior elements, when any description of the principle solution can be used to concretize the functions. An extensive behavior model, for instance, is only used when a more direct progression from function to structure is not possible [84]. Also, the behavior library can be used for its simulation capabilities after having certain structural elements identified. Other reasons for not directly choosing FBS are that the final structure level contains more than just the mechanical system structure, e.g. software or services, supported through the service library. Also, not using the term structure helps to avoid ambiguity e.g. compared to function or behavior structures, which are also structures.

2.4. Functional Modeling

For functions and functional modeling there exist many different definitions. The functions used here are **defined as** “input/output relationship[s] with the flow (noun) describing the in- and outputs and the operator (verb) describing the change between in- and output to express what a system should do” [3]. This is based on the approach by Pahl and Beitz [23] together with the taxonomy defined in the Functional Basis (FB) [15]. At the same time it conforms to the systems engineering definition of functions stated as “transformations of input flows into output flows performed by the system to achieve its mission” [57].

2.4.1. Use of Functional Modeling

Despite the usefulness and benefits of functional models there is generally a lack of practical application of functional modeling in industry [29, 85, 86]. Yet, “considering the impact of the work on industry practice, the use of function has gained ground over the past decade” [87]. This might relate to its solution-neutral and therefore discipline-independent knowledge representation fitting for the development of multi-disciplinary systems.

The use of functional models for the development of mechatronic systems is common and shown through multiple examples by Van der Auweraer et al. in [19]. For the concept development of PSS, functional models are also used, as in [7, 24] or reasoned in [29]. Further related reasoning about functions in product design and functional modeling approaches is given by Wölkl in [13], in the review papers of Erden et al. [88], Deng [84] and with a special focus on functional modeling across disciplines in [89] from Eisenbart et al.

According to Saunders et al. [90] **functional modeling contributes** to achieving innovative products, e.g. by identifying additional functionalities or changed functions in the system. Functional models allow a high-level system overview to ensure fitting abstraction levels, even if the implementation is still unclear or not yet known [59]. Such abstract representations decompose what a system is supposed to do in a solution-neutral way. Hierarchical decompositions are required to derive “components in any complex system [that] will perform particular sub functions that contribute to the overall function” [91]. Besides this enabled search for partial solutions, functional models also allow to trace fulfillment of functional requirements and they reduce the danger of making models too detailed through over-modeling and redundancy [59]. This role of functions as integration elements makes it “possible to describe a system at different levels of detail, focusing on the points of interest to the user while

maintaining coherence of the model" [21], which is identified to be crucial for developing multi-disciplinary systems. To summarize, functional modeling can provide "both a better understanding of increasingly complex systems and possibility for making use of ever increasing computation capabilities" [88].

With functional modeling being used in different disciplines in different ways, there are several **controversies**. In [89], for example, there are seven different functional modeling perspectives identified, which are used in various combinations. Another example for the varying understanding of the term "function" comes from Albers and Zingel [37] and their conducted study. Some derived contrasting statements are illustrated in Table 1.

Functions describe the designer's intention of the purpose of a design	Functions realize functional requirements
	Functions are abstraction of intended and useful behavior of an artifact [84]
Functions can be described in mathematical terms	Functions are abstract specifications of transformations
Functions describe an active behavior	Functions are an interaction of components to achieve a certain behavior
Functions are characteristic tasks, actions, or activities that must be performed to achieve a desired outcome [56]	Functions are transformations of (matter/energy/information) input flows into output flows performed by the system to achieve its mission [57]
Functions describe the effect of the object on the environment [92]	Functions describe internal parameters of the object [92]

Table 1: Contrasting pairs of aspects related to the term "function" (adapted from [37])

As indicated there, a broad **variety of meanings of functions coexists**. Functions can focus on subjective system purpose, its actions, behavior, effects on external elements or the internal interactions. Their representation can vary

from limited sets of verb and noun pairs [15] to mathematical equations or free sentences [2]. Their uniting factor is to bridge between human design intention and physical artifacts, i.e. their “common role of relating goal descriptions of devices with structural descriptions of the devices in a general and interdisciplinary way” [93]. As argued by Vermaas [93] this coexistence is required, because the meaning of specific functions in a functional model depends on the particular development task. To address this coexistence of different functional modeling approaches and to refine the functional modeling approach that is used here, a systematic comparison of different functional models in SysML was conducted [2].

Another minor controversy about functional modeling is its claimed **solution-neutral representation** compared to a more solution-afflicted representation. Here it is concluded that “functions are not completely solution-neutral, but they are also not component-afflicted” [37], which makes them close enough to be solution-independent on a conceptual level where still multiple different concrete realizations may provide the functionality [3].

2.4.2. The Functional Basis (FB)

Also due to the many coexisting functional modeling approaches “it is important to communicate abstract functions in a consistent manner” [87]. Also human design intention is an abstract and subjective concept, which is not easy directly used as the function description [94]. This leads to a need for formalization. One widely accepted attempt to standardize and formalize functional modeling by addressing a lack of semantics is the **Functional Basis (FB)** by Hirtz et al. [15]. It reconciles and evolves previous work [95, 96] to offer a controlled vocabulary with defined semantics to reduce ambiguity of functional models

The **FB contains** 53 verbs for functions together with 45 nouns for flows, each organized in a three-level hierarchy. The highest-level hierarchy terms, e.g. “material”, “energy” and “signal” for flows and “convert”, “connect” or “support” for functions, are the most abstract. The following two hierarchy levels contain more details, e.g. “mechanical energy” or “rotational mechanical energy”. An excerpt of FB hierarchies are given in Figure 20 and Figure 22 with the function library. All terms of the FB have descriptions to define their semantic meanings also through basic examples.

In general **using the FB results in** better designed products [97] and more critical thinking by students in engineering design courses [87]. Work by Caldwell et al. [98-101] to empirically evaluate the use of the FB shows that the second hierarchy level of the FB is the most informative [100], which is used almost exclusively by modelers [101]. Yet, the use of additional free language within a model greatly increases the understanding of the model due to provided context that helps the user [98]. This is especially true for flow nouns, which can not only offer additional knowledge to increase the expressiveness, but also reduce the uncertainty [102]. In general there is an increase of the functional quality of ideas generated by designers for generating high quality concepts [98], especially for compact and pruned models that use the FB.

2.5. Design Libraries as Knowledge Bases

Design libraries are a major method to support reuse through the provision of formalized knowledge. This section presents first the reasoning for focusing on libraries for concept modeling in SysML and second the used knowledge bases to be incorporated into SysML libraries.

2.5.1. Why Libraries?

With a major goal of MBSE being the integration of systems knowledge within a unified representation, e.g. with SysML, it is questioned by Reil [60], why one should have libraries directly in SysML? The answer is that the libraries can be integrated by having them in SysML, since in SysML all conceptual design knowledge is captured and therefore the reusable design knowledge must be provided. **Libraries are a common means of reuse**, for example established in object-oriented software development [40, 41], where systematic reuse is the most effective way to significantly improve development. Such libraries provide collections of basic software functionalities for the designers. The provided knowledge is hereby barely interconnected, in contrast to e.g. frameworks [41]. There are two ways to use elements from object-oriented libraries: their direct instantiation or to derive more detailed objects through inheritance.

In engineering design, for instance of PSS, reuse is also of high importance with “design knowledge obtained from past product cases provid[ing] helpful information to designers, especially in the conceptual design phase” [7]. The generally claimed **benefits of reuse** are faster development, reduced development risk and better understanding of the system through standardization [39, 103]. In engineering design there also exists a potential for increased design quality and productivity through design reuse [44, 104]. For example it is identified that “reusing an existing design can save 30%-80% of design time for new products associated with existed models” [44]. With respect to reusing knowledge resources, e.g. from libraries, their reuse provides the greatest foreseen benefits with advantages of over 20% improvement of time, quality or performance [105]. Fitting to the recommended actions to improve reuse [44], libraries provide resources to prepare and verify designs, they support modeling to reuse the existing design knowledge and they enable the reuse of third party

expert knowledge. Such external knowledge is used as knowledge bases for the libraries, as explained in the following section.

2.5.2. Knowledge Bases Used

Reusing existing third party expert knowledge [44] also supports the quick building of a “critical mass” of reusable components [106], to make managing their reuse worthwhile. Therefore different existing knowledge resources are utilized here and formally modeled in SysML:

- **Functional Basis (FB):**

The FB for the function library of Section 4.2 is introduced previously in Section 2.4.2.

- **LMS Imagine.Lab Amesim:**

LMS Imagine.Lab Amesim [16], or in short Amesim, is a multi-physics simulation tool of Siemens LMS. Amesim models are essentially graphical representations of differential equation systems. These time-dependent physical equations of component behavior are based on bond graph theory [107]. Amesim comes with a database for two types of components, physics-based ones, e.g. mechanical, hydraulic, thermal, or electric elements, and applications oriented elements, e.g. for powertrains or cooling systems. Its provision of well documented, scalable and especially valid, evaluated and proven design knowledge within its database is the main reason for selecting Amesim instead of e.g. Modelica [31]. Here its database is implemented in the behavior simulation library in Section 4.3. An excerpt of an Amesim model is given on the right side of Figure 35.

To clarify the following implementation and usage details in Section 4.3, the underlying **bond graph theory** is briefly presented. It is for graph-based, multi-domain modeling based on the conservation of energy. Along its bonds,

power is transmitted in the form of effort and flow variables, e.g. the electromotive force [15] and current for electrical power or the force and velocity for translational mechanical power [107]. To derive usable equations there must be a causality defined for each bond. Causality means that the node on one side of the bond defines the effort variable and the node on the other side defines the flow variable. When setting up the network of bonds, the causality is forwarded from nodes with fixed causality. If different causalities would be applied to a single bond there exists a causal conflict and the model must be changed accordingly until all causality is valid. Additional information about bond graph theory is available by Borutzky in [107].

- **Service Catalogue:**

The service catalogue by Schmidt et al. [17] is used here. It contains a hierarchy of generic services. Its intention is it to help designers to identify suitable types of services for PSS. The catalogue focuses on industries, which are providing complex technical products. This focus results in certain limitations together with the time-dependency of the data acquisition for the catalogue.

From initially over one thousand services from sources in literature and companies the **catalogue contains** 265 services, grouped into 63 clusters, 19 super clusters in four categories. The four main categories are: “services supporting consumer customers”, “services supporting business customers”, services supporting product” and “services supporting outcome”. The clustering process involves among other things 53 additional customer functions to help identify identical and related services. This underlying functional foundation is also a reason for selecting the presented service catalogue.

Alternative service categorizations [108, 109] exist mostly in classifications of services, which are too abstract to support practitioners in finding new concrete services. Other existing service ontologies [110, 111] also

do not suggest concrete services and more are used for structuring services. Comparable service taxonomies [112, 113] focus only on the differentiation of services by defining criteria. Their “way of identifying new services is not easily applicable for practitioners, as they first have to understand the taxonomy and derive their services from” it [17].

- **eCl@ss Standard:**

The eCl@ss standard [114] is a cross-industry data standard for the classification of products. It has a hierarchical system with properties for a detailed product description. It is used for a SysML structure library by Wölkl [13] for generic and mostly structural elements that realize the modeled functions, as done here, too. The elements in the structure library have additional object *ports* for the functional flows of the function library. These object *ports* serve for the identification of matching functions by offering relations between abstract functions and specific components. This capturing of function-component-relations knowledge is similar to function component matrixes [87].

2.6. Design Patterns

To enable reuse in addition to libraries there are also design patterns to capture recurring solution knowledge. They too fit to the recommendations for action to improve reuse [44], by providing verified solutions for the modification of existing designs into new ones.

2.6.1. What are Patterns?

Based upon the initial use of patterns in architecture in 1977 with the fundamental idea that “all creation is simply an imitation of an original pattern" [115], there are well known and commonly used patterns in software engineering, e.g. by the so called “gang of four” [18]. Their **definition of a design pattern** is “captured expert knowledge in the form of reusable solutions to known problems

within their specific context" [18]. To document these solutions there are several elements mandatory: all patterns need a unique name, a problem description, a description of the provided solution, the forces, i.e. often contradictory considerations that must be taken into account, and the context in which the pattern can be applied. Additional obligatory information can also be documented. For example the resulting context, design rationale or application examples [116].

From **industrial experience of applying patterns** in software engineering [117, 118] it is known that patterns support the communication of complex solutions and are useful to encourage the reuse of best practices extracted from working designs. Yet, their creation is also difficult and time-consuming, requiring successful documentation of the essential parts of working designs. Further claimed benefits of the use of patterns [118] are an increased productivity and program quality together with a skill increase of novice designers, learning from proven solutions. Criticisms of pattern application [119] focus on lacking formal foundations and that patterns target the wrong problem in a sense that solutions should not be copied to avoid resulting inefficiency.

2.6.2. Patterns in Engineering Design

Existing work about the application of **patterns in systems engineering** by Cloutier and Verma [106, 120] takes up the design patterns of object-oriented software development. They provide a framework for their documentation, classification, and management with the goal of the systems engineering community building itself "a maturing source of patterns that can be leveraged for enhanced engineering effectiveness and efficiency" [120]. Work done by Hein et al. [121] and Kruse [39] for instance suggests a two-fold approach to extract patterns for SysML: By analyzing and taking over software patterns and by

extracting patterns from SysML best practices. Other current examples for MBSE patterns exist from Weilkiens et al. [59] with the more high-level “Zig-Zag” development pattern or patterns for SysML diagram layout.

For mechanical engineering, or to be more specific for mechatronic systems, there exist the **solution patterns from Anacker et al.** [122, 123]. They are defined to describe domain spanning principle concept solutions. They describe these concept solutions in the form of partial models that include various aspects to offer coherent subsystems. The following aspects are part of their description: pattern feature characteristics, context, functions, active structure, solution principles and behavior in the form of activities and state transformations. This specification allows a holistic and domain-spanning model representation. It forms a basis for communication and cooperation of the designers from different disciplines during the development process [122].

For **solution patterns in engineering design** in general it is claimed that “most design processes in practice consist of combining known, preferably well-proven solution patterns” [124]. Here a broader definition of solution patterns is used. They are simply defined as aggregations of characteristics and properties with known relations between the two. These characteristics and properties follow hereby the CPM approach of Weber [125] with characteristics standing for directly influenceable structural information of a product and properties describing the product’s resulting behavior. Such solution patterns allow two ways of product innovation: By either replacing one or a few solution patterns in a design or by developing an entirely new solution pattern. Other existing patterns in engineering design are, for example, from Salustri [126], which are very abstract, generic and not model-based, or from Deigendesch [127], which are text-based patterns specifically for micro engineering.

2.7. Related Work

For additional background information there exist also several related approaches to the work in this thesis. **Based on FBS** by Gero [54] and Umeda et al. [55] there exists, for example, the KIEF framework by Yoshioka et al. [128]. KIEF stands for Knowledge Intensive Engineering Framework. It uses a physical concept ontology to integrate engineering knowledge. An approach for concept generation from a functional point of view is also described by Kurtoglu et al. [129]. It derives the structural model directly from a flow-based function structure by relating functions with components according to pre-defined rules. Yet, despite custom computational support both of these two approaches lack in standardization and application. Another FBS-based approach is RFBS with added requirements, for instance as implemented with SysML by Christophe et al. [130]. It focuses on automatic synthesis of conceptual design solutions by reusing knowledge from ontologies. The automation method maps each function to one or more of six abstract organs using an “online sematic atlas, based on contextual closure between verbs” [130]. The combination of organs then leads to different structural possibilities. Yet, this implicitly limits the solution space through the reduction from functional structures to combinations of only six different organs.

Related **MBSE concept modeling approaches** in SysML are, for example, the FUSE method by Hutcheson et al. [131] or the FAS method by Lamm and Weilkiens [132, 133]. FUSE [131], i.e. function-based systems engineering, applies functional modeling to formalize and integrate mapping of customer needs to desired functionality, behavioral modeling as well as the identification, modeling and selection of solutions in SysML. Yet, compared to the work in this thesis it lacks in formalization and modeling support. The FAS method [132, 133] uses *blocks* and IBDs for functional modeling in SysML

instead of *activities* and ACTs used here. The syntax of *blocks* and IBD is also suitable for function structures. Yet, *blocks* are defined in the SysML specification as modular units of the system description, i.e. parts of the system [11]. A function on the other hand is an abstract and qualitative description of what a system is doing with respect to its conversion of input flows into output flows. This matches with the here selected *activities* on ACTs, which are “used to describe […] the flow of inputs and outputs among *actions*” [11]. Also there are no further defined semantics in the FAS method, e.g. from the FB. Instead it focuses on the grouping of functions according to criteria that should be based on conceptual rather than technical aspects, to not move away from a solution-neutral functional model.

Other approaches especially for the **development of mechatronic systems with SysML** are the SysML extension of Chen et al. [134] and the design framework of Wu et al. [135]. Both approaches use functional modeling for an initial domain-independent representation and include some geometry information for early virtual prototyping. Yet, this complexity makes especially the approach of Chen et al. [134] into a collection of many highly specific SysML extensions and consequently even more difficult to learn and properly implement than standard SysML [37]. Similar to FAS, Wu et al. [135] also focuses on the definition of fitting modules while neglecting to integrate knowledge for reuse.

Approaches for the **development of PSS** are, for example, the PSS design process proposed by Kim et al. [24] and the knowledge management method for supporting conceptual design of PSS by Nemoto et al. [7]. The six defined steps of Kim et al. [24] are requirement identification, stakeholder activity design, PSS function modeling, function-activity mapping for PSS concept generation, concept detailing and finally prototyping. These steps are not fundamentally different to some mechatronic design processes, showing again a

fundamental analogy towards mechatronic design, fitting to [29]. The knowledge management method of Nemoto et al. [7] presents a PSS design catalogue knowledge representation with its specific design catalogue viewer to retrieve knowledge. While the potential reuse of formal service knowledge is beneficial, there is limited expressiveness of non-service aspects. Superior in these terms is the model for designing generic services in SysML, by Dhanesha et al. [35]. It presents a highly specific SysML extension for formal service modeling.

Based on the search for an integrated functional modeling framework across disciplines [89] there exists a **DSM-based framework** by Eisenbart et al. [136]. It has a flow-based functional representation at its center and use cases, actors, states with their transformations and interactions adjunct. This is comparable to modeling in SysML with ACT, UC, STM, SD and cross-cutting relations. While there are advantages and disadvantages with respect to readability on both sides, SysML has advantages with its mature tool support, potential for object-oriented reuse and extendibility, which come with a higher learning effort and needed guidance [137].

Considering **approaches that integrate behavior simulation** there is first of all the work of Wan et al. [138]. It automates the mapping between functions and Amesim simulation model components for direct concept creation and simulation. Each simulation element from Amesim defines both structure and behavior. Since all functions are allocated to a single viewpoint of the problem, provided by the used library, the results are not generic and potentially biased [138]. Also, the connectivity of the assembled Amesim simulation elements is only checked with respect to the interface types and not the underlying causality. This aspect of ensuring valid causality is, for instance, achieved by Münzer and Shea in [139], where simulation models are automatically generated based on concept model graphs.

For the **combination of SysML with simulation** besides the integral PAR diagrams with interlinked solvers, there exists the SysML-Modelica transformation specification [140]. It enables the definition of Modelica [31] models for simulation, directly in SysML. This enables powerful simulation capabilities by laborious recreations of the purpose-built simulation modeling language Modelica within the generic graphical modeling language SysML. Another concept design approach that combines simulation with SysML is the logic-based approach by Kerzhner [141] for decision making. It defines problem-specific DSLs for capturing design synthesis knowledge and transforms the more compact SysML representation into a mathematical programming problem for solving. One of its limitations comes from the problem-specific DSL, which results in problem-specific knowledge bases with complex simulation knowledge to be created for each different problem anew. Another limitation is the computational scalability of the simulation, similar to the logical synthesis of concept model graphs in [139]. Because identical architectures can be described by different sets of binary variables, the solver is forced to either search through a large number of identical architectures or intelligently identify existing symmetries.

To conclude, considering the related design approaches, the following **research gap is identified**: A seamless concept design approach for complex multi-disciplinary systems, supported through the integration of generic and proven design knowledge for reuse together with a direct mapping to behavior simulation for concept evaluation. Within this it is important to focus on a concrete and applicable modeling approach that provides extra guidance for SysML modeling for concept design and build on its standardization and formality.

3. Case Study

To validate and evaluate the conducted research different approaches are followed. The validation and evaluation involves checking that the developed design support does address the planned requirements. The general modeling approach with its developed model libraries is validated theoretically [142]. The theoretical validation is based on a case study and tests the general applicability of the developed libraries, patterns and their modeling approach. The used case study is a concept model of a fused-deposition modeling (FDM) based 3D printer. A comparison to known benchmark problems is not used due to the lack of suitable standardized benchmarks in model-based systems engineering. The used modeling tool is Magicdraw v18.1 from NoMagic, Inc. [65] together with its SysML plugin. For the usefulness and usability, the function library as one central part of the approach there is additional experimental validation in form of a user study described in Section 5.

3.1. The Reprap 3D Printer

The 3D printer model used for the case study is based on the **Reprap project** [143], the self-replicating rapid prototyping machine that is intended to be capable of producing all its mechanical parts, except machine elements, by itself. Although this goal of self-replication is not focused on in the case study, the Reprap project nevertheless started a whole community of comparably cheap and simple FDM based 3D printers. This is possible due to the open source GNU Public License (GPL) that freely allows interested people to participate and exchange information.

The underlying principle, the **FDM process** is an additive manufacturing method that produces parts by laying down material in layers. An example FDM based 3D printer set-up is shown in Figure 10. It has a print head that melts a

usually plastic filament and moves two-dimensional above the print plate with the printed part, moving along the third axis. This vertical axis movement creates the individual layers that constitute the part. Common build materials for FDM based printers are acrylonitrile butadiene styrene (ABS) or polylactic acid (PLA).

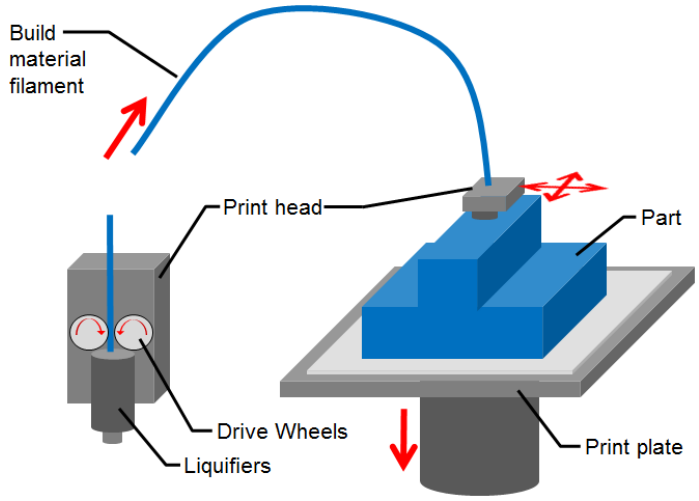


Figure 10: FDM based 3D printer set-up

One example for a 3D printer based on the Reprap project is the **Raptype** [144] from Figure 11. It is developed as part of a student project supervised by the author. Its main goal is it to achieve a high printing speed by reducing the moved mass at the print head. The print head moves on top of linear rails, driven by two stationary motors and two belts in the CoreXY [145] configuration. More information about the CoreXY configuration is given in Section 3.2.

The 3D printer case study was selected for several reasons. First, there is plenty of information available due to the open source community as well as the Raptype student project with its documentation and prototype. Second, it is a multi-disciplinary mechatronic system with many possible variations in its mechanical structure, electrical propulsion and control software. This is needed to show the capabilities of the modeling approach in SysML. Third, there are

plenty of possibilities to extend a 3D printer system into a PSS by offering additional services. Especially maintenance and spare part supply services are often used by industry to improve and ensure a printer's reliability. Finally, as a device it is comparatively simple and wide-known, unlike other systems that were modeled by the author using the approach in this thesis but not selected: the hydrokeratome [2] or the electric car [3, 4].

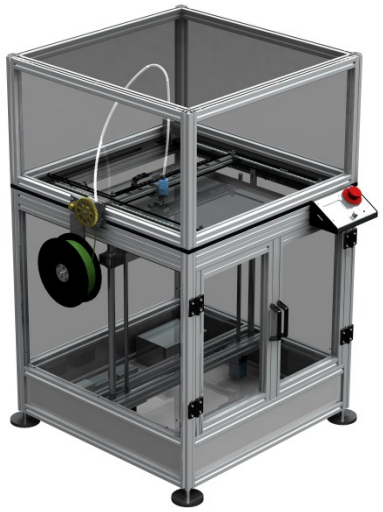


Figure 11: Raptype 3D printer visualization [144]

3.2. 3D Printer Concept Model

The **scope of the 3D printer model** used as a case study is defined as follows: Only the conceptual design of the printer hardware together with accompanying services is partially modeled in SysML. The model is decomposed down to a level where elements from the design libraries are used. Details like single screws, washers and the open source software are left out for simplicity. The modeled information is also not complete and focuses on aspects relevant for the validation of the approach. SysML diagrams in general show only certain highlighted aspects of the model. More model details follow with the description of the modeling approach in Section 4.

An **overview of the concept model** is given in Figure 12 with an excerpt of the model's main *package* structure PKG. Printer variations are modeled according to the model structure of Weilkiens et al. [59]. There are the *packages* for the different configurations and their variations, i.e. those system elements that can differ between the selected configurations. The *package* with the common elements contains elements that are the same for all printer configurations. The integrated design libraries are shown on top of Figure 12.

Based on the levels of concretization in [80] there are different **levels of abstraction** in the model with *packages* for the context level with its *requirements*, system context and *use cases*, the functional level, the principle solution level and the technical solution level for the more physical concretization of the previous principles and functions.

The included **variations** contain alternatives for the printer kinematics, propulsion with position sensors and control. This is shown in Figure 13 together with the variations *package* structure. For the kinematics there are two different variants modeled, which both realize the two-dimensional positioning of the print head. Both variants use belts that are driven by stationary motors to move the print head on top of a sliding carriage by means of linear bearings. There is the HBot [146] design and the CoreXY [145] design. The simpler HBot uses only a single belt but has a asymmetric load on its print head. The similar but more complex CoreXY uses two crossing belts but has a better balance of the forces on the print head. The two designs are schematically displayed in Figure 14.

The propulsion is either realized by stepper motors or continuous DC (direct current) motors, which need different types of control. The continuous DC motors need additional position sensors, either integrated into the linear rails of the print head or into the DC motors, making them into servo motors. In Figure 13 there is a "Constraints" *package* for these constrains between the variants.

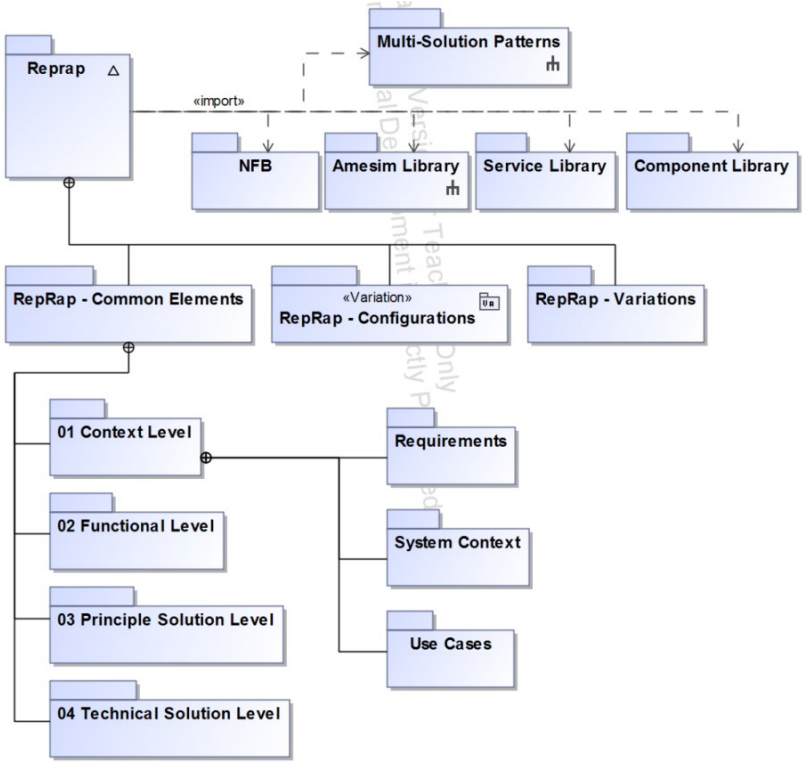


Figure 12: Model overview of case study Reprap

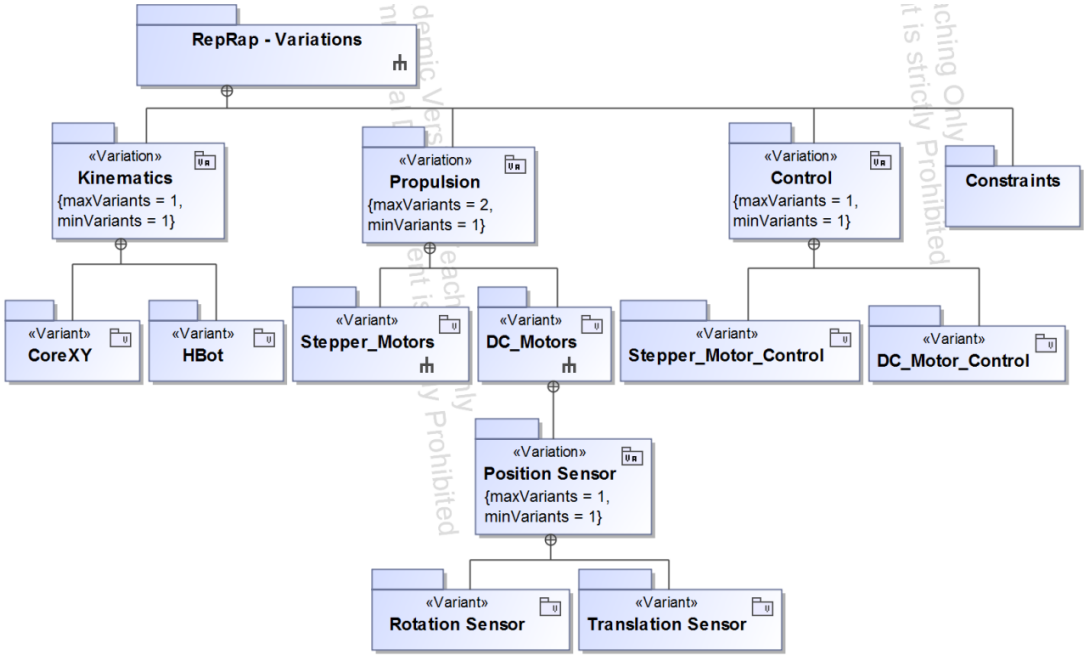


Figure 13: Reprap variations structure

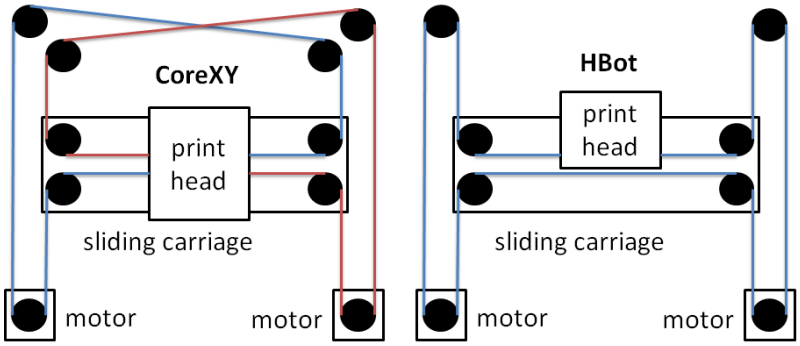


Figure 14: CoreXY (left) and HBot (right) kinematic schemas [1]

Not all possible **configurations** of the variants are modeled but only excerpts of three of them, displayed in Figure 15. There is the “Reprap Basic” redefining its inherited properties with the “HBot Kinematics”, three “Stepper motor” for the propulsion and no position sensor. The “Raptype Servo” uses the “CoreXY Kinematics” with DC motors and rotation sensors. The “Raptype DC+Rails” finally uses the “CoreXY Kinematics” together with DC motors and position sensors on the linear rails.

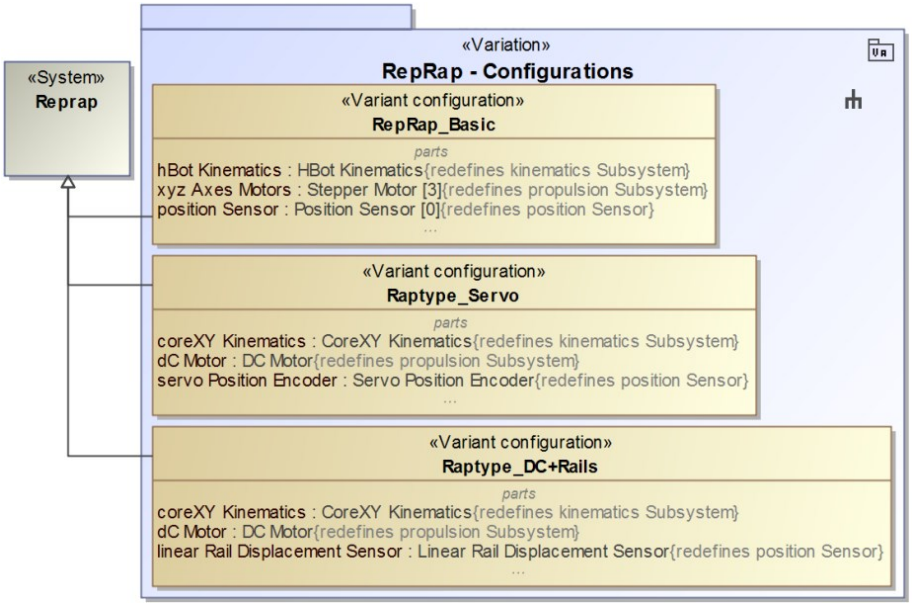


Figure 15: Modeled Reprap configurations

4. Concept Modeling Approach in SysML

This section presents the concept modeling approach in SysML. It includes design libraries and multi-solution patterns for the conceptual design of multi-disciplinary systems. First an overview of the approach is given, followed by the function library, behavior simulation library, service library, multi-solution patterns and the results of their use for the case study. For the libraries and patterns they are each first defined before their use is explained in the case study. The additionally used structure library [13] is introduced in Section 2.5.2.

4.1. Concept Modeling Approach Overview

The general concept modeling approach presented is based on MBSE and engineering design principles. For the **main tasks of concept design**, there is for example the VDI 2221 [14]. These tasks are the task definition and clarification, the determination of functions and their structure and the search for solution principles. The potential to use the MBSE modeling language SysML for these tasks is shown in [34]. It also introduces knowledge bases for SysML modeling as well as possibilities for computational support.

The modeling approach focuses explicitly on conceptual design. Yet, to relate it towards its broader context of a complete development process the **V-model** is used. In particular it is based on the V-model from the VDI 2206 [79], adapted by Eigner et al. [64, 80]. With its focus on multi-disciplinary mechatronic systems it also has potential for the development of PSS, as shown in [29]. There it is argued that domain- and solution-neutral functional modeling is critical for a successful application of the V-model during mechatronic and PSS development, since the general problem of allocating functions to suitable solutions is similar. The used adapted version of the V-model is shown in Figure 16. It is used for a comprehensive system description to enable a “model-based and structured

system description on the left wing of the ‘V’ in the early design phases” [64]. It starts with context and requirement modeling and specification, goes over to concept modeling and first simulation before the discipline-specific modeling at the bottom. The involved disciplines are not only mechanics, electronics and software, but also include service modeling.

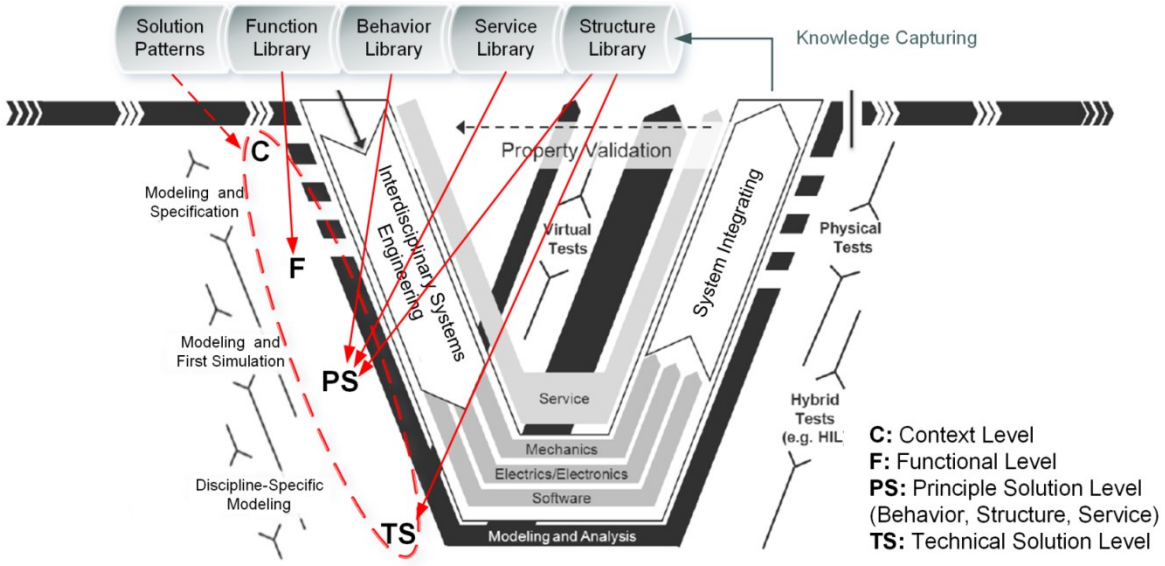


Figure 16: Libraries and patterns in the context of the adapted V-model (based on VDI 2206 [79], adapted from Eigner et al. [64])

The reason for choosing the V-model is to highlight the **knowledge capturing** of the libraries developed in this work and shown on top of Figure 16. There is the function library for functional modeling, the behavior, structure and service libraries for realizing the functions and the solution patterns to document known solutions that incorporate and link various model elements. The reused knowledge, especially in the patterns, comes from proven solutions from previous developed projects, where the models are validated during the right wing of the V-model.

When looking into the concept modeling approach in more detail the interdisciplinary model-based design approach for developing cyber-physical and mechatronic systems [80] is used for its **levels of abstraction** and the consideration of SysML. The model levels are the context level, the functional level, principle solution level and technical solution level. The use of the four level structure to set up the model framework is illustrated in Figure 17. Their use for the SysML case study is shown in Figure 12. The modeling support provided here focuses on the functional and principle solution levels.

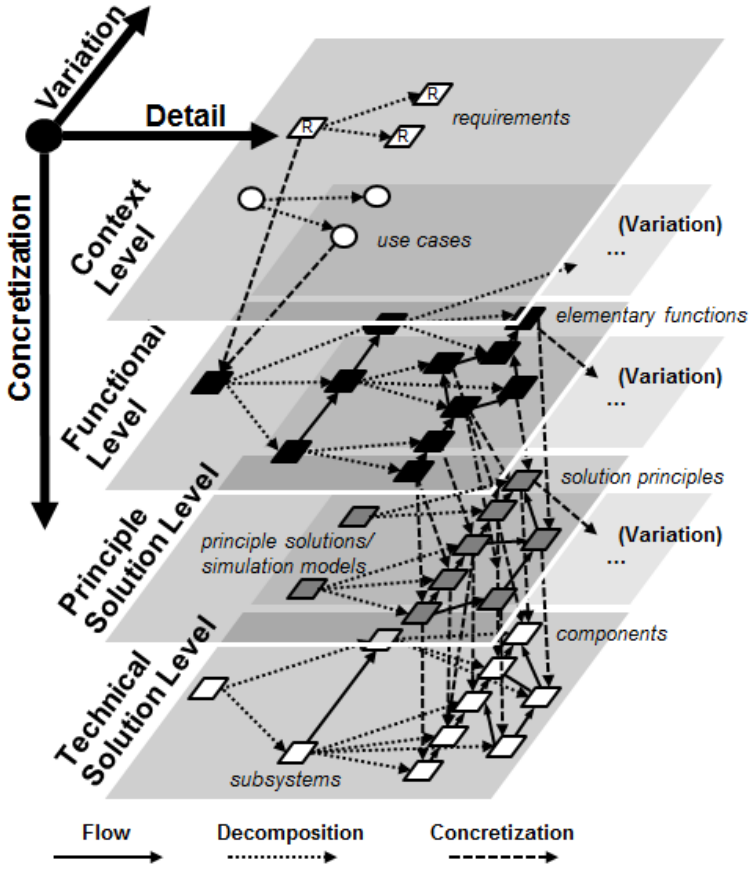


Figure 17: Level structure of model framework (based on [80, 82])

The **axes** on top define the solution space. The “detail” axis stands for the accumulation of information without explicitly restricting the possible solutions. When no further detailing is needed “concretization” takes place in form of a transition to a deeper level. In these transitions “variation” occurs, since multiple alternatives are to be considered during concretization [80]. To provide traceability throughout the model semantic links exist “vertically” between the hierarchical levels, as well as “horizontally” between elements of the same type.

The **context level** in SysML contains object-oriented *requirements* together with the *use cases* and the system context. The *use cases* define the system’s main functions and complement the functional model by offering an alternative and more informal functional representation, as established in a systematic comparison of functional modeling methods in SysML [2]. The system context, for example according on [59], helps identifying the system boundary, its interfaces and interactions to other systems or humans.

Based on this information on the context level, the system’s main functions are defined as black boxes with interfaces for the **functional level**. The functional level’s aim is a solution-neutral decomposition of these main functions. Its resulting functional structures consist of networks of elementary functions and material, signal and energy flows from the function library.

The **principle solution level** is for the systematic identification of possible solution variants, based on different solution principles or working principles. Possible solution principles to fulfill functions are provided with the behavior simulation, structure and service libraries.

The principle solutions are further concretized on the **technical solution level.** This level contains the most concrete representation of the conceptual solution. It describes an abstract solution concept by basic system components to realize the system functions and behavior by applying the principle solution

[80]. For this, it utilizes elements from the structure library [13]. Besides the further concretization it also includes initial concept analysis for evaluation compared to the *requirements*. The system evaluation is supported through concept simulation, planned with the behavior simulation library.

The general **modeling workflow** is displayed in Figure 18. After the initial task definition the requirements engineering takes place to clarify and define the development task. This results in the requirements model and *use cases* of the context level. Starting from a main function, defined as a black box with flow-based interfaces, the functional decomposition takes place next to create a functional model. This step is supported by the function library with its functions and flows. The search for working principles and their combinations is about identifying principle solutions to fulfill the decomposed elementary functions of the functional model. The principle solutions are detailed on the technical solution level by structure, behavior and service models, each supported by a generic design library. The behavior model also supports the model simulation in Amesim for an initial concept evaluation. Solution pattern are alternatively applied to fitting target functions and offer proven solutions in the form of partial models covering multiple levels. Collaboration between the different involved disciplines is necessary to reach a valid solution concept model. The design process usually requires multiple iterations before the concept is fully elaborated in the form of a SysML concept model including Amesim simulation models.

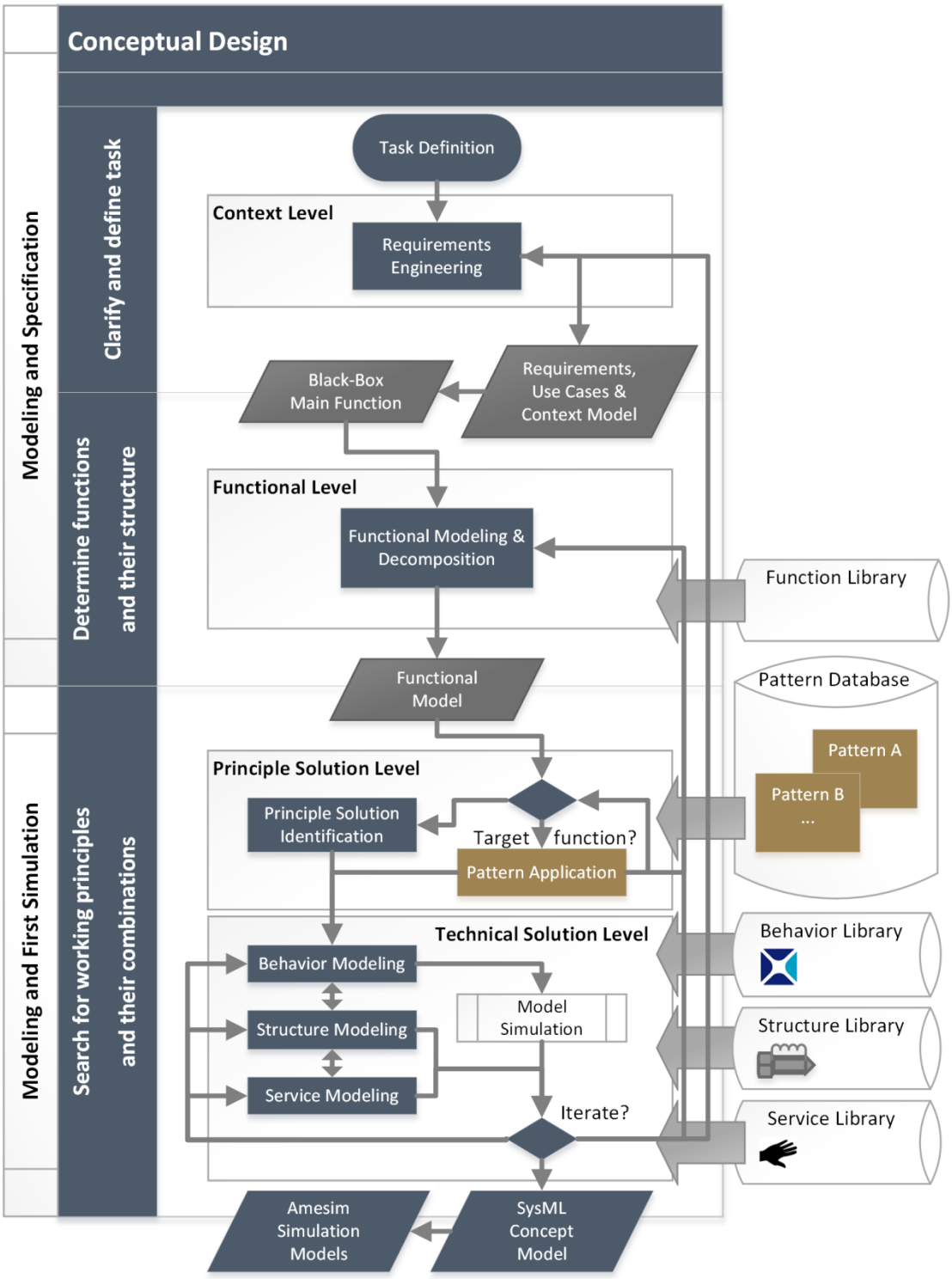


Figure 18: Modeling workflow schema

4.2. Function Library

This section describes the definition and refinement of the SysML function library. Afterwards its use is demonstrated by the case study. The **needs** addressed by the library include modeling guidelines for functional modeling together with improved model formality for better model consistency and avoidance of modeling errors or ambiguity. This way it aims to improve modeling in SysML through enabling a simple reuse of standardized model elements. The library contains the functions and flows of the FB to be combined on SysML ACT.

4.2.1. Function Library Definition and Refinement

Based on the initial definition of the function library by Wölkl [13], its definition in SysML and further refinement in this work is now presented. The function library contains functions and corresponding flows, both based on the FB [15]. This allows an operator-flow formulation of functional structures with the required verbs and nouns being semantically defined in the FB hierarchies.

Two possible ways exist to **incorporate such taxonomies** of functions and flows in SysML [13]. First, it is possible to multiply the functions by the flows, which is useful when the meaning of a function changes depending on the involved flows. However, this also means that there would be a large number of entries in such a library and that updating the library becomes difficult. For example, with the change propagation of one changed flow type through many different functions. The second possible way to model the FB in SysML is by defining two separated parts, one for the functions and the other for the flows. These two parts are then used combined together. This way the number of library elements is limited to only the 45 flow definitions and the 52 function definitions, to be combined during modeling to create elementary functions. Such a setup with less elements also simplifies the effort for searching for certain functions and flows.

An overview over the defined **stereotypes in the function library** that extend SysML, is given in Figure 19. There are the <<BasicFlow>> and the <<ElementaryFunction>> *stereotypes* with their *tagged value* properties. The <<BasicFlow>> *stereotype* is specialized from the SysML *block* and has an additional *tagged value* for the level in the FB hierarchy [13]. The additional *stereotype* <<User-definedFunction>> is for all functions that are not elementary. User-defined functions can be decomposed further into other user-defined functions or elementary functions. This is shown in Figure 19 along the *directed associations* between the elements stating that each <<User-definedFunction>> *activity* can be decomposed by any number of <<ElementaryFunction>> or <<User-definedFunction>> *activities*.

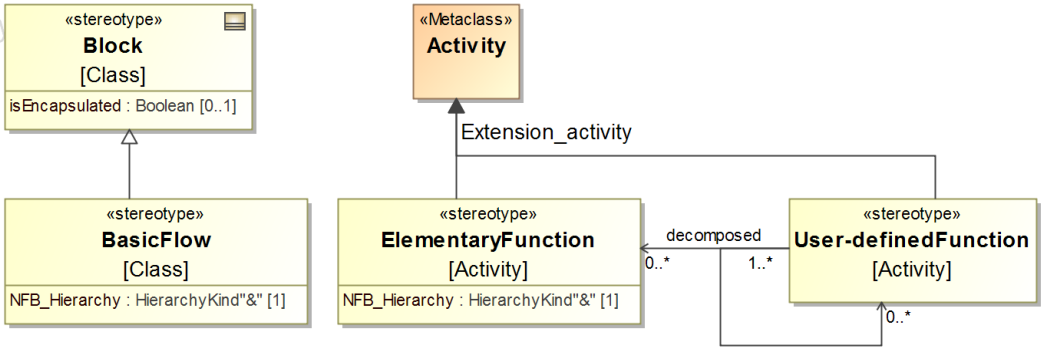


Figure 19: Defined *stereotypes* in function library

The first step of transforming the paper-based three level hierarchies of the FB terms into formal libraries in SysML is the creation of analog structures. The **derivation of flow elements** is displayed in Figure 20 showing the *stereotype* <<BasicFlow>> for the newly created *blocks*. The <<BasicFlow>> elements are created matching to the flow terms from the FB [15], as displayed on the top right corner of Figure 20. According to their hierarchical position, the flow elements are arranged in a tree structure with *generalizations* between

them. This means for example that the “Solid” flow is a kind of “Material” flow that also inherits the properties of the “Material” flow. The “RootFlow” element on top of Figure 20 is added to provide a parent element for all flows. It is used for the most general, i.e. not limited *parameter nodes* in the definition of the functions.

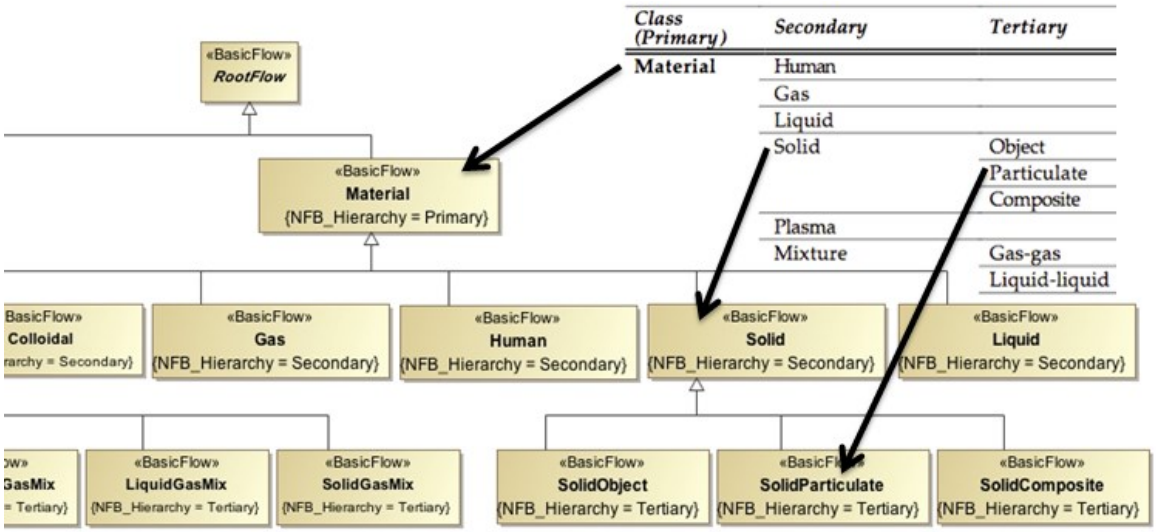


Figure 20: Transformation of some FB flows [15] into the SysML function library hierarchy

Since <<BasicFlow>> elements are specialized *blocks* they can have various properties. As shown in Figure 21, the energy flows have additional *value properties* for **“effort” and “flow” variables**. These effort and flow terms come from bond graph theory [107] as power conjugate complements from Section 2.5.2. The “ElectricalEnergy” flow for example inherits the two properties while redefining them into “electromotive force” and “current” to describe itself.

The implementation of the library allows the **extension of the FB terms** by more task specific ones. This is especially of value for the flow types, due to their identified benefits for increased expressiveness and reduced ambiguity [101]. An example for modeling mechatronic systems is given in Figure 21 with

the extension of the electrical energy flow into one that provides alternating current (AC) and another one for DC. Another example for mechatronic systems is the introduction of more specific signal flows, as shown in [4].

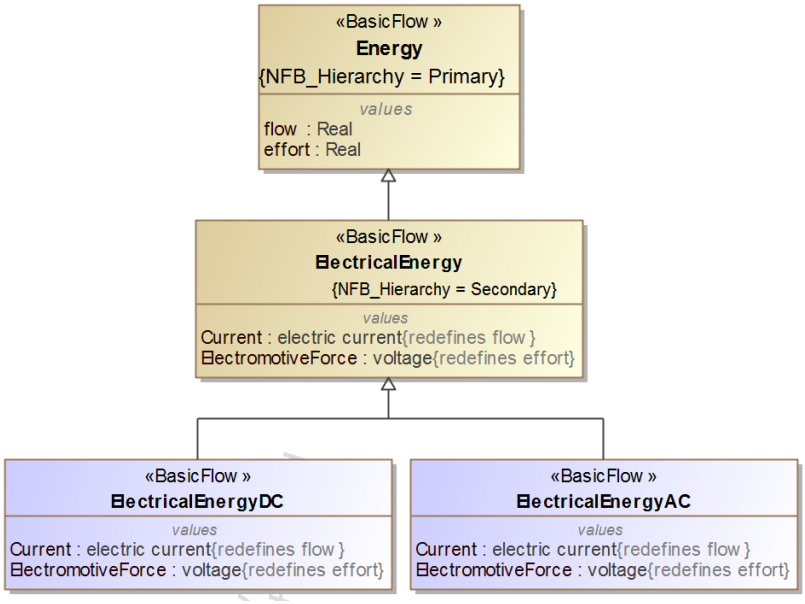


Figure 21: "ElectricalEnergy" flow with redefined effort and flow parameters as well as custom extensions for DC and AC

Analog to the flows, there is a hierarchy structure created for the **function elements** [13], using the <<ElementaryFunction>> *stereotype*. The functions inherit properties from each other, which in case of their interfaces are often redefined to be more specific. The interfaces are defined using *activity parameter nodes*. They have defined types and directions to represent input and output *parameters* that enter or leave a function. The interfaces are not only defined for the functions' main flows, but also for additional auxiliary flows to model supporting flows that enable the interaction of the function with its main flows.

Having all this **information formally specified** in interfaces is a major difference towards the original FB, where such information is only partially and

implicitly given in the function descriptions. Also the addition of auxiliary flows is an extension compared to the originally implemented library [13]. They broaden the modeling freedom by allowing accompanying flows, e.g. to model the needed electrical energy flow for processing a signal flow. For this they are specified with a *multiplicity* of "*0..**"in the library, which means that no such flow is explicitly required.

An **example of the definition of two functions** is given in Figure 22. It shows the functions "Regulate" and "Change" with differently defined interfaces derived from their descriptions. "Regulate" for example allows any material, signal or energy flow to be regulated according to a specific and necessary "SignalFlow". The similar "Change" function changes the flow in a "predetermined

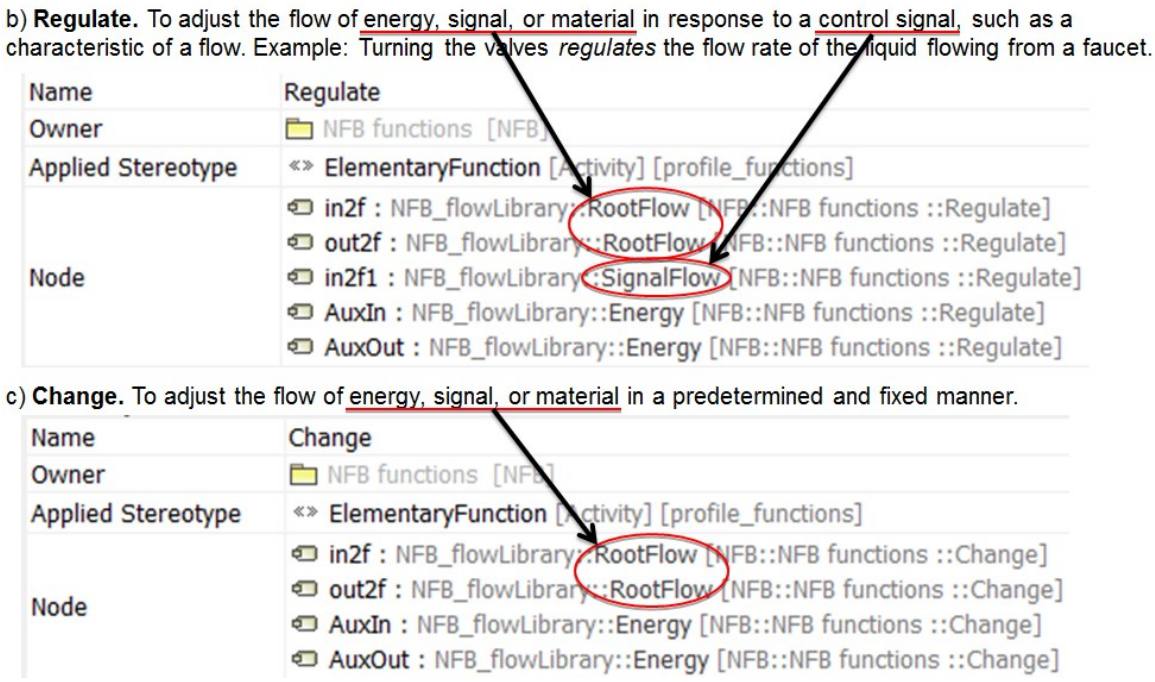


Figure 22: Definition of functions [15] implemented in a SysML library with their inputs and outputs

and fixed manner" [15] and needs no such signal input. The textual description is also added into the library as the *activities'* documentation to make this knowledge available during the modeling process.

Another **improvement of the function library** compared to its initial configuration by Wölkl [13] is its set-up with additional *packages* and *containments*. This change resulted directly from the feedback of the user study from Section 5. The new library set-up is displayed in Figure 23. Flow *blocks* are contained within each other on the right, while function *activities* on the left are contained in nested *packages* with identical names to their superior functions. The function "Regulate" for example is contained in the *package* "Control_ Magnitude", which is its more generic parent function. The extra packages are used to avoid inconsistency when *activities* contain each other without graphical representation or even intention [69]. This new set-up improves identification of related elements across the hierarchical levels in the modeling tool.

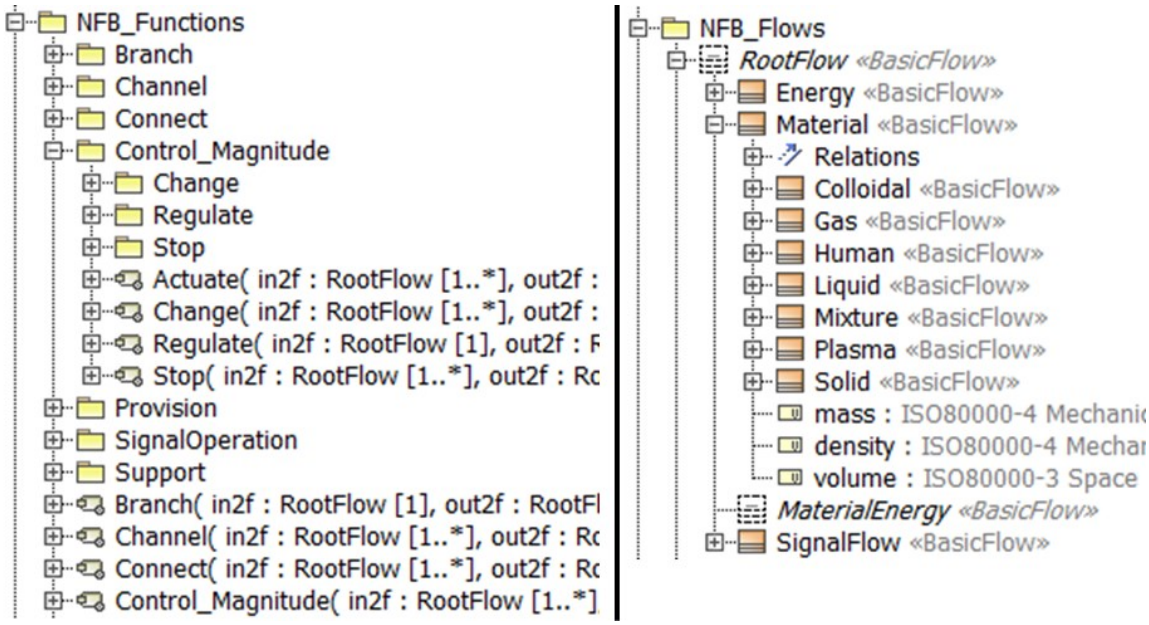


Figure 23: Function library containment (left: functions, right: flows)

To support modeling with the library, two **OCL constraints** (Object Constraint Language) are included. They enable automatic checks to determine whether the created functions are named and if their interface types are more specified than by the default “RootFlow”, which has no concrete meaning.

4.2.2. Function Library Usage

The workflow of using of the function library [4] is shown here by the case study. For the functional modeling, first the system’s **main function is defined** as a black box [23] in the form of a SysML *activity.* It is created as a user-defined function with the system’s inputs and outputs, which have energy, signal or material flow types from the library. The information from the task clarification step of the context level is used for this: the main *use cases* and the system context [2, 59]. For the 3D printer case study this step is visualized in Figure 24.

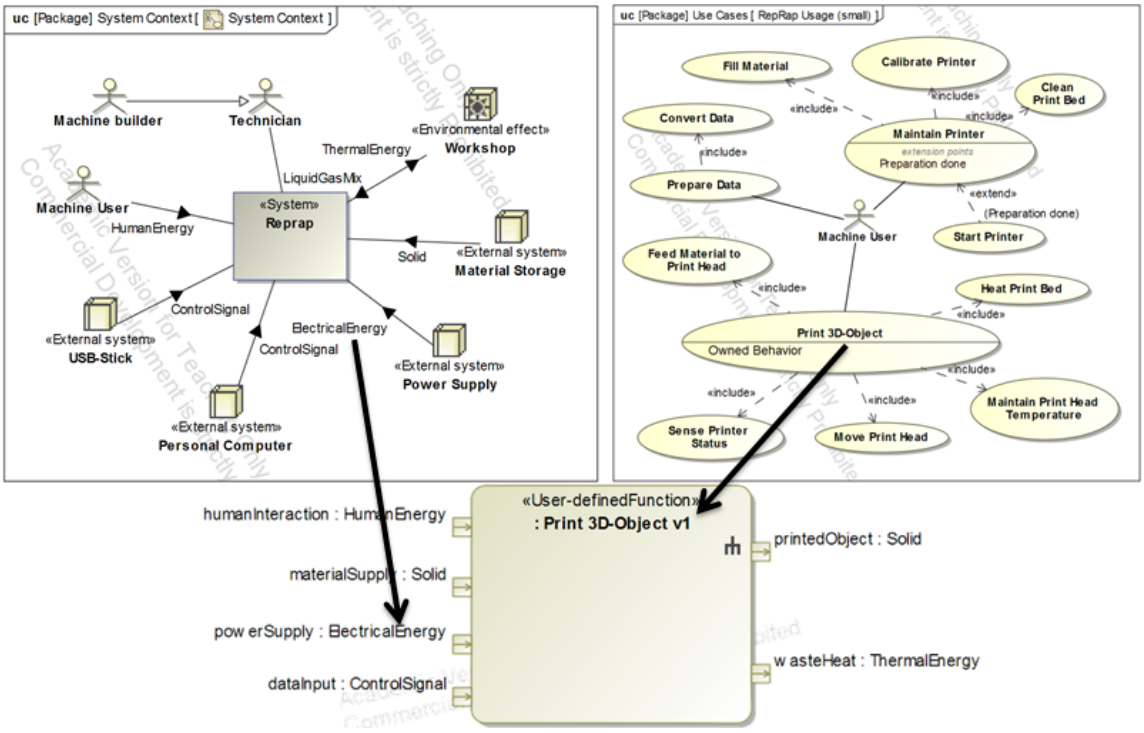


Figure 24: Derivation of main function "Print 3D-Object" of case study

The use case “Print 3D-Object” on the right is used to create the user-defined function “Print 3D-Object” with the inputs and outputs coming from the flows across the system boundary of the context diagram on the left.

This **main function is then decomposed** on its ACT. It is modeled as a network of sub-functions using elementary functions from the library or other nested user-defined functions. The user-defined functions are further decomposed on their own ACTs. This decomposition process follows the general rules of a verb and noun based representation along its energy, material and signal flows, as presented in [23]. It is for example recommended to follow the system’s main flow first during the decomposition and add supporting flows later. More detailed rules for the functional decomposition are presented in [147, 148]. With this approach using *activities* in SysML, the nodes on the diagrams are *actions*, while the edges for the flows between the functions are *object flows*. The a*ctions* themselves cannot be stored in a model library. Consequently, all used *actions* are *call behavior actions*, which refer either to a user-defined function or an elementary function from the library.

An **example elementary function** is given in Figure 25 together with its used library elements. First an unspecified *action* is created, which then is transformed into a *call behavior action* by assigning a type in the form of an *activity*. Here it is the *activity* “Regulate” from the function library, transforming the *action* into an elementary function. This *call behavior action* is then assigned its name and specific flow types. Here it is named “ElectricalEnergy” to describe what the function is doing by relating to its main flow. To specify this main flow the flow types are refined by suitable subtypes. The possible flow types are defined in the library *parameters* and *activity parameter nodes*, as seen on top of the figure. Here the generic input and output “RootFlow” *pins* are specified as “ElectricalEnergy”, which is regulated by a “ControlSignal” flow. The descriptions

of the optional and unused auxiliary input and output *pins* are finally removed from the diagram for improved clarity. Removing their *pins* altogether would result in a model inconsistency between the *pins* of the *call behavior action* and the *activity parameter nodes* defined in the called *activity*.

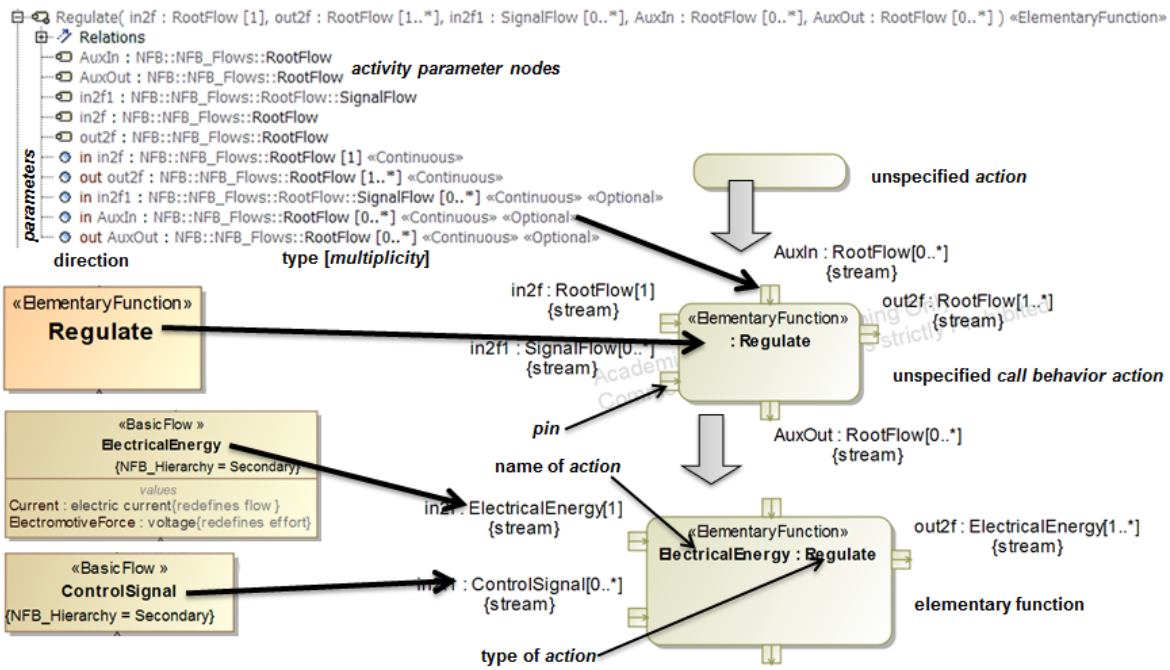


Figure 25: Function library (left) used to define "ElectricalEnergy:Regulate" function as *call behavior action* (right)

This way the auxiliary **predefined *pins*** help at reducing the danger of missing important auxiliary flows, while being mandatory in their usage. With the *pins* it is also possible to rename them to add more details about the flow. The free naming of *action* and *pin* names further increases the information content of the model for improved expressiveness and understanding through provided context [98, 101]. If the type or number of the elementary function's predefined *pins* is not sufficient, for instance when both the flow of pneumatic energy and the accompanying flow of air are needed, there are two solutions. One possibility

is to use user-defined functions instead of elementary ones. The other possibility is to make a custom extension to the library flows with a new subtype of two types of flows, which can then be used on a single *pin* of an elementary function.

Through the **functional decomposition**, the system functionality is described step-by-step in more detail. This makes the model more specific and implies already a particular type of solution, i.e. an FDM process, while in general remaining solution-neutral. During the functional modeling, different variations of functional decompositions are investigated to determine the most promising one. In parallel it is recommended to refine the requirements as well [56]. This iterative process of mutual refinement and decomposition helps in establishing traceability between the functions and the requirements and helps defining the functions “in terms of allocated functional, performance, and other limiting requirements” [48].

The decomposition is partially shown in Figure 26. The **top ACT** of the figure shows the main function “Print 3D-Object”. It contains five user-defined functions similar to the generic decomposition in [149]: “Store & Supply Material” for handling the printing material before its use, “Pattern Material” for preparing the printing material, i.e. melting it in an FDM process, and “Create Primitive” for the layer-wise positioning of the printing material into the targeted geometry. The functions “Provide Energy” and “Control Process” support this by providing the necessary energy and control of the printing process.

The **lower ACT** displays the functional decomposition of the user-defined function “Pattern Material” into elementary functions. They are: “Liquid:Position” for the three-dimensional positioning of the liquid printing material, “Liquid-Solid:Couple” for the coupling of the printing material with the printing table or previously printed material, “Liquid-Solid:Convert” for the solidification process, “Solid:Support” for the support of the printing material by the printing table, “Solid:Export” for manually removing the printed part from the printer and

“ElectricalEnergy-PneumaticEnergy:Convert” for additional forced convection that supports the solidification process. The elementary function “Liquid:Position” is framed to indicate that it is further decomposed and refined depending on the modeled variations introduced in Section 3.2. This happens by *refactoring* it when applying a solution pattern in Section 4.5. *Refactoring* means to restructure a code or model without changing its external behavior.

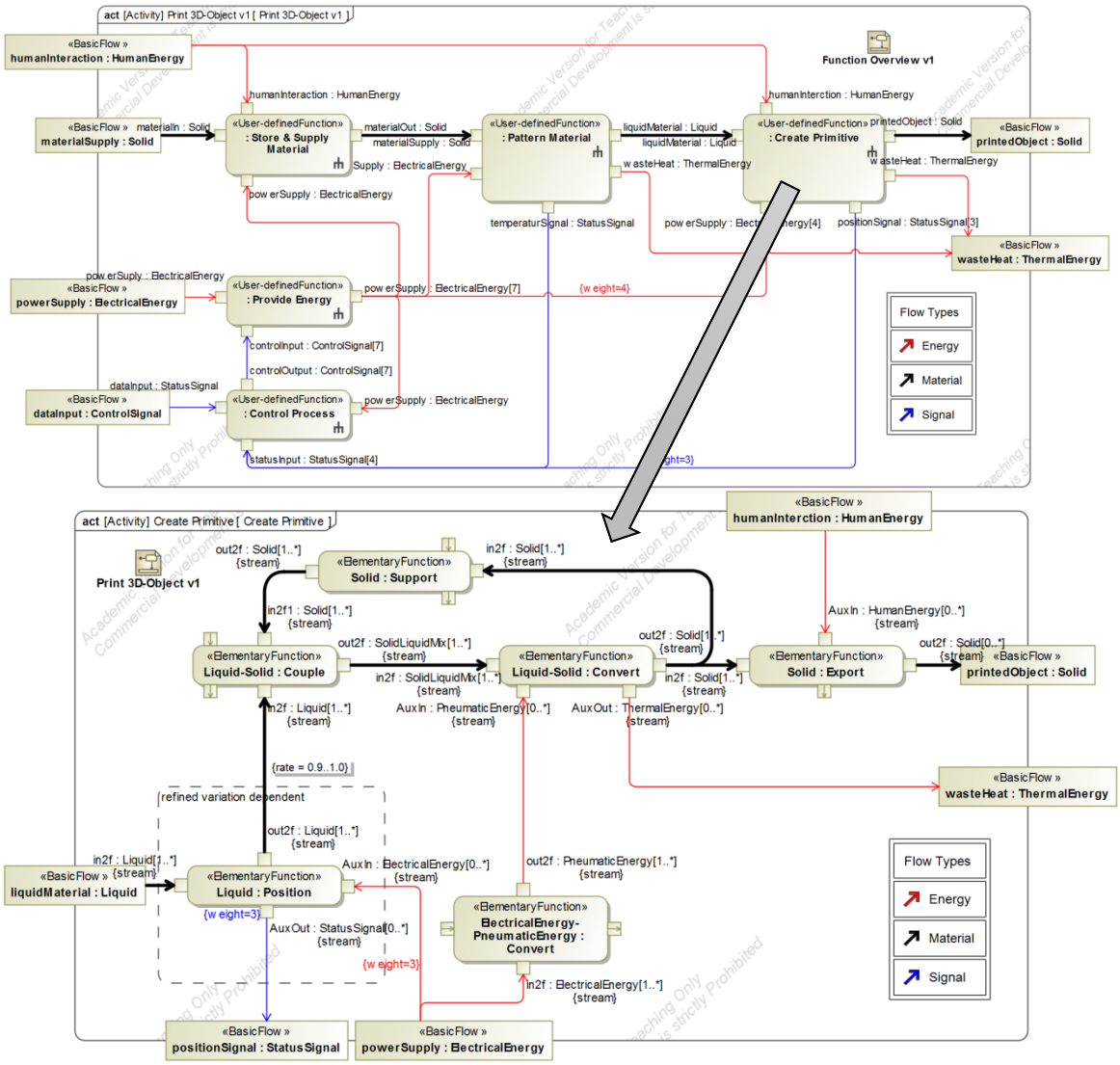


Figure 26: Excerpt of functional decomposition of "Print 3D-Object" main function

For the **object flows** between the *actions* it should be noted that the energy and signal flows to and from “Liquid:Position” have the *weight* of “3” to indicate that three flows of the same type are needed. Extending the library usage in [13] by Wölkl, this follows the definition in UML [69] to specify multiple required flows along an *edge*. Here they are needed for the three dimensions in which the positioning takes place. Similarly, it is also possible to assign flow rates to the edges. Here there is a flow rate between 90% and 100% defined for the positioned material flow after “Liquid:Position”.

The **functional decomposition process ends** when a satisfying level of detail is reached. This level is not necessarily the same for all parts of the system. Using the function library, this level is at the latest reached when only elementary functions are used, since they are not decomposed further. If they are not used, user-defined functions are directly *allocated* to suitable components or principle solutions instead. In general, the functional decomposition “takes place until useful [solutions] have been found” [150].

To add further information about sequencing or variations of the functions there exist **additional ACT modeling elements**. For example in Figure 27 of the case study the *control flow* includes a *decision node* for including or excluding a printer inspection function when doing maintenance that consists of calibrating the printer. This addition is in contrast to the original definition of the function library by Wölkl in [13] or the initial functional decomposition method by Pahl and Beitz [23] and goes into the direction of enhanced functional flow block diagrams (EFFBD) [151]. For more detailed sequential or system status knowledge there are additional STM or SD diagrams to be used. In Figure 27 there are also *swimlanes* used to set *allocations* to other modeling elements that provide the modeled functionality. Here they are used for an initial explicit partitioning [152] to define which functions are fulfilled by which domain, i.e. software or service.

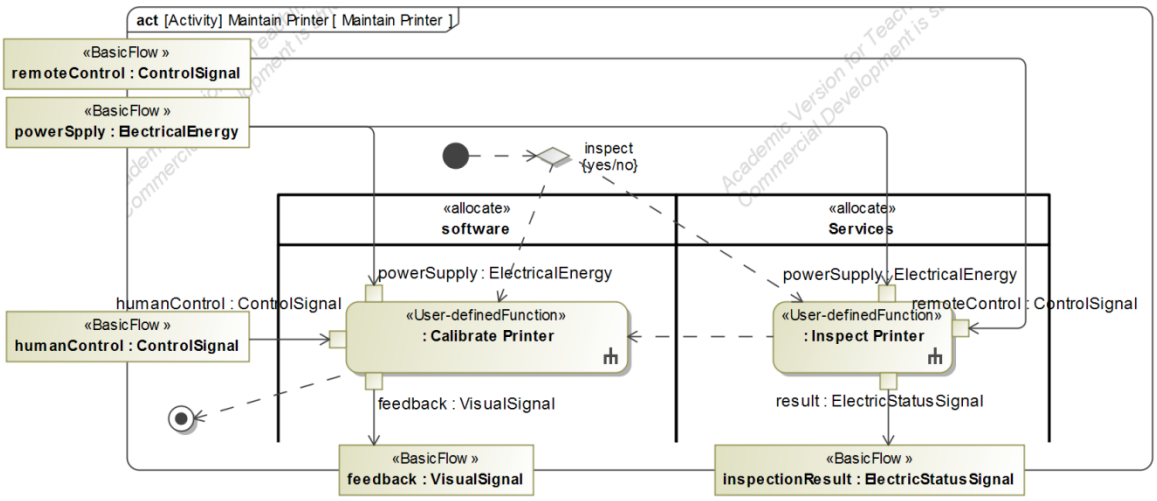


Figure 27: "Maintain Printer" function with *control flow* and *swimlanes*

Besides the two constraints that are defined in the library, there are some **modeling constraints** from the UML and SysML specifications that also provide assistance. They check in the modeling tool if the modeled *object flows* connect *pins* with compatible flow types. The consistency of the *object flows* is fulfilled when the offered flow is the same or a more specialized version of the required flow. This complies with the fact that a flow should not have different objects at its ends. Yet, it can be advantageous to allow more specialized subtypes of flows to be accepted anyways to allow different levels of detail. Examples are given in Figure 28: On top is an offered “Energy” flow connected to a *pin* that needs more specialized “ElectricalEnergy”, which causes an error message. In the middle is a valid flow of “Energy” with identical input and output *pin* types. On the bottom is an “ElectricalEnergy” output *pin* connected with a “Energy” input *pin*. This is also valid since the offered “ElectricalEnergy” is a subtype of “Energy” in the library and hence fulfills the required flow.

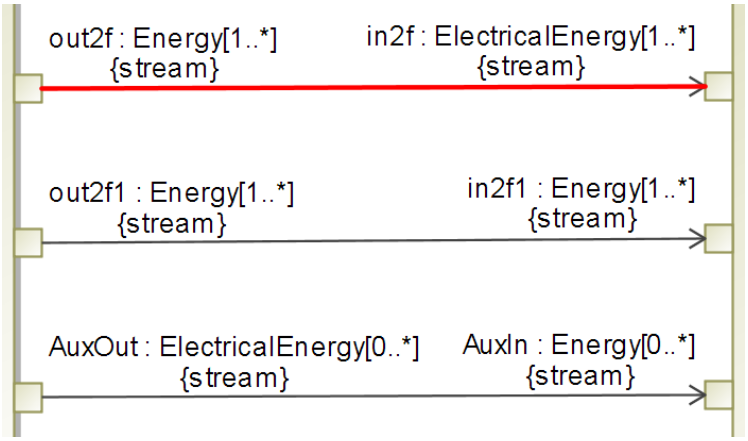


Figure 28: Function flow consistency examples

4.3. Behavior Simulation Library

Following the functional modeling, the search for solution principles and the later system behavior modeling to plan model simulation takes place. These processes are supported by the behavior simulation library [1] presented in this section with its implementation in SysML and its use with the case study.

The designer's **needs** addressed by this library are to support two roles of behavior models: First, the provision of design knowledge as principle solutions that provide and concretize the modeled functionalities. This includes modeling guidance through the provided *stereotypes*. Second, model simulation for early concept evaluation through the provision of the same design knowledge as found in tested and proven elements of simulation models. Due to the conceptual nature of the intended design with the behavior simulation library, its simulation models focus on the representation of expected behavior compared to structure behavior [83], even if properties are derived from the following structural model.

4.3.1. Behavior Simulation Library Definition

The library contains elements from the simulation tool LMS Imagine.Lab Amesim [16] together with *stereotypes* for generic principle solutions and their solution principles. These *stereotypes* defined in the library are shown in Figure 29. On top there is the **<<PrincipleSolution>> *stereotype*** to generally represent how *allocated* functions are realized in principle. It is based on the SysML extension MechML [153], where it is used for morphological matrixes. It is a subtype of the SysML *block* and has two attributes: A *string* typed attribute for its general description and a priority with a certain grade. The grade is an *enumeration* element with the *enumeration literals*: “perfect”, “good”, “satisfactory”, “fair”, “poor” and “fail”. The <<PrincipleSolution>> *stereotype* also has *associations* to *stereotypes* that can further refine and concretize the principle solution. In Figure 29 these are the <<SolutionPrinciple>> *stereotype* and the <<AmesimSimulationModel>> *stereotype*.

The **<<AmesimSimulationModel>> *stereotype*** represents whole Amesim simulation models in SysML, containing simulation elements and further data for the simulation. It is a subtype of the SysML *block*, too. For capturing the data to run simulations, the *stereotype* has multiple attributes defined. One example for an attribute is the “analysis_mode” with its *enumeration literals* “temporal” and “linear”. Other examples are “start_time”, “final_time” and “print_interval” in seconds. They define the analysis mode, runtime and step size as some major simulation parameters. The complete list of attributes and used enumerations, which are all derived from the settings in the simulation tool, is seen in Figure 29. The <<AmesimSimulationModel>> *stereotype* decomposes into two different element types: at least one <<AmesimBlock>> in the role of a simulation element and potential <<SolutionPrinciple>> elements in the role of placeholders, if no <<AmesimBlock>> element is known yet.

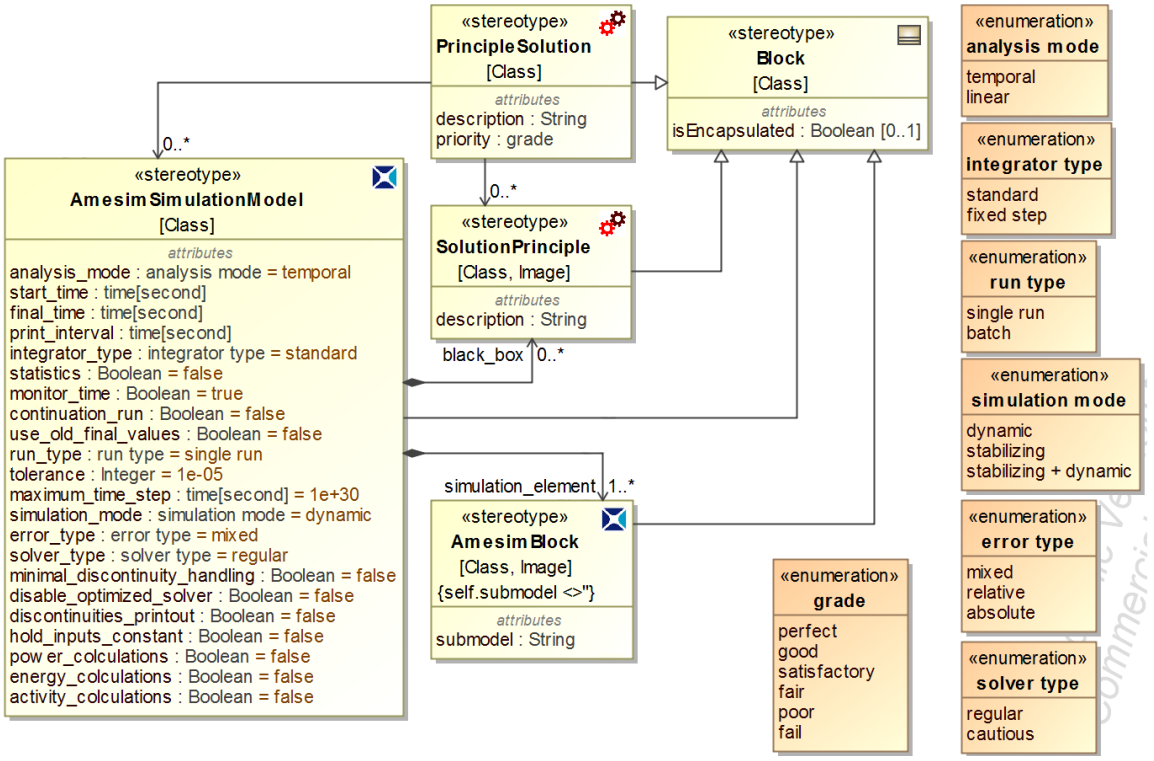


Figure 29: *Stereotypes* in the behavior simulation library

The **<<AmesimBlock>> *stereotype*** is another subtype of the SysML *block*. Additionally, it has the *metaclass* "*Image*", which allows it to be graphically represented by specific icons on SysML diagrams. The *stereotype* contains an attribute for its submodel specification, which is needed to uniquely identify specific elements for the simulation. For this there is an OCL constraint to ensure that a submodel is specified through the modeling tool's validation capability.

The **<<SolutionPrinciple>> *stereotype*** is again a subtype of the SysML *block* and has also the additional *metaclass* "*Image*". It has one attribute for a general description. It is used to represent general solution or working principles to further specify principle solution elements or work as a black box placeholder in simulation models when no fitting simulation elements are known or existing [153].

To contain these stereotypes and other content the behavior simulation library has the following **structure in SysML**. In contrast to the inheritance relations in the function library, the behavior simulation library does not have such global hierarchies. Instead it is structured into nested *package* structures for the library content, its profile, used *value types*, *port* types and super components. The *packages* follow the set-up of the database in the simulation tool for the simulation elements. This is illustrated in Figure 30. The *package* "Mechanical" for example contains all simple mechanical elements. It contains *packages* for e.g. "Rotational", "Translational" and "Transformer" elements. The "Rotational" *package* again contains *packages* for the modeling elements, e.g. "Inertia" for mechanical rotational inertia elements with different interfaces and properties.

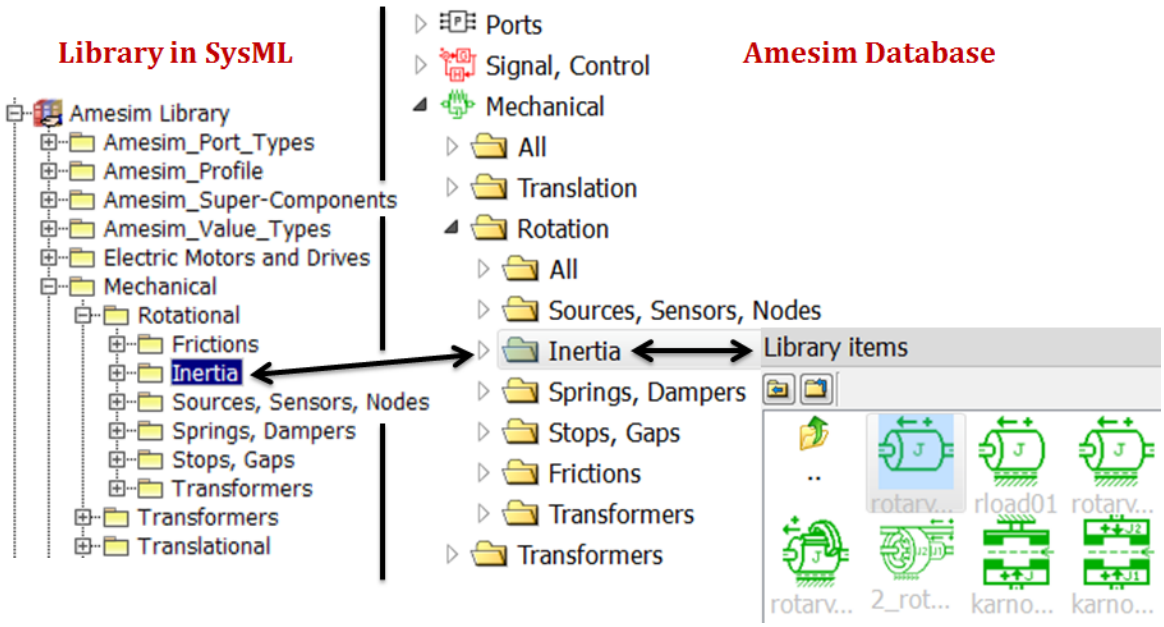


Figure 30: Behavior simulation library implementation in SysML (left) with corresponding database in Amesim (right)

An example for such a simulation element in SysML is given in Figure 31 with a rotary load element. On top is the rotary load element with two ports in the simulation tool database with its four different submodels: “RL02”, “RL03”, “RL02A” and “RL03A”. “RL02” is for a rotary load with two shafts without friction. “RL03” is the same but for the dynamics of a zero inertia. “RL02A” is also for a rotary load with two shafts without friction, but gives the angle as output, which is displayed on the top left of Figure 31. “RL03A” finally is the same as “RL02A” but again for the dynamics of a zero inertia.

Below are the equivalent **<<AmesimBlock>> elements** implemented in SysML together with an additional parent element. This parent element “rotaryload2” not only inherits its properties to all sub elements through *generalizations*, but can also be used in SysML when no specific submodel can be selected, yet. For the specific submodel selection there are the four sub elements with their redefined *ports* and additional properties. Each element in the library also has a textual description in SysML from the Amesim documentation and is assigned with the equivalent icon. The naming of the <<AmesimBlock>> elements follows the names of the elements in the Amesim database and not their often varying name in the tool documentation. Only to differentiate between submodels there are cases that need an addition coming from the documentation. Setting the names of the properties follows the documentation’s variable title instead of the variable name for again a better expressiveness.

The “rotaryload2” element of Figure 31 has certain **properties defined**. It has one constraint for documenting the mathematical formula for the calculation of the rotary acceleration from the elements internal properties and inputs. This equation comes from the Amesim documentation. In SysML it serves only to enhance the understanding of the inner workings of the element. The *value properties* of “rotaryload2” that are passed on to all four child elements are

“rotary acceleration” and “shaft speed port 2”. The property “shaft speed port 2” for instance has a type of “angular velocity[revolution per minute]” following the ISO 80000 [154] with a default value of “0” and a fixed <<interval>> from maximal “1.0E30” to minimal “-1.0E30”. The ISO 80000 [154] is an international standard for the international system of quantities in form of a style guide for the use of physical quantities and units of measurement.

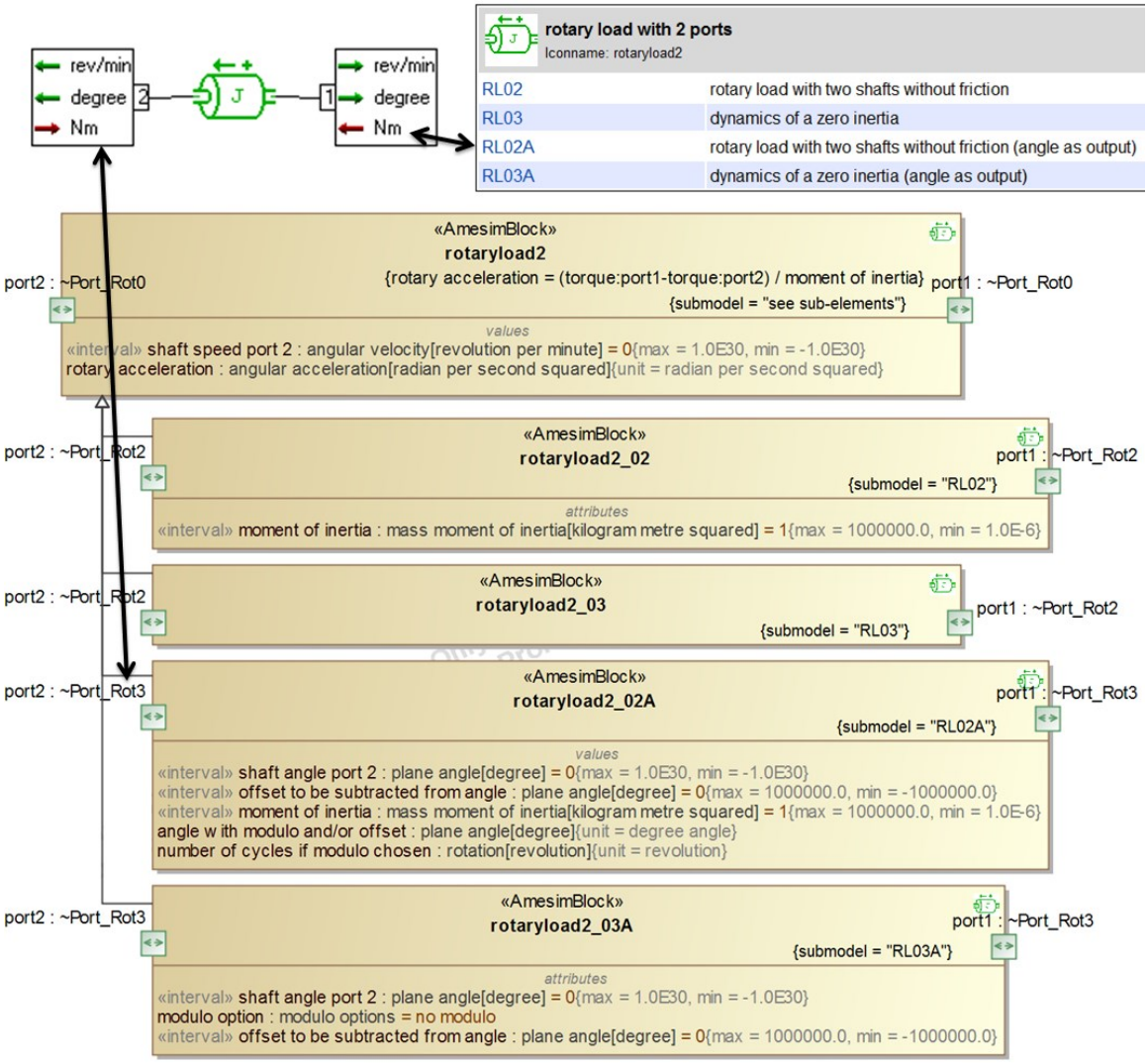


Figure 31: Rotary load elements in Amesim and their implementation in SysML with parent element without specified submodel

The **interfaces** of the rotating load elements in Figure 31 are modeled as *flow ports* with reusable types shown below in Figure 32. The figure shows an excerpt of the library with a hierarchy of *port* types for simple mechanical rotational energy transfer. The *port* types are modeled as SysML *blocks* and have *flow properties* for the transmitted values of the simulation. The *port* type "Port_Rot3" of Figure 32 is for instance used for the rotating load element of Figure 31. It has the following *flow properties*: the outgoing torque in newton meter, the incoming rotary velocity in revolutions per minute and the rotary angle in degree. In addition to the *port* type elements in Figure 32, there are also the corresponding interfaces visualizations from the Amesim database. To model opposite flow directions of library elements their flow ports are set to be *conjugated*, which is displayed in Figure 31.

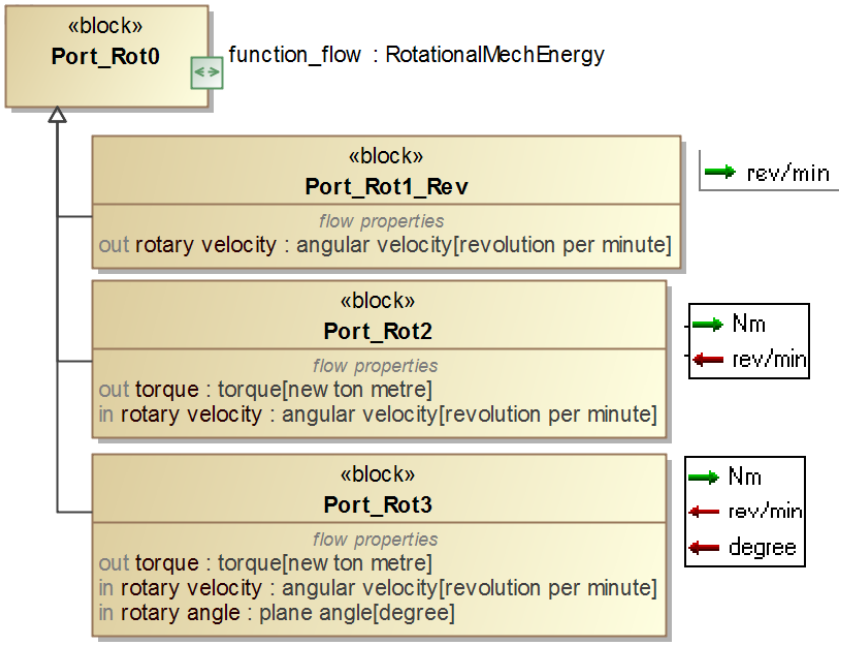


Figure 32: Excerpt of flow port type hierarchy defined in behavior simulation library

The *port* types are arranged in hierarchical relations to inherit an additional nested **function *flow port*** from their most generic parent element. This unspecific parent element without *flow properties* is used for parent <<AmesimBlock>> elements without concrete submodels, as seen in Figure 31. The nested *flow port* has a function flow element from the function library as its type. The function flows are derived from the interfaces of the Amesim simulation element, similar to [138]. The nested *port* with its function flow is there to support an identification of suitable behavior simulation elements by corresponding to the flows of functional models. Flows from the functional model can also have further properties, e.g. their effort and flow parameters of Figure 21, to be referred to in the behavior model.

Although the *flow port* is considered to be deprecated in the current version 1.4 of SysML [11] it is still used here. Because using ***flow ports* has the advantage** of better port compatibility checking compared to standard *ports*. For standard *ports* with normal *connectors*, the compatibility is not ensured to a satisfying degree unless special *binding connectors* are used. Yet, these *binding connectors* only allow identical *port* types to be connected, which is usually not the case within Amesim simulation models. *Flow ports* on the other hand are checked for at least one matching *flow property* with matching type and direction. This complies much better to the modeling of Amesim simulation models. The port compatibility checking is also the reason for the nested function *flow port* instead of an additional *flow property* with an equivalent type from the function library. An additional function *flow property* would result in incompatible ports not being identified, if only their generic function flow, e.g. mechanical energy, fits.

Another **investigated alternative** is the use of nested *ports* for all of the transmitted values instead of *flow properties.* Here, compatibility checking does not take place unless all used nested *ports* are connected, which results in a

significant higher modeling effort and bloated diagrams. Also the *flow properties* allow a better implementation of parent port type elements with unspecified port directions for parent AmesimBlock elements without a specific submodel. *Flow properties* further allow the use of standardized units from the ISO 80000 profile and they can have set *default values* when a parameter is fixed by the simulation element.

When creating AmesimBlock elements from the tool database, several other aspects have to be considered. First, the definition of the properties of the AmesimBlock elements need further ***value types*, *units* and *enumeration* elements**. Standardized elements defined in the ISO 80000 profile, which is provided by the SysML modeling tool, are used where possible. Those simulation elements that need more specific *units* or *enumerations* have them defined in the library. Examples of the rotating load element in Figure 31 are the “offset to be subtracted from angle” property with the type “plane angle[degree]” and the “modulo option” property with the possibilities to choose between “no modulo”, “modulo 360” and “modulo 720”. An excerpt of the *value type*, *unit* and *enumeration* definition in the library is given in Figure 33.

Second, it is possible to include further custom elements in the library. These **supercomponents** are model elements that define partial simulation models. They realize the principle of representing a group of components by a single icon to avoid confusingly large simulation models in Amesim. They can be added to both representations of the database in Amesim and SysML and are used as any other simulation element. For instance they are used by Münzer and Shea in [48] to enable automatically generated simulation models.

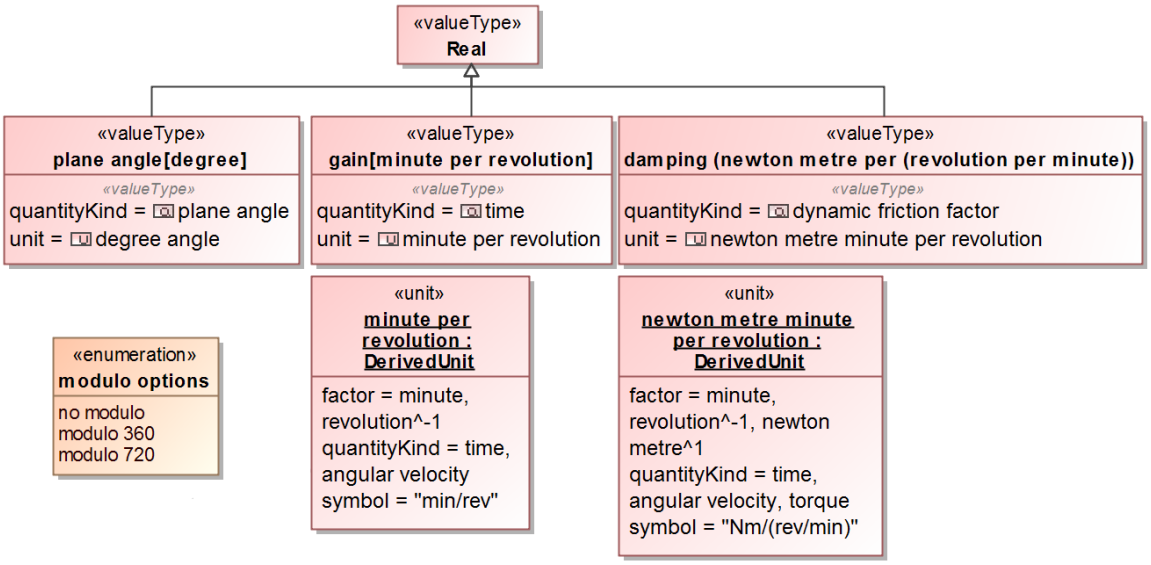


Figure 33: Excerpt of *value type*, *unit* and *enumeration* definition of behavior simulation library

4.3.2. Behavior Simulation Library Usage

The **intended use** of the behavior simulation library is the provision of elements that work as principle solutions to realize the modeled functionalities as well as the supported planning of simulation models directly in SysML. The simulation results can, for instance, be used for a first concept evaluation or to identify additional requirements and system elements. The behavior simulation library is used in the case study to plan the simulation of the two alternative kinematic systems, the HBot and the CoreXY of Section 3.2.

An **example usage** in the case study is given in Figure 34. The shown example is also part of a multi-solution pattern from Section 4.5. It shows on top the <<User-definedFunction>> “Position Liquid” that is decomposed amongst others into the <<ElementaryFunction>> “TranslationalEnergy:Guide”. This function is created by using the function library elements “Guide” and “TranslationalMechEnergy”. It stands for the guidance of the translational mechanical energy in the used belts of the HBot and CoreXY kinematic solutions.

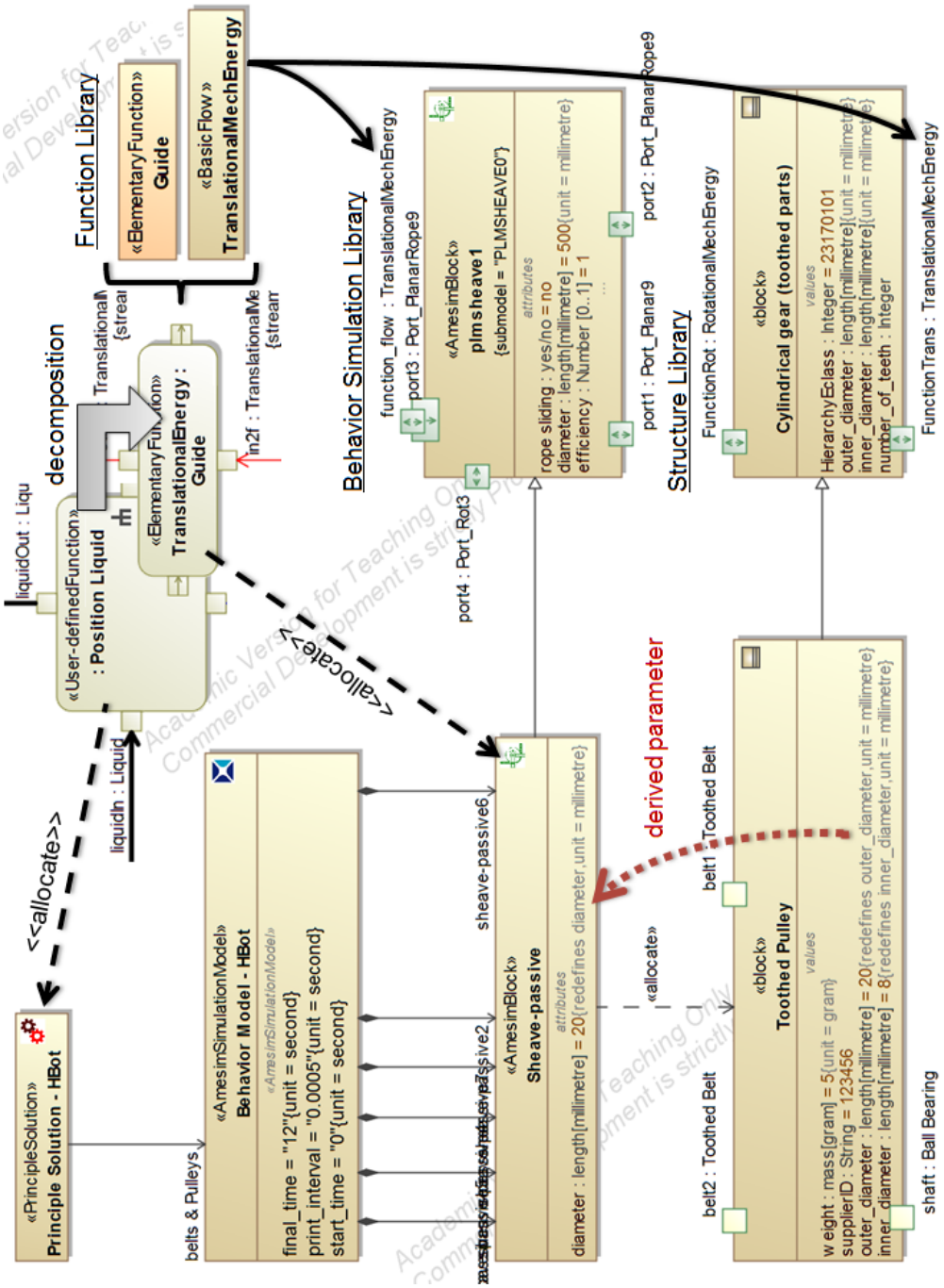


Figure 34: Behavior simulation library usage example

In Figure 34 there is a <<PrincipleSolution>> in the form of the HBot kinematics *allocated* from the user-defined function. This principle solution on the

principle solution level is *associated* to a <<AmesimSimulationModel>> element that further refines and concretizes the principle solution in the role of its “belts & Pulleys”. This **<<AmesimSimulationModel>> element “BehaviorModel - HBot”** represents the later created simulation model in Amesim itself. It contains necessary properties for running the simulation, e.g. the simulated time between “0” and “12” seconds and the used “print_interval” of “0.0005” seconds. As a simulation model it is composed out of <<AmesimBlock>> elements as *part properties*. Here there are six passive sheaves shown, which are *allocated* from the <<ElementaryFunction>> “TranslationalEnergy:Guide” since they provide the behavior of guiding the belt and with it the energy flow. They fulfill the role of a working principle that realizes the function.

These **<<AmesimBlock>> “Sheave-passive” elements** are subtypes of the reused “plmsheave1” element from the behavior simulation library. This *generalization* relationship is created to inherit the properties from the library, while allowing their redefinition into concrete values. In this case a concrete value is the diameter of “20” millimeters, being derived from the parameters of the final realization of the component in form of the “Toothed Pulley” below. The <<AmesimBlock>> element also has interfaces with additional functional flows. In the example the flow of “TranslationalMechEnergy” is shown that relates to the main flow of the elementary function. The required “submodel” is already defined as “PLMSHEAVE0” within the element from the library.

On the bottom of Figure 34 the **structure library** of Wölkl [13] is used for the “Toothed Pulley” *block* that provides parameters for the simulation. It is allocated from the “Sheave-passive” element as its concretization on the technical solution level. The structure library provides its parent element in form of a generic “Cylindrical gear (toothed parts)” with its eCl@ss hierarchy identifier, properties and generic *flow ports* that again relate to the flow of translational

mechanical energy. This concludes the traceability of properties over behavior simulation elements to functions on the example in Figure 34.

The different <<AmesimBlock>> elements that constitute the simulation model as *part properties* are interconnected on IBDs. An example of a **simulation model represented in SysML and Amesim** is given in Figure 35. As illustrated with the concept sketch on top of the figure it shows an excerpt of the complete kinematics model in form of a single pulley guiding the toothed belt. On the left side there is the model in Amesim, whereas on the right side it is illustrated as shown in SysML. Both model representations have the same elements, as highlighted for the rotary load element that serves as an inertia of the passive sheave. For its other open end the rotary load element has a zero torque source. The sheave is fixed in the reference frame, by a fixture that serves as a zero acceleration, velocity and displacement source.

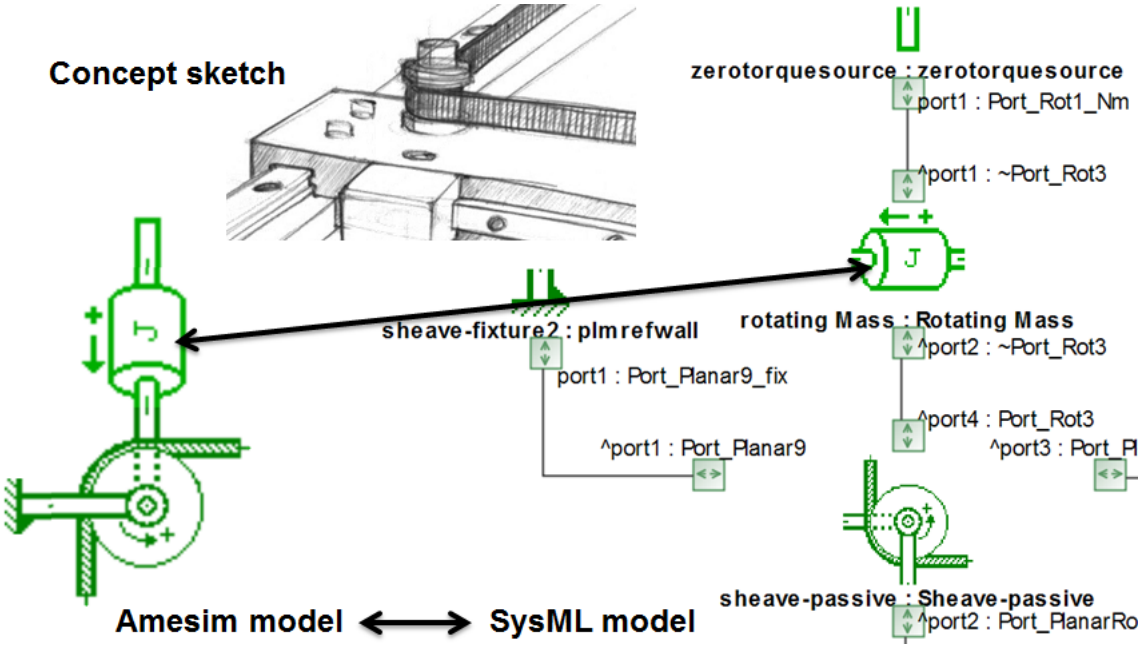


Figure 35: Excerpt of simulation models in Amesim (left) and in SysML (right) (adapted from [1], concept sketch from [144])

The **icons of the simulation elements** in SysML are from the tool database and defined for the elements in the library. Here they are used to represent corresponding *part properties* on IBDs. This helps the designer not only by providing adequate icons that illustrate the single elements, but also by giving a consistent visualization between the different tools. Current issues with the icons are that they are not passed on through inheritance in SysML, meaning that the icon of “Sheave-passive” in Figure 34 is assigned manually. Also it is not possible in the SysML modeling tool to rotate them to match each other with their drawn interfaces, as it is done in Amesim on the left side of Figure 35.

Examples for the **compatibility of different *flow port* interfaces** of the library elements in SysML are shown in Figure 36. The invalid *connector* on top is between two completely incompatible *port* types for rotational energy and translational energy through a rope. Below is another invalid *connector* between the unspecific parent rotary load element of Figure 31 and a *port* of the passive sheave element that handles rotational energy, too. Here the error is caused by the *port* of the rotary load being too generic. It requires further specification through a fitting submodel. The *connector* on the bottom is the correct connection between the two elements with the submodel “rotaryload2_02A” selected. Above it there is an incorrect submodel selected, which is not possible in the Amesim tool but causes no error message in the SysML tool. Since the causality can change each time a new element is added because of Amesim’s bond-graph [107] origin, the generic parent elements without concrete submodel and interfaces from the library should be used initially in SysML.

To simulate a model planned in SysML, the capability exists in the Amesim tool to import IBDs from SysML [155]. A manual mapping is necessary between the *part properties* and corresponding elements of the tool’s database. If

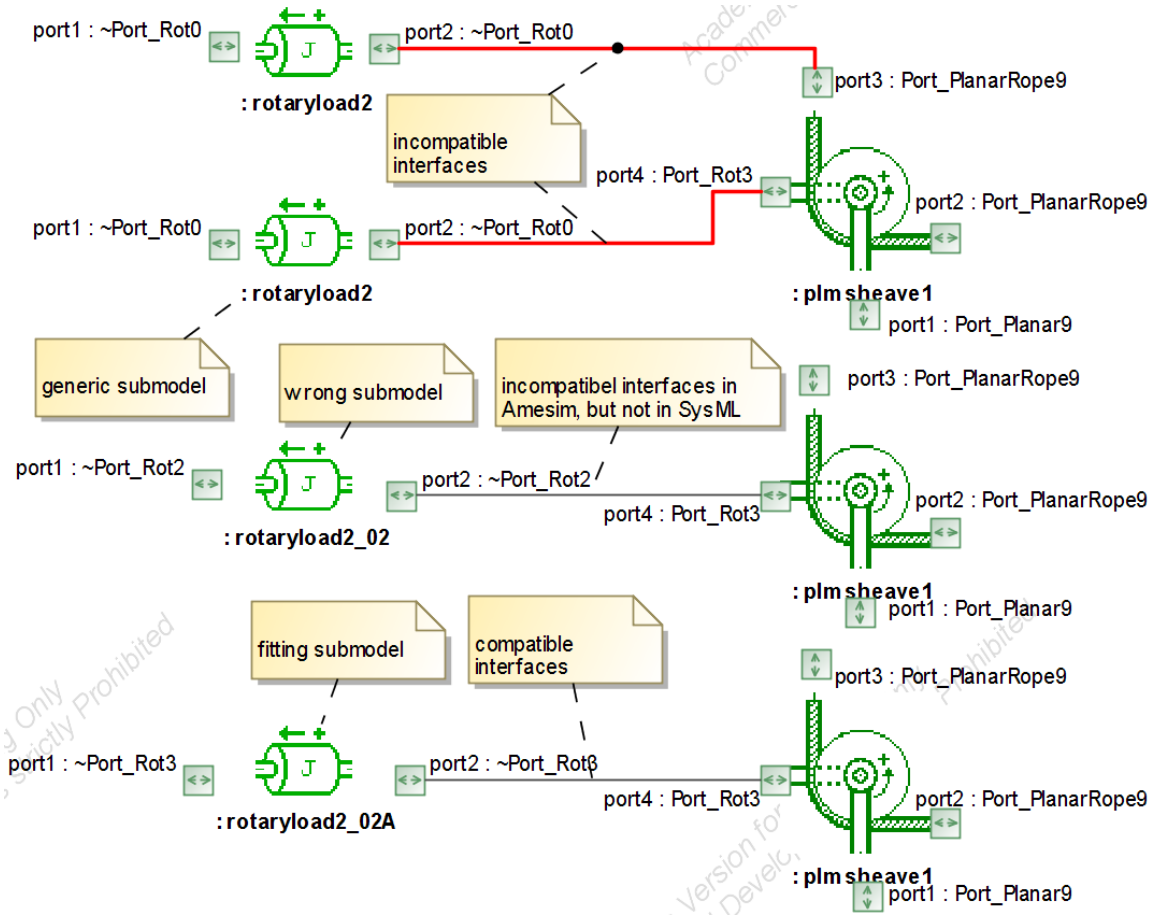


Figure 36: Interface compatibility examples of behavior simulation library elements

the SysML model is only partially complete it is further refined in the simulation tool. To run the simulation of the case study there are approximated stepper motor models together with simple controllers without feedback added to the kinematic models, which are provided by a multi-solution pattern from Section 4.5. The simulation of the DC motor variants of Figure 13 is not shown, due to a missing comparably detailed stepper motor model. Therefore for demonstrating the simulation capabilities with the case study only mechanical aspects of the HBot and CoreXY kinematics are compared.

The **simulation runs** in LMS Imagine.Lab Amesim independently from the SysML model or its modeling tool. For the simulation of the two kinematic concepts of the case study the print head moves in a circular path for a total of 12 seconds with time intervals of 0.0005 seconds. These parameters are also defined in the simulation model representation in SysML, as seen in Figure 34. Two parameters of the simulation results are displayed in Figure 37. The figure shows the orthogonal forces on the linear bearings of the sliding carriage over the 12 seconds time frame. “The graphs appear solid since positive and negative values are reached almost simultaneously due to the approximated stepper motors that have inconsistent stepwise provision of angular momentum resulting in very quick load changes” [1]. Comparing the two alternatives, the load of the HBot is about twice the load of the CoreXY configuration. The HBot reaches around ±6 Newton while the CoreXY only reaches ±3 Newton.

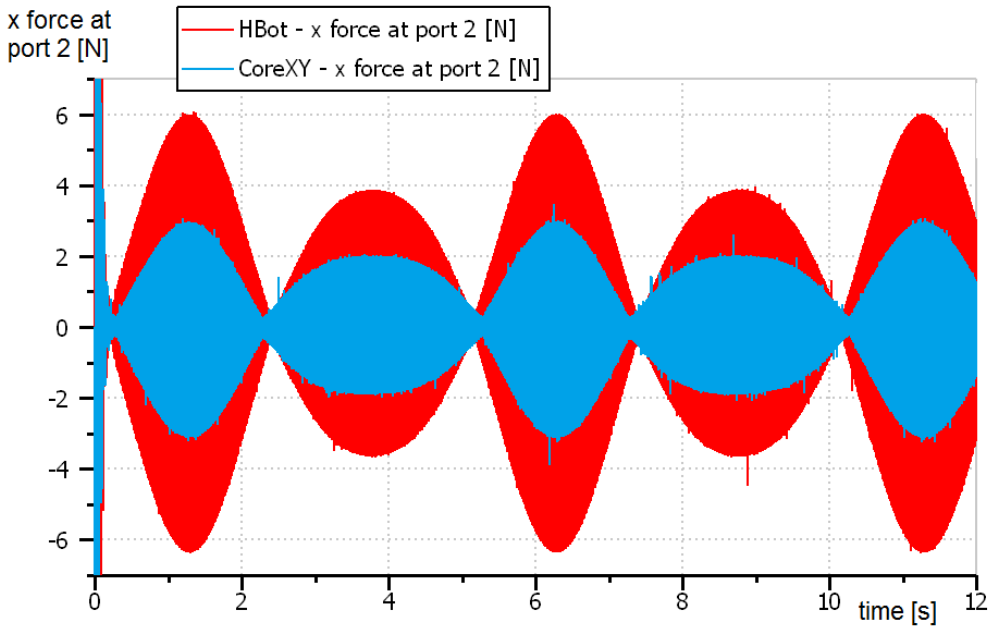


Figure 37: Simulation results of oscillating orthogonal forces on the linear bearings of the sliding carriage for HBot and CoreXY [1]

These **simulation results indicate** that the simpler HBot needs linear bearings that can withstand a higher orthogonal load than the ones for the CoreXY with its more complex two belt configuration. Due to these results an additional requirement is introduced in the model especially for the HBot design to ensure robustness against these orthogonal forces to reach comparable printing performance and precision. This demonstrates the use of the behavior simulation library for an initial simulation of different concepts for potential concept evaluation or trade-off studies.

In the shown examples no element with the *stereotype* **<<SolutionPrinciple>>** is used. As described in Section 4.3.1, it serves as a general solution principle to specify principle solutions or be a black box placeholder in a simulation model. Its use is illustrated in Section 4.4.2 in Figure 42 together with the use of the service library.

4.4. SERVICE LIBRARY

This section presents the service library in SysML. First its definition and implementation is shown before its use is demonstrated with the case study. The service library contains the collection of services from Schmidt et al. [17]. The library serves as a formal repository of proven and accepted services. It addresses the designer's **need** to support the identification of services to realize modeled functionalities in PSSs. This is important since "compared to product design, a broader range of knowledge is required in PSS design because both products and services are included in the design space" [7]. Also in PSS design there are needs identified [28] to better support the links between requirements and actors through services, as well as between the physical product and its services. This can be enabled by using unified SysML modeling for PSS design.

4.4.1. Service Library Definition

To formally capture services in SysML there is the **<<Service>>** ***stereotype*** defined in the service library. It is displayed in Figure 38. It is not only used for the services of the database, but is also to be used to model custom services in SysML. The <<Service>> *stereotype* is a subtype of the SysML *block*.

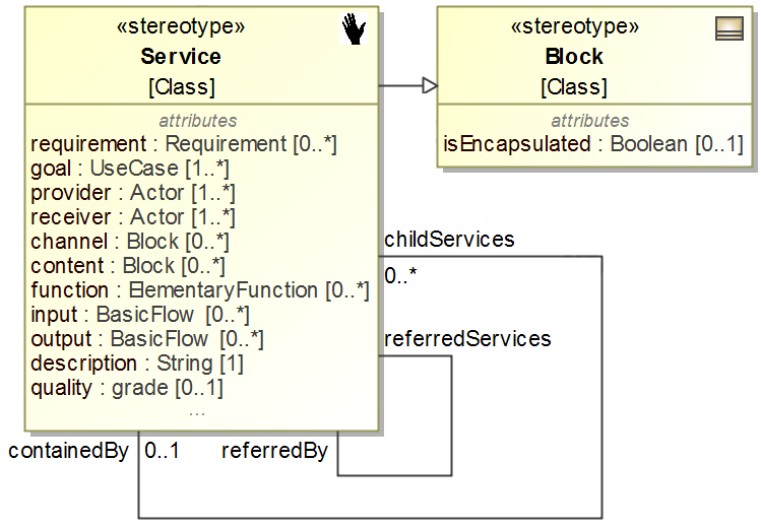


Figure 38: <<Service>> *stereotype* in the service library

To represent services in an object-oriented way the *stereotype* has multiple **properties**, which are based on different approaches in literature [7, 26, 35]. Its properties are: an arbitrary number of *requirements*, at least one *use case* as the goals of the service receiver, at least each one service provider and service receiver as SysML *actors* to represent the entities who provide and perform as well as order and receive the service. Then there is an arbitrary number of channel and content elements as *blocks*, which capture the channels the service uses between the provider and receiver to deliver the service content. The providers, receivers, channels, service contents and their interrelations constitute the service environment [26]. The functions of the service are

formalized by <<ElementaryFunction>> elements together with their input and output <<BasiFlow>> elements from the function library. The textual description of the service and the service quality come last. The service quality property uses the same “grade“ enumeration as seen in Figure 29 to qualitatively state the quality of the performed service. Relations between <<Service>> elements are for referencing related services and for one service containing a number of child services. The formal capturing of these service properties is also beneficial for the documentation of services, since just like functions, services are also very much subjective and depend on the viewpoints of the service providers and receivers [27].

An **example for a <<Service>> element** in the library is given in Figure 39 with the “Remote Inspections” service of the service catalogue [17]. Such services define remote inspections of the systems through their manufacturer, for example to identify potential needs for maintenance or to keep the 3D printers of the case study calibrated for a reliable printing performance. The service element in the library has generic properties defined to match any remote inspections services. The top property is the service channel on which the provider performs their service. Here, it is a “Remote Access” that the system needs to have in order to inspect it remotely. Below follows the parent-child relation towards the “Remote diagnosis” service that is contained in a remote inspection of the system. Next is the service content in the form of an “Inspection Report” that is created and delivered. The textual description describes the service, which is about remotely gaining information of the absent system’s condition. The elementary function is “Sense”. It has the inputs of “Energy” and “ControlSignal” as well as the output of a “StatusSignal”. The service senses the system’s condition by means of the provided energy according to the actuating control signal. The system itself is not modeled as a flow of its own. This fits to the ideas

that functional modeling can be used as a basis for developing PSS, as done in [7, 24] or reasoned in [29]. To provide “Remote Inspections” services a “Technician” is needed, while the receiver is a generic “Service Receiver”. The related *requirement* “Remote Access” is about capturing the need of an adequate remote access to the system. Finally, the service goal in the middle is: “Gain awareness of product condition”. This *use case* is derived from a so called customer function that was used during the creation of the service catalogue [17].

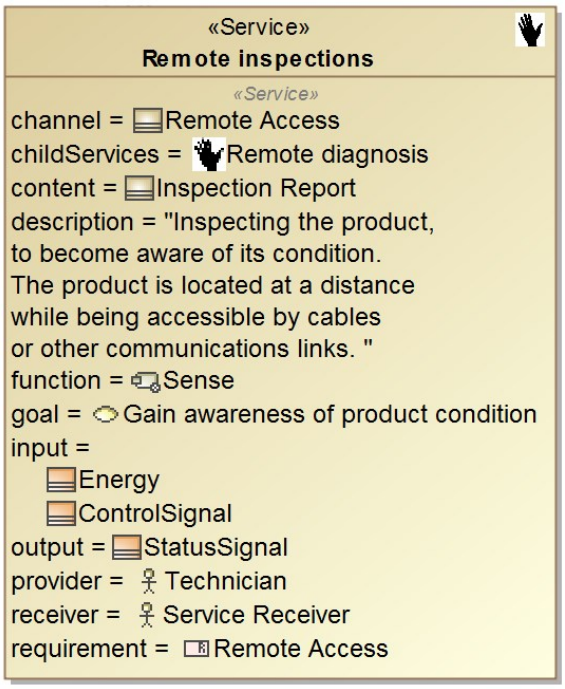


Figure 39: "Remote inspections" service from service library

The **content of the service library** is structured according to the categories, clusters and super-clusters of the original service catalogue [17]. An excerpt of the implementation of the service library in SysML, is seen in Figure 40. On top is the “Generic Service” element as an *abstract* parent element to all contained services. The following three layered hierarchy is contained in

packages for the four main categories of the catalogue. In Figure 40 the *package* for the category “Services supporting Product” is displayed. The contained super-clusters, e.g. “Product Maintenance”, and clusters, e.g. “Product Inspections”, follow. All subordinate elements are related by *generalizations*. Service elements on these levels do not have special properties since they are only for organizing the contained services. For this they have descriptions from the service catalogue as their documentation in SysML. Two examples are given in Figure 40 with *callout notations*. On the lowest hierarchical level of the library are the 265 actual services. Three example services about product inspections are displayed without their properties.

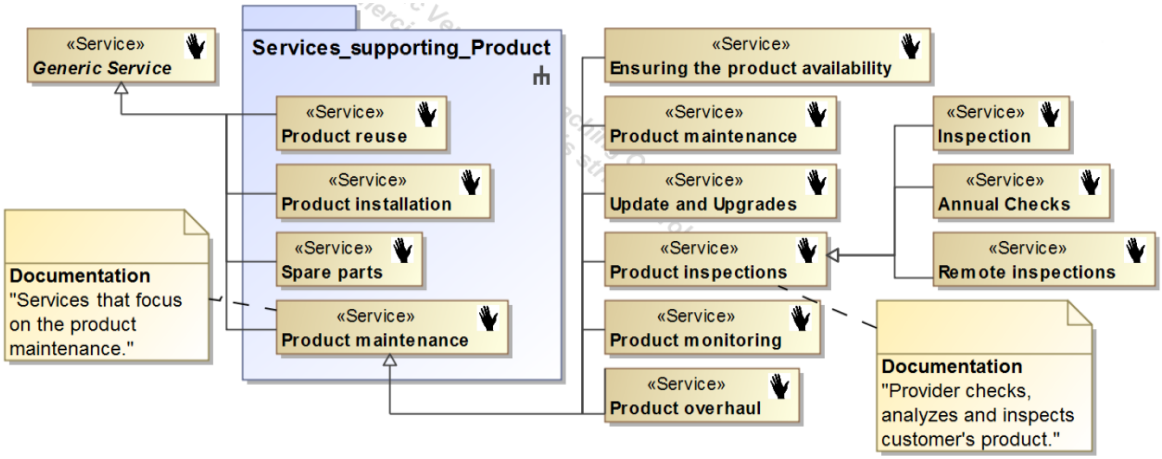


Figure 40: Service library hierarchy excerpt with “Product Inspections” services

Besides the service hierarchy there are **accompanying elements** stored in the service library. For describing the services there are *actors*, *blocks* for channel elements and content elements, service goals as *use cases* and *requirements* needed. They are modeled in separate *packages* with further *generalization* hierarchies. An example of such a hierarchy is displayed in Figure 41 for some used *actors*. Beneath the abstract “Generic Actor” there are two

groups of actors for different “Service Receivers” and “Service Providers”, e.g. the “Technician” of Figure 39.

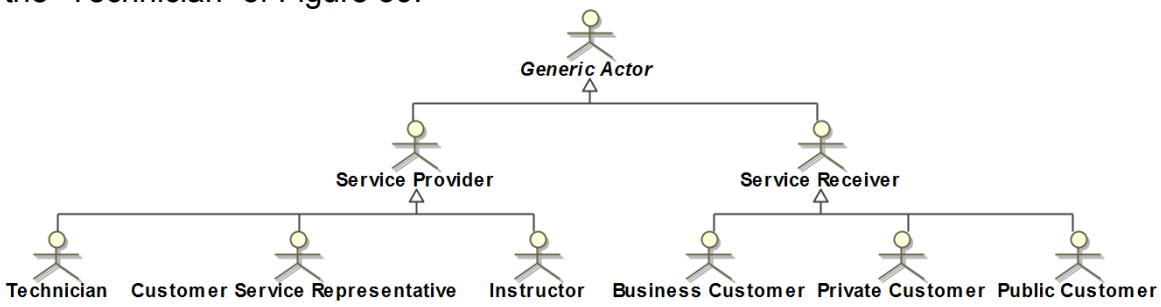


Figure 41: Excerpt of actor hierarchy for service providers and service receivers

4.4.2. Service Library Usage

Generally, the service library is used similar to the structure library with its machine elements and components [13]. However, for the services there is a greater need to tailor them for their specific use and physical system. This goes together with the intended use of the cataloged services as solution principles to realize functions.

Using the service library together with the other libraries presented creates an integrated modeling approach for PSS. This is similar to the six step PSS design process of Kim [24]. The stakeholders and actors with their intended service usage are specified, as seen in Figure 24 with an example from the case study in form of the *use case* “Maintain Printer”. An example for a following functional model is given in Figure 27 showing the “Maintain Printer” ACT with the “Inspect Printer” user-defined function. The service library is now be used to provide services to realize the modeled functionalities. An example from the case study is the *allocation* from the user-defined function “Inspect Printer” to suitable principle solutions, e.g. to inspect the 3D printer by offered “Services”. This is displayed in Figure 42. There the <<PrincipleSolution>> “Service” is further concretized by a generic <<Service>> element in form of “Remote Inspections”

from the service library of Figure 39. Figure 42 also shows the use of <<SolutionPrinciple>> elements of the behavior simulation library to capture two different realizations of the second accompanying principle solution.

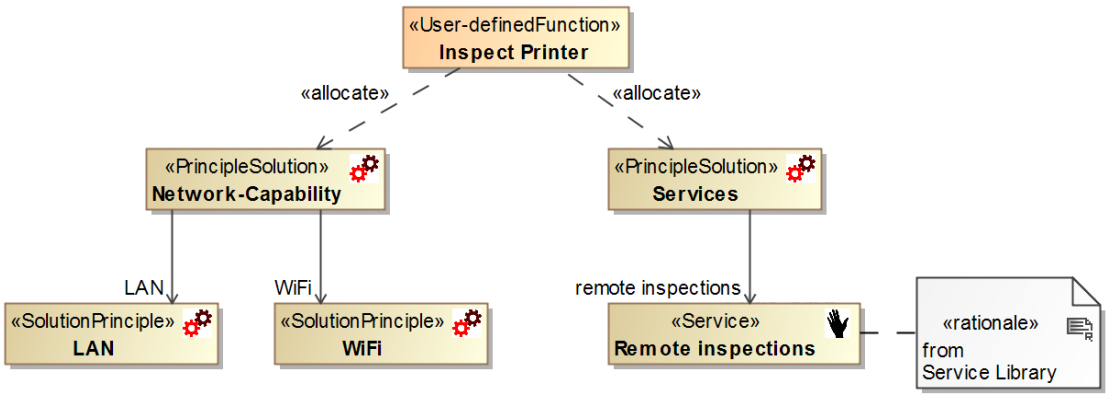


Figure 42: Principle solution example for “Remote Inspections” service of the 3D printer

After the selection of generic services from the library they must be **adapted and detailed** for their concrete application. An excerpt of this is shown in Figure 43 for the custom “Raptype Remote Inspection” service. This service element uses the <<Service>> *stereotype* and inherits from two services from the library: “Remote Inspections” and “Calibration”, which are combined to offer a specific and unique solution. Figure 43 shows the *tagged values* of the “Raptype Remote Inspection” service, e.g. the DC electrical energy input or the “Network Interface Card” as main service channel. It also displays the linkage of the service towards other related elements to allow traceability through the SysML model. The *callout notation* states that the service is *allocated* from different involved <<PrincipleSolution>> elements and the <<CallBehaviorAction>> “PrinterCondition”, which is an elementary function of the user-defined function “Inspect Printer” from Figure 27 and Figure 42. There are also *allocations* towards the service provider “Technician”, the “Network Interface Card” and the

"Remote Control Software". The "Network Interface Card" has a *satisfy* relation towards the "Remote Access" *requirement* that comes originally from the "Remote Inspections" service of the service library.

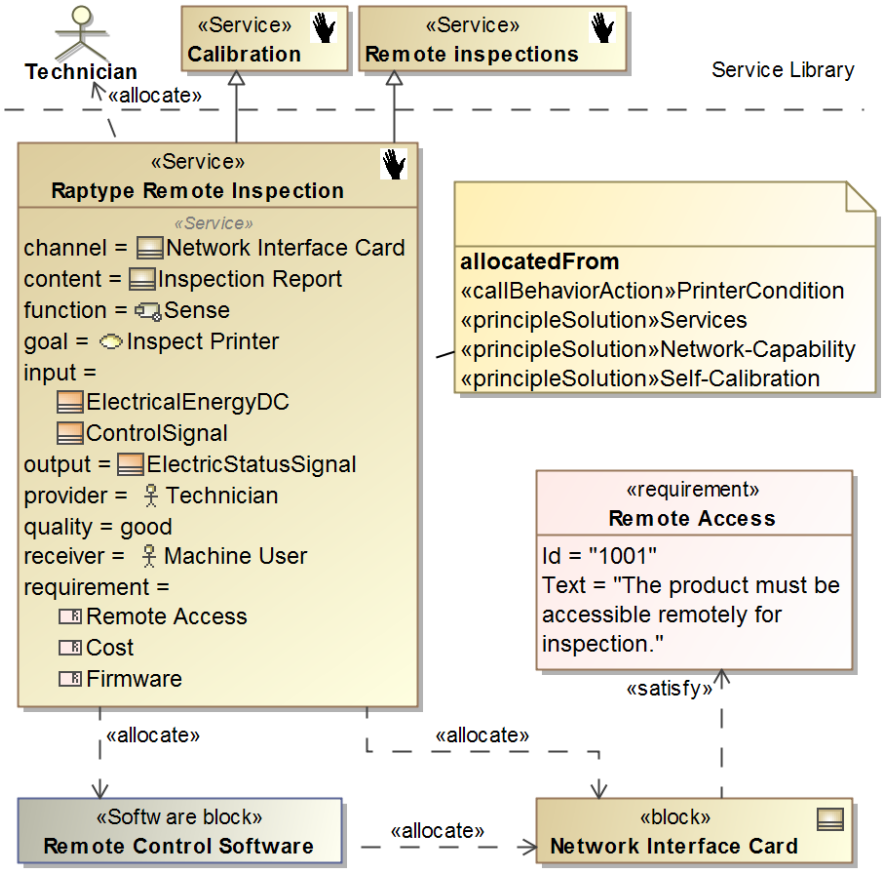


Figure 43: Model excerpt for custom "Raptype Remote Inspection" service, showing its linkage to other model elements

4.5. Multi-Solution Patterns

This section presents the multi-solution pattern concept in SysML. First they are defined and implemented in SysML, then excerpts of an example pattern are shown before its use is demonstrated with the case study. The presentation of the example pattern is separated from the definition of the concept in SysML itself, since unlike the other libraries there is not a given set of

elements to be implemented, but all patterns are to be derived from previously developed systems and are therefore more problem specific.

The patterns are based on the mechatronic solution patterns of Anacker et al. [122] to support the modeling “by describing concept solutions in the form of partial models that correlate library elements with other aspects to offer coherent subsystems” [1]. The patterns address the **needs** of further concept modeling support that goes beyond independent elements from design libraries. They use library elements to offer multi-disciplinary design knowledge from different levels of abstraction. This way the patterns improve reuse by providing verified solutions to be adapted from existing designs [44].

4.5.1. Multi-Solution Pattern Definition

To successfully support the concept design of multi-disciplinary systems using patterns several **requirements** must be met [122]: The multiple disciplines and levels of abstraction, including a solution-neutral view, must be taken into account. Also, creating and using the patterns must allow a dynamic application by a direct integration into the design process together with a growing potential with each successfully modeled system. This is possible through the model-based representation of the pattern in SysML because of its advantages compared to conventional textual pattern descriptions [106, 122].

The **multi-solution patterns are defined** to describe multiple alternative and domain-spanning principle concept solutions to realize target functions. The solutions describe the basic operation mode of the system and its desired behavior by several aspects. These combined aspects form a coherent system by correlating with each other as part of partial models of the pattern. This way they interrelate the pattern context, problem description and multiple solutions together with guidance on use and forces to be considered [1]. Combining

multiple solutions into one pattern is based on the idea of using a solution-neutral functional description as the basis for identifying suitable patterns, especially since the patterns themselves contain functional models for their multi-disciplinary solution representation. Since the functional models are solution-neutral they allow realization by the multiple different solutions of the pattern.

The formal **implementation in SysML** allows the necessary multi-disciplinary model representation and the easy reuse of knowledge from other SysML models and design libraries. Also the formality supports a “uniform pattern representation to combine and compare patterns and to fix the engineering design-knowledge [while] providing essential context know-how” [122]. For the implementation in SysML there are two *stereotypes* defined in Figure 44 to extend the *“Package” metaclass*. There is the <<Design Pattern>> *stereotype* for the patterns with their solution-neutral information and the <<Pattern Solution>> *stereotype* for the individual solutions with all solution-specific information. The different aspects of the pattern description are defined as *tagged values* of the *stereotypes*. They follow the existing frameworks for engineering patterns [106, 116, 122, 123, 156] and are adapted for SysML to allow the inclusion of multiple solutions and the utilization of the design libraries.

The **<<Design Pattern>> *stereotype*** has the following properties: There are possible “alias” *strings* to capture alternative names for the pattern. To uniquely identify a pattern there is the “patterID” *string*. Next, related keywords are stored as additional *strings*. The “problem_description” *string* describes “the specific problem that needs to be solved” [116]. This description is to be independent of the problem context and separate from the constraints on the solution. The “problem_context” is described by its own *string* in the next property. It specifies the circumstances in which the problem is solved, especially

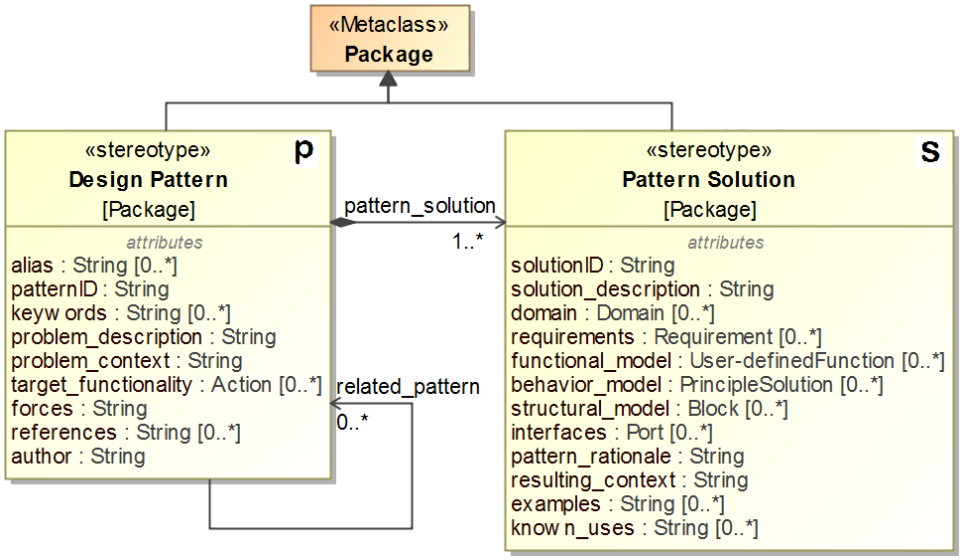


Figure 44: Multi-solution pattern *stereotypes*

with respect to their imposed constraints on the solutions. The "target_ functionality" is modeled as SysML *actions*, preferably by using *call behavior actions* as elementary functions from the function library. The target functions are those solution-neutral functions the solutions of the pattern realize. The *string*-typed "forces" on the pattern explain "the often contradictory considerations that must be taken into account when choosing a solution to a problem" [116]. The importance of the forces relative to each other is determined by the context.

Finally, there is an arbitrary number of *strings* to document "references" of the pattern and one *string* for the "author" of the pattern. Further relations of the <<Design Pattern>> *stereotype* are the recursive relation to any number of related patterns and the relation to the main elements of the patterns: their solutions, of which at least one or more must be included.

The solutions are represented by the **<<Pattern Solution>> *stereotype*** on the right of Figure 44. The solution solves the problem by resolving the higher priority forces at the expense of less important forces as determined by the context. The *stereotype* has the following properties: There is the "solutionID"

string, to uniquely identify a particular solution, similar to the “patternID”. The “solution_description” has also the type of a *string*. It describes the solution in a textual way to accompany and summarize the additional modeling elements. The arbitrary number of involved *domain* elements documents them explicitly [152]. The *requirement* elements introduce necessities the system has to fulfill when selecting the solution.

The “functional_model” contains a further and more solution-dependent functional decomposition of the target functions in <<User-definedFunction>> elements from the function library. This functional model serves for a better understanding of what the solution is actually doing and to maintain traceability from the target functions to the system elements from the pattern. The “behavior_model” is defined by <<PrincipleSolution>> elements from the behavior simulation library of Section 4.3. It can contain simulation models using the elements of the library to offer simulation models for the solution. To allow a seamless integration of simulation models from different patterns there must be general compatibility ensured for the partial simulation models in SysML and Amesim. For this there are rules for the creation of simulation models defined by Münzer and Shea in [48]. The “structural_model” finally contains *blocks* to represent the subsystem structure of the solution on the technical solution level. For this it can use elements of the structure library [13] or the service library of Section 4.4. Coming from the structural model there are the “interfaces” of the solution. They are represented as *ports*. They explicitly document the interfaces of the solution, also to be used for an identification of suitable solutions of a pattern, e.g. by using the main function flows from the function library as it is done for structure library elements, too [13].

The “pattern_rationale” captures “an explanation of why this solution is most appropriate for the stated problem within this context” [116] in form of a

string. The ”resulting_context” is also a *string* for describing the system context after the application of the particular solution of the pattern. This especially includes potential new problems to solve, which can relate to other patterns. Finally, there are *strings* used to capture “examples” of the use of the solution and “known_uses” of the particular documented solution from the pattern.

With these properties the pattern solution models cover their specific solution, while being only partially complete in the context of the whole system. The **partial models** follow general SysML modeling rules and can use elements from the libraries, since they should be captured from previously modeled systems. This includes linkage between the different aspects within the partial models of a solution, e.g. to trace back from the structure and behavior models over the functional models to the requirements.

4.5.2. Multi-Solution Pattern Example

Key aspects of the **multi-solution pattern “2D Kinematics”** are presented here. It includes partial simulation models to evaluate the solutions. The pattern element is shown in Figure 45 with cropped descriptions. The pattern solves the problem of two-dimensional positioning with fitting speed and accuracy. For this it has the target function “PositionMaterial” with a “Material” flow to be positioned by an auxiliary “Energy” flow while giving a “StatusSignal” about its position. The problem context for example introduces a fixed reference frame to take up reaction forces of the moving kinematics. The offered <<Pattern Solution>> elements are: “CoreXY Solution”, “HBot Solution” and “Scissors-Bot Solution”, which all realize the target function and solve the pattern’s problem. The two related patterns are about the provision of mechanical energy in different forms for the different solutions. The properties for references, keywords and alias are also used as seen in Figure 45.

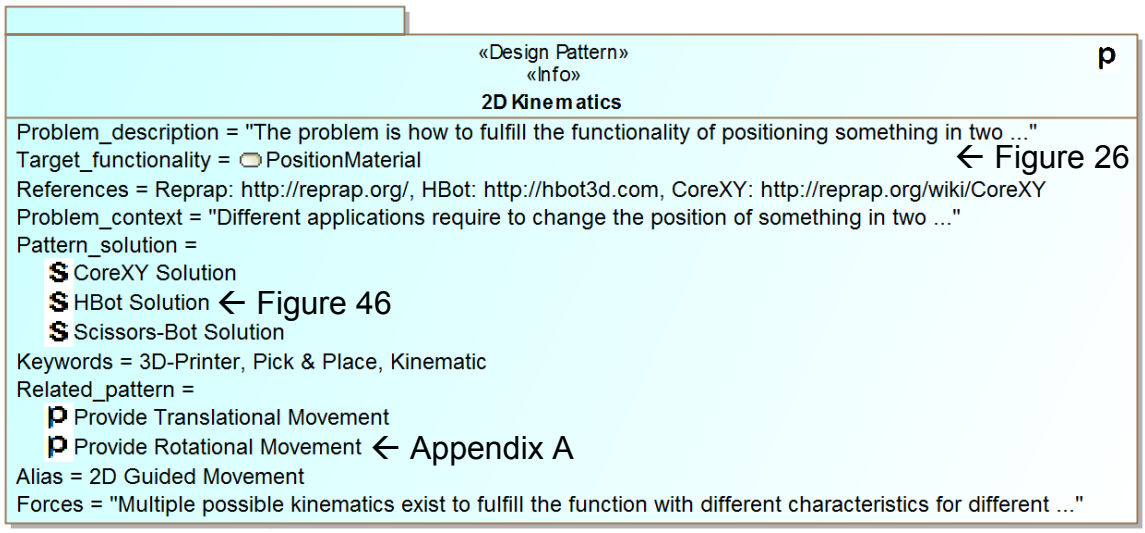


Figure 45: Cropped "2D Kinematics" design pattern (adapted from [1])

For a **<<Pattern Solution>> element** there is the cropped “HBot Solution” of the “2D Kinematics” pattern illustrated in Figure 46. Besides different modeling elements it contains a solution description, to describe the HBot configuration with the sliding carriage and its propulsion by two motors driving a single belt. The pattern rationale explains that such a solution reduces the moved mass due to the fixed motors and that it produces a torque on the moved object, due to asymmetrical force application. The resulting context of the solution application describes the needs to further adapt the model with respect to whatever is actually positioned and that solutions must be found for propulsion and control.

The **partial models** of the solution in Figure 46 include the “Functional_model”, which contains a functional decomposition of the target function together with auxiliary functions needed for this functional decomposition. The *call behavior actions* of the decomposition that represent elementary functions from the function library are displayed on the allocation matrix of Figure 47. Analog, there is the “Structural_model” with its *part properties* displayed in Figure 47. The two *requirements* that come with the

solution are “Belt Tension” that demands a certain tension in the used belt to work and “Orthogonal Load on Linear Bearings” to state that the linear bearings of the HBot solution must withstand a certain orthogonal load. This orthogonal load is determined by the accompanying behavior model with its <<AmesimSimulationModel>> element to conduct a simulation as explained in Section 4.3.2. Excerpts of the simulation model are given with Figure 34 and Figure 35. Finally in Figure 46, there is the “Domain” of the solution set to “mechanical” and its interfaces are explicitly defined flows fitting to the target function. Only the signal output is not specified in the offered solution, since it depends on the used motors, e.g. stepper motors or servo motors.

To support traceability and reasoning within the solution, *allocations* are set between the model elements. An excerpt of the **allocation matrix of the "HBot Solution"** is given in Figure 47. There are different types of *allocations* used in the pattern as shown in the matrix. The different types of *allocations* are presented in Section 2.1.2.

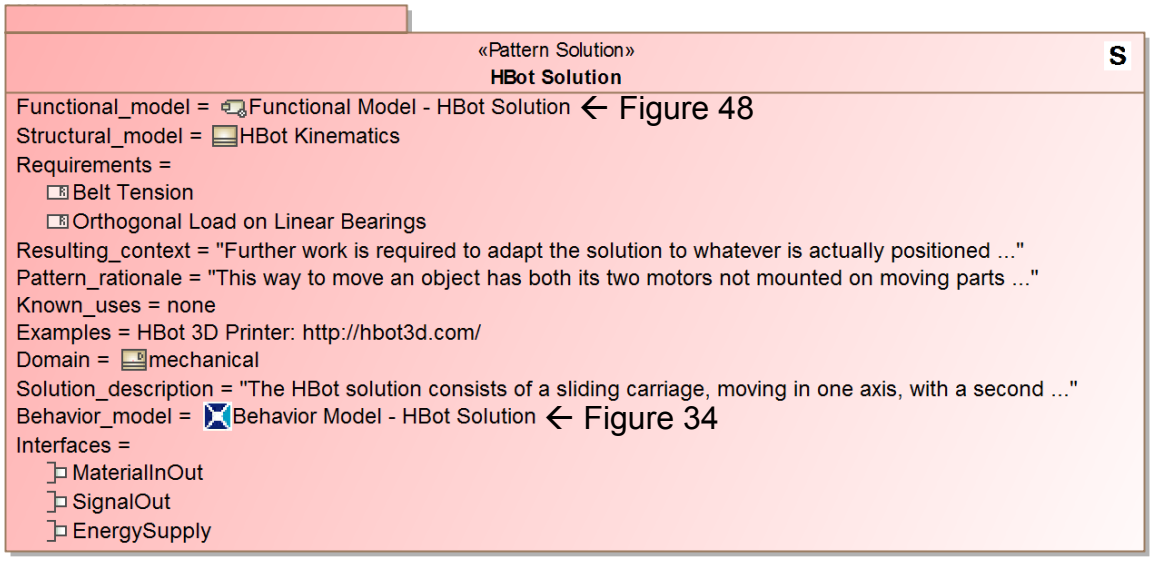


Figure 46: Cropped "HBot Solution" pattern solution (adapted from [1])

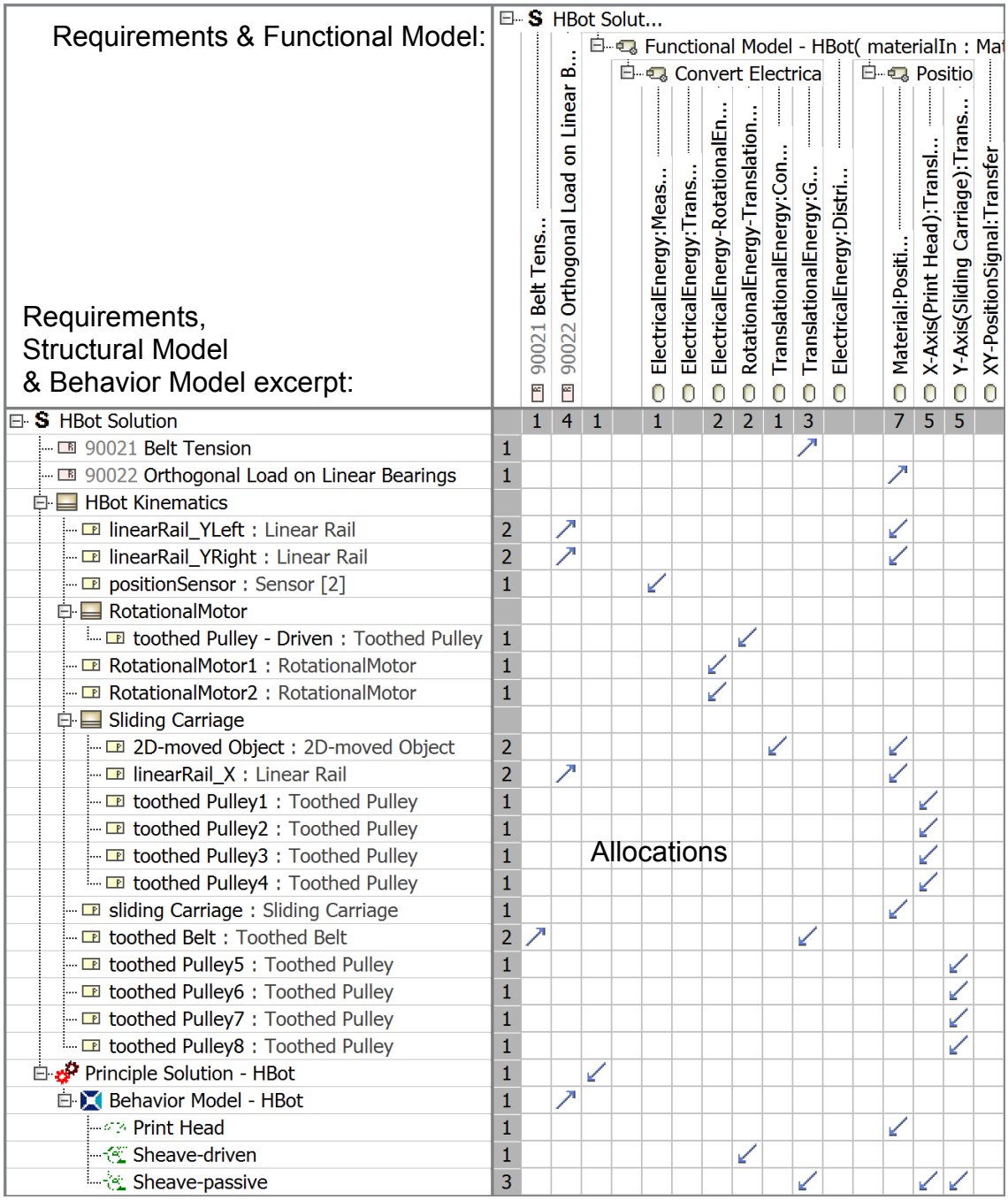


Figure 47: Excerpt of allocation matrix of "HBot Solution"

Allocations from usage to definition are set between the *actions* of the functional model and the customized <<AmesimBlock>> elements of the simulation model, as also seen in Figure 34. *Allocations* of usage are set to trace between the elementary functions and the *part properties* of the structural model, e.g. the "toothed Belt" that realizes the function of guiding translational energy.

Another example for a multi-solution pattern is given in Appendix A – Multi-Solution Pattern Excerpt, with the related "Provide Rotational Movement" pattern that offers DC motor solution and a stepper motor solution. It contains, for instance, further auxiliary information in the form of STM and SD diagrams to specify the sequential and behavior and state transformations of the motors outside of Amesim simulation models.

4.5.3. Multi-Solution Pattern Usage

When using solutions from patterns they must be adapted accordingly to the specific context of the problem. The two HBot and CoreXY solutions of the presented "2D Kinematics" pattern are used in the Reprap model for its print head movement. After selecting a solution from a pattern it gets copied into the system model together with its partial models to be integrated and adapted.

An example of this **adaption process** is shown with the functional model in Figure 48. It is shown here because of its use to identify suitable patterns by the target functions [1] and to refer to the function of the initial model on the lower ACT in Figure 26: "Liquid : Position". This functions gets *refactored* by the functional model of the solution, which then is adapted accordingly. The final result of this process is partially given in Figure 48. On the left side are adapted functions from the pattern, e.g. with the generic "Material" flow of the pattern specified into the "Liquid" flow of molten plastic in an FDM based 3D printer. The user-defined functions, e.g. ":Position Liquid", are further decomposed on other

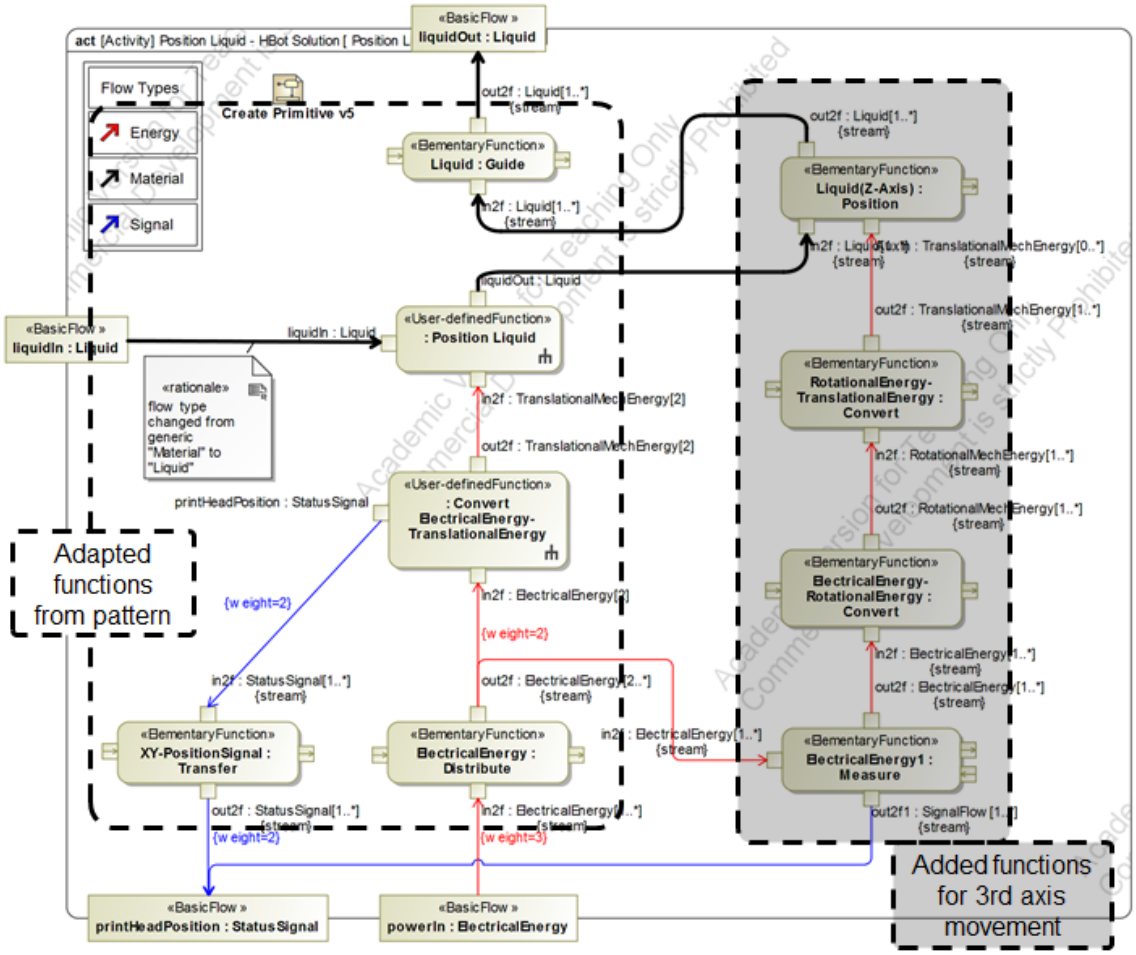


Figure 48: Functional model adaption at pattern application

diagrams and the auxiliary elementary functions, e.g. “Electrical Energy: Distribute”, are enabling them. The functional model is then manually extended on the right side to include functions for the third axis movement of the 3D printer, which is realized here by moving the print table.

The other partial models, e.g. behavior and structure are handled similarly. Necessary adaptions are, for instance, the replacement of the abstract “2D-moved Object” of Figure 47 by a model of the print head or the completion of the simulation model for its use. This means not only a setting of fitting parameters of the provided elements as seen in Figure 34 for the pulley diameter, but also the

addition of essential motors and control, which are not part of the pattern. Motors, for example, can be added by applying the related pattern, as mentioned before. The simulation and its results are shown in Section 4.3.2.

4.6. RESULTS

The concept modeling approach applied to the case study results in several findings related to the used design libraries and multi-solution patterns. Focusing on the demonstration of the presented modeling support, the conceptual design of the 3D printer is only partially completed in SysML. There are model elements of **different domains used**: “Electrical”, “Mechanical” and “Software” for the printer itself and elements from the “Service” domain, as shown in Section 4.4.2. Additionally there are magnetic and optic properties modeled for the different position sensors, e.g. linear rails with optical displacement sensing and servo motors with magnetic induction for determining the angular displacement.

For the **number of reused system elements** there are 38 elementary functions from the function library used in the functional model of the HBot variant alone, twelve of them coming from applying the pattern. They are structured into eight user-defined functions in four levels of hierarchy. For the CoreXY variant there is a similar number of functions, with most of them being identical to the HBot variant. The behavior simulation model, for instance, for the HBot uses eleven different <<AmesimBlock>> elements from the library to create 52 simulation elements as *part properties* interconnected by 59 *connectors.* Most of these elements are provided by the pattern and the model does not contain any elements for propulsion and control. To evaluate the two kinematic variants, the propulsion and control is added only in the Amesim tool itself to run the simulations. Finally, there are four different services from the service library

reused to model two new custom services for remote inspections and on-site maintenance. Further information on how these elements are used in parts of the system model is given in the previous sections that illustrate the use of the presented libraries and multi-solution patterns.

In total there are three **multi-solution patterns** created for the case study. In addition to the "2D Kinematics" pattern of Section 4.5.2 there is the "Provide Rotational Movement" pattern in the Appendix for different electric motors and another pattern for alternative sensors of rotational mechanical energy. All three patterns offer conceptual solutions in the form of partial models, by correlating library elements with more information to offer coherent subsystems. Through their solution-neutral target functions they enable the reuse of alternative multi-disciplinary conceptual solutions for common problems, e.g. the selection of an electric motor or a sensor. Yet, due to the fact that only limited prior system models were available for the case study, most information content of the patterns was created specifically for it, while aiming to be generally applicable.

For creating the concept model using the libraries and patterns the **modeling workflow** follows the concept design tasks of the VDI 2221 [14] with the level structure of Eigner et al. [80], as shown in Section 4.1 with Figure 18. The figure also display the need for design iterations, which are necessary to create a concept model. This fits to the definition of system architecting being a process to iteratively refine technical specifications [157]. Design iterations during the creation of the case study especially involved refinements of the simulation models, coming from their representation and use in the simulation tool.

In the task definition of the **context level**, the main *requirements* of the 3D printer come from Wölkl [13] with certain refinements, e.g. related to the service modeling in Figure 43. The main *use cases* and the context of the case study are

displayed in Figure 24, where also the printer's main function is derived as a black box with flow-based interfaces.

This is the starting point for the functional decomposition to determine functions and their structure on the **functional level**. The function structure provides a solution-neutral and hence domain-independent basis for the following multi-disciplinary system development. Excerpts of the functional model are shown in the Figure 26 and Figure 48 for three layers of the functional decomposition of how to print in 3D and Figure 27 for functions of the service modeling. The model uses flows from the library together with its elementary functions for those functions that are not further decomposed. This reuse not only clarifies the semantic meaning of the functions, but it also supports the modeling through use of predefined functions with fixed and consistent interfaces.

The search for working principles as potential solutions to the elementary functions constitutes the **principle solution level**. The <<PrincipleSolution>> elements are hereby defined and detailed by various system elements. This step is shown in Figure 34 with a principle solution associated to a simulation model, containing <<AmesimBlock>> elements derived from the behavior simulation library. The principle solution HBot realizes the user-defined function of positioning the liquid plastic while subordinate elementary functions are concretized by elements of the behavior simulation model. Another example of a principle solution of the case study is given in Figure 42. A user-defined function is concretized by two principle solutions, which are detailed by generic services from the service library and different solution principles. Reusing common services helps in extending the prior mechatronic system model into a PSS. Alternatively to the shown behavior or service elements it is also possible to directly link towards structural components, e.g. provided by the structure library

[13]. This relates to the idea of FBS that the behavior is only used where it is needed to find concrete solutions [84].

The modeling of the system structure, i.e.components, and behavior models for simulation further concretize the **technical solution**. For the behavior model there is, for instance, the one of the HBot principle solution of Figure 34 refined. Parts of the simulation model in SysML and Amesim are displayed in Figure 35. Two simulation results are shown in Figure 37, where they are used to specify the orthogonal load on the used linear bearings. These results illustrate the use of the behavior simulation library as a means of planning the comparative evaluation and potential trade-offs of the two investigated configurations. Examples of the structural view in the case study are the *part properties* of the HBot kinematics subsystem in Figure 47 and the IBD in Figure 49, which shows the wiring of the basic 3D printer configuration with its “Arduino Board” and stepper motors [144].

Modeling the different 3D printer **configurations and their variants** follows the approach of Weilkiens et al. [59]. The SysML model structure of the case study, e.g. shown in Figure 12, Figure 13 and Figure 15, combines the variation management with the multi-level system representation of [80]. In general there is to say that the configuration management in SysML is possible, yet needs further future improvements, e.g. through enhanced tool support.

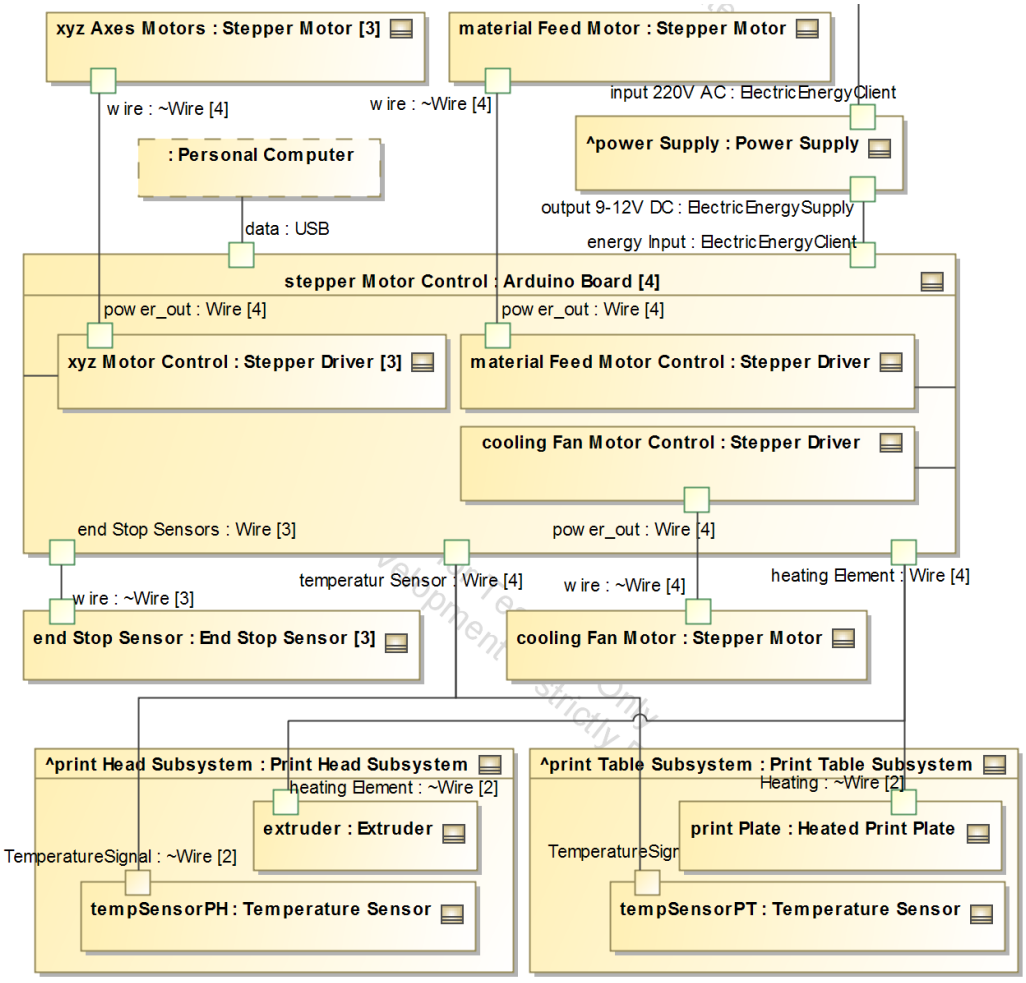


Figure 49: Wiring schema of the basic configuration of the 3D printer with stepper motors

A schematic **overview of the modeling approach** and model structure is given in Figure 50. It shows the main modeling elements, the design libraries they come from and simplified relations between them together with a pattern database. There are for example *allocations* between the different levels, *composition* and *decomposition* relations for the function, behavior and structure models, *associations* to refine principle solutions, and a *satisfy* relation to trace back from the detailed system structure to the *requirements*.

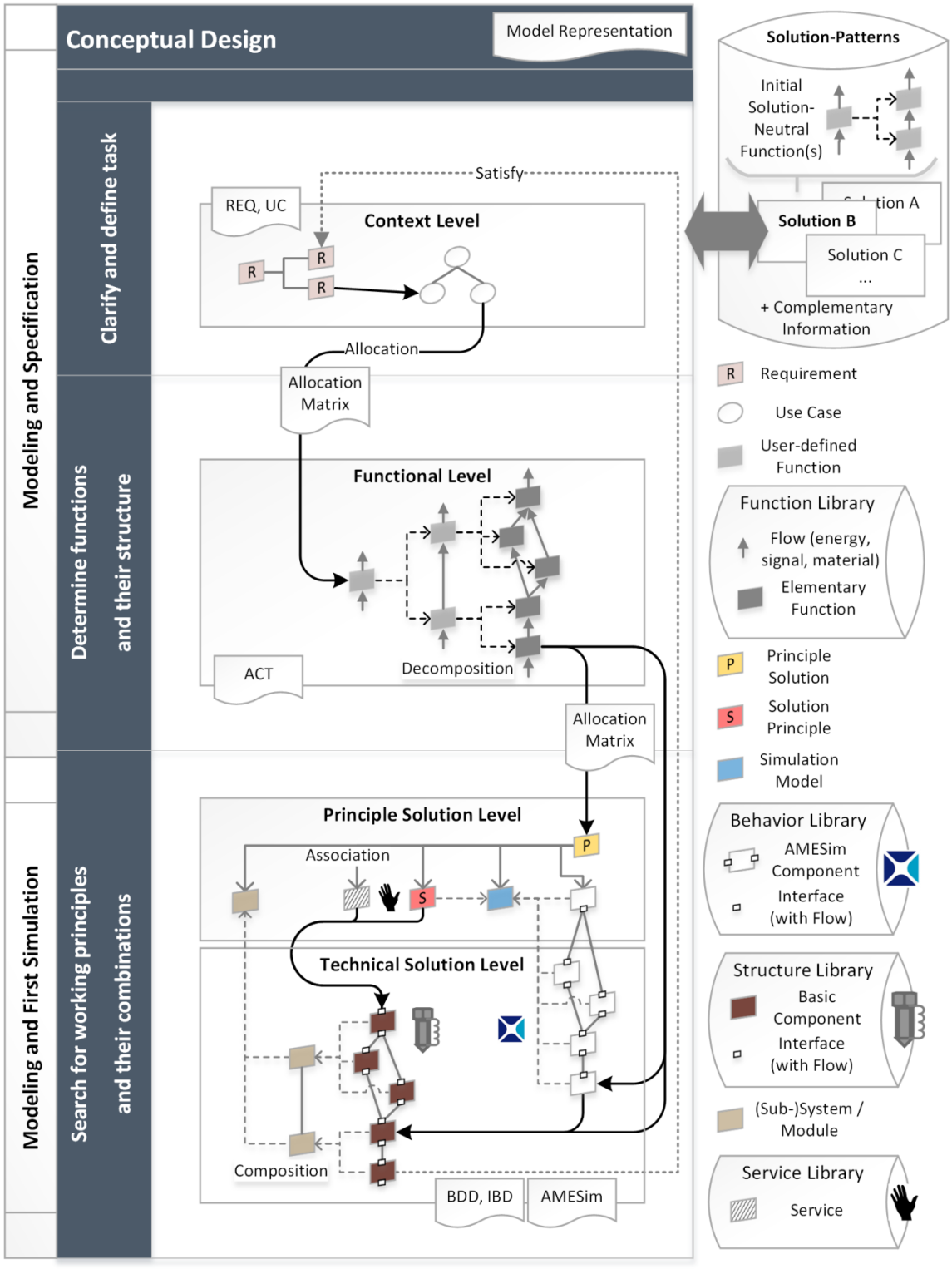


Figure 50: Schematic modeling approach overview

The model-based representation in SysML allows explicit relations between various system elements to represent multi-disciplinary systems. For PSS, for example, there are links shown in Figure 43 between service aspects, the physical system and other elements such as *actors, requirements* or the software to be modeled in UML. Another example for this **traceability** between modeling elements throughout the SysML model is given in Figure 34. There are semantic links between the different levels of abstraction, i.e. “vertically” to the level structure of Figure 17, and there are “horizontal” semantic links between elements on the same level. The “vertical” *allocations* connect functions with elements that represent realizing principle solutions. The “horizontal” links exist for instance as *generalizations* between library elements and their more detailed child elements.

Other examples for the interrelations in the case study model are given in Figure 47 with an allocation matrix or in Figure 51 with a tool generated **relation map**, especially for traceability purposes. It shows the *allocations* from the “Rotational Energy Provision” *requirement* over the elementary function “ElectricalEnergy-RotationalEnergy:Convert” towards the “PrincipleSolution - DC Motor”. This element is *associated* to an Amesim simulation model that contains the simulation element “DC-Motor_emd_DirrectCurrentMachine”. This behavior element is *allocated* to the “DC Motor” *block*, which is also alternatively *allocated* directly from the elementary function. From the “DC Motor” *block* it is traced back towards the *requirement* by a *satisfy* relation. Due to the decision of fulfilling this *requirement* with its function through an electric DC motor the new *requirement* “Electric Power Supply” gets derived that must be fulfilled in a following design iteration.

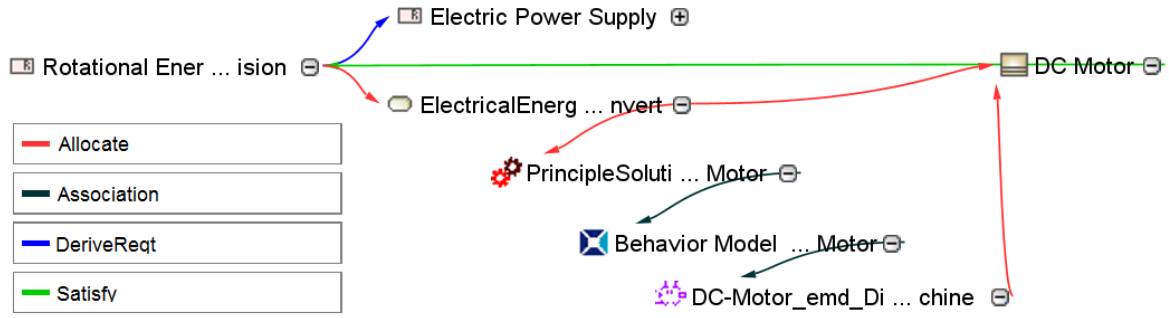


Figure 51: Relation map of "Rotational Energy Provision" *requirement*

To use the elements from the libraries, **specific allocations are required**. These are *allocations* from usage to definition. Such asymmetric allocation are "not generally recommended since [they] can introduce notational ambiguity" [70]. Yet, they are used here since both alternatives, *allocations* of usage and *allocations* of definition do not apply. Using the function library, functions do only exist as *actions*, i.e. using *activities* from the library. *Allocating* to realizing elements from other libraries takes place before these elements are used in the model. Therefore, *allocations* from usage to definition must be used, first.

Finally, when combing all presented design libraries with the multi-solution patterns there is a **total number of ten extra *stereotypes* used**. This, compared to other approaches [72, 134, 140, 141], relatively low number of *stereotypes* is intended to reduce the learning effort of designers when using the presented modeling approach. An overview of the major relations between the used *stereotypes* is given in Figure 52.

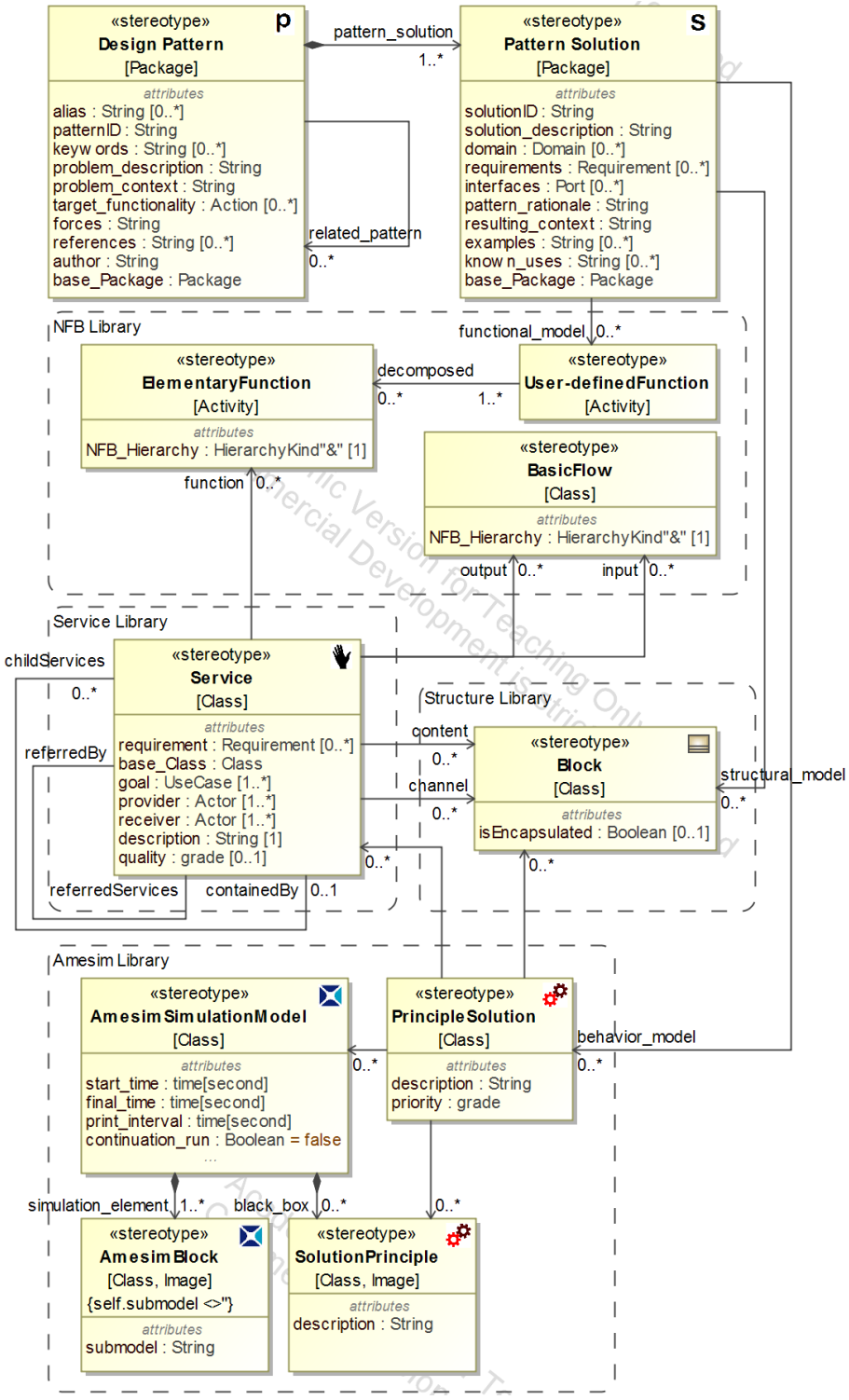


Figure 52: Overview over used *stereotypes*

5. Functional Modeling User Study

Usability has been identified as a key challenge for both, functional modeling [29, 89] and SysML modeling [5, 38]. This section investigates usability of the developed modeling approach in a descriptive user study. It is conducted to analyze the **usability of the SysML function library** developed, to highlight advantages and areas for improvement. The function library serves here as an example for the general library supported modeling approach in SysML. It intends to support the use of the FB and more formal and guided functional modeling. Other benefits are expected from reusing library elements, rather than having to define them all from scratch.

5.1. Experiment Setup

The experiment setup starts with the investigated research hypotheses and the experimental factors. Derived from these follow the individual performance measures and the general procedure of the study including its tasks and instructions. The section ends with the master models as references for these tasks.

5.1.1. Hypotheses and Experimental Factors

To determine the actual effects of the function library as modeling support, its availability is defined as the single **experimental factor** under investigation. This means that the FB is either given to the participants as tables printed out on paper, based on [15], or that it is provided with the library in SysML. The SysML library is then incorporated as a read-only module.

Considering the usefulness of the library, “time, quality and performance are all […] benefiting from reuse” [105] of proven knowledge and model elements. Based on this the following **hypotheses** are derived:

- Using the SysML library reduces the perceived modeling workload.

- Using the SysML library leads to “better” models.
  - Using the library leads to bigger models.
  - Using the library leads to more use of the FB, resulting in more formal models.
  - Using the library leads to less errors in the models.
  - Using the library leads to a broader coverage of the general top-level functions of the system, compared relatively to master solution models.

5.1.2. Performance Measures

Two approaches are used in the conducted user study to complement each other in measuring the modeling performance of the participants. The first is a **general questionnaire and the NASA’s TLX test** [158] to mainly determine the perceived workload and the user acceptance after the tasks. The used questionnaire is given in Appendix B – User Study Questionnaire. Questionnaires and established and recognized tests, such as RSME [159] or NASA-TLX are the best practices when it comes to determining the user acceptance. The TLX test uses six factors, each on a scale from low to high workload, going from zero to 100 in increments of five. The factors are: mental demand, physical demand, temporal demand, performance, effort and frustration. These factors are weighted towards each other by letting the participants compare them pairwise in a random order at the end of the experiment.

The second approach is an **analysis of the created models**. In general there exist different approaches to assess the quality of functional models as a mean to determine the modeling performance. The ISO 9241-11 [160] standard distinguishes between effectiveness, efficiency and user acceptance. A common method to measure the effectiveness for object-oriented modeling is by counting

different model elements and their connections, as for example done for entity relationship diagrams in [161]. The model size can also take incorrect model elements separately into account to consider the model correctness, too. Besides this there are relevant partial solutions defined and used for comparison to determine the grade of task completion for the modeling effectiveness [162]. For modeling efficiency, modeling time is an additional factor along with modeling effectiveness [160]. In this user study the maximum modeling time is fixed for the participants. This implies that the effectiveness measure already includes a statement about modeling efficiency [162].

To conclude, the modeling effectiveness is mainly determined here by using the model analysis parameters of covered top-level functions to determine the **grade of task completion**. The reasons for this are that the model completeness and correctness only consider the model size [162], but neglect the information content contained. Determining a grade for the model correctness is also not chosen since it requires the analysis of semantic modeling errors. The collection of semantic errors, e.g. contradicting the common sense of how a required system might work, is not done to avoid potential bias when determining what counts as such an error. The same argument applies for not choosing another method of establishing the quality of functional models from Nagel et al. [163]. It defines problem specific questions that have only yes or no answers, which are counted. A correct and unbiased selection of answers might be ensured through multiple evaluators, but an unbiased selection of questions seems to be more problematic.

In total, the following **parameters** are determined from the models:

- Number of functions as nodes
- Number of flows, i.e. edges connecting the functions
    - Their combined number for the model size

- Number of functions from the FB
    - Relative number of functions from the FB
- Number of flow types from the FB
- Number of functions from the SysML function library
- Number of flow types from the SysML function library
- Relative number of top-level functions, compared to the full master models
- Relative number of top-level functions, compared to the pruned master models (for both master models see Section 5.1.4.)
- Number of syntax errors in SysML, including the connectivity of the functions and flows

5.1.3. Experimental Procedure

The experiment is executed as part of a tools course teaching activity for mechanical engineering students at the ETH Zurich. Of the 11 participants, 8 are bachelor students and the remaining 3 are doctoral students in mechanical engineering. All participants have no prior knowledge or experience of using SysML, creating functional models or the used modeling tool. The modeling is supervised by the author and questions were answered about using SysML, general functional modeling and the modeling tool, but not about how to do the specific modeling task with its functional decomposition. The procedure of the experiment is displayed in Figure 53. It contains three afternoon sessions with the first two serving as a learning phase for the participants. On day 1 SysML is introduced with the PKG, the UC and the ACT as a first step into MBSE. Additionally the method of functional modeling with functions from the FB is presented together with guidelines to decompose a target main function [23]. On day 2 the REQ, the BDD and the concept of cross-cutting relations in SysML are given together with the function library and its use.

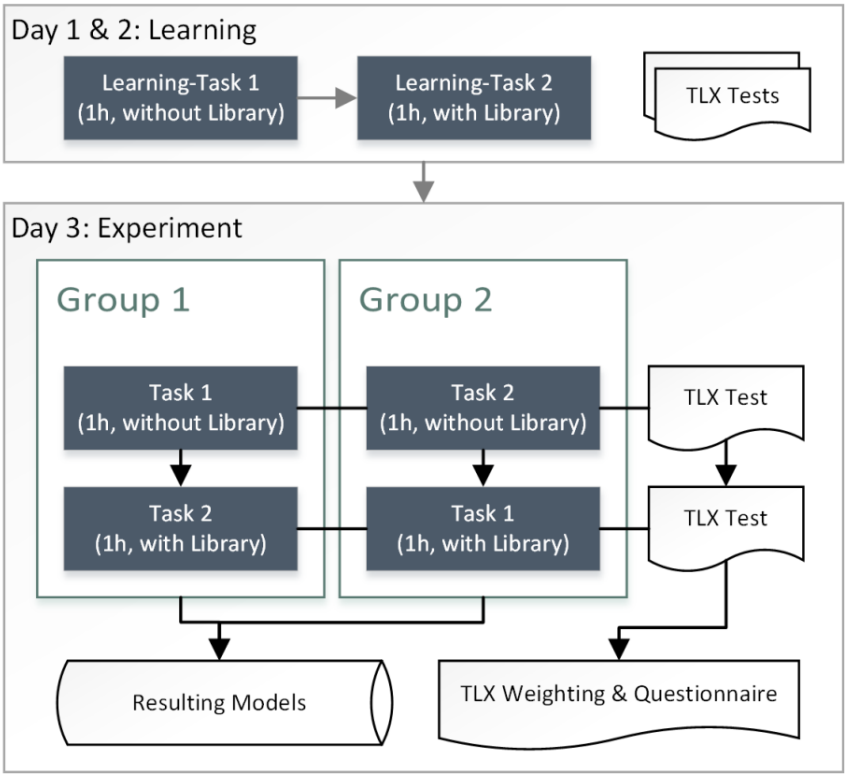


Figure 53: User study experiment plan

The tasks for **learning functional modeling** on day 1 and 2 are the creation of the two main aspects of an electric bike, to drive with support from the electric motor on day 1 and to break while recuperating energy on day 2. These tasks are selected to be fairly known by the participants and to cover a mechatronic system. The first task is done without the library and the second task is done with it to have the participants learn both ways equally. The maximum time to complete the functional models is one hour each. Instructions given to the participants consist out of general goal descriptions together with multiple exemplary models, provided on paper and as an SysML model in the modeling tool. When not having the library, the participants are nevertheless instructed to use the terms of the FB, as provided on paper. When having the library, it is assumed to be used, too. After the required introduction of the TLX

test with its scale and six factors, it is done for each task to familiarize the participants with it. To gain initial insights into the behavior of the participants, a pilot study is done with three participants for day 1 of the course. Because the analysis of the results in Section 5.2 focuses exclusively on the modeling on day 3, these three participants are treated equally as the rest for the analysis.

On day 3, where the actual **experiment takes place**, two modeling tasks are carried out by the participants, as displayed in Figure 53. Both tasks are about the functional models of a small mechatronic consumer device, an automatic bean-to-cup coffee maker. The two tasks represent the main aspects of such a coffee maker, with modeling of the functionality of brewing fresh coffee and grinding coffee beans into coffee powder. The modeling support through the library is not provided for the first task and given for the second task. Half of the randomly assigned participants do the brewing coffee task first and the grinding coffee beans task second, while the other half of the participants performs the two tasks in reversed order. This way a potential difference in the complexity and required effort between the two tasks is compensated, while the maximum case size can be used to test for the experimental factor of modeling with or without library. On day 3 there are further TLX tests performed for each task and the concluding weighting of the TLX scales is done as required. For this the scale factors are randomly pairwise compared, it is counted how many times one of the factors is determined to be more important, which then results in an average weight for each factor. Besides this procedure for the experiment, the participants are introduced to the SysML IBD as part of the tools course.

5.1.4. Master Models

The master models define potential target solutions for the two modeling tasks of brewing coffee and grinding coffee beans. They are made to provide an

agreed-on **mean for comparison of the participants' solutions**. Their direct content in form of their elementary functions and how they are connected with each other is not used for the comparison to avoid bias by different ways of creating these models. Likewise, there are two different configurations of the functional models used: One complete model and a pruned model, following the rules of Caldwell et al. [164]. Examples for these pruning rules are to remove all "Import" and "Export" functions or all "Couple", "Join", and "Link" functions referring to any type of "Solid". Pruned models are models with a reduced number of functions to focus on the essential information and improve readability. Three of the total four models are given in Figure 54, Figure 55 and Figure 56.

Figure 54 shows the complete functional model of the **grinding coffee beans task**, as implemented on an ACT using the function library. It uses 13 functions, 12 elementary ones decomposing the single main function, and 18 *object flows* connecting the functions. It also shows, framed by dotted lines, the six aspect, which define the top-level functions. They serve as more objective parameters in a comparison to access model quality. They are needed to be covered for fulfilling the main function. To grind coffee beans they include the following six aspects of human control, providing and converting energy, providing and grinding coffee beans and exporting the resulting grounded coffee.

The **pruned version** of the functional model of grinding coffee beans is shown in Figure 55 with only 6 elementary functions that still cover 5 of the original 6 top-level functions. The functionality of exporting the grounded coffee is completely removed by the pruning rules. The pruned model consists in total out of 7 functions and 12 *object flows*.

The **task of brewing coffee** consists of 8 top-level functions, covering aspects of human control, energy, water and coffee powder provision, heating

the water, combining it with the coffee powder, separating them again and exporting the brewed coffee. For this 18 functions are used together with 27 *object flows*. The pruned version of the brewing model, as displayed in Figure 56, uses only 10 functions with 18 *object flows* and has again one top-level function less, because of omitting the export of the coffee.

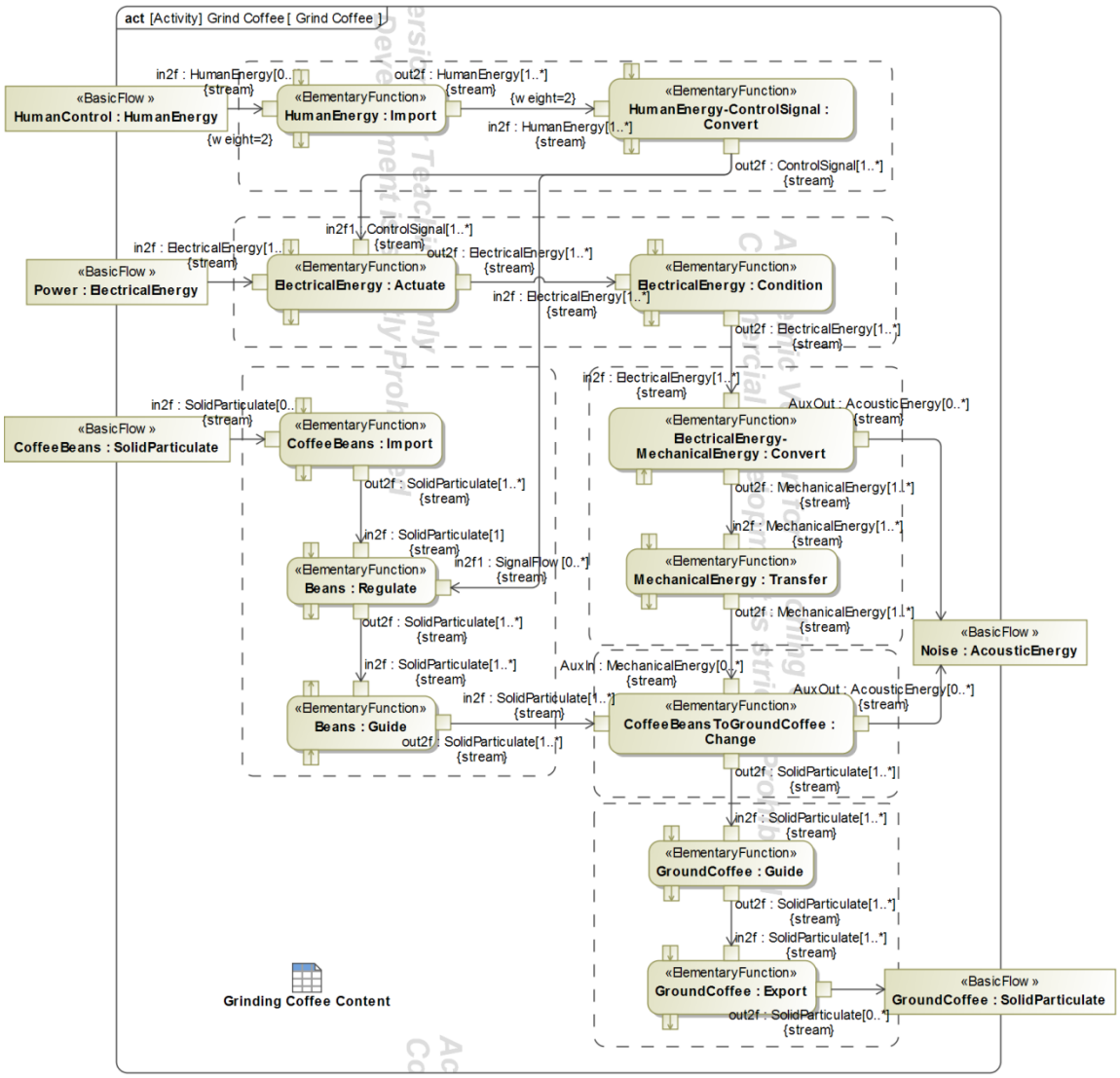


Figure 54: Full functional model of “Grind Coffee Beans” task, highlighting its six top-level functions

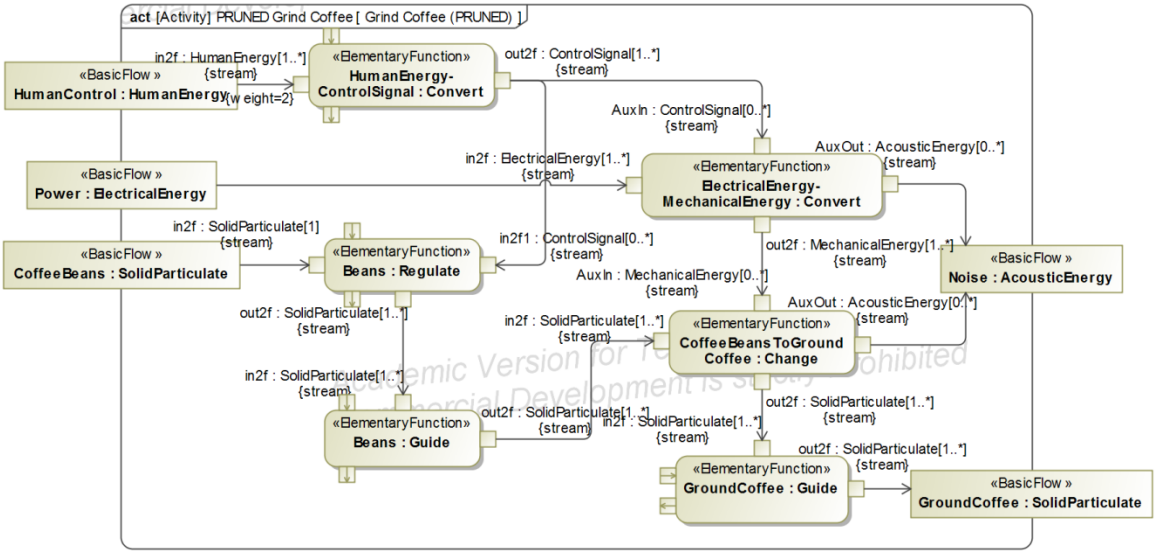


Figure 55: Pruned functional model of “Grind Coffee Beans” task

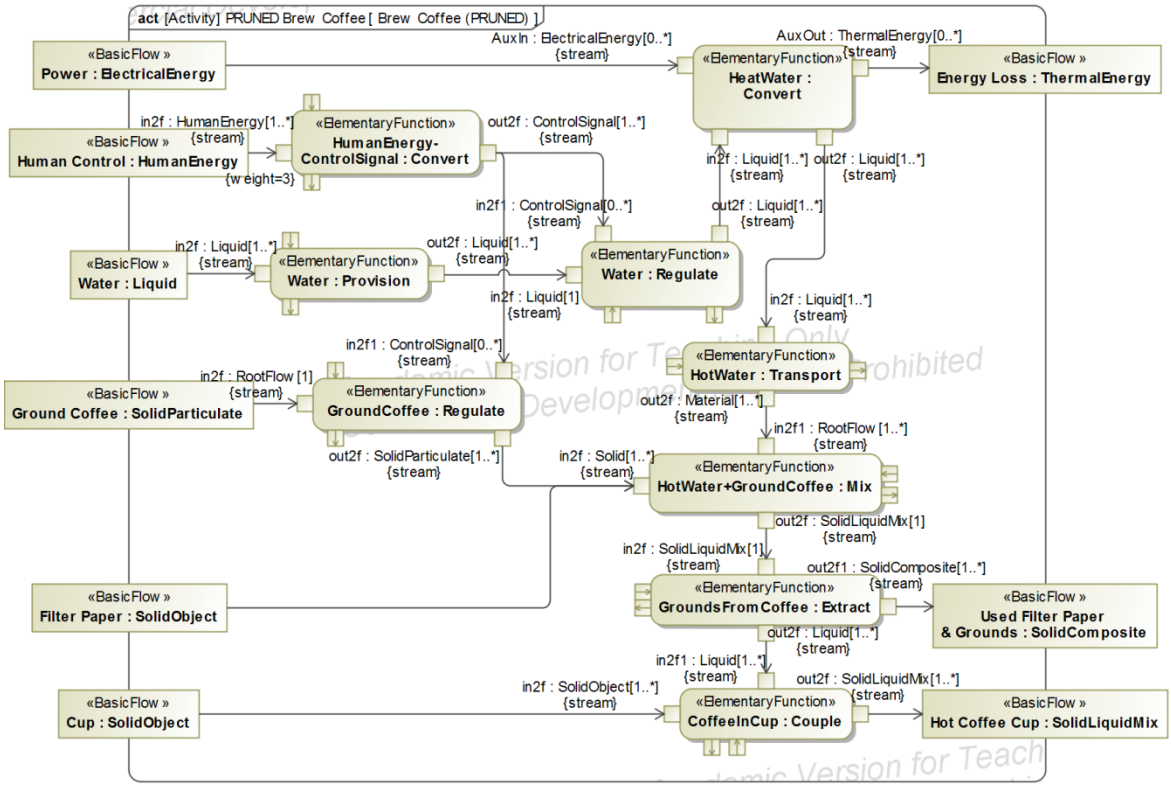


Figure 56: Pruned functional model of “Brew Coffee” task

5.2. Experiment Results

To look at the modeling effectiveness with respect to the use of the function library, the results of the questionnaires about user acceptance and perceived workload are described together with the results of the different functional modeling tasks.

5.2.1. Questionnaire and TLX Test Results

The given **questionnaire** contains questions to assess the participants prior experiences with SysML, the modeling tool, functional modeling and concept modeling using these elements. It also asks about their importance for early product development phases. The average approval rating is around 60 on a scale from 0 to 100, with 0 meaning complete disagreement and 100 meaning complete agreement. Only the aspect of functional modeling as a means to define what a system is supposed to do before defining its structure is rated highly important with an average approval rating of 87. The questions about the use of the function library reached approval ratings around 59 to 60 for the statements that modeling with the library in SysML improved the resulting model and the modeling process.

The general **results of the TLX tests** are displayed in Figure 57, showing the means of weighted ratings for the perceived workload for each of the four measurements. The first two measurements are for the learning phase on day 1 and 2, with day 1 providing the FB only on paper and day 2 providing it with the SysML library. The last two measurements are from day 3 of the course, with again initially not providing the library before providing it. The values for the mean overall workload are: 53.6 on day 1, 60.1 on day 2, 41.1 on the first measurement of day 3 and 49.6 on the second measurement of day 3. These lower values for the perceived workload on day 3 indicate a learning effect of the participants. The standard deviations, coming from the 11 participants, are: 10.4,

10.7, 11.5 and 14.5 in the same order. The standard errors, which are indicated from left to right in Figure 57 are: 3.1, 3.2, 3.5 and 4.4. The standard deviation quantifies the scatter of the individual data points, i.e. how many of the values vary from one another, while the standard error of the mean quantifies how precisely the true mean of the population is known. It takes into account both the sample size and the value of the standard deviation.

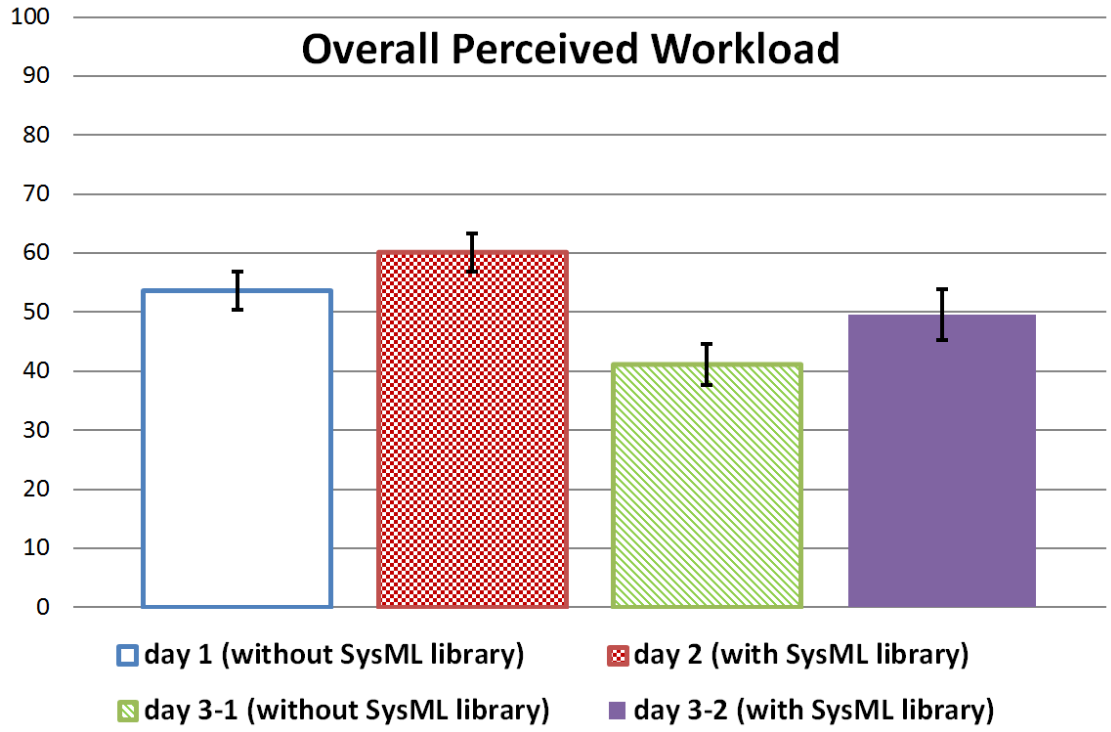


Figure 57: Overall perceived overall workload with standard errors from the TLX tests by the means of weighted ratings

More details of day 3 with the results for the different factors and their weightings are shown in Figure 58 and Figure 59. For the tasks of performing functional modeling in SysML the following weights are derived for the six factors: 3.6 for mental demand, 0.5 for physical demand, 2.5 for temporal demand, 2.8 for performance, 3.4 for effort and 2.4 for the frustration aspect. These values are

displayed relatively as width of the columns on the horizontal axis in Figure 58 and Figure 59. The mean workload values for the six factors in Figure 58 are: 45.5 for mental demand, 13.6 for physical demand, 37.3 for temporal demand, 41.8 for performance, 44.5 for effort and 20.9 for the frustration aspect. Combined with their individual weights they make up the overall workload value from Figure 57, the 41.1 for the first measurement on day 3. For the second measurement on the third day with the library, the following values are derived: 52.7 for the mental demand, 27.3 for the physical demand, 41.8 for temporal demand, 50.1 for performance, 52.3 for effort and 40 for the frustration aspect. These values are shown in Figure 59 together with again the same weighting factors for the width of the columns. They also make up the overall workload value of 49.6 for the second measurement of day 3 of Figure 57.

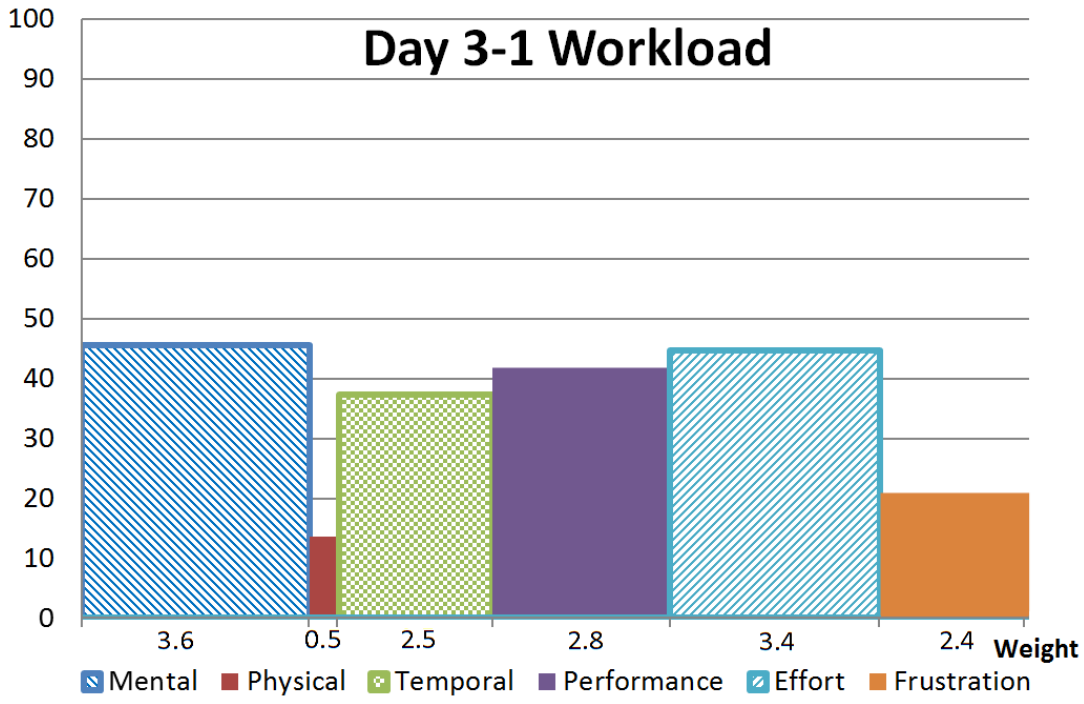


Figure 58: Perceived workload on day 3 without library support, shown for each factor with its relative weighting

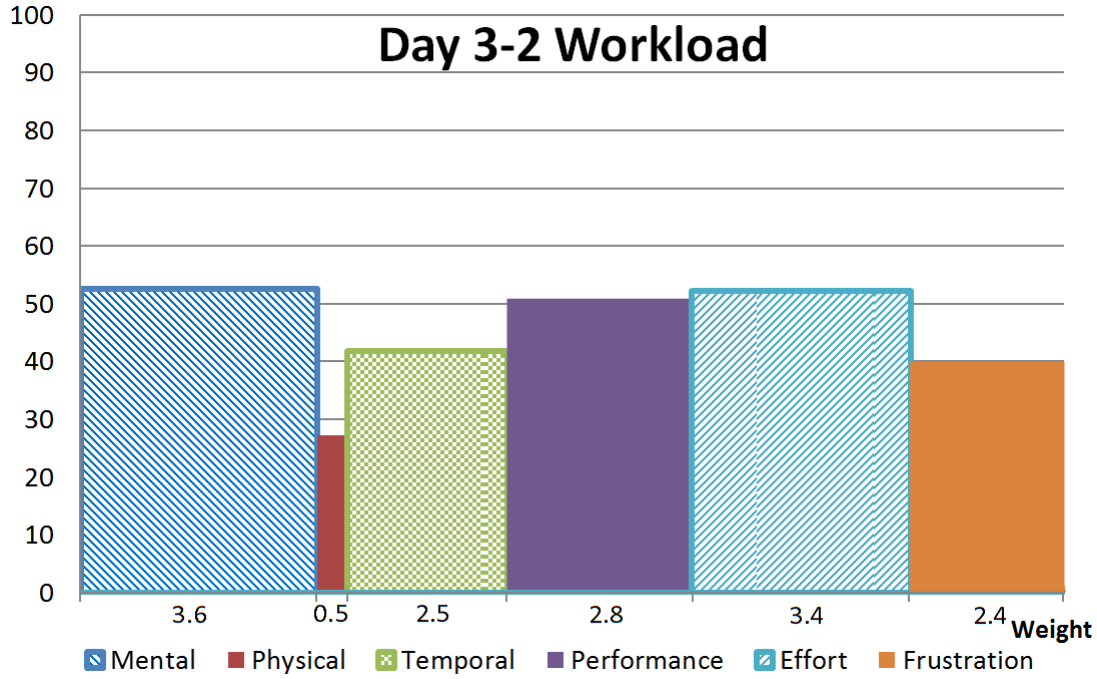


Figure 59: Perceived workload on day 3 with library support, shown for each factor with its relative weighting

5.2.2. Model Analysis Results

The model analysis uses a total of 22 models, coming from the 11 participants, each performing two tasks. IBM SPSS Statistics v22 is used to check for normal distribution of the data, to perform Wilcoxon tests [165] and check for correlations between the parameters according to Spearman [166].

The **results are summarized in Table 2**, showing the mean values of the number of functions, flows, the model size, the functions of the FB and their ratio, the flow types of the FB, the functions and flows from the library, the covered top-level functions with their ratio compared to the two master models and the number of syntax errors. For each of these parameters the mean value is given for modeling with and without the library, as well as for modeling the two tasks of grinding and brewing. Table 2 also includes the standard deviations and standard errors for the modeling with and without function library for each task.

Table 2 shows that the **mean model sizes** for the two modeling tasks are not equal, with grinding coffee beans having an average size of 18.91 and brewing coffee having 24.18. Yet, due to the two groups performing both tasks in reversed order there is no impact on the results to be expected. Also the workload measurements and questionnaires indicate a comparable task difficulty.

(N = 11)	Mean Value				Standard Deviation		Standard Error	
	without library	with library	grinding task	brewing task	without library	with library	without library	with library
functions	8.09	8.82	7.82	9.09	3.36	1.66	1.01	0.50
flows	12.18	14.00	11.09	15.09	4.26	4.60	1.29	1.39
model size	20.27	22.82	18.91	24.18	7.39	5.71	2.23	1.72
FB functions	2.82	7.09	5.00	4.91	3.31	2.70	1.00	0.81
ratio of FB functions	0.31	0.78	0.60	0.49	0.30	0.26	0.09	0.08
FB flow types	3.27	4.91	3.82	4.36	2.20	2.59	0.66	0.78
functions from library	0.00	7.09	3.91	3.18	0.00	2.70	0.00	0.81
flows from library	0.00	4.55	2.55	2.00	0.00	1.81	0.00	0.55
top-level functions	3.27	4.91	3.45	4.73	1.42	1.51	0.43	0.46
ratio of top-level functions (full master models)	0.46	0.71	0.58	0.59	0.18	0.19	0.05	0.06
ratio of top-level functions (pruned master models)	0.54	0.83	0.69	0.68	0.21	0.22	0.06	0.07
syntax errors	0.64	0.36	0.64	0.36	1.03	0.67	0.31	0.20

Table 2: Model analysis results with standard deviation and error

The **Wilcoxon test** [165] is a non-parametric statistical comparison of the average of two samples, similar to the t-test. Unlike the t-test the Wilcoxon test works with metric data that has no normal distribution and comes from two dependent measurements. Although two different tasks are part of the experiment, the two data sets of modeling with and without the library are dependent from each other, since the same participants are involved. The number of data points is equivalent to the number of participants: N = 11. Also

the data is not normally distributed, which is checked for the differences of the two samples by Kolmogorov-Smirnov tests and graphically by histograms. Finally, the Wilcoxon test is recommended for relatively small sample sizes, as in this experiment. It checks, if the null hypothesis that the average signed rank of two dependent samples is zero, i.e. that the factor under investigation has no impact. Its p-value is the probability for the real mean value differences between two compared groups being equal or bigger than the actual observed results. The Z-value is a standardized and normally distributed signed number of standard deviations by which the value of a data point is above the mean value of what is being observed or measured. Results are statistically significant if p is smaller than the critical value of 5% and Z is outside its critical values of ±1.96. The results of the Wilcoxon test show several statistically significant relations, as displayed in Table 3.

(N = 11)	**mean values**				**Z**	**p**
	without library	with library	grinding task	brewing task		
functions	8.09	8.82	7.82	9.09	-0.625	0.532
flows	12.18	14.00	11.09	15.09	-0.972	0.332
model size	20.27	22.82	18.91	24.18	-0.868	0.385
FB functions	2.82	7.09	5.00	4.91	**-2,301**	**0.021**
ratio of FB functions	0.31	0.78	0.60	0.49	**-2,803**	**0.005**
FB flow types	3.27	4.91	3.82	4.36	**-2,113**	**0.035**
functions from library	0.00	7.09	3.91	3.18	**-2,820**	**0.005**
flows from library	0.00	4.55	2.55	2.00	**-2,869**	**0.004**
top-level functions	3.27	4.91	3.45	4.73	-1.901	0.057
ratio of top-level functions (full master models)	0.46	0.71	0.58	0.59	**-2,346**	**0.019**
ratio of top-level functions (pruned master models)	0.54	0.83	0.69	0.68	**-2.667**	**0.008**
syntax errors	0.64	0.36	0.64	0.36	-0.816	0.414

Table 3: Wilcoxon test results (significant results: bold)

The number of used **functions from the FB** is significantly higher, with the test statistic Z = -2,301 and the asymptotic p-value p = 0.021. The mean value goes from 2.82 functions to 7.09 functions from the FB. This relation is illustrated by a box plot in Figure 60 with error bars for the 95% confidence interval and two outliers. Analog, there is an even stronger significant relation for the relative number of functions from the FB, with mean values going from 0.31 to 0.78, Z = -2,803 and p = 0.005. This is shown in Figure 61. These two results indicate that the availability of the library influences the use of the FB. This also means that poor use of the FB is observed when it is only provided on paper compared to the SysML library.

Having or not having the library also significantly influences the number of functions used from the library. Here Z = -2,820, p = 0.005 and the mean values go from 0 to 7.09. This means that at an average number of 8.82 functions per model there are 7 of them reused from the library, if provided. This also indicates good acceptance of the SysML library.

Similar significant results are observed regarding the **flow types** that show that the number of flows used from the library rises from a mean value of 0 to 4.55 different flow types, if the library is provided. Here Z = -2,869 and the p-value is p = 0.004. With a total of average 4.91 different flow types from the FB used, this indicates again a good acceptance of the library by the participants. The use of the flow types from the SysML library goes together with a higher use of flow types defined in the FB. For this there is Z = -2,113, p = 0.035 and the means are 3.27 without library and 4,91 with library, as displayed in Table 2.

Figure 60: Number of functions from the FB with and without function library (with error bars and two outliers)

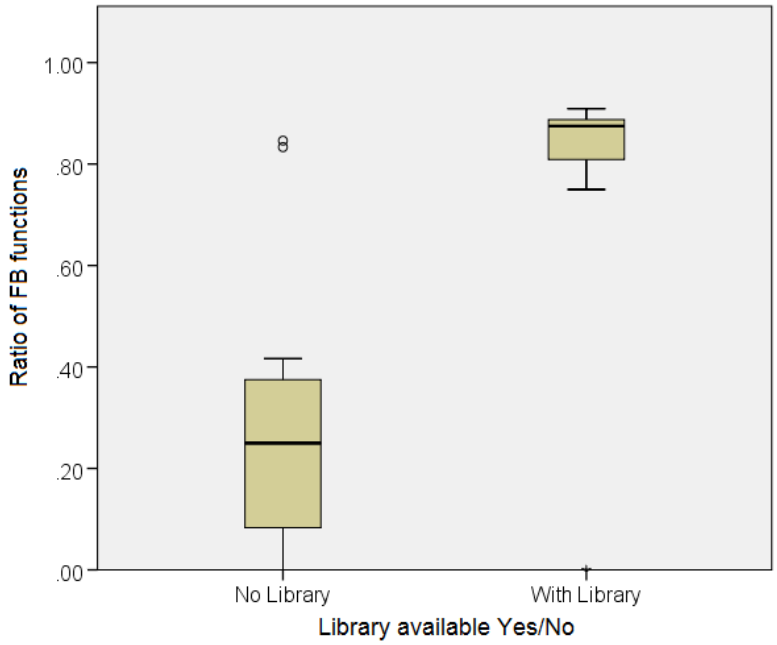


Figure 61: Relative number of functions from the FB with and without function library (with error bars and three outliers)

To make statements about the modeling effectiveness and efficiency the grade of task completion [162] is used by comparing the created functional models to the master models and their top-level functions. For the **relative number of top-level functions** there exists a statistically significant relation depending on the availability of the function library. The covered ratio rises from 0.46 to 0.71 for the comparison with the full master models and from 0.54 to 0.83 for the pruned master models, when providing the library. For the full master model Z = -2,346 and p = 0.019. For the pruned version there is Z = -2.667 and with p = 0.008, which is even below the lower critical threshold of 1%. This relation is displayed in the plot of Figure 62. It indicates that the modeling with the library is more efficient, since its resulting models provide a broader coverage of the necessary top-level functions of the modeling task.

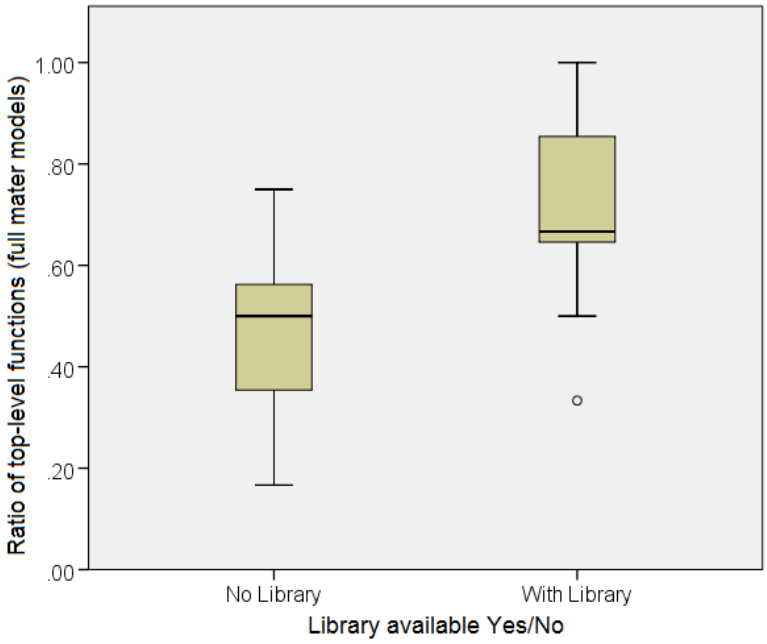


Figure 62: Relative number of covered top-level functions compared to the full master models with and without function library (with error bars and one outlier)

For the **other parameters** of Table 2 there are no important and statistically significant correlations found that relate to the provision of the function library. Yet, the tendency is visible of bigger and therefore more complete models with less modeling errors when having library support. Also, the relatively low number of syntax errors mainly comes from the modeling tool itself with its inherent constraints.

Bivariate correlations are calculated according to Spearman [166] because the data is not normally distributed. It needs ranked data as input. The number of data points is equivalent to all analyzed models: N = 22. The strength of the correlations is determined by Spearman's correlation coefficient r_s. It goes from -1 for a perfect negative association of ranks, over 0 to indicate no association between ranks, to +1 for a perfect positive association of ranks. Following Cohen [167] the absolute value of r_s being above 0.1 equals a weak effect, above 0.3 equals a medium effect and above 0.5 is a strong effect. Again the p-value is calculated to determine statistical significance.

A total of 47 statistically **significant bivariate correlations** are identified for the investigated parameters, as displayed in Table 4. One example for a noticeable significant correlations exists between the number of functions used from the SysML library and the relative number of functions from the FB. The correlation is strong with a coefficient $r_s = 0.802$ and highly significant with $p < 0.001$. It indicates again that the initial assumption that the participants use the FB provided on paper must be rejected. Functions from the FB are only used when they are provided by the SysML function library.

Other significant and strong relations exist for the **ratio of functions used from the FB**. There is one correlation for the number of flows from the FB with $r_s = 0.641$ and $p = 0.001$ and another one for the model size with $r_s = 0.505$ and $p = 0.017$. A strong correlation exists also between the number of flows from the FB

(N = 22) Top value: r_s Lower value: p-value	functions	flows	model size	FB functions	ratio of FB functions	FB flow types	functions from library	flows from library	top-level functions	ratio of top-level functions (f.)	ratio of top-level functions (p.)	syntax errors
functions	-											
flows	**0.763** **0.001**	-										
model size	**0.879** **0.001**	**0.971** **0.001**	-									
FB functions	**0.600** **0.003**	**0.489** **0.021**	**0.567** **0.006**	-								
ratio of FB functions	0.402 0.064	**0.503** **0.017**	**0.505** **0.017**	**0.902** **0.001**	-							
FB flow types	0.352 0.108	**0.506** **0.016**	**0.508** **0.016**	**0.602** **0.003**	**0.641** **0.001**	-						
functions from library	0.347 0.113	0.400 0.065	**0.438** **0.042**	**0.782** **0.001**	**0.802** **0.001**	**0.961** **0.001**	-					
flows from library	0.303 0.170	0.396 0.068	0.422 0.051	**0.745** **0.001**	**0.781** **0.001**	**0.568** **0.006**	**0.961** **0.001**	-				
top-level functions	**0.465** **0.029**	**0.648** **0.001**	**0.631** **0.002**	**0.540** **0.009**	**0.536** **0.010**	**0.582** **0.004**	**0.572** **0.005**	**0.569** **0.006**	-			
ratio of top-level functions (f.)	**0.476** **0.025**	**0.578** **0.005**	**0.585** **0.004**	**0.621** **0.002**	**0.616** **0.002**	**0.707** **0.001**	**0.665** **0.001**	**0.685** **0.001**	**0.925** **0.001**	-	-	
ratio of top-level functions (p.)	**0.472** **0.027**	**0.559** **0.007**	**0.570** **0.006**	**0.594** **0.004**	**0.584** **0.004**	**0.705** **0.001**	**0.648** **0.001**	**0.672** **0.001**	**0.905** **0.001**	-	-	
syntax errors (all negative r_s)	0.338 0.124	0.379 0.082	0.420 0.052	0.096 0.670	0.116 0.607	0.213 0.342	0.347 0.113	0.282 0.204	0.305 0.168	0.310 0.161	0.326 0.138	-

Table 4: Spearman test results for bivariate correlations (significant results: bold)

and the model size with r_s = 0.508 and p = 0.016. These interlinked correlations allow the argumentation that either the better performing participants, i.e. those that made larger models, also used more functions and flows from the FB or that use of the FB itself results in more complete and larger models.

Yet, regarding the aspect of model quality there is the **grade of task completion** favored before the model size for the model completeness. For this there exists a highly significant and strong correlation between the ratio of functions from the FB and the ratio of covered top-level functions with $r_s = 0.616$ and $p = 0.002$. The data points for these two parameters are displayed in Figure 63 with respect to the provision of the function library. In Figure 63 there is to note that with a higher relative number of FB functions there comes a higher relative coverage of the required top-level functions. Figure 63 also displays that the provision of the function library leads to higher relative use of FB functions and covered top-level functions, which is shown before in Figure 60 and Figure 62. Therefore, it can be reasoned that the SysML function library leads to relatively more terms used from the FB and hence to more formal models. Also it leads to a higher grade of task completion with more covered top-level functions.

Equivalent results are derived when using the **pruned master models** for comparison instead of the full versions. Its existence to avoid potential bias through a single reference point is therefore superfluous in this case. An example for this is shown for the significant relation of the relative number of top-level functions depending on the availability of the function library.

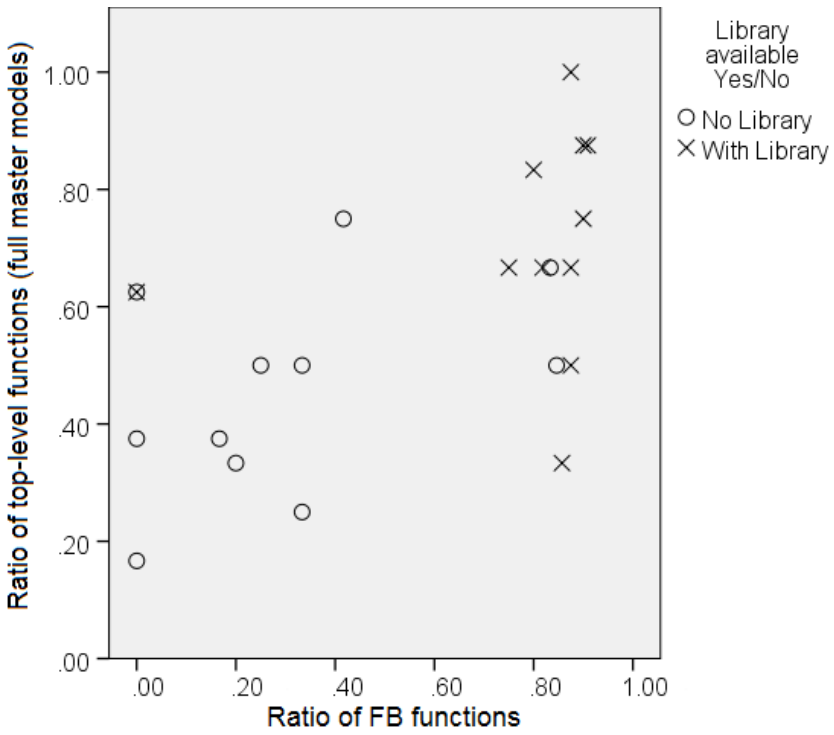


Figure 63: Correlation between ratio of FB functions and covered top-level functions relative to full master models

6. Discussion

In this thesis design libraries and multi-solution patterns are presented to support the concept design in SysML. The following section discusses their use for the case study with its benefits and limitations. There are several **general benefits** expected when reusing knowledge and models in object-oriented modeling. For example when reusing UML models and architectures in software development, there are certain advantages to be expected [39, 41]: a better and faster development, better understanding of the model and reduced development risk. This is investigated in the user study of Section 5 for the function library, which is discussed in the following Section 6.1.1.

These reuse benefits come with certain **general disadvantages** in the form of additional effort for setting up and maintaining the libraries and patterns with their content. Also the designers need training about using the patterns and the libraries with their *stereotypes* and imposed formality. A demonstration for this initially higher workload with library support is given with the user study. Consequently, to keep this additional learning effort small, it is aimed to limit the number of SysML extensions, as indicated in Figure 52.

When reusing solutions from patterns or model elements from libraries, these model elements can also restrict the modeling freedom e.g. with their higher formality. While the reuse provides a certain modeling and design guidance, it can also restrict innovation. Following the definition of innovation from Weber and Husung [124] there are two possibilities for creating product innovation: by a novel combination of existing elements and by creating something completely novel. The first way is supported by the offered reuse knowledge, while its provision might mislead the designers away from an innovation in the form of something completely novel.

6.1. Modeling with Library Support

Modeling with the presented libraries raises the **model formality** due to reusing formally captured model elements with their defined interfaces that capture design knowledge from existing and established knowledge bases. The higher model formality has advantages of better consistency and a reduced ambiguity for a “clear communication and sustainable documentation” [37] together with a potentially better understanding of the model. It further enables computational support, e.g. model consistency checking or eventually automated model synthesis [9, 45]. Further formality comes from the use of a standardized modeling language in the form of SysML. Extending SysML through the *stereotypes* in the libraries allows a wide range of applications, including complex and multi-disciplinary systems [34].

6.1.1. Modeling with the Function Library

Using the function library comes with certain modeling guidelines to support functional decomposition by reusing proven design knowledge. This design knowledge formalizes the syntax and semantics of functions and flows from the FB [15] into standardized SysML model elements. This offers the advantages of improved model formality, better model consistency and avoidance of modeling errors as well as model ambiguity for handling complex multi-disciplinary systems in a solution-neutral way.

One example of the **benefits of the raised formality** of the function library is an improved capability to reuse partial functional models by their standardized interface types from the library. This is for example the case when applying a solution from a pattern. It goes together with a certain model consistency checking of compatible flows in SysML, as seen in Figure 28. Another example is the use of these material, signal and energy flows in the

other libraries to support a possible identification of suitable realizations of functions and maintain traceability between the model levels.

In respect to **modeling restrictions** through the function library with the FB there exists a general difficulty to express certain more static systems without specific energy, signal and material flows. This is because the origin of the FB and its flow-based representation lies in more traditional mechanical engineering [15, 23]. Yet, here there are still *use cases* for a more design purpose oriented functional view, user-defined functions that are not from the FB, the mandatory auxiliary function interfaces and other additions for EFFBDs to use for more modeling freedom. This way modeling flexibility is kept while utilizing defined elementary functions for flow-based models.

Compared to **similar functional modeling approaches**, e.g. FUSE [131] or the FAS method [132, 133] in SysML it should be noted that those approaches lack formalization and modeling support in form of clear defined functions. Yet, the FAS method includes certain additional tool support for grouping of functions, which would be beneficial to be combined with the function library. Compared to the initial version of the library by Wölkl in [13] there is especially the more specified workflow [4], the improved modeling freedom due to auxiliary function interfaces and additions to the function structure from EFFBDs as well as an improved library structure.

This improved structure results from the functional modeling user study of Section 5. The conducted study investigates the practical **usability of the function library.** It considers two hypotheses: the library reduces the workload for the participants and it leads to better functional models. The first hypothesis must be rejected, due to the results of the TLX test. The perceived workload increases for the untrained modelers when using the library. This seemingly contrasts the expected reuse benefit of reducing modeling effort by reusing

predefined elementary functions and defined interfaces, instead of creating them each time anew. In respect to the hypothesis of creating better models, the results show that the library does lead to better models with a significantly higher ratio of covered top-level functions. This is important, since according to Eckert [86] functional modeling support must produce immediate benefit for the users, which is indicated by the improved models.

A major factor for both of these results is that with the very high acceptance of the library, the use of the FB is equally high. This is in contrast to the case where the library is not available, where the participants are using the formal terms of the FB less. This might be caused by the simple convenience of reusing elements provided with the function library in SysML. The conclusion can be drawn that the increased workload as well as the better model quality are very likely a result of the guidance and more formal representation of the functions of the FB itself and not directly caused by the function library. This way the library shows its benefits and usefulness by bringing inexperienced designers towards a more correct use of the FB, achieving more formal and complete functional models.

This reasoning does also fit to **comments of the participants** from the questionnaire: It is said that having the “library forces you to break down the activity further” and that it “triggers [the] thought process”, leading to a “resulting model [that is] more fundamental”. Yet, at the same time the participants felt “very limited by the functions of the library” because of “having to look up functions [and] decide which are appropriate”, which both comes from using the FB and not the function library itself. Therefore the claimed benefit of a higher model quality and completeness of the system seem reasonable. Also the benefit of faster modeling through reusing predefined elementary functions with their

interfaces could most likely not be shown because of the one-sided use of these defined terms from the FB.

For the **conduction of the user study** itself it has been confirmed that the three half days of an ETH tools course is too little time for introducing MBSE, functional modeling, SysML and its modeling tool. Yet, the TLX test results show an improvement for the third day indicating a learning effect for the students.

6.1.2. Modeling with the Behavior Simulation Library

For the **use of the behavior simulation library** certain advantages are identified. It supports the designer by providing proven design knowledge from Amesim ready for reuse. This guides and supports the behavior modeling within the integrated modeling approach in SysML. The design knowledge can be used either as a repository to find solution principles to realize the modeled functionalities or also to plan Amesim simulation models within the SysML model.

The enabled modeling of **Amesim simulation models** directly in SysML supports the capability to simulate aspects of multi-disciplinary systems at an early conceptual design stage, for example to evaluate the system concepts, to perform trade-off studies or even to optimize system parameters. The main aspect of the support is the improved linkage and traceability between the simulation model and other system knowledge in SysML, as shown in Figure 34. Although there is knowledge used from the following structural model, the simulation model represents the expected behavior [83] for the system's expected actions for guidance and evaluation of potential design solutions on a conceptual level. The contrasting structure behavior would include properties of the system that are measured, calculated or derived from the observation a specific design solution, e.g. on the right half of the V-model [79].

The use of the behavior simulation library elements as a knowledge resource to **search for solution principles** supports the identification of behaviors that realize the modeled functions before their concretization, similar to FBS. The library offers hereby formally captured behaviors from multiple disciplines and grades of abstraction, ranging from fundamental physics-based elements to more specific application oriented ones. The elements come with illustrating icons and documentation that contains underlying equations and principles to remove ambiguity for better usage. The addition of functional flow types to the interfaces of the Amesim elements further enhances the linkage between them and the functions that they realize.

Comparing the here presented behavior modeling to **other approaches** in literature it is to note that this is basically a generic library to plan and formally model simulation models in SysML. It does not include the simulation itself. SysML4Modelica [140], a very detailed SysML profile to allow model transformations between SysML and Modelica, for example tries to recreate complete Modelica models with all the required equations within SysML. This makes the modeling in SysML more complicated with a great number of very specific *stereotypes*. It also currently lacks in providing the existing knowledge of Modelica databases. Kerzhner's approach [141] of using a domain-specific language in SysML consequently only considers domain- and application-specific simulation knowledge that must be created anew for each different project. Finally, creating the simulation models directly on SysML PARs lacks the simulation power and flexibility of custom simulation tools as well as their inherent simulation knowledge.

One identified **disadvantage** of using the behavior simulation library is the limited tool support for the model transformations between SysML and Amesim. Going from SysML to Amesim requires additional software [155] and a manual

mapping for each element. It does not use the existing and detailed submodel information in the SysML model that comes with the library. Yet, even more problematic is the not supported transformation from Amesim back to SysML for iterations of the simulation models or to integrate simulation results.

Other disadvantages exist regarding the modeling in SysML itself. The consistency checking in SysML is lacking compared to Amesim, despite offering a certain level of model consistency, as shown in Figure 36. A reason for this lies in the bond-graph origin of Amesim with its causality, which also complicates the modeling in SysML with changing submodels. Since the causality and therefore the submodels may change whenever an element is added to the model it is recommended to initially model without specific submodels assigned. This modeling with unspecific parent elements without concrete submodel is less convenient in SysML than the modeling in Amesim, where elements without fitting submodels cannot be connected and all submodels are specified after the modeling. Finally, there are still some model elements lacking in the simulation tool database, for example to represent stepper motors or the material transport with joint melting in the print head of the 3D printer case study. If needed, such more specific simulation elements must be added together with their underlying differential equations in the simulation tool.

6.1.3. Modeling with the Service Library

There are two main advantages of the service library. First it provides **service modeling in SysML** itself with the extension of SysML in form of the defined *stereotype* and its properties. This relates to the representation of PSS design information as one foundation of PSS design [26], by suggesting formal and object-oriented SysML models with defined service elements. The raised

formality with the service *stereotype* also provides modeling guidance and documentation for understanding and reuse.

SysML is identified to be capable to model and represent PSS through its generic modeling capability of multi-disciplinary systems that includes requirements, the system structure, its behavior as well as the service's behavior and involved stakeholders. Based on literature [29] as well as on the modeling of the case study it is argued that there exists a general analogy between PSS and mechatronic design, especially when using functional models. The modeling in SysML explicitly allows necessary linkage from the services towards their stakeholders, contents and other properties as well as towards other PSS elements, since all model elements are part of a single unified SysML model. This allows traceability not only between solution-neutral functions and their services, but also backwards to *requirements* and *use cases* or forwards to structural elements and software as illustrated in Figure 43. The traceability through the model relates to the identified issues in current PSS design to especially "represent design information to relate service receivers […] and service providers (in terms of their products and services)" [26], to be able to gain "reasoning capabilities to answer queries by tracing back" [26].

The second main advantage of the service library is the supported modeling through the **provision of generic services** with the library elements. These services help designers to identify suitable services for their products by providing realistic possibilities in form of a checklist [17]. The service library reduces the required effort for the PSS design process step of information collection and organization [26] and it supports the modeling in SysML, by providing reusable system elements for specialization. Compared to the original catalogue [17], these service elements are enhanced in SysML with further generic information in form of their properties. Examples are the captured linkage

to solution-neutral functions and *use cases*, which are already used for the service catalogue build-up, or the generic relations to service stakeholders.

An **issue with the service library** lies in its implementation with these service properties, also because they are not part of the initial service catalogue. This added information must be modeled generally enough to be generally applicable, as are the services themselves. Examples are given with the properties in Figure 39, e.g. a generic “Inspection Report” for the service content or the “Remote Access” requirement.

Compared to **other PSS development approaches**, it is similar to an implementation of the process by Kim et al. [24] in SysML and has improved modeling capabilities of non-service aspects compared to the method of Nemoto et al. [7] that also uses a knowledge database for its services. Also there is no source of knowledge stated for the formalized services in its specific design catalogue viewer. A common issue in literature [7, 24, 28] is the lacking evaluation of service quality and performance in respect to its benefits for the service receiver. This is neither addressed by the service library, nor by the multi-physics simulation models of behavior simulation library. The service *stereotype* of the library only has a grading for the quality property, as seen in Figure 43. These properties are usually not generic and are therefore not assigned to the elements in the library.

6.2. Modeling with Multi-Solution Patterns

It is observed that the **multi-solution patterns meet their requirements** of being able to contain multiple solutions with partial models that cover multiple levels of abstraction, fit to the solution-neutral problem description and are able to include multiple domains. Their copy and paste application during the modeling process allows a dynamic selection of solutions and their required

adaption to the concrete context for a broad range of possible applications [122]. The coverage of multiple domains is tested by the different modeled patterns. The rotation sensor pattern for example includes sensors that use electronics, optics, magnetism and mechanics to measure the rotational motion. Different level of abstraction exist for example between this rotation sensor pattern that documents alternative single components, and the presented “2D Kinematics” pattern that contains whole subsystems with its multiple pulleys, belts and bearings.

The presented multi-solution patterns have the main difference **compared to Anacker et al.** [122, 123] of containing multiple solutions at once. This is also the main difference compared to pattern definitions in systems engineering [106, 116, 156]. Having multiple solutions in one pattern contrasts to the usual definition of patterns, which allows only a single solution per pattern. Conventional pattern definitions need multiple independent patterns for multiple exchangeable solutions. Yet, this causes the issue of how designers should choose one of multiple alternative patterns, since they become “confused if several patterns have similar or identical problem descriptions” [116]. Also offering multiple solutions at once supports the creation of alternatives through the repeated application of patterns with their different solutions. Other differences compared to the solution patterns of Anacker et al. [122, 123] are that here the functional model is not a simple hierarchy, but consists of a network structure with energy, signal and material flows fitting for mechatronics. Such flow-based representations are argued to fit better to especially mechatronic systems, with their explicit signal and energy flows [74]. Further, the patterns here are implemented with the modeling language SysML for a more formal and standardized approach. Also, they can contain simulation models with the behavior simulation library, instead of only solution principles.

Several **advantages of applying a pattern** are identified. Reusing their partial models that capture the essential parts of working designs, results in fewer modeling steps being necessary, since these partial models must only be adapted and not created from scratch. Having the essential parts of solutions formally documented in patterns also provides an effective mean for communicating complex concepts effectively between designers, as shown for software patterns in industry [117]. This improved communication goes together with a potential skill increase of novices that are encouraged to use the contained best practices, as well as a raised understanding of the system by the designers [118]. More specific advantages come from the included functional decomposition. It supports traceability throughout the partial models, the understanding of how the offered solution works, the identification of auxiliary supporting functions for the selected solution and the model consistency, by ensuring that “the functional description of a solution pattern as well as the definition of the desired system [stay] comparable" [122].

Identified disadvantages of the presented multi-solution patterns include the currently limited tool support for pattern application, resulting in manual copy and paste operations. A potential improvement for this exists in an extension of the pattern application capabilities of the modeling tool [65] for these multi-solution patterns. The creation of the patterns itself remains an issue, too. From industrial experience with software design patterns it is known that “good patterns are difficult and time-consuming to write” [117]. This is even more true, when not only complex system models with various aspects including simulation must be captured, but even captured to offer multiple comparable solutions at once. Also the knowledge to be incorporated must not only exist, but it must be identified as suitable for being formally documented as a pattern.

7. Summary and Future Extensions

The presented approach in SysML uses four libraries and multi-solution patterns together with corresponding simulation models to **support model-based concept design of multi-disciplinary systems**, e.g. mechatronic systems and PSS. It addresses the main initial research question from Section 1.2 of how to provide improved support for multi-disciplinary concept design.

Improved support is provided by offering generic and formal design knowledge in the form of libraries and patterns for reuse in SysML. With respect to the research question of which **knowledge to integrate** there are three libraries presented in detail. Their incorporation of specific existing design knowledge and its implementation in SysML implicitly addresses the research question about the levels of formality and detail. The design libraries are for functional modeling, behavior modeling and structure modeling [13], to follow loosely the FBS model with an additional service library. The function library is based on the verbs and nouns of the FB [15]. The behavior simulation library is based on the Amesim simulation tool [16]. It addresses the research question of enabling a link between the modeling approach and model simulation by an analog representation of simulation elements in both databases and therefore both models. The service library for service modeling and identification is based on a current service catalogue by Schmidt et al. [17]. To link the provided knowledge together the function library serves as a basis that is used throughout the other libraries to model functions and to define function flow interfaces.

Further, the **multi-solution patterns** contain multiple concept solutions in the form of partial models. These models build coherent subsystems by correlating library elements with other aspects. Such correlations within patterns are based on having a unified and model-based system representation in SysML.

SysML enables hereby to link together knowledge from different domains, levels of abstraction and sources, e.g. from the libraries.

The libraries as well as the patterns are implemented using a limited number of *stereotypes* to simplify their introduction to their users. Their cooperation as well as the provided design knowledge also define a modeling workflow shown by the case study. This addresses the research question for a modeling workflow to use for the developed approach.

To demonstrate the supported modeling approach and answer the research question as to whether it is able to model mechatronic systems and PSS a 3D printer case study is conducted. The results of the **case study** show that the various elements from the libraries together with multiple solutions taken from patterns can be used to create different concept variations. Traceability through the concept model in SysML is shown for example with relations between functions and services or between requirements and representations of Amesim simulation models. Their simulation results quantitatively show differences between two investigated solutions to move the print head of the 3D printer.

To answer the final research question about how the approach can support designers, an initial **user study** is conducted. The results show improved model quality when using the FB with the SysML function library through a higher grade of task completion with significantly more covered top-level functions and increased formality. At the same time it shows a higher workload for adapting to the more formal functional modeling approach.

7.1. Contributions

Compared to the previous work by Wölkl [13], the **function library** has been significantly improved. This includes its use together with other SysML

elements and within the other libraries and patterns to provide a comprehensive modeling approach. Additionally, there are suggested extensions of the terms of the FB for mechatronic systems. The integration of the FB in SysML contributes by improving the reusability and consistency of functional models. Its defined syntax and semantics increase the formality of SysML ACTs while reducing ambiguity. Adjunct to the function library the user study provides an initial validation of the function library.

The contributions of the **behavior simulation library** lie in its support to plan powerful simulation models directly in SysML. The reuse of provided simulation knowledge supports simulation in a commercial simulation tool to provide a first concept evaluation. The library also contributes to concept modeling in SysML by offering generic behavior elements that concretize functions as solution principles, similar to the FBS model. The library not only formally captures the simulation knowledge of the Amesim database in SysML, but it extends this knowledge by additional function flows. These properties contribute to the identification of suitable library elements by corresponding to the flows of functional models.

The **service library** contributes to service modeling in SysML. It contains an extension of SysML that allows the formal representation of services including their properties in a standardized and object-oriented way. This supports the necessary traceability to other non-service system elements within SysML, e.g. from functions. The service library also contributes to the PSS modeling by offering a collection of common services within SysML to help PSS designers identify suitable services for their systems. This includes the addition of further generic information to the provided services for their object-oriented description, e.g. in the form of functions or requirements.

The presented **multi-solution patterns** contribute by offering a means to formally capture and then reuse engineering solutions in the form of partial SysML models. The patterns support concept model creation by utilizing the libraries through the provision of already interconnected system elements. The second main contribution of the multi-solution patterns is the novelty of capturing multiple solutions, each with different possible viewpoints and levels of abstraction, in one pattern. This supports the notion that multiple solutions exist for solution-neutral functions. It also supports the investigation of alternative concepts, as shown with the case study.

Finally, the complete **modeling approach** and workflow, utilizing four design libraries and solution patterns is shown. It offers consistent support of multi-disciplinary concept modeling in SysML by including proven and formally captured design knowledge for reuse and a process to build concept models. The unified model-based system representation within SysML also enables traceability among model elements and levels, e.g. from functions to services.

7.2. LIMITATIONS

The first main limitation of the presented work is the additional **effort for introducing and maintaining** the libraries. Even with intentional small-scale SysML extensions there is additional effort required to utilize the libraries and patterns, as shown with the modeling workload in the user study. Further effort is needed to maintain the libraries and especially to derive and formally document additional useful multi-solution patterns.

A second **limitation with respect to modeling** is a lack of support to find suitable elements in the libraries. For example, when using the structure library it is difficult to find suitable elements to realize functions, since the suggested port matching of the function flow *ports* is not sufficient [13]. The function flow *ports* of

the pulley representations of Figure 34 for example are different for the behavior simulation library and structure library. Both of them include the required flows of the elementary function and can therefore be found via port matching. Yet, deriving one from another might be challenging and there are many other elements in the libraries to be expected to offer these same basic interfaces.

Another **functional modeling limitation** comes with the FB and its difficulty to express systems without specific energy, signal and material flows. In the presented approach it is mainly addressed by including previous *use case* models, as explained in Section 4.2.2. and in [2].

Finally, there are **modeling limitations when including simulation** capabilities in SysML with the behavior simulation library. First, there is a limited expressiveness of the modeling elements from the Amesim database, which for example does not include a stepper motor model. Second, there is limited compatibility checking between behavior simulation library elements in SysML and the handling of changing submodels. Third, there is limited support for the necessary two-way model transformations between SysML and Amesim.

7.3. Future Extensions

There are several potential future extensions now identified. First of all, there are bidirectional and more automated model transformations between SysML and Amesim required. These model transformations need further investigation for utilizing the captured design knowldege in SysML, e.g. the identification of specific submodels and their properties. Also, for the **behavior simulation library** there might be an improved compatibility checking through additional OCL constraints to be investigated.

For an improved application of the **multi-solution patterns** there exists potential to integrate them into the used modeling tool, similar to already offered common software engineering patterns by [18].

Further potential modeling support comes with the **integration of computational design synthesis**. Computational design synthesis enables automatic model generation while utilizing formalized knowledge [9, 49] and circumventing or respectively automating the search for suitable elements in the libraries. To include automation raises the chance of finding novel and creative solutions from having more generated concepts [51]. The standardization and graphical nature of SysML also might improve usability aspects of potential synthesis approaches [139] in return. Yet, at the same time it is argued that designing combines art and technology and cannot be fully automated. Some tasks might be suitable for automated support, but the main work is based on the experience of the designer and guided by heuristics [59].

Finally, there are **additional user studies** to be done to further validate the presented approach. Based on the findings of the conducted user study there is further work required to re-evaluate the increase of modeling speed for more experienced designers on a broader scale. Also the other libraries, the patterns and the total approach would need further practical application and validation.

8. Conclusion

Rising complexity of today's multi-disciplinary systems leads to the need for improved concept design since the concept phase offers the biggest potential to ensure a successful development process. One promising way for enhanced concept design is by model-based systems engineering (MBSE). The presented modeling approach combines formalized knowledge with object-orientation and the multi-disciplinary modeling power of SysML to support function-based concept design and evaluation through simulation. The thesis contributes to knowledge in the area by providing usable design libraries for supporting functional modeling, behavior modeling to plan concept simulation in an external simulation tool and service modeling for PSS design. The additional multi-solution patterns formally capture multiple alternative concept solutions in the form of partial models to support the reuse of common subsystems in addition to single library elements.

The concept modeling approach is demonstrated through a 3D printer case study model in SysML that combines elements from the libraries with solutions from patterns. In addition to the mechatronic models, it includes models for additional inspection services and representations of two kinematic solutions for simulation. All of this information is captured in a unified and object-oriented model, thus, enabling traceability between the different model elements. To initially evaluate the approach, a user study is conducted for the function library, indicating that by using the SysML function library the model quality and completeness is improved.

REFERENCES

[1] Kruse, B., and Shea, K. Design Library Solution Patterns in SysML for Concept Design and Simulation. 26th CIRP Design Conference. Stockholm, Sweden; 2016. p.620-625.

[2] Kruse, B., Gilz, T., Shea, K., and Eigner, M. Systematic Comparison of Functional Models in SysML for Design Library Evaluation. 24th CIRP Design Conference. Milano, Italy; 2014.

[3] Kruse, B., Münzer, C., Wölkl, S., Canedo, A., and Shea, K. A Model-Based Functional Modeling and Library Approach for Mechatronic Systems in SysML. ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. Chicago, IL, USA; 2012.

[4] Kruse, B., Münzer, C., Wölkl, S., Canedo, A., and Shea, K. Workflow and Modeling Conventions for Function and Product Structure Modeling of Mechatronic Systems in SysML Using Libraries. Mechatronics 2012. Linz, Austria; 2012.

[5] Beihoff, B., Oster, C., Friedenthal, S., Paredis, C. J. J., Kemp, D., Stoewer, H., Nichols, D., and Wade, J. A World in Motion – Systems Engineering Vision 2025. INCOSE. 2014.

[6] Eigner, M., Gerhardt, F., Gilz, T., and Nem, F. M. Informationstechnologie Für Ingenieure. ISBN: 3642248934. Springer-Verlag; 2012.

[7] Nemoto, Y., Akasaka, F., and Shimomura, Y. A Knowledge Management Method for Supporting Conceptual Design of Product-Service Systems. ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. 2013.

[8] Ehrlenspiel, K., Kiewert, A., Lindemann, U., and Hundal, M. S. Cost-Efficient Design. ISBN: 9783540346487. Springer; 2007.

[9] Chakrabarti, A., Shea, K., Stone, R. B., Cagan, J., Campbell, M. I., Hernandez, N. V., and Wood, K. L. Computer-Based Design Synthesis Research: An Overview. Journal of Computing and Information Science in Engineering, 2011. 11:2.

[10] Andreasen, M. M. 45 Years with Design Methodology. Journal of Engineering Design, 2011. 22:5. p.293-332.

[11] OMG. OMG Systems Modeling Language (OMG SysML). Version 1.4. No. formal/2015-06-03. 2015.

[12] Haskins, C. Systems Engineering Vision 2020. No. INCOSE-TP-2004-004-02, INCOSE. 2009.

[13] Wölkl, S. Model Libraries for Conceptual Design. Thesis, Munich, Germany: Technische Universität München. 2012.

[14] VDI-Gesellschaft. VDI 2221: Methodik Zum Entwickeln Und Konstruieren Technischer Systeme Und Produkte. No. 03.100.40. 1993.

[15] Hirtz, J., Stone, R. B., Mcadams, D. A., Szykman, S., and Wood, K. L. A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts. No. 1447, Washington, D.C., USA: NIST. 2002.

[16] Siemens PLM Software. LMS Imagine.Lab Amesim. 09.01.2017. http://www.plm.automation.siemens.com/de_ch/products/lms/imagine-lab/amesim/

[17] Schmidt, D. M., Malaschewski, O., Jaugstetter, M., and Mörtl, M. Service Classification to Support Planning Product-Service Systems. Asian Design Engineering Workshop (A-DEWS) Hong Kong, China; 2015. p.34-39.

[18] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. ISBN: 0201633612. Addison-Wesley Professional; 1994.

[19] Torry-Smith, J. M., Qamar, A., Achiche, S., Wikander, J., Mortensen, N. H., and During, C. Challenges in Designing Mechatronic Systems. Journal of Mechanical Design, 2013. 135:1.

[20] Van Der Auweraer, H., Anthonis, J., De Bruyne, S., and Leuridan, J. Virtual Engineering at Work: The Challenges for Designing Mechatronic Products. Engineering with Computers, 2013. 29:3. p.389-408.

[21] Alvarez Cabrera, A. A., Foeken, M. J., Tekin, O. A., Woestenenk, K., Erden, M. S., De Schutter, B., Van Tooren, M. J. L., Babuška, R., Van Houten, F. J. a. M., and Tomiyama, T. Towards Automation of Control Software: A Review of Challenges in Mechatronic Design. Mechatronics, 2010. 20:8. p.876-886.

[22] Komoto, H., and Tomiyama, T. Multi-Disciplinary System Decomposition of Complex Mechatronics Systems. CIRP Annals - Manufacturing Technology, 2011. 60:1. p.191-194.

[23] Pahl, G., and Beitz, W. Konstruktionslehre: Grundlagen Erfolgreicher Produktentwicklung. Methoden Und Anwendung. ISBN: 9783540340607. Springer; Berlin, Germany: 2007.

[24] Kim, Y., Lee, S., Jin, H., Shin, J., Park, J., Lee, Y., Kim, C., Seo, B., and Lee, S. Product-Service Systems (PSS) Design Process and Design Support Systems. In: *Functional Thinking for Value Creation*, ISBN: 9783642196898. Springer, 2011.

[25] Törngren, M., Qamar, A., Biehl, M., Loiret, F., and El-Khoury, J. Integrating Viewpoints in the Development of Mechatronic Products. Journal of Mechatronics 2013. p.745-762.

[26] Komoto, H., and Tomiyama, T. Systematic Generation of PSS Concepts Using a Service CAD Tool. In: *Introduction to Product/Service-System Design*, ISBN: 9781848829091. Springer, 2009.

[27] Shimomura, Y., and Tomiyama, T. Service Modeling for Service Engineering. In: *Knowledge and Skill Chains in Engineering and Manufacturing*, Springer, 2005.

[28] Vasantha, G. V. A., Roy, R., Lelah, A., and Brissaud, D. A Review of Product–Service Systems Design Methodologies. Journal of Engineering Design, 2012. 23:9. p.635-659.

[29] Müller, P., Schmidt-Kretschmer, M., and Blessing, L. T. M. Function Allocation in Product-Service Systems-Are There Analogies between PSS and Mechatronics? AEDS 2007 Workshop. Pilsen, Czech Republic; 2007.

[30] Bossak, M. A. Simulation Based Design. Journal of Materials Processing Technology, 1998. 76:1. p.8-11.

[31] Modelica Association. Modelica® - a Unified Object-Oriented Language for Systems Modeling. Language Specification. 2012.

[32] Phoenix Integration, Inc. ModelCenter. 24.11.2016. www.phoenix-int.com/

[33] Thomke, S., and Fujimoto, T. The Effect of “Front-Loading” Problem-Solving on Product Development Performance. Journal of Product Innovation Management, 2000. 17:2. p.128-142.

[34] Wölkl, S., and Shea, K. A Computational Product Model for Conceptual Design Using SysML. ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. San Diego, CA, USA; 2009.

[35] Dhanesha, K. A., Hartman, A., and Jain, A. N. A Model for Designing Generic Services. IEEE International Conference on Services Computing (SCC'09). 2009. p.435-442.

[36] Kim, Y., Wang, E., Lee, S., and Cho, Y. A Product-Service System Representation and Its Application in a Concept Design Scenario. 1st CIRP Industrial Product-Service Systems (IPS2) Conference. 2009.

[37] Albers, A., and Zingel, C. Challenges of Model-Based Systems Engineering: A Study Towards Unified Term Understanding and the State of Usage of SysML. In: *Smart Product Engineering*, ISBN: 3642308163. Springer, 2013.

[38] Friedenthal, S., and Burkhart, R. M. Evolving SysML and the System Modeling Environment to Support MBSE. INCOSE, Insight, 2015. 18:2, p.39-41.

[39] Kruse, B. Modell-Basierte Satellitenentwicklung: Wieder- Und Weiterverwendung Objektorientierter Systemmodelle Nach Der Konzeptphase. Thesis, Garching: Technische Universität München. 2010.

[40] Griss, M. L. Software Reuse: Architecture, Process, and Organization for Business Success. 8th Israeli Conference on Computer Systems and Software Engineering. 1997. p.86-89.

[41] Chughtai, A., and Vogel, O. Software-Wiederverwendung - Theoretische Grundlagen, Vorteile Und Realistische Beurteilung. In: *Software Management*, ISBN: 3642627129. Springer, 2002.

[42] Gajski, D. D. Embedded System Design : Modeling, Synthesis and Verification. ISBN 9781441905031. New York : Springer; 2009.

[43] Lienig, J. Layoutsynthese Elektronischer Schaltungen — Grundlegende Algorithmen Für Die Entwurfsautomatisierung. ISBN 9783540299424. Springer Berlin Heidelberg; Berlin, Germany: 2006.

[44] Jackson, C., and Maura, B. The Design Reuse Benchmark Report: Seizing the Opportunity to Shorten Product Development. Boston, MA, USA: Aberdeen Group, Inc. 2007.

[45] Bonev, M., Hvam, L., Clarkson, J., and Maier, A. Formal Computer-Aided Product Family Architecture Design for Mass Customization. Computers in Industry, 2015. 74:1. p.58-70.

[46] Li, B., Xie, S., and Xu, X. Recent Development of Knowledge-Based Systems, Methods and Tools for One-of-a-Kind Production. Knowledge-Based Systems, 2011. 24:7. p.1108-1119.

[47] Prasad, B. Designing Products for Variety and How to Manage Complexity. Journal of Product & Brand Management, 1998. 7:3. p.208-222.

[48] Münzer, C., and Shea, K. A Simulation-Based CDS Approach: Automated Generation of Simulation Models Based from Generated Concept Model Graphs. ASME 2015 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. Boston, MA, USA; 2015.

[49] Cagan, J., Campbell, M. I., Finger, S., and Tomiyama, T. A Framework for Computational Design Synthesis: Model and Applications. Journal of Computing and Information Science in Engineering, 2005. 5:3. p.171-181.

[50] Münzer, C., Helms, B., and Shea, K. Automatically Transforming Object-Oriented Graph-Based Representations into Boolean Satisfiability Problems for Computational Design Synthesis. Journal of Mechanical Design, 2013. 135:10.

[51] Kudrowitz, B. M. Haha and Aha! Creativity, Idea Generation, Improvisational Humor, and Product Design. Thesis, http://hdl.handle.net/1721.1/61610 Massachusetts Institute of Technology. 2010.

[52] De Weck, O. L. Feasibility of a 5x Speedup in System Development Due to Meta Design. ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. 2012. p.1105-1110.

[53] Blessing, L. T. M., and Chakrabarti, A. DRM: A Design Research Methodology. ISBN 9781848825864. Springer; 2009.

[54] Gero, J. S. Design Prototypes - a Knowledge Representation Schema for Design. AI Magazine, 1990. 11:4. p.26-36.

[55] Umeda, Y., Takeda, H., Tomiyama, T., and Yoshikawa, H. Function, Behaviour, and Structure. IEA/AEI-90. Boston, MA, USA; 1990. 1. p.177-193.

[56] Walden, D. D., Roedler, G. J., Forsberg, K. J., Hamelin, R. D., and Shortell, T. M. Systems Engineering Handbook. San Diego, CA, USA: INCOSE; 2015.

[57] BKCASE-Editorial-Board. The Guide to the Systems Engineering Body of Knowledge (Sebok). Hoboken, NJ, USA: The Trustees of the Stevens Institute of Technology; 2016.

[58] Tepper, N. A. Exploring the Use of Model-Based Systems Engineering (MBSE) to Develop Systems Architectures in Naval Ship Design. Thesis, Cambridge, MA, USA: http://hdl.handle.net/10945/24368 Massachusetts Institute of Technology. 2010.

[59] Weilkiens, T., Lamm, G. J., Roth, S., and Walker, M. Model-Based System Architecture. *Wiley Series in Systems Engineering and Management*, ISBN 9781118893647. Hoboken, New Jersey : John Wiley & Sons, Inc.; 2016.

[60] Reil, R. What Makes Model-Based Systems Engineering Transformational? INCOSE, INSIGHT, 2015. 18:3, p.16-17.

[61] Qamar, A. An Integrated Approach Towards Model-Based Mechatronic Design. Thesis, Stockholm, Sweden: KTH-Royal Institute of Technology. 2011.

[62] Königs, S. F., Beier, G., Figge, A., and Stark, R. Traceability in Systems Engineering – Review of Industrial Practices, State-of-the-Art Technologies and New Research Solutions. Advanced Engineering Informatics, 2012. 26:4. p.924-940.

[63] London, B. A Model-Based Systems Engineering Framework for Concept Development. Thesis, Cambridge, MA, USA: Massachusetts Institute of Technology. 2012.

[64] Eigner, M., Dickopf, T., Apostolov, H., Schaefer, P., Faißt, K.-G., and Keßler, A. System Lifecycle Management: Initial Approach for a Sustainable Product Development Process Based on Methods of Model Based Systems Engineering. In: *Product Lifecycle Management for a Global Market*, ISBN: 3662459361. Springer, 2014.

[65] No Magic, Inc. Magicdraw. 23.09.2016. http://www.nomagic.com/products/magicdraw.html

[66] Estefan, J. A. Survey of Model-Based Systems Engineering (MBSE) Methodologies. INCOSE Technical Data, 2008. 25:8.

[67] Kasser, J. E. Seven Systems Engineering Myths and the Corresponding Realities. Systems Engineering Test and Evaluation Conference. Adelaide, Australia; 2010.

[68] OMG. UML 2.0 Infrastructure: Version 2.4.1. 2011.

[69] OMG. UML 2.0 Superstructure: Version 2.4.1. 2011.

[70] Friedenthal, S., Moore, A., and Steiner, R. A Practical Guide to SysML: The Systems Modeling Language. ISBN 9780123743794. Elsevier, Morgan Kaufmann OMG; Amsterdam [u.a.]: 2009.

[71] Petrasch, R., and Meimberg, O. Model Driven Architecture: Eine Praxisorientierte Einführung in Die MDA. ISBN 3898643433. dpunkt-Verlag; 2006.

[72] Kaiser, L. Rahmenwerk Zur Modellierung Einer Plausiblen Systemstruktur Mechatronischer Systeme. Thesis, Paderborn, Germany: Universität Paderborn. 2013.

[73] Holt, J., and Perry, S. SysML for Systems Engineering. ISBN 0863418252. Institution of Engineering and Technology; Stevenage: 2008.

[74] Eigner, M. Überblick Disziplin-Spezifische Und -Übergreifende Vorgehensmodelle. In: *Modellbasierte Virtuelle Produktentwicklung*, ISBN 9783662438152. Springer, Berlin, Heidelberg; 2014.

[75] VDI-Gesellschaft. VDI/VDE 2422: Entwicklungsmethodik Für Geräte Mit Steuerung Durch Mikroelektronik. No. DE18949719. 1994.

[76] Gajski, D. D. Construction of a Large Scale Multiprocessors. No. UIUCDCS-R-83-1123, Urbana, IL, USA: Cedar Project, Laboratory for Advanced Supercomputers, Dept. of Computer Science, University of Illinois at Urbana-Champaign. 1983.

[77] Whitney, D. E. Why Mechanical Design Cannot Be Like VLSI Design. Research in Engineering Design, 1996. 8:3. p.125-138.

[78] Boehm, B. Guidelines for Verifying and Validating Software Requirements and Design Specifications. No. USC-CSE-79-501, North Holland: Euro IFIP 79. P. A. Samet. North-Holland Publishing Company. 1979.

[79] VDI-Gesellschaft. VDI 2206: Entwicklungsmethodik Für Mechatronische Systeme. Düsseldorf, Germany. 2004.

[80] Eigner, M., Dickopf, T., and Huwig, C. An Interdisciplinary Model-Based Design Approach for Developing Cybertronic Systems. 14th International Design Conference, Design 2016. Dubrovnik, Croatia; 2016. p.1647-1656.

[81] Pohl, K., Hönninger, H., Achatz, R., and Broy, M. Model-Based Engineering of Embedded Systems: The Spes 2020 Methodology. ISBN: 3642346146. Springer; 2012.

[82] Helms, B., and Shea, K. Computational Synthesis of Product Architectures Based on Object-Oriented Graph Grammars. Journal of Mechanical Design, 2012. 134:2.

[83] Kannengiesser, U., and Gero, J. S. Is Designing Independent of Domain? Comparing Models of Engineering, Software and Service Design. Research in Engineering Design, 2015. 26:3. p.253-275.

[84] Deng, Y.-M. Function and Behavior Representation in Conceptual Mechanical Design. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2002. 16:5. p.343-362.

[85] Summers, J. D., Eckert, C., and Goel, A. K. Function in Engineering: Benchmarking Representations and Models. International Conference On Engineering Design, ICED13. Seoul, Korea; 2013.

[86] Eckert, C. That Which Is Not Form: The Practical Challenges in Using Functional Concepts in Design. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2013. 27:3. p.217-231.

[87] Arlitt, R. M., Stone, R. B., and Tumer, I. Y. Impacts of Function-Related Research on Education and Industry. In: *Impact of Design Research on Industrial Practice: Tools, Technology, and Training*, ISBN 9783319194493. Springer International Publishing, Cham; 2016.

[88] Erden, M. S., Komoto, H., Van Beek, T. J., D'amelio, V., Echavarria, E., and Tomiyama, T. A Review of Function Modeling: Approaches and Applications.

AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2008. 22:2. p.147-169.

[89] Eisenbart, B., Gericke, K., and Blessing, L. T. M. An Analysis of Functional Modeling Approaches across Disciplines. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2013. 27:Special Issue 03. p.281-289.

[90] Saunders, M. N., Seepersad, C. C., and Hölttä-Otto, K. The Characteristics of Innovative, Mechanical Products. Journal of Mechanical Design, 2011. 133:2.

[91] Simon, H. A. The Sciences of the Artificial. ISBN: 0262264498. MIT press; 1996 (1st edition: 1969).

[92] Chandrasekaran, B., and Josephson, J. R. Function in Device Representation. Engineering with Computers, 2000. 16:3-4. p.162-177.

[93] Vermaas, P. E. The Coexistence of Engineering Meanings of Function: Four Responses and Their Methodological Implications. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2013. 27:3. p.191-202.

[94] Chen, Y., Zhang, Z., Xie, Y., and Zhao, M. A New Model of Conceptual Design Based on Scientific Ontology and Intentionality Theory. Part I: The Conceptual Foundation. Design Studies, 2015. 37:1. p.12-36.

[95] Stone, R. B., and Wood, K. L. Development of a Functional Basis for Design. Journal of Mechanical Design, 2000. 122:4. p.359-370.

[96] Szykman, S., Racz, J. W., and Sriram, R. D. The Representation of Function in Computer-Based Design. ASME Design Engineering Technical Conferences, 1999.

[97] Oman, S., Gilchrist, B., Rebhuhn, C., Tumer, I. Y., Nix, A., and Stone, R. Towards a Repository of Innovative Products to Enhance Engineering Creativity Education. ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2012. p.207-218.

[98] Caldwell, B. Evaluating the Use of Functional Representations for Ideation in Conceptual Design. Thesis, Clemson, SC, USA: http://tigerprints.clemson.edu/all_dissertations/875 Clemson University. 2011.

[99] Caldwell, B. W., Sen, C., Mocko, G. M., Summers, J. D., and Fadel, G. M. Empirical Examination of the Functional Basis and Design Repository. In: *Design Computing and Cognition'08*, ISBN: 1402087276. Springer, 2008.

[100] Sen, C., Caldwell, B. W., Summers, J. D., and Mocko, G. M. Evaluation of the Functional Basis Using an Information Theoretic Approach. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2010. 24:01. p.87-105.

[101] Caldwell, B. W., Sen, C., Mocko, G. M., and Summers, J. D. An Empirical Study of the Expressiveness of the Functional Basis. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2011. 25:3. p.273-287.

[102] Sen, C., Summers, J. D., and Mocko, G. M. Topological Information Content and Expressiveness of Function Models in Mechanical Design. Journal of Computing and Information Science in Engineering, 2010. 10:3.

[103] Mili, H., Mili, A., Yacoub, S., and Addy, E. Reuse-Based Software Engineering: Techniques, Organization, and Controls. ISBN: 0471398195. Wiley-Interscience; 2001.

[104] Girczyc, E., and Carlson, S. Increasing Design Quality and Engineering Productivity through Design Reuse. 30th International Design Automation Conference. 1993. p.48-53.

[105] Duffy, A. H. B., and Ferns, A. F. An Analysis of Design Reuse Benefits. International Conference On Engineering Design, ICED99. 1998. p.799-804.

[106] Cloutier, R. J., and Verma, D. Applying the Concept of Patterns to Systems Architecture. Systems Engineering, 2007. 10:2. p.138-154.

[107] Borutzky, W. Bond Graph Modelling and Simulation of Multidisciplinary Systems – an Introduction. Simulation Modelling Practice and Theory, 2009. 17:1. p.3-21.

[108] Gaiardelli, P., Resta, B., Martinez, V., Pinto, R., and Albores, P. A Classification Model for Product-Service Offerings. Journal of Cleaner Production, 2014. 66 p.507-519.

[109] Tukker, A. Eight Types of Product–Service System: Eight Ways to Sustainability? Experiences from SusProNet. Business Strategy and the Environment, 2004. 13:4. p.246-260.

[110] Maximilien, E. M., and Singh, M. P. A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing, 2004. 8:5. p.84-93.

[111] De Groot, R. S., Wilson, M. A., and Boumans, R. M. A Typology for the Classification, Description and Valuation of Ecosystem Functions, Goods and Services. Ecological Economics, 2002. 41:3. p.393-408.

[112] Lee, S., and Park, Y. The Classification and Strategic Management of Services in E-Commerce: Development of Service Taxonomy Based on Customer Perception. Expert Systems with Applications, 2009. 36:6. p.9618-9624.

[113] Wemmerlöv, U. A Taxonomy for Service Processes and Its Implications for System Design. International Journal of Service Industry Management, 1990. 1:3. p.20-40.

[114] eCl@ss e.V. eCl@ss - Classification and Product Description. 17.01.2017. http://www.eclass.de/

[115] Alexander, C., Ishikawa, S., and Silverstein, M. A Pattern Language: Towns, Buildings, Construction. ISBN: 0195019199. Oxford University Press; 1977.

[116] Meszaros, G., and Doble, J. A Pattern Language for Pattern Writing. Pattern Languages of Program Design, 1998. 3:1. p.529-574.

[117] Beck, K., Crocker, R., Meszaros, G., Vlissides, J., Coplien, J. O., Dominick, L., and Paulisch, F. Industrial Experience with Design Patterns. 18th International Conference on Software Engineering. 1996. p.103-114.

[118] Zhang, C., and Budgen, D. What Do We Know About the Effectiveness of Software Design Patterns? IEEE Transactions on Software Engineering, 2012. 38:5. p.1213-1231.

[119] Shvets, A. Design Patterns: Explained Simply. sourcemaking.com; 2015.

[120] Cloutier, R. J. Applicability of Patterns to Architecting Complex Systems. Thesis, Castle Point on Hudson, Hoboken, NJ, USA: Stevens Institute of Technology. 2006.

[121] Hein, A., Kruse, B., Lopez, R. P., and Brandstätter, M. Object-Oriented Modeling Methods: Enable Model Reuse for Hardware Systems. Tag des Systems Engineerings 2010. Munich, Germany; 2010.

[122] Anacker, H., Schierbaum, T., Dumitrescu, R., and Gausemeier, J. Solution Patterns to Support the Knowledge Intensive Design Process of Intelligent Technical Systems. International Conference On Engineering Design, ICED13. Seoul, Korea; 2013.

[123] Anacker, H., Dumitrescu, R., and Gausemeier, J. Design Framework for the Integration of Cognitive Functions Based on Solution Patterns. In: *Design Methodology for Intelligent Technical Systems*, ISBN: 9783642454349. Springer, Dordrecht; 2014.

[124] Weber, C., and Husung, S. Solution Patterns – Their Role in Innovation, Practice and Education. 14th International Design Conference, Design 2016. Dubrovnik, Croatia; 2016. p.99-108.

[125] Weber, C. Modelling Products and Product Development Based on Characteristics and Properties. In: *An Anthology of Theories and Models of Design*, ISBN: 9781447163374. Springer London, 2014.

[126] Salustri, F. A. Using Pattern Languages in Design Engineering. International Conference On Engineering Design, ICED05. Melbourne, Australia; 2005.

[127] Deigendesch, T. Kreativität in Der Produktentwicklung Und Muster Als Methodisches Hilfsmittel. Thesis, IPEK - Institut für Produktentwicklung: Karlsruher Institut für Technologie (KIT). 2009.

[128] Yoshioka, M., Umeda, Y., Takeda, H., Shimomura, Y., Nomaguchi, Y., and Tomiyama, T. Physical Concept Ontology for the Knowledge Intensive Engineering Framework. Advanced Engineering Informatics, 2004. 18:2. p.95–113.

[129] Kurtoglu, T., Campbell, M. I., Arnold, C. B., Stone, R. B., and Mcadams, D. A. A Component Taxonomy as a Framework for Computational Design Synthesis. Journal of Computing and Information Science in Engineering, 2009. 9:1.

[130] Christophe, F., Sell, R., Bernard, A., and Coatanéa, E. OPAS: Ontology Processing for Assisted Synthesis of Conceptual Design Solutions. ASME 2010 International Design Engineering Technical Conferences & Computers and

Information in Engineering Conference - 35th Design Automation Conference. New York; 2010. p.249-260.

[131] Hutcheson, R. S., Mcadams, D. A., Stone, R. B., and Tumer, I. Y. Function-Based Systems Engineering (FUSE). International Conference On Engineering Design, ICED07. 2007. p.28-30.

[132] Lamm, G. J., and Weilkiens, T. Funktionale Architekturen in SysML. Tag des Systems Engineering 2010. Munich, Germany; 2010. p.109-118.

[133] Lamm, J. G., and Weilkiens, T. Method for Deriving Functional Architectures from Use Cases. Systems Engineering, 2014. 17:2. p.225-236.

[134] Chen, R., Liu, Y., Cao, Y., and Xu, J. A SysML-Based Modeling Language for Mechatronic System Architecture. ASME 2015 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. Boston, MA, USA; 2015.

[135] Wu, J. C., Poppa, K., Leu, M. C., and Liu, X. F. Integrated Function Structure and Object-Oriented Design Framework. Computers in Industry, 2012. 63:5. p.458-470.

[136] Eisenbart, B., Gericke, K., Blessing, L. T. M., and Mcaloone, T. C. A DSM-Based Framework for Integrated Function Modelling: Concept, Application and Evaluation. Research in Engineering Design, 2016. p.1-27.

[137] Eisenbart, B., Mandel, C., Gericke, K., and Blessing, L. T. M. Integrated Function Modelling: Comparing the IFM Framework with SysML. International Conference On Engineering Design, ICED15. Milano, Italy; 2015.

[138] Wan, J., Canedo, A., Faruque, A., and Abdullah, M. Functional Model-Based Design Methodology for Automotive Cyber-Physical Systems. IEEE Systems Journal, 2015.

[139] Münzer, C. Constraint-Based Methods for Automated Computational Design Synthesis of Solution Spaces. Thesis, Zurich, Switzerland: ETH Zurich. 2015.

[140] Paredis, C. J. J., Bernard, Y., Burkhart, R. M., Koning De, H.-P., Friedenthal, S., Fritzson, P. A., Rouquette, N. F., and Schamai, W. An Overview of the SysML-Modelica Transformation Specification. INCOSE International Symposium, 2010.

[141] Kerzhner, A. A. Using Logic-Based Approaches to Explore System Architectures for Systems Engineering. Thesis, Athens, GA, USA: Georgia Institute of Technology. 2012.

[142] Bracewell, R. H., Shea, K., Langdon, P. M., Blessing, L. T. M., and Clarkson, P. J. A Methodology for Computational Design Tool Research. International Conference On Engineering Design, ICED01. Glasgow, Scotland, UK; 2001. p.181-188.

[143] Jones, R., Haufe, P., Sells, E., Iravani, P., Olliver, V., Palmer, C., and Bowyer, A. Reprap – the Replicating Rapid Prototyper. Robotica, 2011. 29:01. p.177-191.

[144] Aebischer, M., Beer, Y., Grossenbacher, F., Nieland, A., Rüttimann, L., and Vogel, A. 3D-Printing: Raptype - Building the Future. Zurich, Switzerland: EDAC, ETH Zurich. 2014.

[145] Moyer, I. E. Core(X,Y). 05.02.2016. http://corexy.com/

[146] 3D Printers Sp. z o.o. Hbot 3D Printers. 05.02.2016. http://hbot3d.com/

[147] Nagel, R. L., Vucovich, J. P., Stone, R. B., and Mcadams, D. A. Signal Flow Grammar from the Functional Basis. International Conference On Engineering Design, ICED07. Paris, France; 2007. p.28-31.

[148] Sridharan, P., and Campbell, M. I. A Study on the Grammatical Construction of Function Structures. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing), 2005. 19:3. p.139–160.

[149] Williams, C. B., Mistree, F., and Rosen, D. W. A Functional Classification Framework for the Conceptual Design of Additive Manufacturing Technologies. Journal of Mechanical Design, 2011. 133:1.

[150] Anacker, H., Dorociak, R., Dumitrescu, R., and Gausemeier, J. Integrated Tool-Based Approach for the Conceptual Design of Advanced Mechatronic Systems. 2011 IEEE International Systems Conference (SysCon). 2011. p.506 - 511.

[151] Bock, C. SysML and UML 2 Support for Activity Modeling. Systems Engineering, 2006. 9:2. p.160-186.

[152] Jansen, S. Eine Methodik Zur Modellbasierten Partitionierung Mechatronischer Systeme. Thesis, Bochum, Germany: Ruhr-Universität Bochum. 2006.

[153] FAS4M. Mechanics Modeling Language (MechML), Version 1.0. 2016.

[154] Wikimedia Foundation, Inc. Iso/IEC 80000. 19.09.2016. https://en.wikipedia.org/wiki/ISO/IEC_80000

[155] Siemens PLM Software. LMS Imagine.Lab Sysdm. 09.01.2017. https://www.plm.automation.siemens.com/de_ch/products/lms/imagine-lab/sysdm/

[156] Pfister, F., Chapurlat, V., Huchard, M., Nebut, C., and Wippler, J. L. A Proposed Meta-Model for Formalizing Systems Engineering Knowledge, Based on Functional Architectural Patterns. Systems Engineering, 2012. 15:3. p.321-332.

[157] Komoto, H., and Tomiyama, T. Computational Support for System Architecting. ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, 2010.

[158] Hart, S. G., and Staveland, L. E. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. Advances in Psychology, 1988. 52 p.139-183.

[159] Oswald, W. D., and Roth, E. Der Zahlen-Verbindungs-Test:(ZVT); Ein Sprachfreier Intelligenz-Schnell-Test. Verlag für Psychologie Hogrefe; 1978.

[160] DIN German Institute for Standardization. Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs). Part 11: Guidance on usability (ISO 9241-11:1998). 1999.

[161] Genero, M., Poels, G., and Piattini, M. Defining and Validating Measures for Conceptual Data Model Quality. Advanced Information Systems Engineering. 2002. p.724-727.

[162] Vogel-Heuser, B. Usability Experiments to Evaluate UML/SysML-Based Model Driven Software Engineering Notations for Logic Control in Manufacturing Automation. Journal of Software Engineering and Applications, 2014. 7:11. p.943-973.

[163] Nagel, R. L., Bohm, M. R., Linsey, J. S., and Riggs, M. K. Improving Students' Functional Modeling Skills: A Modeling Approach and a Scoring Rubric. Journal of Mechanical Design, 2015.

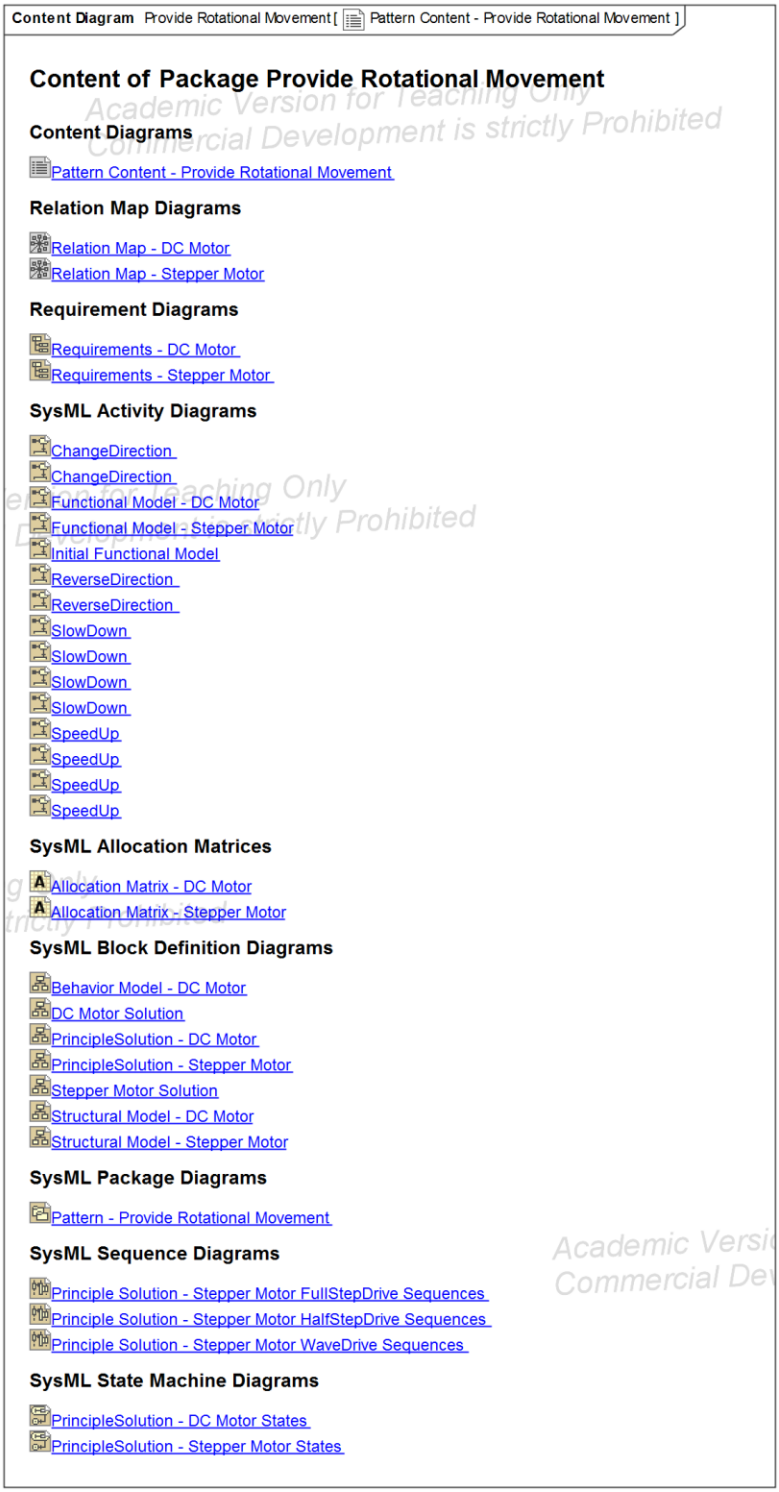
[164] Caldwell, B. W., Thomas, J. E., Sen, C., Mocko, G. M., and Summers, J. D. The Effects of Language and Pruning on Function Structure Interpretability. Journal of Mechanical Design, 2012. 134:6.

[165] Wilcoxon, F. Individual Comparisons by Ranking Methods. Biometrics Bulletin, 1945. 1:6. p.80-83.

[166] Spearman, C. Demonstration of Formulae for True Measurement of Correlation. The American Journal of Psychology, 1907. p.161-169.

[167] Ruiz-Cabello, J., Vuister, G. W., Moonen, C. T., Van Gelderen, P., Cohen, J. S., and Van Zijl, P. C. Gradient-Enhanced Heteronuclear Correlation Spectroscopy. Theory and Experimental Aspects. Journal of Magnetic Resonance (1969), 1992. 100:2. p.282-302.

APPENDIX A – MULTI-SOLUTION PATTERN EXCERPT: "PROVIDE ROTATIONAL MOVEMENT"



Content Diagram Provide Rotational Movement [Pattern Content - Provide Rotational Movement]

Content of Package Provide Rotational Movement

Content Diagrams

Pattern Content - Provide Rotational Movement

Relation Map Diagrams

Relation Map - DC Motor
Relation Map - Stepper Motor

Requirement Diagrams

Requirements - DC Motor
Requirements - Stepper Motor

SysML Activity Diagrams

ChangeDirection
ChangeDirection
Functional Model - DC Motor
Functional Model - Stepper Motor
Initial Functional Model
ReverseDirection
ReverseDirection
SlowDown
SlowDown
SlowDown
SlowDown
SpeedUp
SpeedUp
SpeedUp
SpeedUp

SysML Allocation Matrices

Allocation Matrix - DC Motor
Allocation Matrix - Stepper Motor

SysML Block Definition Diagrams

Behavior Model - DC Motor
DC Motor Solution
PrincipleSolution - DC Motor
PrincipleSolution - Stepper Motor
Stepper Motor Solution
Structural Model - DC Motor
Structural Model - Stepper Motor

SysML Package Diagrams

Pattern - Provide Rotational Movement

SysML Sequence Diagrams

Principle Solution - Stepper Motor FullStepDrive Sequences
Principle Solution - Stepper Motor HalfStepDrive Sequences
Principle Solution - Stepper Motor WaveDrive Sequences

SysML State Machine Diagrams

PrincipleSolution - DC Motor States
PrincipleSolution - Stepper Motor States

«Design Pattern»
«Info»
Provide Rotational Movement

Alias = Electric Motor
Problem_description = "The problem to solve is how to convert electrical energy into specific required rotational mechanical energy."
Author = "Benjamin Kruse"
Pattern ID = "P-BK-003"
Problem_context = "Many applications require rotational mechanical energy in some form.
This energy can be provided by converting electrical energy using the Biot-Sarvat Law in different ways."
Target_functionality = ElectricalEnergy-RotationalEnergy
References = DC:
https://en.wikipedia.org/wiki/DC_motor, Stepper:
https://en.wikipedia.org/wiki/Stepper_motor, Biot-Savart Law:
https://en.wikipedia.org/wiki/Biot%E2%80%93Savart_law, Rotary Stepper Motors:
http://www.haydonkerk.com/LinearActuatorProducts/StepperMotorLinearActuators/RotaryStepperMotors/tabid/72/Default.aspx
Forces = "Multiple possible electric motors exist to fulfill the function with different characteristics for different applications.
There are especially differences regarding the electricity input (e.g. DC/AC), the control and the capability to hold with a certain momentum."
Keywords = Electric Motor, Rotational Energy
Pattern_solution = DC Motor Solution
Related_pattern =
- 2D Kinematics
- Measure Torque
- Sense Torque
- Sensing Rotational Energy
- Measuring Rotational Energy

Version = "v01"

«Pattern Solution»
DC Motor Solution

Interfaces =
- wasteEnergy
- fixture
- ObjectIn
- ObjectOut

Behavior_model = PrincipleSolution - DC Motor
Known_uses = none
Resulting_context = "Further work is required to adapt the solution to a more specific motor. Also solutions must be fund for its control,
e.g. by adding an encoder to create a servo motor for precise control of angular or linear position, velocity and acceleration."
Functional_model = Functional Model - DC Motor
Pattern_rationale = "Brushed DC motors were the first important application of converting electric power into mechanical energy,
since they use a simple direct current power source and no not necessarily need additional electronics or control.
Their speed and torque characteristics can be altered to provide steady speed or speed inversely proportional to the mechanical load.
Yet, powerful DC motors usually must be colled, which is almost always done using forced air.
Another disadvantage is the wear down of the brushes that might require replacement. Alternative brushless DC motors use power electronic devices."
Solution_description = "The DC motor solution converts direct current electrical power into rotational mechanical power.
For this it relies on the forces produced by magnetic fields, which are generated by an electric current.
It consists of a brushed DC electric motor to periodically change the direction of current flow in the motor together with a permanent magnet stator."
Domain =
- electrical
- magnetic
- mechanical

Structural_model = DC Motor
Requirements =
- Electric Power Supply
- Electromagnetic Compatibility Testing
- Rotational Energy Provision

Examples = DCX35L GB KL 36V Motor, used in the Raptype 3D printer
Solution ID = "S-BK-003-01"

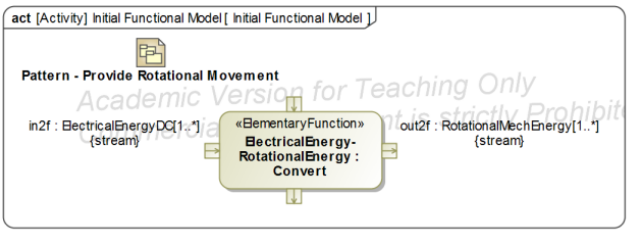
«Pattern Solution»
Stepper Motor Solution

Interfaces =
- wasteEnergy
- ObjectIn
- ObjectOut
- ObjectIN2

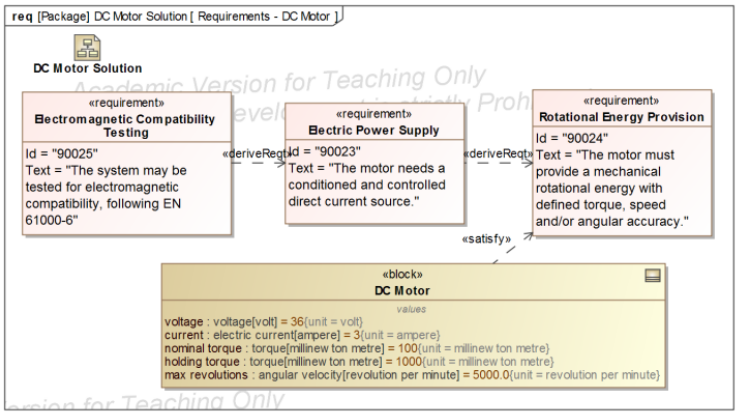
Behavior_model = PrincipleSolution - Stepper Motor
Known_uses = none
Resulting_context = "Further work is required to adapt the solution to a more specific motor, including two alternative wirings: uni-polar and bi-polar.
Also solutions must be fund for its control, usually by special stepper motor controllers to initiate steps and hold the torque in between."
Functional_model = Functional Model - Stepper Motor
Pattern_rationale = "Advantages of stepper motors are mainly theit low cost for control, a high torque at startup
and low speeds, their ruggedness and simplicity of construction.
Disadvantages are the requirement for a dedicated control circuit, the higher current usage than DC motors and a reduced torque at higher speeds."
Solution_description = "The stepper motor solution converts alternating current electrical power into rotational mechanical power.
For this it relies on the forces produced by magnetic fields, which are generated by an electric current.
It consists of a toothed electromagnet stators around a central gear-shaped rotor. Modern steppers are of hybrid design,
having both permanent magnets and soft iron cores. The electromagnets are energized by an external driver circuit or a microcontroller."
Domain =
- electrical
- magnetic
- mechanical

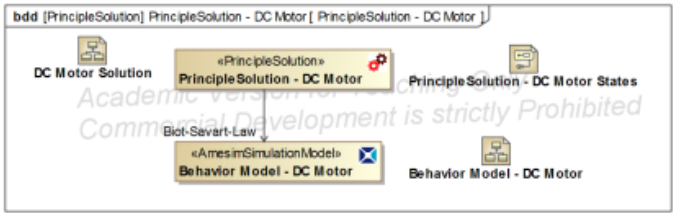
Structural_model = Stepper Motor
Requirements =
- Electric Power Supply
- Electromagnetic Compatibility Testing
- Rotational Energy Provision
- Minimal Step Size

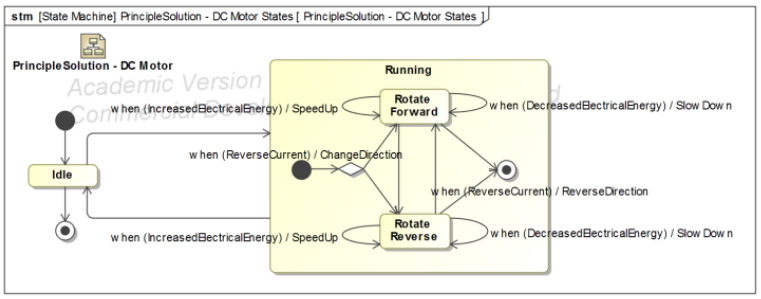
Examples = MotionKing 17H2A Motors, used in the Raptype 3D printer
Solution ID = "S-BK-003-02"

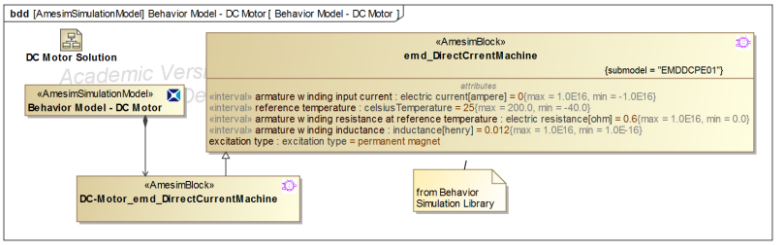


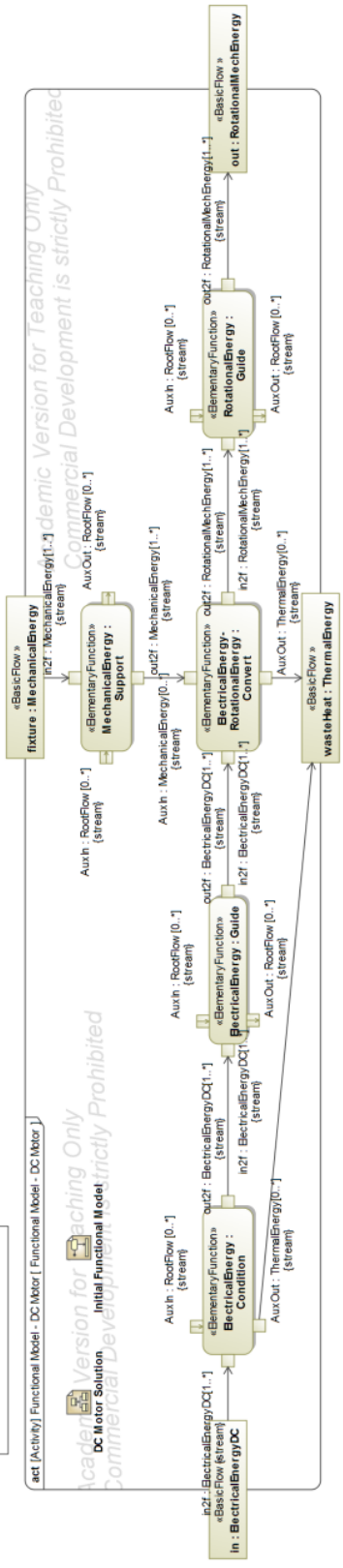
DC Motor Solution:

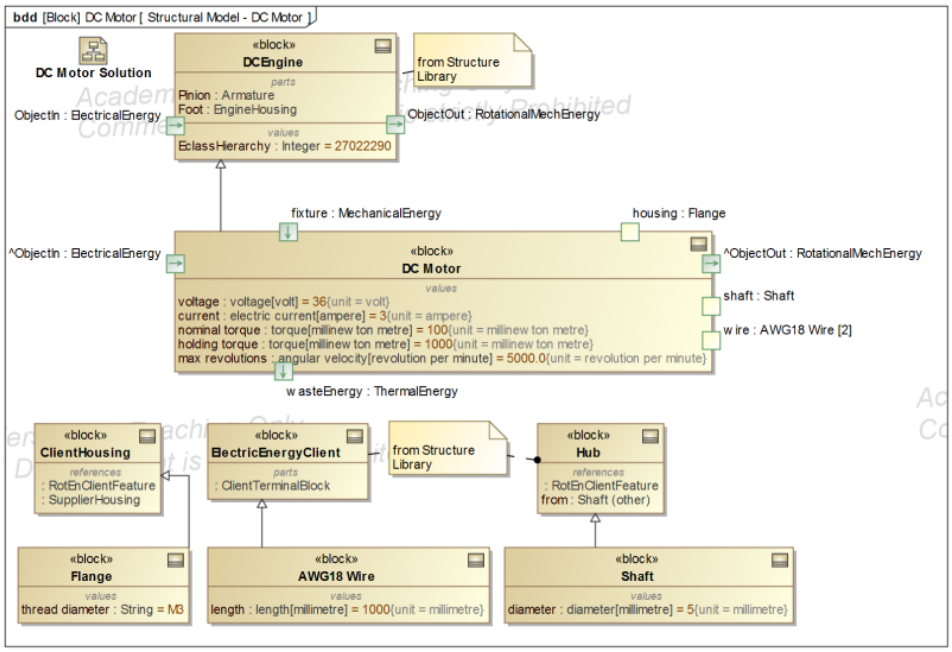


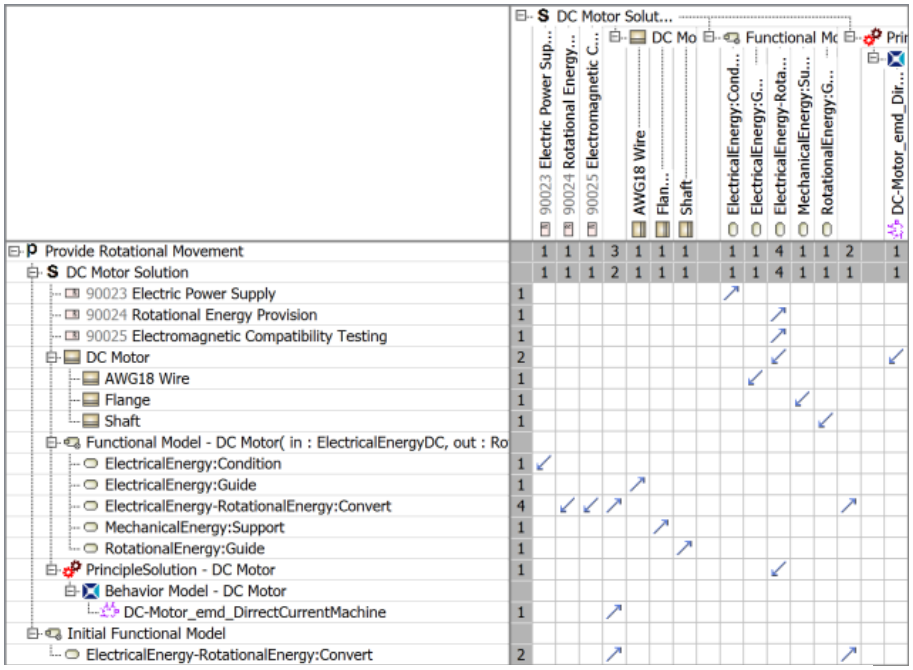


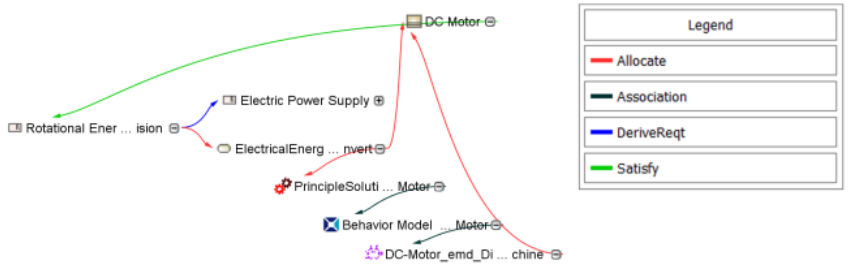




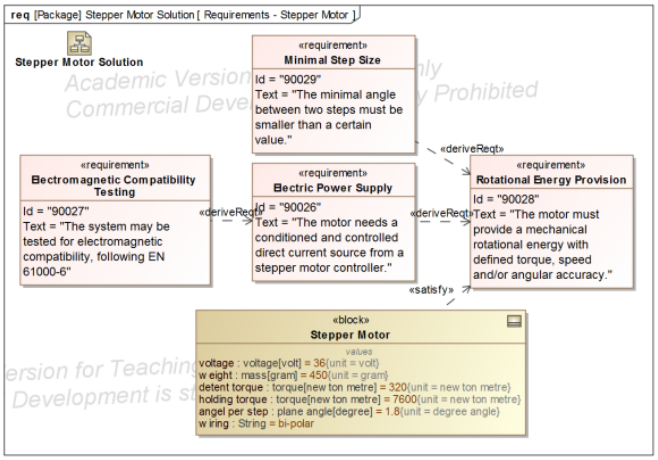


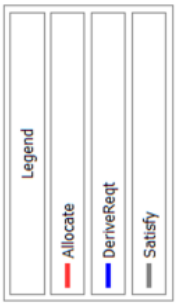

bdd [Block] DC Motor [Structural Model - DC Motor]
DC Motor Solution
«block»
DCEngine
parts
Pinion : Armature
Foot : EngineHousing
values
EclassHierarchy : Integer = 27022290
from Structure Library
ObjectIn : ElectricalEnergy
ObjectOut : RotationalMechEnergy
fixture : MechanicalEnergy
housing : Flange
^ObjectIn : ElectricalEnergy
«block»
DC Motor
values
voltage : voltage[volt] = 36{unit = volt}
current : electric current[ampere] = 3{unit = ampere}
nominal torque : torque[millinew ton metre] = 100{unit = millinew ton metre}
holding torque : torque[millinew ton metre] = 1000{unit = millinew ton metre}
max revolutions : angular velocity[revolution per minute] = 5000.0{unit = revolution per minute}
^ObjectOut : RotationalMechEnergy
shaft : Shaft
w ire : AWG18 Wire [2]
w asteEnergy : ThermalEnergy
«block»
ClientHousing
references
: RotEnClientFeature
: SupplierHousing
«block»
ElectricEnergyClient
parts
: ClientTerminalBlock
from Structure Library
«block»
Hub
references
: RotEnClientFeature
from : Shaft (other)
«block»
Flange
values
thread diameter : String = M3
«block»
AWG18 Wire
values
length : length[millimetre] = 1000{unit = millimetre}
«block»
Shaft
values
diameter : diameter[millimetre] = 5{unit = millimetre}

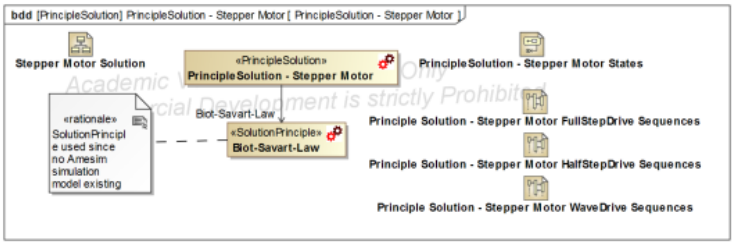

DC Motor Solut...
DC Mo
Functional Mo
Prin
90023 Electric Power Sup...
90024 Rotational Energy...
90025 Electromagnetic C...
AWG18 Wire
Flan...
Shaft
ElectricalEnergy:Cond...
ElectricalEnergy:G...
ElectricalEnergy-Rota...
MechanicalEnergy:Su...
RotationalEnergy:G...
DC-Motor_emd_Dir...
Provide Rotational Movement
DC Motor Solution
90023 Electric Power Supply
90024 Rotational Energy Provision
90025 Electromagnetic Compatibility Testing
DC Motor
AWG18 Wire
Flange
Shaft
Functional Model - DC Motor(in : ElectricalEnergyDC, out : Ro
ElectricalEnergy:Condition
ElectricalEnergy:Guide
ElectricalEnergy-RotationalEnergy:Convert
MechanicalEnergy:Support
RotationalEnergy:Guide
PrincipleSolution - DC Motor
Behavior Model - DC Motor
DC-Motor_emd_DirrectCurrentMachine
Initial Functional Model
ElectricalEnergy-RotationalEnergy:Convert

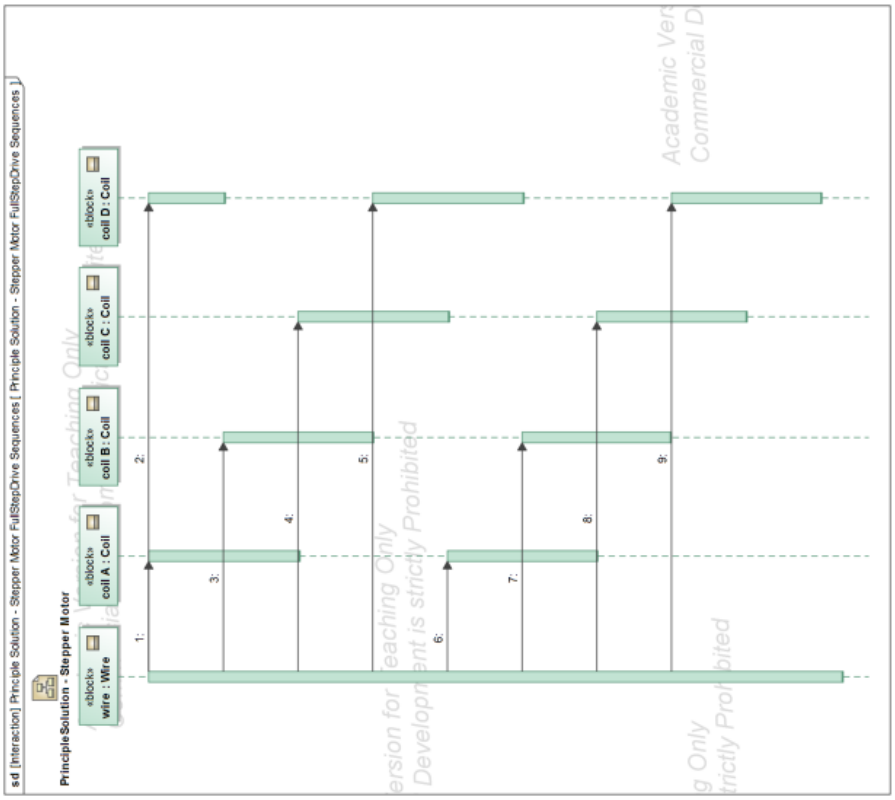

DC Motor
Electric Power Supply
Rotational Ener ... ision
ElectricalEnerg ... nvert
PrincipleSoluti ... Motor
Behavior Model ... Motor
DC-Motor_emd_Di ... chine
Legend
Allocate
Association
DeriveReqt
Satisfy

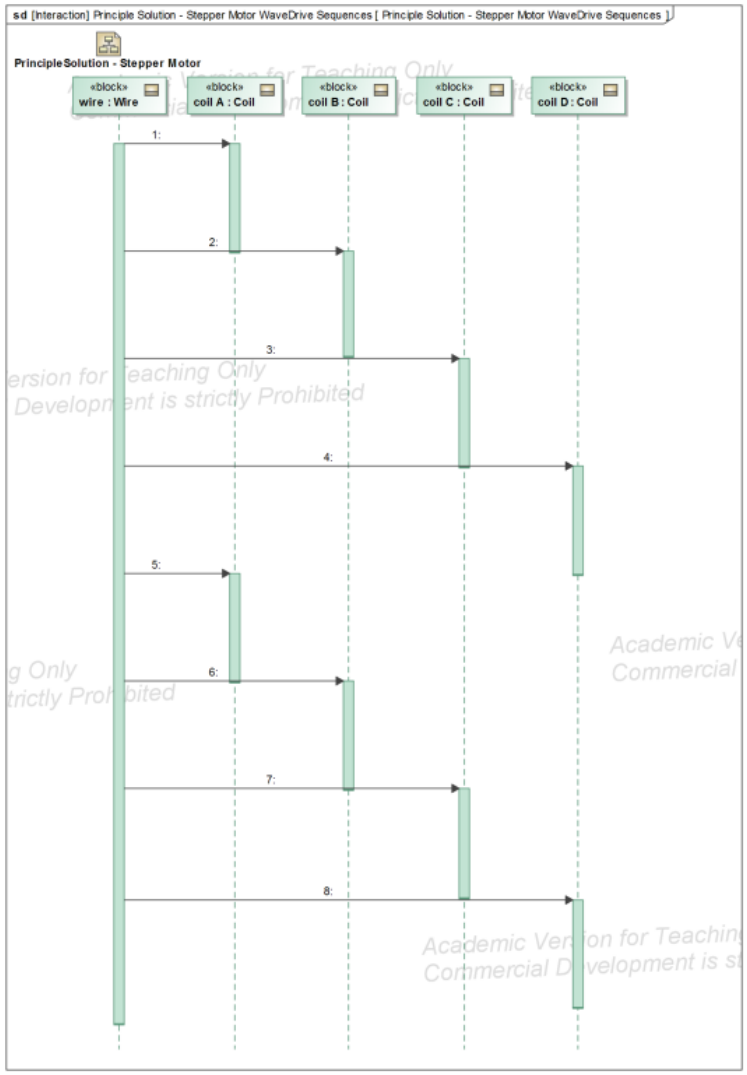
Stepper Motor Solution:

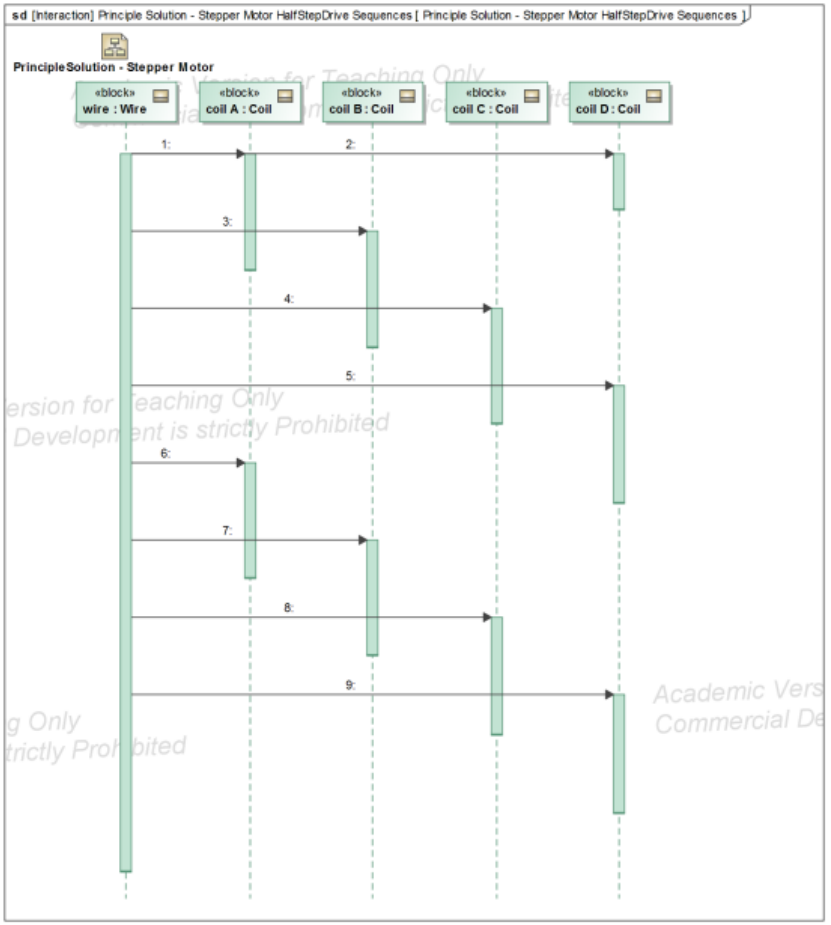


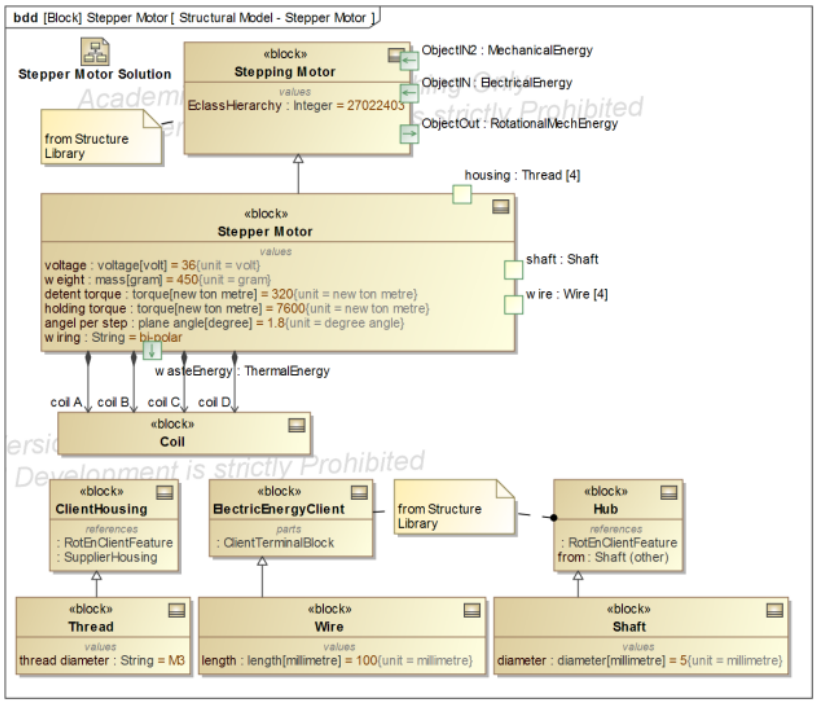


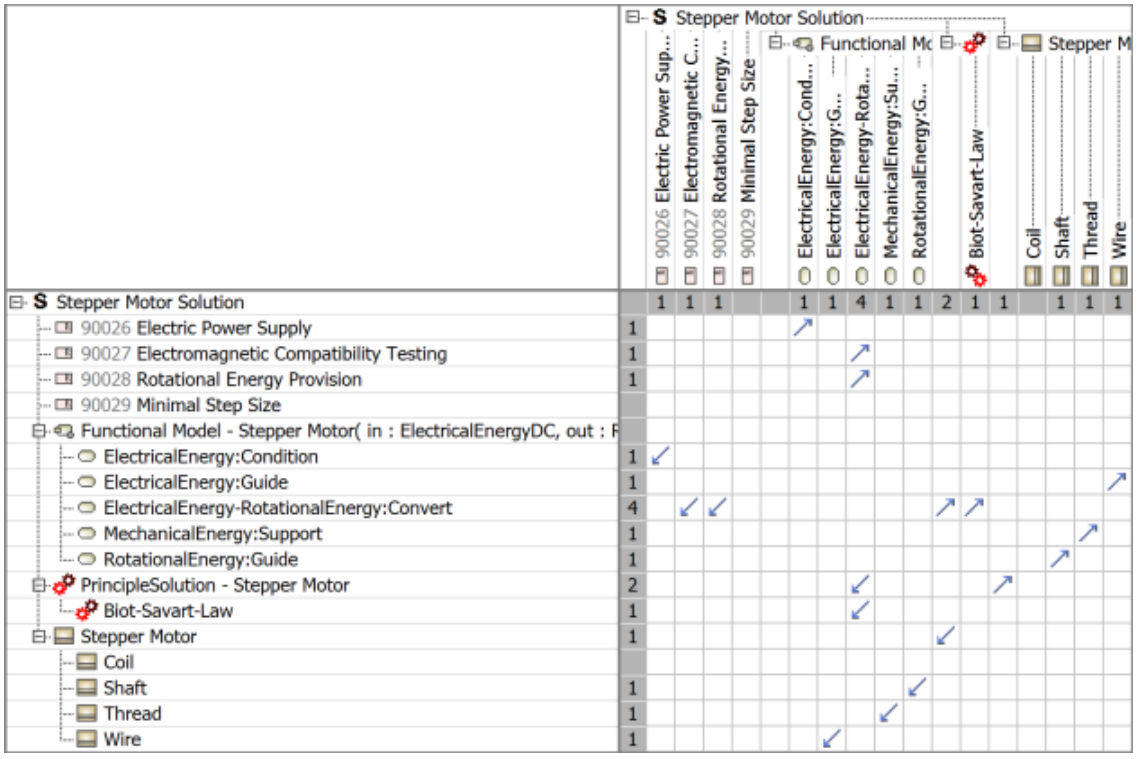





sd [Interaction] Principle Solution - Stepper Motor WaveDrive Sequences [Principle Solution - Stepper Motor WaveDrive Sequences]
PrincipleSolution - Stepper Motor
«block» wire : Wire
«block» coil A : Coil
«block» coil B : Coil
«block» coil C : Coil
«block» coil D : Coil
1:
2:
3:
4:
5:
6:
7:
8:


sd [Interaction] Principle Solution - Stepper Motor HalfStepDrive Sequences [Principle Solution - Stepper Motor HalfStepDrive Sequences]
PrincipleSolution - Stepper Motor
«block» wire : Wire
«block» coil A : Coil
«block» coil B : Coil
«block» coil C : Coil
«block» coil D : Coil
1:
2:
3:
4:
5:
6:
7:
8:
9:


bdd [Block] Stepper Motor [Structural Model - Stepper Motor]
Stepper Motor Solution
«block»
Stepping Motor
values
EclassHierarchy : Integer = 27022403
ObjectIN2 : MechanicalEnergy
ObjectIN : ElectricalEnergy
ObjectOut : RotationalMechEnergy
from Structure Library
housing : Thread [4]
«block»
Stepper Motor
values
voltage : voltage[volt] = 36{unit = volt}
w eight : mass[gram] = 450{unit = gram}
detent torque : torque[new ton metre] = 320{unit = new ton metre}
holding torque : torque[new ton metre] = 7600{unit = new ton metre}
angel per step : plane angle[degree] = 1.8{unit = degree angle}
w iring : String = bi-polar
shaft : Shaft
w ire : Wire [4]
w asteEnergy : ThermalEnergy
coil A
coil B
coil C
coil D
«block»
Coil
«block»
ClientHousing
references
: RotEnClientFeature
: SupplierHousing
«block»
ElectricEnergyClient
parts
: ClientTerminalBlock
from Structure Library
«block»
Hub
references
: RotEnClientFeature
from : Shaft (other)
«block»
Thread
values
thread diameter : String = M3
«block»
Wire
values
length : length[millimetre] = 100{unit = millimetre}
«block»
Shaft
values
diameter : diameter[millimetre] = 5{unit = millimetre}


Stepper Motor Solution
90026 Electric Power Sup...
90027 Electromagnetic C...
90028 Rotational Energy...
90029 Minimal Step Size
Functional Mc
ElectricalEnergy:Cond...
ElectricalEnergy:G...
ElectricalEnergy-Rota...
MechanicalEnergy:Su...
RotationalEnergy:G...
Biot-Savart-Law
Stepper M
Coil
Shaft
Thread
Wire
Stepper Motor Solution
90026 Electric Power Supply
90027 Electromagnetic Compatibility Testing
90028 Rotational Energy Provision
90029 Minimal Step Size
Functional Model - Stepper Motor(in : ElectricalEnergyDC, out : F
ElectricalEnergy:Condition
ElectricalEnergy:Guide
ElectricalEnergy-RotationalEnergy:Convert
MechanicalEnergy:Support
RotationalEnergy:Guide
PrincipleSolution - Stepper Motor
Biot-Savart-Law
Stepper Motor
Coil
Shaft
Thread
Wire

APPENDIX B – USER STUDY QUESTIONNAIRE

Please read the following questions carefully. If anything is unclear feel free to ask.

Name: _____

PRIOR EXPERIENCE:

Prior experience with SysML (or UML): □ No □ Yes

If Yes, please specify: _____

Prior experience with Magicdraw (or similar software): □ No □ Yes

If Yes, please specify: _____

Prior experience with Functional Modeling: □ No □ Yes

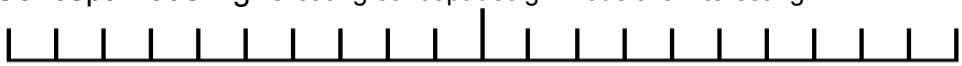
If Yes, please specify: _____

Prior experience about Model-based Systems Engineering: □ No □ Yes

If Yes, please specify: _____

GENERAL QUESTIONS:

Concept Modeling: Creating concept design models is interesting.

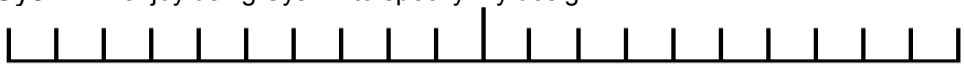


Disagree Agree

Concept Modeling: I think that formal modeling of system concepts is important.

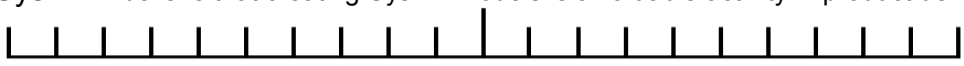
Disagree Agree

SysML: I enjoy using SysML to specify my design.



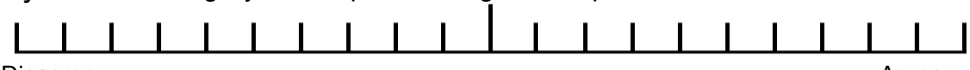
Disagree Agree

SysML: I believe that creating SysML models is a valuable activity in product development.



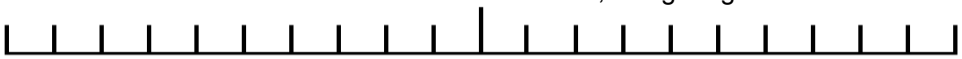
Disagree Agree

SysML: Learning SysML helps me design better products.



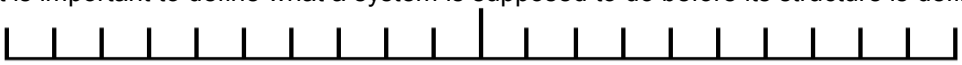
Disagree Agree

Tool: I was able to create the models as I intended, using Magicdraw.



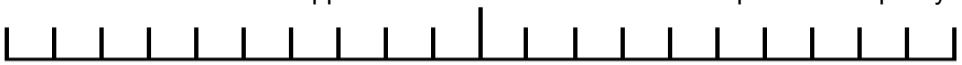
Disagree Agree

Functional Modeling:
It is important to define what a system is supposed to do before its structure is defined.



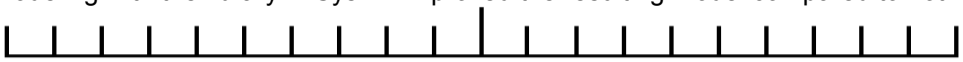
Disagree Agree

Functional Basis: The application of the Functional Basis improved the quality of the model.



Disagree Agree

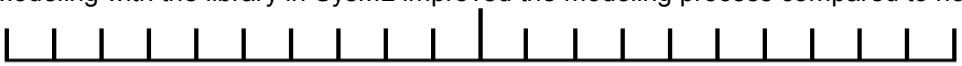
Function Library:
Modeling with the library in SysML improved the resulting model compared to not having the library.



Disagree Agree

Please explain your answer:_____

Function Library:
Modeling with the library in SysML improved the modeling process compared to not having the library.



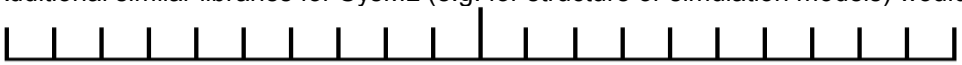
Disagree Agree

Please explain your answer:_____

Function Library:
I prefer having the library when doing functional modeling in SysML compared to not having it.

Disagree Agree

Libraries:
Additional similar libraries for SysML (e.g. for structure or simulation models) would be advantageous.



Disagree Agree

Function Library: What did you like most when doing functional modeling using the library?

Answer here: _____

Function Library: What could be improved regarding functional modeling using the library?

Answer here: _____