TECHNISCHE UNIVERSITÄT MÜNCHEN

Fachgebiet Anwendungen der Virtuellen Produktentwicklung

Model Libraries for Conceptual Design

Stefan J. Wölkl

Vollständiger Abdruck der von der Fakultät für Maschinenwesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr-Ing. U. Lindemann
Prüfer der Dissertation:	1. Univ.-Prof. K. Shea, Ph.D., Eidgenössische Technische Hochschule Zürich | Schweiz
	2. Univ.-Prof. Dr.-Ing. B. Vogel-Heuser

Die Dissertation wurde am 16.10.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 6.02.2013 angenommen.

Listed Publications are part of this thesis, recent publications are presented first.

Kruse, B.; Münzer, C.; Wölkl, S.; Shea S.; Canedo, A.; Workflow and Modeling Conventions for FBS Modeling of Mechatronic Systems in SysML using Libraries, International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. Chicago, IL, 2012

Wölkl, S.; Shea, K.: Modellbibliotheken für die Funktions- und Komponentenmodellierung in der Konzeptentwicklung mit SysML, Wissenschaftsforum Intelligente Technische Systeme 2011, 8.Paderborner Workshop Entwurf Mechatronischer Systeme, HNI Verlagsschriftenreihe Band294, Paderborn, 2011

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

DANKSAGUNG

Diese Arbeit entstand im Rahmen meines Engagements am Fachgebiet Anwendungen der virtuellen Produktentwicklung neben meiner hauptberuflichen Tätigkeit in der Technischen Entwicklung der Audi AG. Mein besonderer Dank geht an Prof. Dr. Kristina Shea, die mir durch ihre Aufnahme in das Fachgebiet die faszinierende Welt der wissenschaftlichen Arbeit eröffnete. Durch ihre scharfsinnigen, aber stets fairen Reviews entstand aus dem „wishy-washy“ der ersten wissenschaftlichen Gehversuche eine wissenschaftlich erstzunehmende Arbeit. Außerdem schaffe ich es nun durch ihr „Practice, practice, practice“! Präsentationen in einem verständlichen Englisch zu halten. Thanks!

Die Anregung zum Promotionsvorhaben gab Dr. Felix Tropschuh. Meine Vorgesetzten der Audi AG unterstützten mein Projekt, indem sie mir eine flexible Zeiteinteilung zugestanden und die Durchführung der Modellierungsstudie bewilligt haben.

Die Mitarbeiter des Lehrstuhls für Produktentwicklung nahmen mich als Externen freundschaftlich auf und boten ein perfektes Übungsfeld für akademische Vorträge und den akademischen Wettbewerb. Die Kollegen des Lehrstuhls standen mir stets mit Rat zur Seite und bereicherten manchen Tag mit heiteren und kulinarischen Mittagsrunden.

Dank gilt den VPD-Kollegen für die hervorragende Zusammenarbeit, Unterstützung, Anregungen und Hinweise zur Literatur und das ehrliche und aufrichtige Feedback zu wissenschaftlichen Themen. Besonders brachte mich der intensive Austausch über die Modellierungsmethoden und Best Practice mit Benjamin, Bergen, Clemens und Torsten voran. Dankeschön!

In Promotionsflauten ließ meine Frau Karolin den Wind der Begeisterung wieder aufkommen, organisierte die familiären Aktivitäten so, dass unsere Kinder Maria, Simon, Andreas und Veronika stets gut versorgt waren und mir genug Zeit blieb, um zwischen fordernden Berufs- und Familienleben diese Arbeit abzuschließen. Der starke Zusammenhalt in der Familie und die intensive Unterstützung durch die Großfamilie Finger machten es möglich, dass ich auch an Wochenenden forschen konnte. Herzlichen Dank, Maria, Richard, Hilde und Katrin! Bei meinen Eltern bedanke ich mich, dass sie mir durch ihre weitsichtige Entscheidung die akademische Laufbahn ermöglicht haben und mir auch zeigten, was es bedeutet große Felder zu beackern und die Früchte zu ernten.

Mein Dank gilt allen Freunden und Arbeitskollegen, die mir durch Zusprüche, wohlformulierte und durchdachte Kritik und großen Einsatz Mut gemacht und mich bestärkt haben. Aus ganzem Herzen: Danke!

Manching, April 2013 Stefan Wölkl

CONTENTS:

MODEL LIBRARIES FOR CONCEPTUAL DESIGN

Figures

1 Introduction

In product design the single tasks to specify and design a product or system are defined in the design processes. In many of them the overall task spans from the first idea to the complete product ready for sale. Conceptual design is an early phase where the problem is identified and decomposed into manageable pieces. Then for any particular problem, solutions are searched for. These are synthesized, meaning assembled, to create the whole artifact. Though stated in guideline VDI 2221 [1993], an integrated computational support is not common in industry today. Instead, the single tasks supported by paper based models are shifted to the computer as spread sheets and text documents. If single tasks are supported by a modeling tool, like requirements management, these tools are only connected via result documents to other tasks. A core part of conceptual design is functional modeling. From a function structure, components that embody the function can be identified and synthesized into a solution. This is at the same time a methodical approach and a creative task. The creativity can be supported by providing the right components for the embodiment. Since with conceptual design, the function of the product and its core properties are laid out, it is important to cover the majority of modeling in conceptual design in an integrated way.

At the beginning of the nineties original theories for mechatronic design appeared [BUUR 1990]. Mechatronic design is the academic field that describes the processes, methods and tasks to design mechatronic products and systems. These are products that integrate functionally interacting artifacts of the domains of mechanical, electric and software engineering, e.g. a numerical controlled milling machine. With mechatronic design, the products became more complex and the inter domain communication is a key factor for designing mechatronic products. Thus, it gets more difficult to manage consistent documents and models throughout conceptual design. Still functional modeling is a common task of all involved disciplines.

Systems engineering is a discipline that also deals with the specification, design and certification of domain spanning and complex systems. In contrast to mechatronic design, systems engineering is not limited to the abovementioned domains but also can integrate astrophysics, chemistry or others. In a systems engineering viewpoint, physical products are often part of a complex system like mobile phones that are part of a world spanning communications system. The methods for architecting software intensive systems and the need for gapless documentation of systems engineering lead to the Model Based Systems Engineering (MBSE) approach. MBSE is a systems engineering paradigm with the aim to create a holistic computer based system model instead of creating paper documentation in parallel to disconnected models [ESTEFAN 2008]. A MBSE Methodology describes the related methods, tools, processes and resources. Most of the model based approaches make use of a modeling language capable of supporting consistent models that provide a system overview. One of the most widely used modeling languages is the Systems Modeling Language (SysML) that is also used in this thesis.

In this work, the conceptual design process in product and mechatronic design and MBSE approaches are compared. Common tasks and terms are discovered. Tasks of the methodical approach of conceptual design are transferred to SysML models. Thus, the majority of conceptual design tasks can be realized as an integrated computational model. SysML model

libraries are introduced to improve the consistency of functional models, enable the comparison of function structures and provide databases for conceptual design tasks. The approach is validated using a simple mechanical product, a luggage compartment cover, and a more complex mechatronic product, a 3D-desktop printer.

1.1 Motivation for integrated modeling and model libraries

Conceptual design faces a general problem: there are many methods of great maturity described in text books. However, there is hardly commonly used dedicated computational support for single tasks. Since ages it is commonly known that the costs and efforts of conceptual design are contrasted by the costs that are allocated to the whole product. This fact is stressed in academia as well as in industry, presented as charts as well as formulated as a heuristic. "…all serious mistakes are made in the first day" [MAIER & RECHTIN 2009].

At current state of conceptual design poor integration of models and tools can be observed. Models for requirements, component structure, cost, weight, function, behavior and space are separate, but not independent. Kreimeyer [2010] states that processes and documents are interlinked. With documents, it is difficult to obtain an overview of their interrelations. Portions of one document are manually transferred to another document. Thus, updating one document needs an update of other documents. An evaluation, if all components that embody a function are budgeted, is difficult since the functional model, component model and budget model are written down in different documents. Moreover, in automotive design, the tasks are distributed to different departments upfront. Thus, the teams designing the models describing an integrated function are also distributed. The author experienced it as a difficulty to ensure the consistency of the functional model and the component structure due to distributed documents. In the worst case, a particular component is not considered in conceptual design and an option cannot be offered to the customer. Motivated to find an improvement, it was searched for approaches that allow integrated modeling in conceptual design. In an empirical study, the MBSE approach together with SysML showed the potential to provide computational support for conceptual design [WÖLKL & SHEA 2009].

Functional modeling used as a paper-based method for the selection of principle solutions, i.e. sketches of artifacts that make use of a physical effect, set the requirements for formality in functional modeling. ROTH [2000] and KOLLER [1998] meet that requirement with a lexicon, i.e. a collection of all possible building blocks for functional modeling, of formally defined functions allowing the mapping of principle solutions. Although in the approach of this work a direct mapping of functions to components is proposed, the formal modeling of functions is necessary to exploit mapping to components. Integrating sets of predefined elements in the modeling environment for SysML requires modeling libraries, e. g. for functional modeling and component modeling.

In an industry project the, MBSE approach was tested by modeling the luggage compartment cover with SysML. Interviewing the engineers, it proved that modeling in conceptual design still was paper based, but in electronic development the requirements are described in a requirement management tool and state machine and activity diagrams (both possible with SysML) are required from the supplier to document the concept of how the control unit

works. In the same project students modeled conceptual tasks with SysML. They first followed the approach of Systems modeling (SYSMOD) [WEILKIENS 2006], then adopted modeling according guideline VDI 2201 [1987]. They complained about the fact that they had to define a huge number of elements before they could actually use them. Providing modeling elements for reuse is also an aspect that requires model libraries to improve the usability of SysML modeling.

1.2 Contributions

Similarities of Model Based Systems Engineering (MBSE) approaches and Systematic Product Development [VDI 2221 1993] are discovered. Requirements management, functional modeling and modeling of a product structure are included in both. MBSE approaches provide modeling with languages and integrated modeling that are not included in product development approaches so far. Due to the similarity, MBSE approaches can be adapted to product development tasks and modeling languages can be used. An overview of MBSE Approaches is given in Chapter 2.

The National Institute of Standards Technology (NIST) Functional Basis provides comprehensive function and flow taxonomies. Ambiguities and missing explicit information have been discovered that required additional information for the definition of a function model library.

The concepts of functions have been defined and transferred to SysML. Based on this, the object oriented concept of inheritance and the concrete SysML syntax of call behavior actions are exploited to build up a model library. The definition of flows and functions as library elements guides the user during the selection of appropriate functions, i.e. by the documentation of functions. In the model library, predefined flows, i.e. material, energy and signal and their specifications, help to set a more specific flow during the modeling.

The component library is set up and filled with elements of E-Cl@ss a product and service classification standard. Meaningful components for synthesis are transformed to SysML blocks and augmented with ports. Although only a small portion of the PSCS was transferred to the model library, it was difficult to keep the overview. Browsing large model libraries and finding library elements by combination of attributes are the limiting factors.

The concept for the identification of components by the mapping of incoming and outgoing flows to input- and output- ports is defined. In this concept, the interacting flows of functions and the interacting ports of components have the same type. The components can be searched for in the model library according to the flows of the functions.

Abovementioned findings are shown with a modeling example and validated with a validation study. The validation study is more complex and needs an additional layer of decomposition that can be managed with the library support. It demonstrates the concepts and shows that the approach is generally applicable.

2 Literature and related work

This thesis involves the domains of systems engineering and product design. The domains are highly intertwined. The first reviewed topic is literature on design processes with a focus on conceptual design and on systems engineering approaches. Product modeling and architectural description methods are tightly linked to design processes and the systems engineering process respectively. Models are used for problem description, evaluation of design alternatives and finally product documentation. Functional modeling and component modeling are common tasks of both systems engineering and product design. Literature on functional modeling and the definition of function in systems engineering reveals differences in terms of function and behavior. Modeling languages are the enablers to build highly interrelated models that cover several areas of an integrated product model. Lexicons, catalogs, taxonomies, model libraries and repositories are examples for information sources for engineering tasks that are reviewed here.

2.1 Design processes

In this section an overview of relevant processes of design and systems engineering is presented. Similarities and differences of the domains of mechanical design, mechatronic design and systems engineering are highlighted and differences are resolved.

2.1.1 Design processes in product design

A general task of engineering design is to create artifacts that satisfy human needs. [PAHL et al. 2007] Although mechanical engineering can be considered to start from 1875 [RAULEAUX 1876], the first decades were devoted to machine theory. From the 1960s genuine theories of the process of synthesizing or creating machines and mechanical products began to appear. [ANDREASEN 1991] Another decade later, the way of human thinking was considered. The systematic approach of Pahl and Beitz was developed in the 1970s and is one of the most representative design processes of the European school. [PAHL et al. 2006]

Systematic product development starts with a general description of the design process and ends with detailed methods to realize concrete steps of single tasks within that process. Pahl and Beitz structure their design process into four phases: “clarification of task”, “conceptual design”, “embodiment design” and “detailed design”. This approach is also integrated in several American textbooks [ULLMAN 2002, ULRICH & EPPINGER 1994]

Common concepts of conceptual design stages from the mentioned authors are requirements, functional modeling and working structure modeling. First the task is described and requirements are collected and ordered. Then inputs and outputs of the system or product are determined and the main function is identified. The main function of the product is broken down into solution neutral sub-functions. At the lowest break down level, physical effects realized by working principles are assigned to the functions. This procedure assures a great

variety of different designs. The best designs can be selected and handed over to embodiment design.

Next to the mainstream trace there are some others. Hubka & Eder [1988] describe technical systems through four viewpoints: "**processes or transformations**" that mirror the technology of the product (mechanical, electrical, etc.), "**functions**", providing the effects for the processes, "**organs**", that create the effects by applying a working principle and interact with other organs and "**machine parts**" that embody the organs and can be assembled to the system or machine. Although to the term function is used in Hubka & Eder's [1988] work and abovementioned design processes, the applied concepts of functional modeling and functional decomposition is different. Modeling the overall function is similar to modeling processes or transformations. The significant difference is that processes are described in terms of purpose functions, meaning functions that describe an "expedient effect of an artifact" [BUUR 1990]. Then the decomposition of functions into sub functions is only possible when a means for its realization has been selected. A functions/means tree is the preferred diagram to present this. It is contrasting the systematic approach of Pahl and Beitz, where the overall function can be decomposed into sub functions without assigning means to functions.

Axiomatic Design [SHU 1998] is a theoretical design approach that starts the development of a product from functional requirements (FR) that are linked to design parameters (DP) and to process variables (PVs). Several axioms guide the designer through the design process. The most important one is the independence axiom. It states that FRs have to be independent from other FRs. Design parameters can be components as well as single surfaces that provide FRs. Axiomatic design is a process that requires rigorous formal modeling and analysis of relations between single model elements. The allocation of FRs to modules is done with design matrix operations. A common point to this thesis is the linking of functions (or FRs) to working principles (or DPs) and the use of function (or FR) as a decisive step in synthesis. Although Aximotic Design uses a way of linking from requirements (FRs) to components (DPs) it uses a completely different metamodel describing the relations among them.

A consolidated view of systematic processes and methods for product development is the VDI guideline 2221 entitled "Systematic Approach to the Design of Technical Systems and Products". [VDI 2221 1987] It was developed in Germany in 1986, revised in 1993 [VDI 2221 1993] and contains a design process that is illustrated by examples. It is the result of a joint effort of academia and industry to provide an efficient and commonly accepted design process as a standard. The process shown there is high level and greatly influenced by the approach of Pahl & Beitz. This standard is also interesting for this research since it describes the requirements for computational support.

The overview in Fig. 1 illustrates the product development process in the VDI guidelines 2221 and 2222. On the left hand side, the lifecycle phases of products and systems are provided. Design is a short but decisive phase at the beginning of the product lifecycle. The middle column breaks down "Design" into three phases, each of which is broken down into a series of stages. These stages in turn are decomposed to single tasks that have to be completed. Conceptual design is chosen for the scope of this work as it is generally thought to be the most difficult for computational modeling and is the area where more support is needed.

As the guideline is generic for the mechanical engineering domain and the most common in Germany, it is used to define basic model requirements for supporting conceptual design. These include modeling requirements, functions and function structures, working principles and working structures as well as physical effects. At this level, these main tasks can be found in various conceptual design processes, thus making the models developed applicable as a starting point for modeling within other design processes.

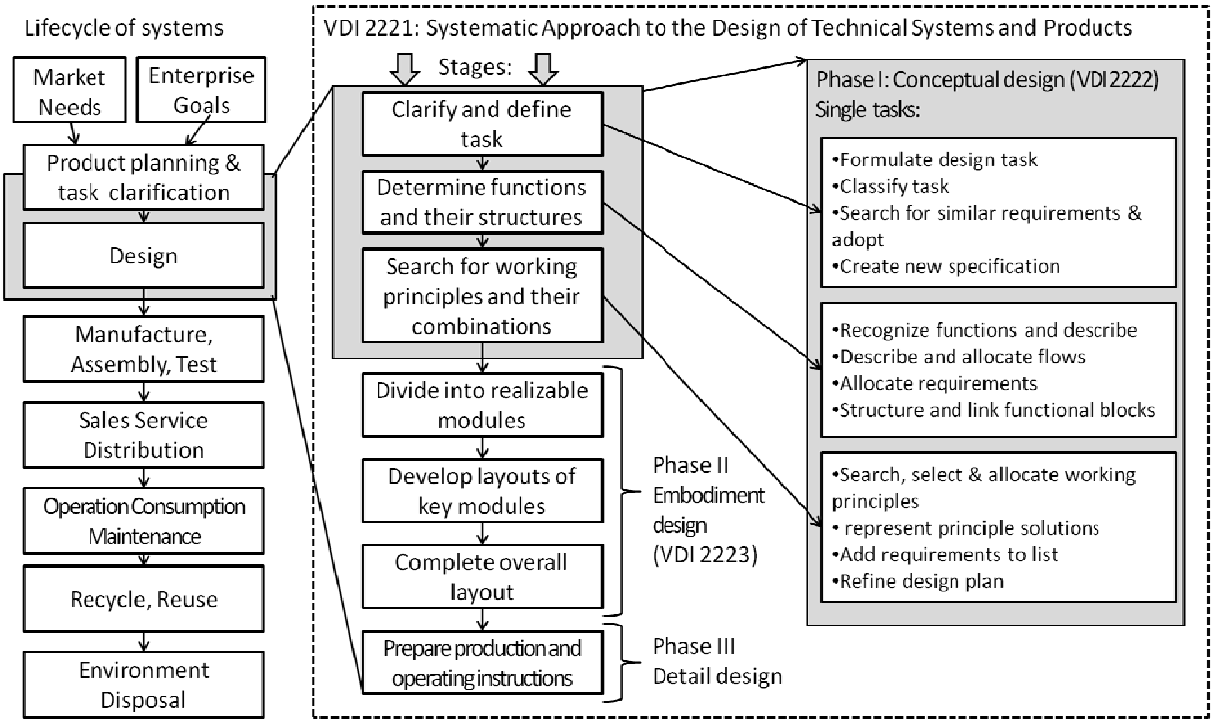

Figure 1: Overview of VDI guidelines for product design, adopted form [VDI 2221 1993, VDI 2222 1997]

Phase I: Conceptual design is broken down into single tasks (Figure 1, right hand side). The first task is to formulate and clarify the design task. Clarification is done by discussion with stakeholders. Formulation involves the definition of a short text that exposes the main goals of the task. Typically this text should be agreed on with all stakeholders and developers. Using a modeling tool for this particular task is probably unnecessary. However, it makes sense to document the result of this task in the product model to provide formalized documentation of this stage [WEILKIENS 2006]. It should be placed at a high level in an integrated product model.

The first requirements can usually be inferred from the overall task description. If there is no existing list of requirements from previous designs, the designer may use a table of categories to identify types of requirements. Requirements can be asked for by stakeholders of the product. Requirements are formulated in a solution neutral manner. The last task in this stage is to draft a design process plan, often supported by project management tools.

The second stage begins with defining the overall function and basic sub-functions along with

their structures. The overall function can optionally be stated form the user perspective. The decomposition into sub-functions can then be accomplished with appropriate decomposition methods. According to Pahl et al. [2007] a function is a solution neutral description of transitions from inputs to outputs represented by a verb-noun pair. Consequently it is important in this stage to remain solution neutral at a high level of abstraction.

The decomposition of a primary function into basic sub-functions can also be used in adaptive design. It deals with the adoption of a legacy product to an extended purpose, e.g. a printer that is extended with a scanner to work as a photocopier. The focus of modeling functions in this case is on varying the structure of functions by means of replacing, reordering, adding or deleting functions. For new products, function structure alternatives need to be generated and evaluated.

Assigning flows to functions establishes the necessary connectivity that is then realized by working principles. Flows can be material, energy and signal. The appearance of flows, either discrete or continuous, can be modeled as well. Relating requirements is a means of linking required information to functions to simplify the selection of physical effects and principle solutions in the next stage.

The third and last stage of conceptual design is decomposed into several tasks. In the first task working principles are allocated to the abstract functional structures worked out in the previous stage. Traditionally the task of allocating working principles is alleviated by collections of physical effects and design catalogs of working principles and matrices to find working principles by input and output variables described by ROTH [2000] and Koller [1998]. Working principles are driven by a physical, chemical or biological effect that is represented by a mathematical equation. The advantage of utilizing only known physical effects is that the number of applicable effects is limited. To find innovative solutions it is essential to find all feasible effects that realize working principles. Requirements, in the sense of design constraints, are used to determine if a working principle can be applied. For some effects the input-output ratio within given constraints is too weak for them to be applied.

Still principle solutions have to be combined to create the overall product structure. Here, the interfaces between combined principle solutions must be matched by type and by value. For example, a working principle with an output of mechanical rotational energy defined by rotational velocity and torque can only be combined with a working principle with the same input types and values. If outputs of one do not match the inputs of another principle solution, a transformation of energy function has to be added to the function structure. This auxiliary function has to be realized with a working principle as well, e.g. the effect of cohesion embodied in a gear pair.

Another task is to present the working principle in an easily understandable manner. Sketches help paper-based methods and also portray initial dimensions. In this stage new requirements can be discovered that must be added to refine the previous collection of requirements.

During conceptual design several principle solutions for the overall product are created. The final task of conceptual design, though not mentioned in Figure 1, is the selection of appropriate solutions. If performance and cost of several solutions are similar, the selection of the design favorite is a challenge to the designer.

This design process as outlined in the VDI 2221 [1993], especially the part for conceptual design is the basis for systematic modeling in this thesis. The VDI guideline has been selected since it is a consolidated design process with contributions from academic and industry and for the information that is given on computational support.

As shown above, different processes and approaches exist. Nevertheless, all of them provide some ordering of tasks to reach a good design. Although the design processes for mechanical design are mature and understandable to designers of that domain, interaction with other domains today is common. The following section presents two design processes for mechatronic systems.

2.1.2 Approaches to design of mechatronic systems

Mechatronic design plays a special role in the field of design. It is a rather young discipline that includes the three domains of mechanical engineering, electronic engineering and software engineering. Engineers of the three disciplines have to work together in an efficient way. To manage this, the VDI guideline “Design Methodology for Mechatronic Systems” proposes the V-Model (see Figure 2) as the procedural model at macro level. [VDI2206 2006] Adapted to mechatronic design the system design and system integration is carried out together with involved domains and the design tasks are accomplished concurrently in the involved domains. The “V” is run through iteratively several times for conceptual design, embodiment design and detail design. The methods and processes are taken from the involved domains, e.g. guideline VDI 2222 [1997] for mechanical design. Major challenges of mechatronic design are the synchronization of the involved designs and developing an understanding of the other involved domains. A property of systems is complexity. Drivers for complexity are number of different elements amount of different relations. Mechanical, electric, electronic and software components are of different types (due domain) and have different kind of relations. This justifies using the term mechatronic system although the system can be an integrated product e.g. a laptop computer or a numerically controlled milling machine.

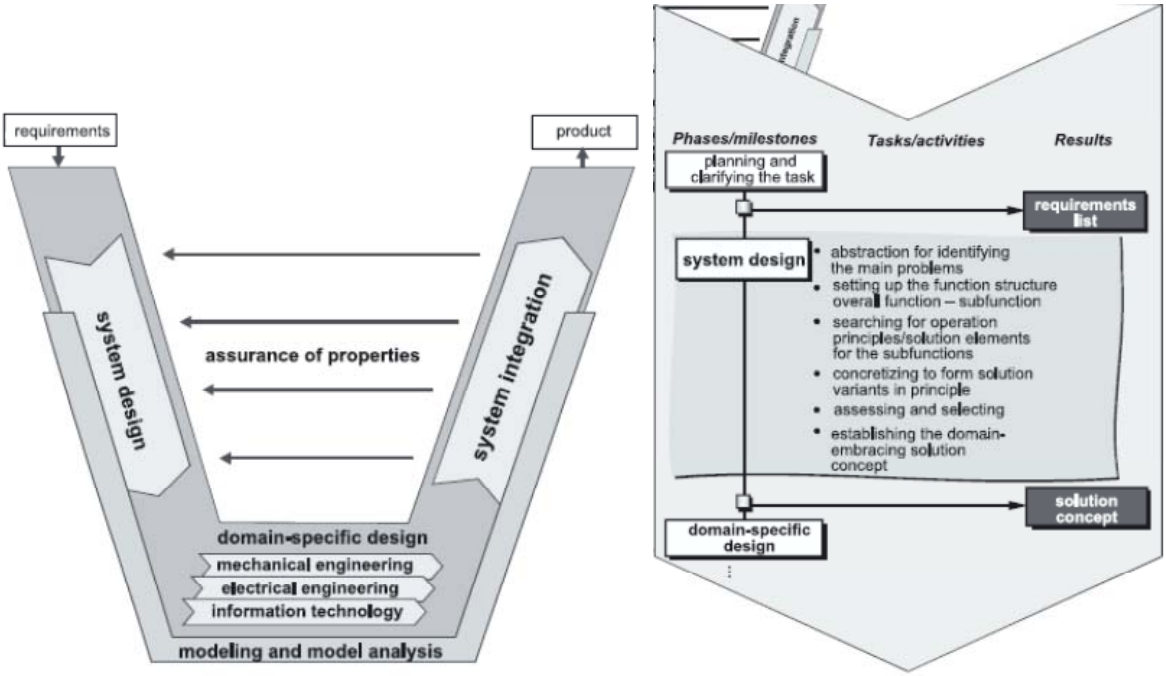

Figure 2: V-Model and tasks of system design for mechatronic design according to VDI 2206 [2006]

Functional modeling plays a central role in mechatronic design since it serves as a common language understandable to all involved domains. [BUUR 1990] Only with an abstract model of the purpose of a mechatronic product, a sensible partitioning into the three sub-domains mechanic, electronic and software is possible. It is part of the system design of the V-model depicted in Figure 2.

The "Specification Technique for Self-Optimizing Systems" is a methodology for complex mechatronic systems. [FRANK 2006, GAUSEMEIER et al. 2009] It provides an adopted procedural model with the stages task clarification, conceptual design of the system, conceptual design of modules and integration of modules. The single stages are organized into sequences of tasks that guide the engineers through the design process. The result of each stage is documented in one or more of the interrelated coherent partial models. They finally form an integrated product model. They are named: requirements, application scenarios, functions, behavior, active structure, shape, system of objectives and environment. In order to keep the integrity, most of the partial models are modeled with the Mechatronic Modeling Language derived from Unified Modeling Language (UML).

Mechatronic products mark a transition from traditional mechanical design towards systems engineering. The core methods to cope with the complexity are the common specification, the decomposition into subsystems and the integration into the whole system. It is related to this thesis through the tasks in the specification phase that are the same as for mechanical design, namely requirements modeling, functional modeling and component modeling. Additional challenges of mechatronic design are communication among experts of multiple domains and

the partitioning of the system into domains.

2.1.3 Systems engineering approaches

In contrast to product design, with roots in machine theory, systems engineering is a rather young discipline that deals with the specification, design and implementation of systems. Systems engineering mostly follows a top-down bottom-up process. In the top down process the system is specified form the whole systems down to details. The designed artifacts are implemented and integrated in a bottom up manner. A simple model for that approach is the V-Model as shown in Figure 4. According to standard IEEE 1220 [2007] systems engineering is defined "as the total technical effort responsible for establishing the product design, as well as the concurrent establishment of the development, test, manufacturing, support, operations, training, distribution, and disposal processes. Thus, the term systems engineering implies that a total system perspective should be applied to the development of a system." This definition is overlapping with integrated product design.

The systems engineering process (SEP) [ISO/IEC26702 2007] is in some points similar to the design process outlined in Figure 1. The differences arise from the different definitions of a system and product. A system is "a set or arrangement of elements [people, products (hardware and software) and processes (facilities, equipment, material, and procedures)] that are related, and whose behavior satisfies operational needs and provides for the lifecycle sustainment of the products."[ISO/IEC26702 2007] A product can be either a system itself, e.g. a car, or can have an is-part-of relation to a system, e.g. a car is part of a postal service. People and processes are considered as elements of a system as well. Taking this viewpoint, the factory that produces products is a system, e.g. a car manufacturing plant, as well as a company that provides a service that includes the use of a product, e.g. the postal service. MBSE approaches integrate all core tasks of the SEP and provide the modeling methods for it.

Therefore, the main activities of systems engineering are first scoping, which is to define what is inside and what is outside the system. Requirements analysis then detects conflicts of requirements and consolidates requirements. Further the partitioning of the system, a task that is referred to as "Decomposition/allocation trade-offs and impacts" in Figure 3. Partitioning involves the design of interfaces as well as the assignment of functions to a domain, e.g. hardware or software [ISO/IEC26702 2007]. An example to clarify the difference of systems engineering and product design is the architecture of the distribution system of the catalytic agent "Ad Blue" that is used for cleaning exhaust gases of diesel engines. Designing the architecture of the distribution system via service centers and gas stations either as liquid from a hose for trucks or in a special container for passenger cars is a systems engineer's job. The design of the interface at the vehicles to allow the filling of the agent into the vehicle is the product designer's job.

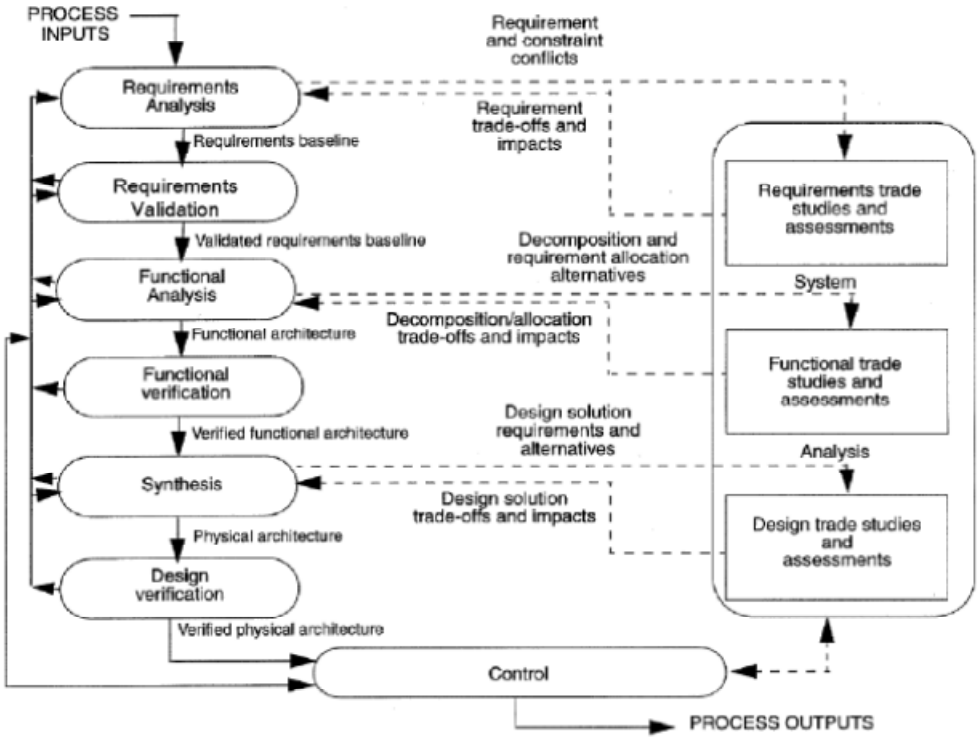

Figure 3: The Systems Engineering Process (SEP) [ISO/IEC 26702]

Figure 3 depicts the SEP. Since systems are more complex than stand-alone products, validation and verification plays a major role. In product design, each following task can be seen as a verification of the preceding task. At systems the number of elements and their interrelation at any level of requirements or functional description is too manifold to allow the following step without verification of the previous step. The assessment of requirements is necessary to detect constraints and conflicts and to eliminate them. For requirements validation, the requirement tradeoffs and their impacts are shown to the client (here: the person who ordered the system) who approves the baseline. It is the basis to for the following steps in the SEP. This does not completely avoid iterations but gives the chance to reduce them due to the involvement of the client. Another challenging point is the involvement of humans as operators or users. This is not visible in the overview of the SEP in Figure 3 but the design and involved constraints of human/system interfaces is explicitly stated in [ISO/IEC26702 2007]. Therefore the SEP integrates steps of verification and validation that are not emphasized in the same way in product design. Nevertheless, common tasks exist with product design processes in terms of requirements analysis, functional analysis, functional verification and concept synthesis. The design process according to VDI 2221 deals with synthesis at a more detailed level which is necessary to provide the required subsystems for systems.

2.1.4 The Model Based Systems Engineering methodology

Model Based Systems Engineering (MBSE) is a subset of systems engineering. A MBSE methodology can be characterized as the collection of related processes, methods and tools used to support the discipline of systems engineering in a “model-based” or “model-driven” context. Estefan [2008] compiled MBSE methodologies and related tools in a survey.

The initial work on MBSE is found in Model Based Systems Engineering: An introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design [WYMORE 1993]. It can be seen as the first mathematical system theory for modeling discrete systems. The main idea of MBSE is to model a system in a consistent way and query the model according to certain viewpoints instead of preparing and distributing huge numbers of documents. With that vision a lot of time that has been used for creating documentation can be saved since it will be created form models on a keystroke [JONES et al. 2010].

Some of the leading MBSE Methodologies according to Estefan [2008] are presented below. Common to all of them is the use of modeling languages that is shared with this thesis.

IBM Telelogic Harmony-SE is a MBSE methodology for developing highly integrated systems and software systems. The Harmony Process applies the classical “V” lifecycle model, see Figure 4. It is different from the V-Model depicted in Figure 2 that is adapted to mechatronic design. The collaboration of the system architects is supported by a central repository for requirements and models. It uses a “service request-driven” modeling approach supported with Object Management Group Systems Modeling Language (OMG SysML). [HOFFMAN 2006] In there the system structure is described using block diagrams. Communications between blocks is realized with messages (service requests). The three main tasks and work products are “requirements analysis”, “systems functional analysis” and “architectural design”. Harmony-SE focuses on software engineering. Functional analysis needs a description of existing service requests which is not common for mechanical design.

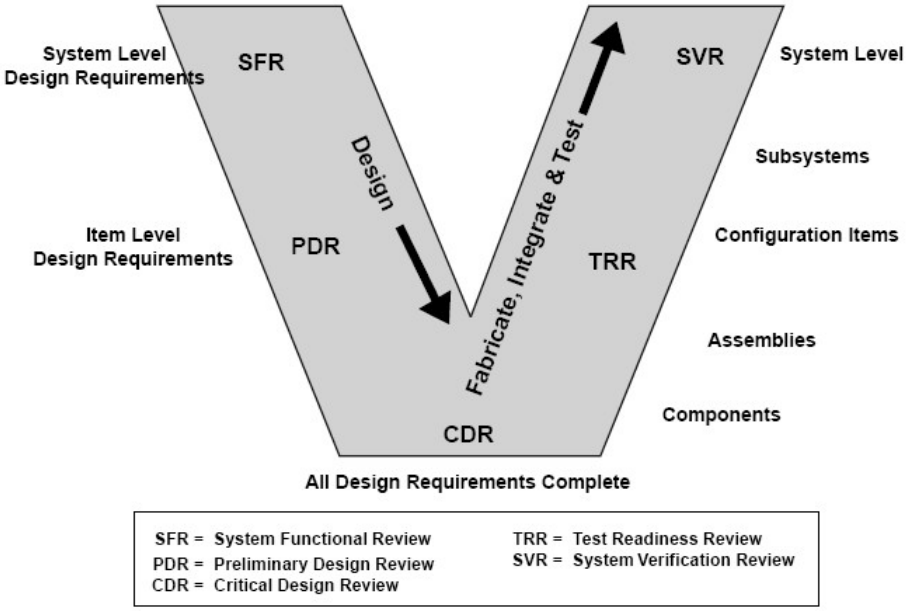

Figure 4: Classical V-Model extracted from [DOD 2001],

The **Object Oriented Systems Engineering Method (OOSEM)** is an approach that initially was developed in a joint effort of Lockheed Martin and the Systems and Software Consortium and refined by the INCOSE OOSEM Working Group [FRIEDENTHAL et al. 2008]. It is a top down, scenario driven process that supports the specification, analysis, design and verification of the system. It makes use of object oriented concepts and uses SysML. It is a very elaborate and detailed process that is intended to integrate with software development, hardware development and test. The development activities of OOSEM are "stakeholder needs analysis", "definition of system requirements", "definition of logical architecture", "synthesis of candidate allocated architectures", "optimization and evaluation of alternatives" and "validation and verification of the system". The full description of OOSEM process illustrated with an example can be found in "A Practical Guide to SysML" [FRIEDENTHAL et al. 2008]. Functional analysis is not part of OOSEM although shown in the same book. It can be integrated as a kind of stakeholder needs analysis that in turn influences the logical architecture. The logical architecture is a solution neutral structure that can be realized by a structure of solution elements. Nevertheless the methodology provides a clear set of detailed tasks that have to be done in the single activities. Common with the thesis is the analysis of system requirements, the use of a solution neutral structure called logical architecture and the search for realizing elements.

The **IBM Rational Unified Process for Systems Engineering (RUP SE)** "is both a process framework and a process product from IBM Rational and has been used extensively in government and industry to manage software development projects" [ESTEFAN 2008]. The

process is divided into the four phases of “Inception”, “Elaboration”, “Construction” and “Transition”. Involved engineering activities are business modeling, requirements modeling, analysis and design, implementation, test and deployment [BALMELLI et al. 2006, KRUCHTEN 2003]. UML and SysML diagrams are used for modeling the system architecture. Concurrent engineering and iterative design and highly reconfigurable discipline workflow templates facilitate the collaboration in systems engineering projects. To manage the multitude views and concerns on a system that are addressed with RUP SE, an architecture framework organizes the models involved. Tool support is provided by the RUP SE modeling center, a plug-in for the eclipse tool suite. The common point with the thesis is the modeling language and the use for various engineering disciplines including mechanical design.

The **Vitech Model-Based Systems Engineering Methodology** uses a System Definition Language (SDL) and a System Design Repository. There is a strong tie to the CORE product suite provided by Vitech Corporation. [CHILDERS & LONG 2004] The primary systems engineering activities are source and requirements analysis, function and behavior analysis, architecture synthesis and design validation and verification. The major difference to the other methodologies is the process to engineer a system horizontally in complete converging layers before vertically. It is called the “Onion Model”. A link to the thesis is given due to the use of requirements modeling, functional modeling and synthesis. The difference is given by the use of SDL, however still it is a language that is formally controlled.

MBSE is linked to this work in two ways. All of the leading methodologies include engineering activities that (partially) cover the main modeling tasks of the thesis. All of them use a modeling language mostly OMG SysML/UML. For further discussion of MBSE methodologies the interested reader may read the survey of Estefan [2008].

Both systems engineering and product design use modeling to better understand the task, analyze the problem and synthesize the system or product. The MBSE approach seeks to apply models for any essential task of systems engineering. Tool support is provided for all MBSE activities. Consistent modeling requires the integration of different models. Since modeling is important to both product design and system engineering the following section present the research efforts in integrated product modeling.

2.2 Product modeling

“Product modeling is the heart of product development activities that determine the engineering productivity and industrial competitiveness.” [KRAUSE et al. 1993]

A model is defined as a simplified image of an object in order to analyze a certain aspect of the original object. [STACHOWIAK 1973] Further, according to Strahinger [1998], a meta-model is the model of a model where the super-ordinate model describes the modeling elements (i.e. modeling language) that are used for modeling the sub-ordinate model. In engineering different types of models are known. A differentiation for the formality of description methods is given in [VDI /VDE 3681 2005], where the three levels “formal”, “semiformal” and “informal description methods” are differentiated. Models can be built with description methods, thus the terms are used synonymously. Formal models have a

mathematical basis, a completely defined syntax and clear semantic interpretation. Semiformal ones miss a mathematical basis but have the other two attributes and informal models do not need to have a mathematical basis neither a clear semantic interpretation nor a complete syntax. Models can be virtual (mental models, software, algorithms) or material (scale models and mock ups). Common to all models is that they are created for a certain purpose (model pragmatism). In the case of functional modeling it is to analyze and understand what flows of material, energy or signal and different functions are needed to specify the product or system in a qualitative way. Since models are abstractions of reality, a model has to be validated against reality at some time. Validation of virtual models and simulation can be carried out by comparison of results with hardware tests. Throughout the design process, models become more and more detailed. At the transition to hardware prototypes and production, the semantic meaning of a model changes from abstraction to blueprint or stereotype.

There are three views on product information, namely "product definition", "product presentation" and "product representation" [ANDERL & TRIPPNER 2000]. Product information for product definition is administrative and organizational information for an unambiguous identification of a product, e.g. classification number, the identification of release states and versions and the product structure. Information for product representation is used for the computational modeling of product attributes e.g. the shape is modeled by geometric models like wireframe, surface, B-rep, solid or other models. The way product representation information is presented to the designer is called product presentation; e. g. a geometric representation is presented as a shaded image or a drawing. The term model representation is used to refer to computer internal information and model presentation when diagrams are shown.

2.2.1 Product models and integrated product models

The term product model is more specific. Krause et al. [1993] define: "Products are materialized, artificially generated objects or groups of objects forming a functional unit. The materialization may contain mechanical **parts**, electrical and electronic **components**, hydraulic or pneumatic **components** or other **elements**. If the product has a computer with a CPU and a memory, the computer hardware and software contributing to the purpose of the **product** are part of it. Products are made of different materials and manufactured by different processes in a great variety of sizes." Using the above definitions of product and model, the term product model can be interpreted as the "logical accumulation of all relevant information concerning a given product during the product life cycle". [KRAUSE et al. 1993] The phases of the product lifecycle are shown in Fig.1 on the left hand side.

With this definition it is seen that, depending on the complexity of the product, a large number of different models have to be stored, related, and accessed during the whole lifecycle of the product. A clarification of the terms "**components**", "**parts**" and "**products**" is necessary. These terms express the degree of complexity in a system context. Table 2-1 shows the hierarchy of the system elements adopted from Hubka and Eder [1988]. It is based on the inner complexity of the elements. Applying the definition of complexity as the amount and diversity of elements and their relations, a part, consisting of one piece, has zero complexity.

The complexity grows with components that are assembled from some parts. The original hierarchy had four levels since it was product centered. It has been extended by the fifth level, system of systems, which is the most complex level. Within this hierarchy, it is not possible to distinguish clearly between the levels [BUUR 1990], however it gives an orientation of the grade of complexity of modeling artifacts and their possible decompositions. The approach of this thesis mostly applies to the levels zero to three.

Table 2-1: Levels of complexity adopted from [HUBKA & EDER 1988]

Level	Name	Description	Example
0	part	single piece, made without assembly operations	flange, shaft
1	component	simple subsystem assembled from parts and other components	electronic components, sensors, actuators, bearings
2	module, sub-assembly	subsystem assembled from components and parts, embodies an independent function of a product	print head, linear modules, circuit board
3	product	assemby of modules, components and parts that performs a closed function	printer, notebook PC
4	(product) system	interacting products that perform a number of functions	office systems including PC, network and printers
5	system of systems	Interacting systems, products and organizations that provide a service or product	production systems, e.g. a car production plant

Further Krause defines: “Product modeling, in its complete sense, […] consists of two interrelated aspects: product models and process chains.” [KRAUSE et al. 1993] The latter are commonly referred to as product development workflows or product modeling processes. Throughout the design process, the single tasks change and thus the product models also change. The modeling sequence in the focus of this thesis is the sequence of requirements modeling, functional modeling and developing the initial component structure. These partial models are at the beginning of the product development process. A term that expresses a similar fact is model evolvement [MAIER & RECHTIN 2009]. Model evolvement is the change of models throughout the development process. It is fueled by refinement, verification of the client’s needs or validation of the models.

The first attempts to create a framework for an integrated product model date back to 1984 when the development of the Standard for the Exchange of Product Model Data (STEP) as successor of previous attempts in product model standardization started. [ANDERL & TRIPPNER 2000, KEMMERER 1999] The initial release of the standard in 1994 was focused on the process chain of 3D geometric modeling, the simulation of physical properties and the transition to production. [ISO/10303 1994] Within the standard, product model data is given as a common abstract description of product attributes (geometry, material, ID, etc.) and their relations. Due to its importance for various applications and stages throughout the product lifecycle it is called an integrated product model [GRABOWSKI et al. 1993]. Still integrated product models in practice are limited to the design stages from embodiment design to production preparation. There is a strong bias towards geometric modeling (CAD), validation (CAE), and subsequent production (CAM) and inspection (CAT). However the initial idea of an integrated product model goes further. Conceptual design was considered with additional

models as well as requirement modeling and functional modeling. [GRABOWSKI & ANDERL 1991] Figure 5 gives an overview of the partial models that cover certain engineering tasks.

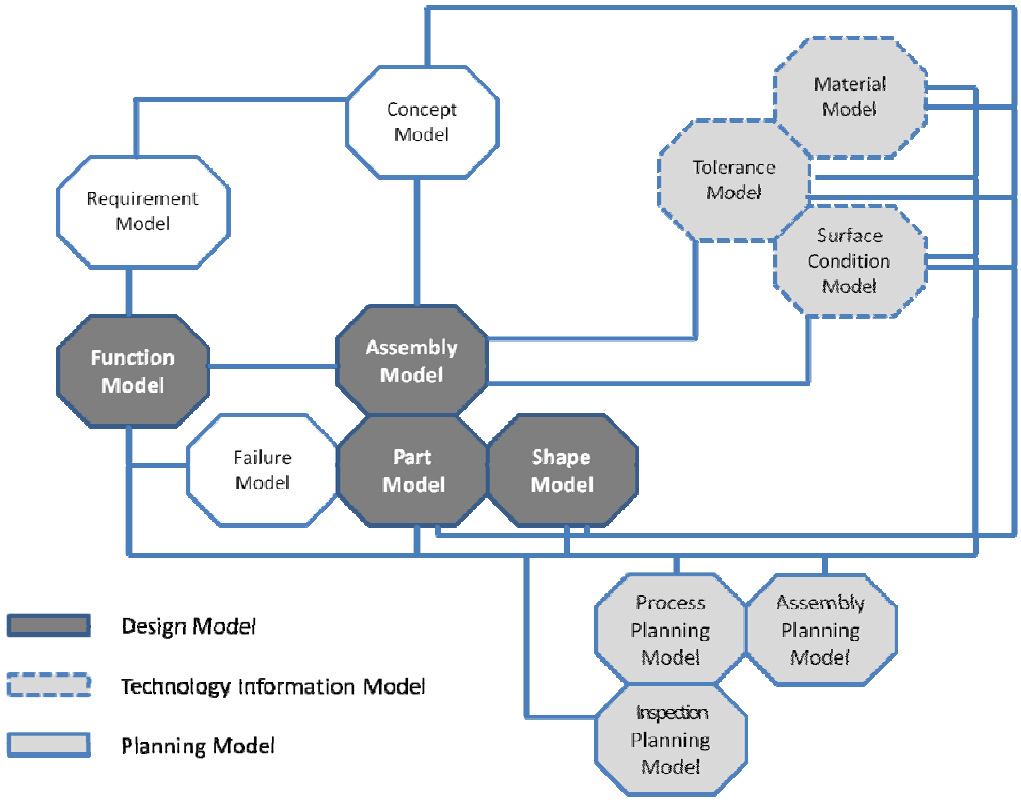

Figure 5: Integrated Product Model adopted form [GRABOWSKI & ANDERL 1991]

The plethora of different product information models has to be managed. This includes the management of access rights, conversion of data and presentation of data. This functionality is provided in Product Data Management (PDM) systems.

Faced with the challenge to integrate product information of different engineering research projects, the Core Product Model (CPM) is developed. [FENVES et al. 2004] It is a UML based meta-model for product information throughout the product life cycle. The CPM presents a taxonomy of classes that allows building product models with core entities (artifact and feature) and core properties (function, behavior, material, geometry form and transfer function). Transfer function refers to a function as an input output relation. The different demands in different product life cycle stages are respected. Fenves [2004] shares a similar viewpoint on an artifact with Friedenthal [2008] describing the meaning of a SysML block. A SysML block is a modeling element to describe the structural element of a system. Both can represent a product, be a part of a product, or be a unique entity in an environment as an engine with a certain code within a vehicle with the VIN 9089875. The core product model is a framework for an integrated product model.

Together with its extensions the CPM serves as a framework for Product Lifecycle

Management (PLM). It is generally defined as "a strategic business approach for the effective management and use of corporate intellectual capital" [SUDARSAN et al. 2005]. The PLM concept holds the promise of seamlessly integrating all the information produced throughout all the phases of a product's lifecycle. In other words, PLM is the framework to organize an integrated product model. The extensions to the CPM are the Open Assembly Model (OAM), the Design Analysis Integration Model (DAIM) and the Product Family Evolution Model (PFEM). In some aspects there is a resemblance to the integrated product model of Grabowski (see Fig. 5). The framework is a concise proposal to guide the development of future PLM systems and is directed towards model driven approaches.

In the last years, efforts have been made to integrate systems engineering models with the standard for the exchange of product model data (STEP). The Application Protocol 233 "Systems Engineering" has been released as draft international standard since 2009 [ISO/DIS 10303-233 2009]. Within this draft relevant models of systems engineering and their interrelation are described. Some of the model driven approaches are covered with this draft.

In a paper about product information exchange, Fenves et al. [2005] give an overview of research in product modeling of the last quarter of a century. They identify three categories of information content for product information exchange named "geometry information", "design information" and "lifecycle information" **Geometry information** and its handling with PDM systems has been standardized in 1993 with the standard for the exchange product information (STEP) [ISO/10303 1994] and is state of the art today. **Design information** includes all information necessary for design as requirements, constraints, design rationale and design rules etc. **Lifecycle information** includes all information that is created during the lifecycle from initial concepts to product disposal and its management. Most of current research is devoted to design information. The standards are still incomplete. To integrate the whole range of lifecycle information into a PLM framework is a future challenge.

Next to the STEP related work on integrated product models there are two examples of integrated product models beyond geometry. The Methodology and tools Oriented to Knowledge based engineering Applications (MOKA) [SAINTER et al. 2000] exploits STEP, Knowledge Interchange Format (KIF) [GENESERETH & FIKES 1992] and other knowledge engineering efforts to create an integrated product model. MOKA holds the views of structure, function, behavior, technology and representation as related modeling domains. To support formal modeling, the MOKA Modeling Language (MML) an extension to the Unified Modeling Language (UML) is deployed [BRIMBLE & SELLINI 2000].

Another attempt to a feature based integrated product model is reported in [BRUNETTI & GOLOB 2000]. It is the attempt to cover all activities for conceptual design following the approach of Pahl & Beitz [2007] and integrating methodical support for all tasks of conceptual design as specified in guideline VDI 2221 [1993]. It is an example for a monolithic integrated product model that integrates all tasks of conceptual design via features. The introduction of such tools in industry is faced with existing legacy tools that can be incompatible.

2.2.2 Systems architecture descriptions

Like an integrated product model attempts to integrate the product information throughout the product lifecycle, a system architecture model collects the outcomes of the systems engineering process. Since the focus of systems engineering is different, it is necessary to point out the relation between a product model and a systems architecture description. In this context, a system architecture is "*the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution*." [IEEE1471 2000] Ann architectural description is defined as the collection of all products to document system architecture. In this case, products refer to documents, models (both virtual and hardware) and other recordings, e.g. photographs, audio tape, video etc.

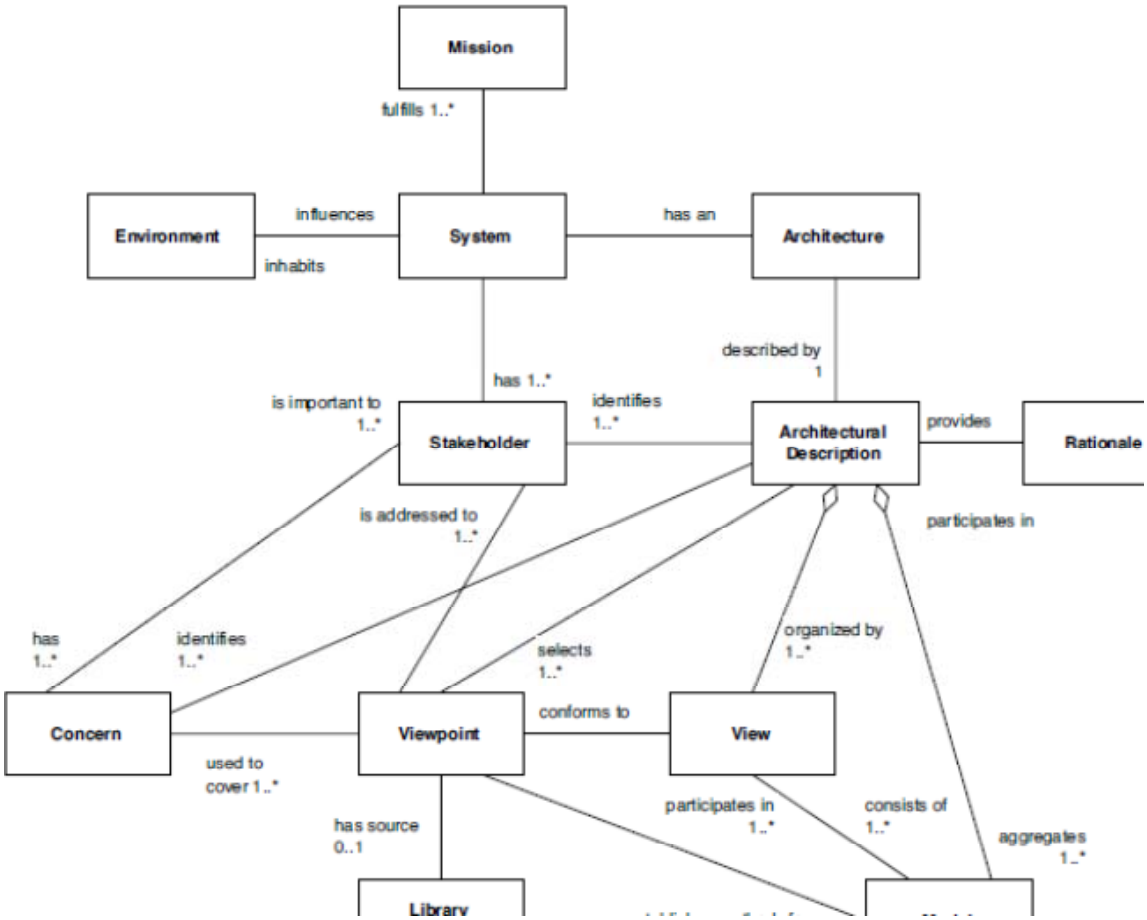

Figure 6: Meta-model of an architectural description [IEEE1471 2000]

Figure 6 shows the meta-model of an architectural description. In this case, meta-model "is simply a defined set of concepts and terminology along with their relationships to one another." [HOLT & PERRY 2008] An integrated product model can be seen as an architectural description. Partial models of an integrated product model are integrated. Where the

integrated product model takes into account customer relation management as lifecycle information, the system architecture refers to stakeholders. Stakeholders can be any person or legal body that has interest in the system, e.g. client, costumer, operator, manufacturer, government, etc. The viewpoints are used to cover the concerns of the stakeholder. Conforming views are used to query the model. This requires a certain flexibility of how the model is composed. The models have to give answers to the questions formulated in the views. Maier and Rechtin [2009] identify "purpose/objective", "data", "behavior", "managerial", "form" and "performance" as common views on systems. One view can contain several models. The overall task is described in the mission that has to be fulfilled by the system.

Architecture frameworks provide a collection of viewpoints and models. They follow a certain purpose. The Enterprise Framework also known as the Zachmann Framework is an architecture framework for businesses. [EACOE 2011] It includes the organizational structure to govern the enterprise as well as models for the business or product of the organization. The Framework provides meta-models, description of the modeling purpose (views) and integrates the single models into a concise system model

The Department of Defense (US) and the Ministry of Defense (GB) both deployed frameworks for the assessment and procurement of Systems of Systems. [DoD/CIO 2012, MoD 2010] Since the focus of these frameworks is to gain an overview and standardize the views on the systems, details like functional modeling are not considered. The main purpose is to have comparable information for the systems to acquire from competing suppliers.

2.2.3 Summary of product modeling and architectural descriptions

Product modeling and architectural descriptions (AD) are not independent from one another. If a product is part of a system, the system AD can give input like requirements and functional description to the product model. The viewpoints of requirements and functional modeling can be found in both system models and product models. Especially the MBSE approaches show that functional modeling is a core activity of systems engineering as well as product design. Product models should assist communication especially among different domains, among systems engineers and the involved domains of product design.

2.3 Functions in product design

The term function exists in many fields of daily use and science. Here the focus is on the field of engineering. In this field, different semantic meanings of the term function exist. For instance, in software engineering, function refers to a subroutine and in control engineering to a mathematical function. Even in the domain of engineering design there are several approaches to function. **Function Representation** (FR) is a research branch related to artificial intelligence that provides the ontology of function related terms targeted at functional reasoning. [Chandrasekaran 2005] **Function(al) Modeling** (FM) is a research stream that deals with the modeling of system functions using function primitives and applying modeling formalisms. Especially FR related research discovered the ambiguity of

the terms function and behavior [CHANDRASEKARAN 2005, CHANDRASEKARAN & JOSEPHSON 2000]. Ullman [2002] presents an effective differentiation in: Function is the abstraction of the artifact under design to deliver an intended performance at a known input. Behavior is the description of the physical properties of an artifact that deliver the actual performance at a known input. Performance in this context is the output within a certain period of time. Function is often referred to as intended behavior. In functional modeling, function is used to capture intention or purpose of an artifact under design. [PAHL et al. 2007] In general functions can be formulated by any person e.g. the customer. The designer's task is it to verify and translate that informally articulated functions into a transformation (or transfer) function and find embodying artifacts. An early distinction of function types is made between purpose function and transformation function. A purpose function is "the ability of a machine to create an expedient effect" [ANDREASEN 1991, BUUR 1990]. In contrast to that "the transformation function (or process) […] is the action which changes an object from an input to a desired output state." [BUUR 1990] The distinction of different types of functions in FR goes beyond a simple yet ambiguous differentiation since for computational manipulation the terms have to be clear and distinct. The ontology for reasoning relates the terms of human need, environment, mode of deployment, function as effect device function, behavior, etc. [CHANDRASEKARAN & JOSEPHSON 2000] "In the device centric view the focus is (…) on its internal configuration namely on its structure." [CHANDRASEKARAN & JOSEPHSON 2000] Transformation functions are used for building a structure thus they can be seen as functions from device centric view. From the view of FR, functional modeling provides to different sets of function primitives. [CHANDRASEKARAN 2005]

A review of functional modeling is given in [ERDEN et al. 2008]. It gives a good overview of modeling but does not cover all function modeling efforts.

2.3.1 Approaches to functional modeling

There are several independent approaches to functional modeling. In German engineering design, functional modeling dates back to 1970 when Rodenacker made use of functional modeling. He introduced the generally applicable functions: "branch", "join" and "guide" and the flows "material", "energy" and "signal". These function primitives are used for modeling sequences or networks of functions to decompose the overall function of a machine. In the following two decades, several independent functional modeling approaches were developed as a part of a design process. ROTH [2000]proposes a set of eight basic functions together with a graphical presentation. His work covers functions for transforming material, energy and signal as processed objects. The concept of function is an input-output relation. He argues that material and signal transformation always needs energy. Further he introduces rules for the creation and manipulation of function structures. The general idea is that the modeler first models the sequence of the main flow, then all other flows that are enablers for the main flow, e.g. the main function is "guide material", then the main flow is material with an enabling "energy" flow.

At the same time Koller [1975] proposes a set of 24 elementary functions grouped by the processed flows. He uses a different graphical presentation from ROTH. The main objective for their function structures is the selection of principle solutions out of their design catalogs.

They use the input output relations of functions to search for stored solutions. Koller explicitly makes use of auxiliary flows in his functions. This opens the way to more complex structures and allows the representation of control structures in functional modeling. Pahl uses a flow block presentation [PAHL et al. 2007] together with a set of five generally applicable functions. All three provide concise descriptions of functions and propose functional modeling as a paper-based method.

Hundal [1990] implemented a software tool to support the tasks of conceptual design. His notion of function is based on the authors mentioned above. He groups 39 physical functions that mostly come with explicit input and output information into 6 basic functions. He uses channel, change magnitude, connect, branch, store/supply and convert as basic functions. They are identical to ROTH's "Allgemeine Funktionen" [ROTH 2000][1]. In a following process he uses the input output relation of physical functions to select physical effects from the database. The flows are detailed to specific forms of material, energy and signal and added by geometrical properties like displacement, velocity, acceleration, etc. Hundal's work is close to the concept in this thesis. He uses databases for functions and solutions but does not use a formal modeling language.

A different set of functional primitives is provided by Modarres and Cheon [1999]. In their view, important things of engineering are **commodities** where **values** can be assigned for e.g. energy, mass, momentum, etc. Physical systems process commodities from one form to another according to conservation principles. Functions describe the contribution of a part to the conversion of commodities in a system. The conservation laws are applied inside one control volume, e.g. conservation of mass. A storage device can act as a sink or source for mass. The function primitives are different from the flow centered view on function namely: "generate", "destroy", "maintain", "control", "transform" and "transport".

A different concept of functional modeling is used in the theory of inventive problem solving, also known as TRIZ. [ALTSHULLER 1994, TERNINKO et al. 1998] The main purpose of the artifact is expressed by at least one primary useful function (PUF) that is in relation with other useful functions that are needed to create the PUF. Through side effects, harmful functions (HF) can occur. A harmful function disturbs the system and in the worst case the system cannot fulfill its purpose. The search for functions of a system is stopped when at least one PUF and one primary harmful function (PHF) is identified. Both types of functions are related by three kinds of relations: function A causes, is used to eliminate or is required for function B. The function model is used to identify contradictions in a relational network. With the identification of contradictions, a matrix indicates possible solutions that solve the contradiction and lead to innovative solutions. TRIZ provides no function primitives but encourages formulating the problem in one's own simple words. Further the definition of function as intended behavior is not valid for harmful functions, since no engineer would consciously intend to design a harmful (contradicting the system purpose) function. TRIZ has great strength with design analogies. In an excessive survey Altshuller [1994] examined more than 40,000 patents and extracted their principles of solution. This results in a domain spanning catalog of neutrally formulated principles and physical effects that can be generally applied.

[1] Hundal partitioned the basic function slightly different from Roth, this leads to 6 basic functions instead of 8.

In function-behavior-state (FBS) modeling, Umeda and Tomiyama [1995] define function as a "description of behavior recognized by human through abstraction in order to utilize it". The relation of function to behavior is many to many. Behavior in this context is defined by sequential one or more changes of state over time. A state is defined by a triplet of entities, attributes and relations among them. State changes in turn are invoked by physical phenomena. FBS modeling is also a guiding design process that relates three core modeling tasks of conceptual design. The Knowledge Intensive Engineering Framework (KIEF) is ongoing research since 1995 that provides formalized knowledge to the FBS design process [YOSHIOKA et al. 2004]. The starting point is a hierarchical functional decomposition. In a next step, the functions are mapped to physical features. KIEF enriches the information by causal dependency reasoning. It is based on a basic ontology and a meta-model.

Recently Komoto and Tomiyama presented a paper on the extension of KIEF towards system architecting[2] [KOMOTO & TOMIYAMA 2010]. They limit their system to complex mechatronic products and point out that system architecting tools have different views (functional, behavioral, structural). The new contribution compared to KIEF is the integration of interval based logic to introduce a qualitative functional ordering and a spatial modeler that automatically can create and position oblongs with faces normal to the main coordinates X, Y and Z. With that kind of spatial model, the product or system is packaged. For the evaluation of consistency of temporal and spatial intervals some rules are implemented. However, the requirements management and the link to an overall task are missing. The tool meets the requirements for conceptual design and presents an elaborated support for it. The differences of terms of the KIEF ontology to the systems engineering semantic dictionary according to ISO/DIS 10303-233 [2009] make a comparison with other model based approaches difficult.

Gero [1990]describes a design process called Function-Behavior-Structure (FBStr). It is different from the FBS of Umeda and Tomiyama [1995]. He states that a direct transformation from "function" (F) to "design description" (D) is not possible. As a consequence he infers that the basis for D is "structure" (S) and "structure" can be analyzed for its "behavior" (B_S). From function the "expected behavior" (B_E) can be derived. By comparison of B_S with B_E, design synthesis can be done by the following sequence: $F \rightarrow B_E \leftarrow \rightarrow B_S \rightarrow S \rightarrow D$. This means that B_E is derived from F and then can be compared to B_S that is analyzed from S that can be transformed to D. The term function is not described into detail and is just related to requirements and set in relation to expected behavior. The schema is implemented as a software prototype. FBStr is related to this thesis due to the similarity of the design process that transforms a function to a design description. The difference is through the use of a formal modeling language and the use of model libraries to link structure directly to functions.

The NIST functional Basis (NIST-FB) is a reconciliation of two independent research efforts [HIRTZ et al. 2002]: The NIST effort to develop generic taxonomies of engineering functions and flows [SZYKMAN et al. 1999] and the functional basis effort from Stone and Wood [STONE & WOOD 2000]. Szykman's motivation is to improve the comparability of engineering functions in the larger context of artifact design. He proposes a schema for

[2] It is called system architecting in contrast Maier & Rechtin's "Systems Architecting". They are different by definition.

computational representation and suggests implementing the schema with the extensible markup Language (XML) or another computational language. Further he compiles taxonomies of functions (130 items) and flows (100 items) from various previous research efforts in engineering functions from different authors. A taxonomy is a set of similar terms that form a hierarchical relation. This is in contrast to a lexicon that contains all words used in a certain context. The functional basis is a lexicon for functional modeling that contains two taxonomies. He admits that it is difficult to cover all concepts of functional modeling with a small taxonomy of functions. Stone and Wood [2000] underline the usefulness of a functional basis for product architecture design, semantic function structure generation, archival and communication of design information, comparison of product functionality and further tasks. They created the functional basis from the same set of sources as Szykman yet this resulted in different taxonomies. Additionally they provide an explanation of terms like product function (overall function), sub- function, functional basis etc. For function decomposition they propose black-box to white-box decomposition. In this kind of decomposition a node of a diagram is further detailed with the same kind of diagram that shows the inner elements (nodes and relations) of the parent node. The reconciliation led to a far smaller functional basis because redundant functions were eliminated and the terms were listed as synonyms. Both efforts show a three level hierarchy for their taxonomies of functions and flows, which was taken over to the NIST functional basis. The three-level hierarchy is denoted from top down as “class”, “secondary” and “tertiary”. Principally the two taxonomies are independent. This basically allows an unrestricted combination of functions and flows. With the NIST functional basis generality is claimed, thus it can be used for various concepts of functional modeling. For this reason the functional basis is used as the blueprint for the model library of functions and flows.

Nagel et al. [2009] has presented a software prototype tool for functional modeling, called “functionCAD”. They use the taxonomies of the NIST-FB for the representation of flows and functions. The software supports modeling processes (function structures) inside a control volume and connecting different processes via flow connections. The black-box white-box decomposition principle can be applied. His work is related to modeling function structures with library support. The major difference is that Nagel allows unrestricted connection of single functions with respect to number and types of flows. Only the direction of the flow is specified. To control functional modeling he used the grammar rules defined with the function design framework (FDF) [NAGEL et al. 2008]. In the modeling examples, a formal modeling language and functional modeling as a design task are used among others. Thus function structures can refer to requirement or component models. Further the number of flows interacting with a function is restricted in number and type.

The content of topological information in function models is investigated by SEN et al. [2009] He compares the unconditional probability of the connection of two functions with a flow with the informed connection of two flows. Informed connection refers to additional information that is needed to establish a valid connection, e.g. direction and type of the flow. The main result is that topological information makes functional modeling more expressive and the rules that contain that information help the designer to connect functions properly. It is related to this work since SysML syntax requires maintaining the flow and direction information that Sen et al. refer to as “informed connection”.

In a user study, Thomas et al. [2009] studied the impact of abstraction levels on the interpretability of a function structure. He uses the functional basis to model different artifacts together with the design repository (http://designengineeringlab.org/delabsite/repository.html). In his study, student groups identify the product from function structures at three different levels of abstraction. There is a clear correlation between the use of context specific terms and the number of products that can be identified. For synthesis, more abstract terms are beneficial, since they allow exploring a wider solution space. The level of abstractness is linked to the number of functions as well as the terms used for functional modeling. Functional modeling at the secondary level of the NIST-FB, as chosen in this thesis, is a viable tradeoff between interpretability and solution neutrality. This information is relevant for the right selection of abstraction in the modeling example.

2.3.2 Comparison of different function lexicons

The contents of different function lexicons are compared to determine the best choice for the function library in this work. The term lexicon is chosen here to include all sets of functions, even those that come without a hierarchical relation and cannot be called taxonomy. In the previous section, several functional modeling efforts are presented. Some of the authors created dictionaries with concise descriptions and examples. In a first comparison, the number of theoretically available function primitives is considered. In Figure 7, a comparison by number is shown including the work by Rodenacker, ROTH, Koller and NIST-FB. The need for a common dictionary for functions is generally given for several reasons. A dictionary lists relevant terms in an ordered way and explains the terms. First is the subjective nature of functions. Dictionaries help to disambiguate and explain the functions with examples. Another reason is the coverage of different engineering disciplines working together. The conception of functions may be different for similar terms. Establishing a common basis helps to avoid conflicting terms and to discover gaps.

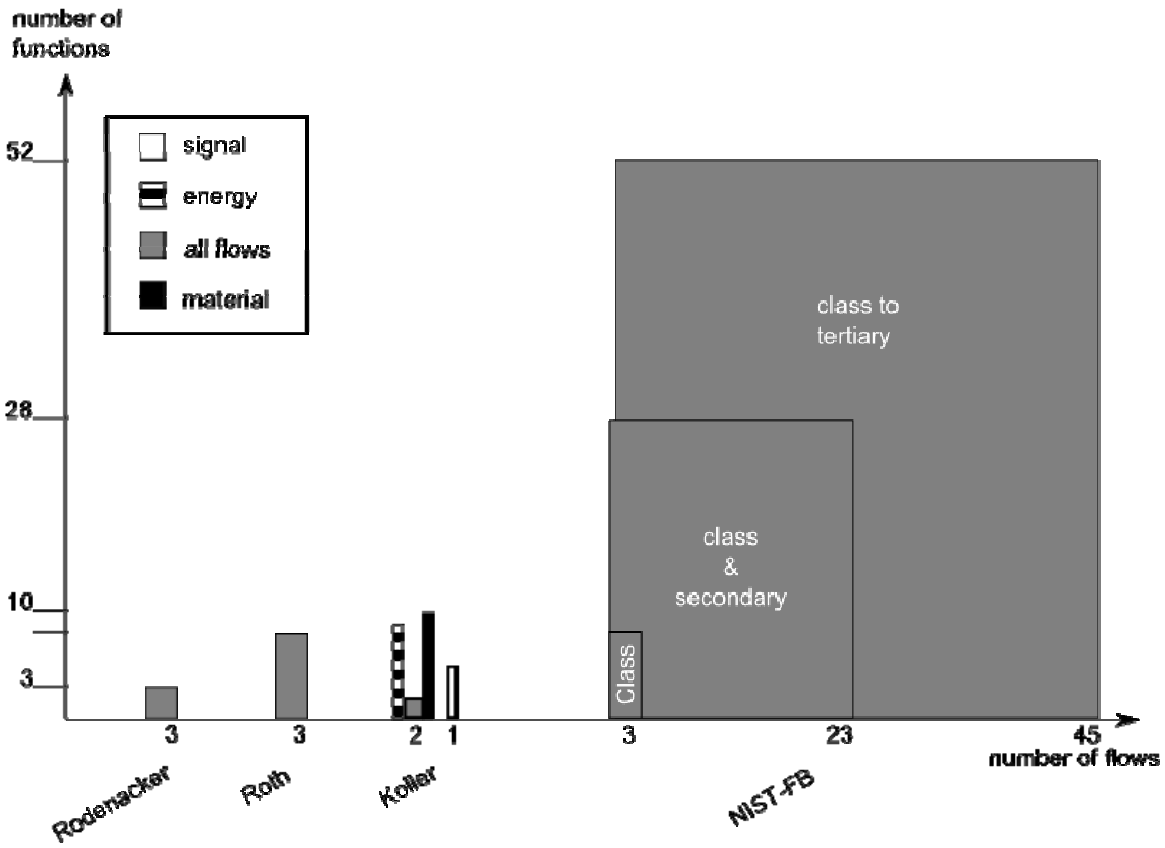

Figure 7: Number of function primitives of different authors. The areas represent the overall number of theoretical function-flow combinations

At the class level of the NIST-FB, the number of function primitives of ROTH are equal (8 functions by 3 flows), the contents of function terms are different, but the flows "material", "energy" and "signal" are the same. Rodenacker started with three functions, and Koller reached up to 10 functions. NIST-FB reaches a great number of function primitives due to the 3-level hierarchy of functions and flows of the functional basis. The levels of the hierarchies are named "class", "secondary" and "tertiary" from top to bottom.

The lexicon of Koller [1998] is different. He proposes four sets of elementary functions for pure material, energy and information and one for the connection of material and energy. The four sets of functions are separate lexicons, therefore presented as separate bars in Figure 7. The involved terms are disjoint and defined by what can be done with the flow, e.g. with "decrease" the granularity of ice cubes can be changed by crushing them. All functions and flows are at the same level. The main purpose of this function taxonomy is to provide access to a design catalog for principle solutions. The presentation of both flows and functions is a symbolic one. ROTH [2000] proposes a similar set of functions for the same purpose.

For a function lexicon that includes the definition of flow types that are involved in a function, every function has to be defined separately. The NIST-FB allows combining flows and functions almost unrestrictedly. Therefore the theoretical number of function primitives

that can be created from the combination of the flow and function taxonomy of NIST-FB is more than 2000. The number of necessary elements in a model library is 45 flows and 52 functions. The combination of flows and function does not work with the definition of functions by Koller, since the definition and examples are different with respect to the flows.

The NIST-FB is a promising candidate for implementation as a model library. With a large body of functions, all terms regarding mechatronic design should be covered. In recent research, the most used function taxonomy that complies with the term function as used in this thesis is the NIST functional basis. However, this does not guarantee that this function lexicon is used in industry as well. In industry, the term is used specifically in different branches; even different companies may have different concepts and different glossaries of functions.

In this thesis the focus is on model libraries to support functional modeling and component modeling. The three-level hierarchy of the NIST FB allows modeling on different levels of abstraction. The coverage of NIST-FB is empirically validated for mechatronic products. This is done using examples from the design repository at Oregon State University (http://designengineeringlab.org/). Therefore, the NIST-FB is selected to implement a model library for flows and functions.

2.3.3 Functional modeling in Systems Engineering

Engineering design as well as systems engineering employs functional modeling. However, there are fundamental differences in the use of terms. Application Protocol 233 “Systems Engineering” of the standard for exchange of product data [ISO 10303-233 2009] is the standard in draft describing the terms needed for product data representation. In informative annex H2: Technical Discussions-Concept Model for Systems Engineering function is described as: ”the entity in the context of modeling that transforms an input set of elements into a set of output elements that may be the same or measurably different from the input set”. [ISO/DIS 10303-233 2009] In the semantic dictionary, an element is described as “anything on which repeated measurements can be made for the engineering purposes of interest.” Behavior consists of function, input-output (I/O) and function ordering (see Fig. 8). It is defined as “what an element is to do or does in response to excitation it receives from external elements in its environment.” [ISO/DIS 10303-233 2009] Inputs and outputs are the elements that enter and leave a function. A function can be a member of the system or the environment. Function ordering is the description when a function is activated or terminated. Consequently, behavior is a sequence of functions with specific inputs and outputs. This is expressed in Fig. 8 where function, I/O and function ordering have a composition relationship to behavior. Systems engineering thus favors a rigorous input-output relationship of function that meets the definition of a transfer function.

Following the draft of AP233 [2009], behavior is different from Ullmann’s [2002] definition as the response of an artifact to an input created due to the physical properties of the artifact. It is also different to the meaning of the term behavior in engineering design. The meaning of behavior in engineering design is taken over by physical property for systems engineering. “*A physical property is what an element exhibits […] in response to excitation and stimulation*

from auxiliary measurement entities that are not part of its context." [ISO/DIS 10303-233 2009] It can be expressed with a differential equation.

The reason for the distinct view on function and behavior lies in the requirement of systems engineering of "modeling … the response of the system to excitation. Function based behavior provides a model for response to excitation that is executable in tools. This provides a time line for that response and records logical errors in the model that prevent execution." [ISO/TS10303-1453 2009] Function behavior modeling has a long tradition in systems engineering. Relevant diagram types are the Functional Flow Block Diagram (FFBD), enhanced FFBD (EFFBD), N2 (N squared), Integration Definition for functional Modeling (IDEF0) and Behavior Diagram (BD). Application Protocol 233 sets the requirements to include these types of diagrams. This results in the meta-model for behaviors presented in Figure 8.

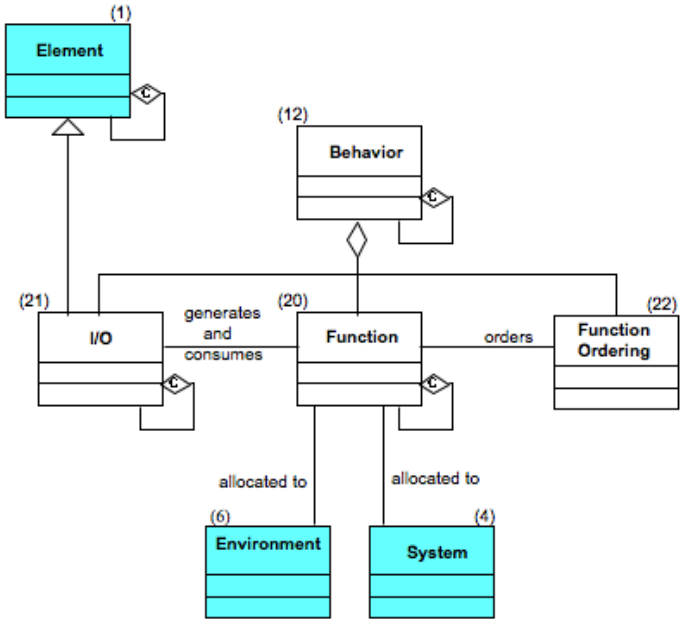

Figure 8: Behavior- function context according to ISO 10303 AP 233

To cope with the different use of terms in engineering design and systems engineering, an overview of terms and meanings is provided in Table 2-2. Relevant terms for functional modeling and their meanings are sorted in a way to compare terms of the systems engineering concept model against terms in product design.

In the following text, redundant terms are trailed with (SE) for systems engineering and (PD) for product design for disambiguation. Complete compliance exists in the terms "transfer function" to "function" (SE). "Function as desired effect" and "purpose function" should be related to requirements. A desired effect is what a client can specify and the demands of clients should not be ignored but treated as requirements. As engineers are professionals they should be able to sort out other meanings of functions and derive the functions (SE) from

requirements and overall task description. The meaning of function according to Umeda and Tomiyama [1995] expresses function (PD) as “abstracted behavior” (PD). This still is vague since the terms of behavior and abstraction have to be clarified. A direct matching to systems engineering terms is not possible here since this is a different viewpoint.

More difficult to define is the term “behavior”, since it is used differently in both domains. In this thesis, working with the term “physical property” (SE) instead of “behavior” (PD) is more relevant. In product design there is no completely compliant term to behavior (SE), yet it is reasonable to have a term that describes the response of an element to things in its environment. The closest matching of “behavior” (SE) is to “overall function” or “product function”. However, behavior (SE) is more than just a simple overall function since the output can change in response to changed input, e.g. actuating the crank of a car jack clockwise lifts the car, not actuating keeps the car in position and actuating it counterclockwise lets the car down. In the viewpoint of product design, “behavior” (SE) and “physical property” (SE) are highly intertwined since a “behavior” (SE) only can be created by designing elements with the appropriate physical properties of the parts comprising the product or system, e.g. designing the lead screw and nut with appropriate physical properties (friction, pitch, Young’s modulus) for the use in a car jack.

Table 2-2: Comparison of relevant terms

Product design		Systems engineering [ISO/DIS 10303-233 2009]	
function	purpose function [Buur 1990]	N/C	
	desired effect [CHANDRASEKARAN & JOSEPHSON 2000]	N/C	
	transfer (transformation) function [BUUR 1990] transformation of inputs into outputs [Roth 2002, Koller 1975, Pahl et al. 2007]	function	the entity in the context of modeling that transforms an input set of elements into a set of output elements that may be the same or measurably different from the input set
	description of behavior recognized by human through abstraction in order to utilize it [Umeda & Tomiyama 1995]	N/C	
functional decomposition	Overall function, product function, process	behavior	what an element is to do or is not to do in response to excitations it receives from the external elements in its environment
behavior	description of the physical properties of an artifact that deliver the actual performance at a known input [Ullman 2002]	physical property	what an element exhibits or does not exhibit in response to excitation and stimulation from auxiliary measurement entities that are not part of its context
	Values of state variables at an instant or interval of time [CHANDRASEKARAN & JOSEPHSON 2000]		
	transitions of states due to physical phenomena [Umeda & Tomiyama 1995]		
		N/C = no compliance	

The different definitions of terms in the domains of systems engineering and product design may lead to fundamental misunderstandings when engineers of different background collaborate in a project. The biggest discrepancies however are not stemming from text books but from journal papers documenting research. The systems engineering terms are explained in one single source that is a draft international standard. They are defined in a semantic dictionary and are free of contradictions. The context for the terms of the dictionary is the modeling of systems.

To avoid misunderstandings, for the remainder of the thesis the terms “function”, “behavior”

and “physical property” as defined for systems engineering in Table 2-2 will be used.

2.3.4 Summary of functions in product design and systems engineering

With this overview, a cross section of different functional modeling types and the relation to function representation is presented. The approaches of systematic product design apply the flow oriented concept of functional modeling. TRIZ functional modeling and functional modeling based on conservation principles are different approaches. This outlines the variety of concepts for functional modeling in engineering design. The function primitives for modeling that have been identified and used in the different concepts finally have been collected and reconciled to a dictionary consisting of two taxonomies called the NIST functional basis.

Although identified as a common activity, systems engineering has a different viewpoint on function and behavior modeling. This difference together with different approaches in engineering design has led to different meanings of the terms function and behavior. The meanings of the terms have been resolved for this thesis. A basis for functional modeling accepted in both product design and systems engineering is created. Core requirements for functional modeling are the concepts of (transfer-) function, function ordering and input-output relationships.

2.4 Modeling Languages for Product and System Modeling

Modeling languages are description methods. They are the next higher level to programming languages and have a tradition in the domains of software and electronic engineering. There are three levels of formality according to VDI/VDE guideline 3681:

- “Formal description method; has a mathematical basis and a completely defined syntax as well as a clear semantic interpretation.” [VDI /VDE 3681 2005] An example is a mathematical formula or bond graph models.
- “Semi-formal description method; has a completely defined syntax as well as a clear semantic interpretation but no mathematical basis.” [VDI /VDE 3681 2005] Modeling languages like UML, SysML or any other domain specific language (DSL) are examples for modeling languages.
- “Informal description method; has a syntax which is not complete in principle as well as a semantic which does not need to be clear. Example: Natural language.” [VDI /VDE 3681 2005]

Using this definition, modeling languages are semi-formal description methods. The added formality compared to informal description methods and the graphical presentations of the modeling languages are the main drivers for using modeling languages. Formal methods, e.g. predicate calculus or Boolean algebra, are not as easily understandable as graphs. Therefore

they are not appropriate for a domain spanning communication. Finally modeling languages have became common with the rise of object-oriented methods thus they expose the necessary maturity for engineering applications. In the following sections the modeling languages UML and SysML are presented.

2.4.1 The Unified Modeling Language

The origin of this description method stems from object-oriented software engineering, where the efforts of BOOCH, RAMBOUGH and JACOBSON [1998] led to the deployment of the Unified Modeling Language (UML) [OMG 2010c] in 1995. It is a "*graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML provides a standard way to write a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components*." [BOOCH et al. 1998] The elements of the language are based on the Meta Object Facility (MOF). Due to its origin in software engineering it is hardly used in mechanical design.

The OMG group defined the four layer architecture of modeling, see Figure 9. It describes the relationships between instances, models and meta-models. In the context of modeling the role of a metamodel is "to define the semantics for how model elements in a model get instantiated." [OMG 2011] The elements of the MOF in the topmost level are used to define the language specification of UML or SysML in M2 level. Domain specific adoptions, e.g. the definition of an additional stereotype, can be made at this modeling level. In level M1 the "user model" is situated. This is what is commonly referred to as "model" since it is the abstraction of an object of interest. The "run-time instances" at level M0 are the objects that are modeled with UML. They are outside the language specification.

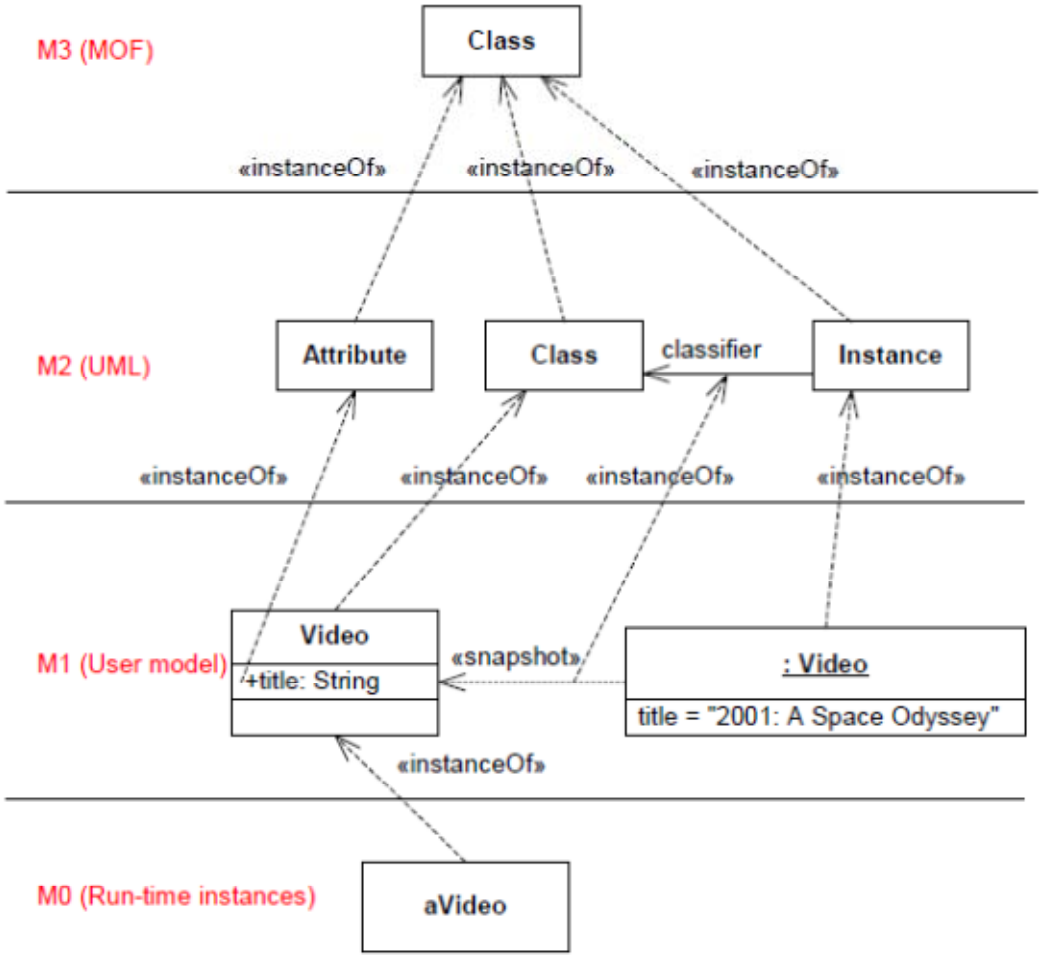

Figure 9: The four layer architecture of modeling according to [OMG 2011]

A model library is situated in the M1 level of the four layer architecture of UML

Although some scientists see potential in using single diagrams to express a certain aspect of conceptual design in product design [FUCHS 2004], the potential of modeling in an integrated way using UML and Computer Aided Software Engineering (CASE) tools for product design is barely exploited.

In a case study of modeling a hardware system, Bahill and Daniels [2003] point out that in large engineered system projects at first the use of UML tools and object-oriented methods is rejected by the customers but they finally recognized the great strength in using UML for domain spanning communication and documentation. In the study they present a home air conditioning and ventilation system that is modeled to a great extent with UML. Finally they conclude that UML and CASE tools can be used for hardware and algorithm design and are excellent communication tools.

2.4.2 Examples for the use of modeling languages in product design

At least in two engineering domains, modeling languages are a common means for conceptual design. With the introduction of the object oriented paradigm, UML became common for the modeling of software systems. The other domain is integrated circuit design where the "Very High Speed Integrated Circuit Hardware Description Language" VHDL is used for the design of integrated circuits. A VHDL model describes the logical functions (basic: AND, OR, NOT composed: NAND, NOR, XOR, flip-flop etc.) of the circuit; the patterns for conductors, transistors and insulators on the chip can be taken from a pattern-library. In this thesis, the approach is similar namely the selection of library components according to modeled functions.

In addition to UML there exists also some design specific languages (DSL) derived from UML. For Mechatronic design GAUSEMEIER [2009] deployed the "Specification Technique for the Description of Self-Optimizing Mechatronic Systems". He uses the Mechatronic Modeling Language extended from UML. The modeling language allows integrating the majority of the partial models used in the specification technique. Up to now, Gausemeier presented a process for mechatronic design and the relevant partial models for supporting the tasks but no supporting libraries for function and components.

During the MOKA Project, the MOKA Modeling Language (MML) was developed. [BRIMBLE & SELLINI 2000] The main purpose is to manage useful knowledge with a formal model. In this context, a formal model is graphical, object oriented and one abstraction level above application code. It was realized by adding predefined classes, views and attributes to UML. The classes used for MML mirror core terms of engineering like behavior, function, structure, principal solution and technical solution.

2.4.3 The Systems Modeling Language (SysML)

The Object Management Group describes SysML as "…a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametric, which is used to integrate with other engineering analysis models. SysML leverages the OMG XML Metadata Interchange (XMI) to exchange modeling data between tools, and is also intended to be compatible with the evolving ISO 10303-233 systems engineering data interchange standard. [OMG 2012] The modeling elements are called concrete syntax; a description can be found in the SysML specification. [OMG 2010b] A big portion of the concrete syntax is from UML 2. A part of UML 2 is excluded from SysML, e.g. time diagrams. Another portion is added to SysML as new syntax e.g. requirement diagrams or as model libraries or profiles to set constraints on the language.

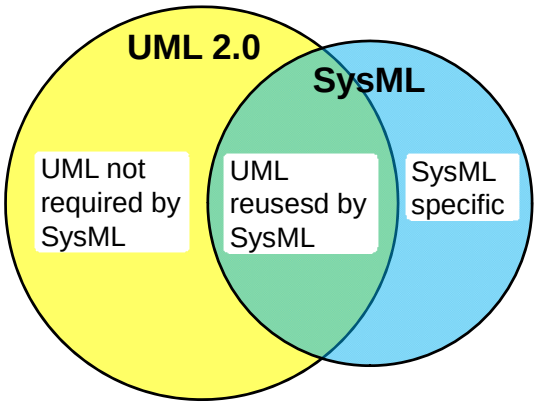

Figure 10: Language composition of SysML according to [OMG 2010b]

The reuse of UML is shown in Figure 10. Basically SysML contains all the forms of diagrams that are necessary to follow systems engineering tasks. The interested reader may refer to the user guides for SysML [FRIEDENTHAL et al. 2008, WEILKIENS 2006] or consult the OMG webpage [OMG 2012] for tutorials and further information.

Like UML, SysML is not a tool but needs a CASE tool. They are tools for modeling with modeling languages that are one level above programming languages and can be automatically parsed into certain object oriented programming languages like Java, C+ or C#. Of course that does not save one from programming but simplifies the work since, for instance, classes of a UML class diagram can be transformed into class definitions with all attributes and operation calls. There are open source tools like Eclipse (www.eclipse.org) that can be extended with the Papyrus plug-in for SysML. The plug-ins provides the definition of the syntax, properties of diagrams and methods that can be applied with the modeling language. A List of further SysML compatible tools can be found on the OMG-homepage [OMG 2012]. For modeling the examples of this work the CASE tool Magic Draw is used.

To work with CASE tools is still not very common in mechanical engineering. However in mechatronic projects the use of UML/SysML-specific diagrams is growing.

The grade of formality that is reached with a modeling language depends on the semantic basis that is used. In case of non-restricted use of natural language there is room for ambiguities. The use of semantic dictionaries and taxonomies provides additional formality for working with modeling languages.

2.4.4 SysML related research

With the development of SysML, a growing amount of research exsists. KERZHNER & PAREDIS [2009] describe the synthesis of hydraulic components to a working hydraulic circuit. For the realization of that approach, a library with predefined components and their interaction points, called ports, is created. A meta-model (at M2 modeling level) defines the structure for each component and the possible relation to other components. With a graph

transformation for each seed of a circuit, components will be added until no further addition of components is possible. What kind of component is added is determined by a probabilistic rule set. The benefit of this kind of synthesis is the fast generation of hydraulic circuit variants. On the cost side some effort has to be put in the set up of the meta-model and the sequence of adding components, since these two elements determine the output of the graph transformation process. The paper exploits port modeling and graph transformation for synthesis. In this thesis, synthesis is guided by function structures and uses flows between functions and ports for determining embodiments.

Also related to modeling with SysML but more directed towards the evaluation of physical properties of a concept JOBE et al. [2008] describe the integration with simulation tools. They propose a framework for organizing different simulation models together with SysML. They show how to organize analysis models of different disciplines with a simple model of a log splitter. Various aspects of analysis mirror the formalism of simulation and analysis like algebraic or DAE simulation or domain specific interests like hydraulic, electric or mechanical. SysML provides the design model and organizes the relations into the single aspects of the analysis models. It is proposed to store the analysis models in a model library. The work is pointed towards reuse of models. This way, the physical properties of a system can be evaluated with appropriate analysis tools and simulation can be reused. The related point with this work is the use of model libraries that in this case are for synthesis at the function and component levels.

Several efforts are taken to integrate the simulation language Modelica with SysML [JOHNSON et al. 2008]. The reason lies in the specialization of Modelica as a simulation tool for continuous dynamic systems and the ability of SysML to maintain high level relations such as requirements to functions and to super systems. SysML lacks the possibility to express fine grained behavior and geometry. An integration of SysML with Modelica and other simulation tools can facilitate the evaluation of the physical properties of systems. The paper shows that an automated link to a simulation tool is possible. This is relevant to evaluate the solution candidates, which is not covered in this thesis but important for the design process.

In a recent paper ALVAREZ CABRERA et al. [2009] present a method to express the models needed for FBS with SysML. They also use the NIST-FB as a source for function primitives to reduce the combinatory explosion of the qualitative process reasoning. However FBS methodology present function structures as function hierarchies. In contrast, this work follows a flow representation. It is acknowledged that using a lexicon of function facilitates the mapping to behaviors. Initially FBS considered storing the function-behavior relation without a taxonomy of functions [UMEDA & TOMIYAMA 1995]. This shows that the NIST-FB is general purpose and can be also used for other approaches to functional modeling than the flow oriented approach. Further Alvarez Cabrera aims to represent FBS modeling with the KIEF ontology using SysML. They do not attempt to use the syntax of SysML to support modeling nor do they propose a way to deal with the conflicting terms for function and behavior of the KIEF ontology and SysML.

2.4.5 Summary: use of modeling languages

Modeling languages are used in many engineering disciplines. The benefits of modeling languages are the clear semantic interpretation and the ability to express higher level relations (UML, SysML, etc.). In software engineering, it is common practice to derive a part of the programming code, e.g. classes in java, directly from a UML classes. At the research level there are several examples where modeling languages are successfully used in mechatronic and hardware design. Ongoing research is performed in the integration of SysML with simulation tools and automated synthesis. Only little research is done in the area of model libraries with SysML. This thesis contributes to this area in the fields of functional and component modeling.

2.5 Knowledge bases for engineering tasks

In the past sections design processes, product modeling and the specific terms of function are reviewed. Yet engineering is a knowledge intensive process [KLEIN 2000]. Therefore sources of information and knowledge are essential. In this section, different sources of information for conceptual design will be presented.

2.5.1 Design catalogs

Design catalogs can be divided into several categories: solution catalogs, process catalogs, object catalogs and relation catalogs. [ROTH 1994] Solution catalogs provide solutions to a certain problem. To access the solution, the problem has to be formulated in a certain way. A process catalog provides a series of actions, e.g. a sequence of manufacturing steps or steps for the variation of a shape and an object catalog provides collections of objects that contain relevant objects independent from a task, for instance machine elements and material properties. This work contributes to solution catalogs since they are accessed through the functional model that is created in conceptual design. A different presentation from design catalogs is used. Due to the use of a modeling language, look up tables are avoided.

Solution catalogs:

- KASTRUP & KOLLER [1998]: Working principles are provided for mechanical, electro-mechanical and fluid-mechanical physical effects. Working principles are principles that make use of a physical effect and described in a conceptual way. A distinction is made between energy transforming effects and material handling effects. The catalog is accessed through functional modeling with function primitives called basic functions. They represent the input output relation and a description of what the device should do (transformation function). With state variables of input and output e.g. force or velocity a working principle can be selected from a look-up table. The principle is presented in terms of a mathematical equation (as far as possible) and a sketch of the working principle. The working principles are presented in a condensed and commonly applicable form. It still requires the interpretation and creativity of the designer to make a sensible artifact out of it. A great deal of

knowledge is necessary to make multi-stroke devices out of the principles that are formulated as single-stroke principles. [KOLLER & KASTRUP 1998] The working principle also should spark the designers ideas not to rely only on own experience.

- ROTH [2000] also provides a catalog of working principles but is mainly focused on energy handling effects. The working principles are described in a similar way and are identified by the same process as in Koller/Kastrup. However the lexicon of function primitives is different from Koller & Kastrup's. Less working principles are provided. Another catalog deals with logical functions as they are needed for information processing. It is not integrated in the lexicon of transformation functions but forms a separate lexicon. A solution catalog is given for mechanical adders, flip flops, switches, etc.

Both solution catalogs are design specific for mechanical engineering. They are provided as paper based catalogs. In the existing form they need human creativity to develop the working principles into components. They have some relevance in education but the relevance for industry use is not known. With a growing number of components off the shelf (COTS) that can be purchased economically, a bought solution is often preferred. This is especially true for machine elements and norm parts where the problem reduces to a selection or configuration problem.

Component catalogs:

Component catalogs are relevant for this research since the approach attempts to directly identify components out of a model library. As far as possible the component library should have an identical structure as preexisting catalogs that are in use. The social impact of introducing a new modeling tool or modeling paradigm to an organization is serious. Therefore, as much as possible should be re-used from known systems, in this case catalogs.

Component catalogs contain information about standard parts like machine elements, locks, electric components, etc. For the modeling of parts, a source of information about component names and relevant attributes is needed. Component catalogs can be divided into supplier catalogs and overview catalogs. One of the most known overview catalogs is Roloff-Matek [GRÜNDER 2009]. It provides components with some specific attributes and its suppliers. The designers can select a supplier according to the range of values they need for the given attributes. The access to the catalog names is only via the component name. It gives a great overview over many components and suppliers but not a complete specification. The catalog itself is ordered like E-Cl@ss [KOZIEL et al. 2006], a product and service classification standard (PSCS). The objective of the catalog is to provide a fast overview about available components and its suppliers.

Supplier catalogs are made by the suppliers of the components. Today, most of the suppliers offer online catalogs for their products e.g. machine elements from supplier INA Schaeffler [LINDNER 2012]. In supplier catalogs values for the specific attributes are given. Often suppliers even provide geometry CAD data and formulas to calculate wear behavior. In this case, the catalog can also be searched with the e-Cl@ss hierarchy number.

Component catalogs exist for electronic components, hydraulic and pneumatic components and a huge number of mechanical components. Overview catalogs like Roloff-Matek provide

not only an overview but also an interval of values of the key attributes of the components. During design this interval will shrink until a component with a specific part number is selected.

2.5.2 Product and service classification standards (PSCS)

The best known product and service classification standards are the United Nations Standard Products and Service Code (UNSPSC) for almost all commercial goods and services, Proficlass for construction and facility services, ETIM for Electric devices and procurement, RosettaNet for service Processes and E-Cl@ss. The latter provides terms for products and services that are updated every few months.

PSCS provide taxonomies of products and services and is fueled by electronic procurement rather than by design synthesis. This means that terms are not used exclusively; auxiliary agents like lubricants are included as well as complete production machinery. Many attributes are defined but only some are of interest for design synthesis.

PSCS provide a basis for supplier catalogs by providing ordering frameworks for components and processes. They are mostly driven by a community of users that provide and propagate the classification standards. The E-Cl@ss community is an association of more than 120 members of industry, industrial associations and public institutions. In case of E-Cl@ss, the classification is based on ISO 13584 [ISO13584 2010] and on DIN 4002. Although both standards stem from the attempt to define attributes for a parts library in context of an integrated product model, they were boosted by the need of classification for electronic procurement. E-Cl@ss provides a hierarchy of Classes, subclasses and commodity classes with lists of relevant attributes to compare the components of different suppliers. They do not provide specific values to the attributes.

In a closer examination, PSCS face the problem that they are structured for e-commerce rather than for design. This leads to a mixture of services, auxiliary agents like lubricants, etc. and components within one hierarchy. HEPP and ABRAMOWICZ [2007] presented an otology to identify the component and divide it from services or others. With this ontology it is simple to divide artifacts from services, a division that is not provided by the hierarchical layers structuring the E-Cl@ss classification.

2.5.3 Design repositories

“Design repositories are the modern successor of file cabinets where information of past design is stored. They store descriptions of past designs together with their rationale in a form suitable for browsing, retrieval and direct use.” [SZYKMAN et al. 2000] Most of the PDM systems of today face the problem that they either do not store design rationale or have to retrieve information about design from many places. The most relevant work on design repositories is the NIST design repository that is transferred to the design repository of University of Rolla. [KURTOGLU et al. 2009] Today the design repository is hosted at the Oregon State University. The data of the more than 100 products has been collected by students. They disassembled the products and measured the parts by hand to retrieve

information. In a reverse engineering manner they reasoned about product functions and function structures and modeled them. The design repository is used to empirically prove the usability of the NIST-FB and approaches of automated design. During a study for research in design automation, KURTOGLU et al. [2009] propose a way to automatically match embodiments for functions. In that work they ordered components in a taxonomy that parallels the hierarchy levels as in the NIST-FB. Primary level term “channelers” contains “importers/exporters”, “transferrers” and “guiders”. The components were placed in the taxonomy according to the most obvious function the component would realize. The parallel hierarchy is used to identify suitable embodiments for functions. The implementation of automated design was tested against a test group of students manually selecting components out of the design repository for the embodiment of a function structure. The automated synthesis implementation found more solutions than the students but all the solutions that were found by students have been included in the results of the automated search.

The link of this work to the design repository is manifold. First, it is highly intertwined with the NIST-FB. Second, the approach to link functions to components was demonstrated with the design repository. Thus, it is a source of knowledge, although the contents have been pulled together in reverse engineering manner instead of documenting the design rationale during designing, which is the original idea of a design repository. On the other hand there are points that contrast this work. The design repository is made from scratch for research purposes; this work uses PSCS information that is available in industry. Consequently, the information about components is ordered for commercial purposes and cannot be accessed directly for conceptual design. The thesis also provides a new approach to identify embodiments for functions from the flows and corresponding ports of components.

2.6 Summary of background section

In Table 2-3 the most important sources of literature are compiled. To date, there are only few attempts to integrate modeling approaches of systems engineering with design tasks in product design. Due to that reason there is no implementation of a semantic dictionary for functions as a SysML model library. Design catalogs are first paper based methods linking functions to embodiments. FBS methodologies provide an integrated tool support but do not fit into the semantic world of other approaches. The functional basis efforts led to a function modeler as a standalone tool. On the other hand, functional modeling is an integral part of systems and product models. This is a gap; therefore the approach is the implementation of model libraries for functional and component modeling with SysML.

Table 2-3: Comparison of relevant literature

					Conceptual design models			
	Reference	Approach/ Implementation	Scope of application	Example	requirements	function	physical property	components
Design catalogs	Koller & Kastrup	Design catalogs / paper based	energy, material & signal handling mechanical engineering			formal set of primitives; explanations & examples	as working principles through I/O of primitives	morphologic matrix to combine partial solutions
	Roth 2002	Design catalogs / paper based	energy, material & signal handling mechanical engineering	car jack	check lists	formal set of primitives; explanations & examples	as working principles through I/O of primitives	morphologic matrix to combine partial solutions
	Hundal 1990	Design catalog / software prototype	energy, material & signal handling	temperature sensor		6 categories of basic functions mechanism to add functions	working principles taken from solution database	solution matrix with evaluation
Functional Basis	Szykman et al. 1999	function representation and literature review	generality claimed for various concepts of functional modeling			100 functions; 130 flows; no explanation, 3 layer hierarchy		
	Stone & Wood 2000	Literature reviews and induction	generality claimed for various concepts of functional modeling			3 layer hierarchy, explanations and examples		
	Hirtz et al. 2002	reconciliation of previous efforts	generality claimed for various concepts of functional modeling			52 functions, 45 flows, synonyms, examples and descriptions		
	Nagel et al. 2009	Function Modeling / software prototype	modeling function structures			52 functions, 45 flows, synonyms		
Product modeling	Grabowski & Anderl 1991	product modeling	theory of organization of product data information		yes	part of design model		Concept model, assembly model, part model
	Fenves et al. 2004	Core Product Model / UML ER-model	organization of product information for product life cycle		yes	function and transfer function	attribute of artifact	structure of artifacts
FBS based	Yoshioka 2004	Knowledge Intensive Engineering Framework (KIEF)/ software prototype visual works/ Smalltalk	whole product life cycle; Integration of FBS modeler with other modelers	Photocopier		as subjective conception of behavior	"behavior" as state change due to physical effect	Physical feature
	Alvarez Cabrera et al. 2	KIEF, presented with SysML	FBS Modeling with SysML, mechatronic design	electric DC motor		as subjective conception of behavior	presented as parametric diagram	Physical feature
	Komoto 2010	KIEF	System architecting for mechatronic products	Photocopier		as subjective conception of behavior	"behavior" as state change due to physical effect	physical feature and geometric primitives to allocate space for artifacts
MBSE Methodologies	Hoffmann 2006	IBM Telelogic Harmony SE	Large systems architectures claimed; software bias	Service request example	repository; meth. support	sequence/ activity diagrams		block diagram
	Friedenthal et al. 2008	OOSEM /CASE tools supporting SysML	Large systems balanced HW/SW	Home security system	requirement diagrams	behavior diagrams	physical properties	structure diagrams
	Balmelli et al. 2006	RUP SE /RUP modeling center based on eclipse	Large systems architecture framework		repository	behavior diagrams		structure diagrams
	Childers & Long 2004	Vitech MBSE/ CORE modeling environment	Large systems		source & rqm analysis	function and behavior analysis		architecture synthesis

3 Modeling example and validation study

In this section the case studies are presented. The first study, called modeling example is the mechanical part of a luggage compartment cover. It is used to show the modeling tasks of the design process that can be performed with SysML modeling. The second study is a more complex mechatronic product that requires more detailed functional modeling. It is used for the validation of the approach.

3.1 Modeling Example: Luggage compartment cover

The example used for the first modeling study is a luggage compartment cover, a subsystem of a vehicle. It is part of many station wagon cars and protects the luggage compartment and its contents from sunlight and viewing.

The basic system as shown in Figure 11 is purely mechanical. It is operated by a vehicle occupant to be opened and closed. It can be disengaged by pulling the retaining pins out of the guiding tracks. The designer's task is to add some ergonomic comfort to this system. The system is to be modified so that by slightly touching the cover it opens automatically and the retaining pins are guided in the tracks. In case the rear seats are folded down, in order to transport bigger luggage, the cartridge should be removable.

The cover consists of a self-retracting spring loaded canvas roll. Self-retraction occurs when the retaining pins are released from the tracks. To make the roll hold firmly, it is inserted into a cartridge that is attached to the upper front end of the vehicle's luggage compartment just behind the rear seats.

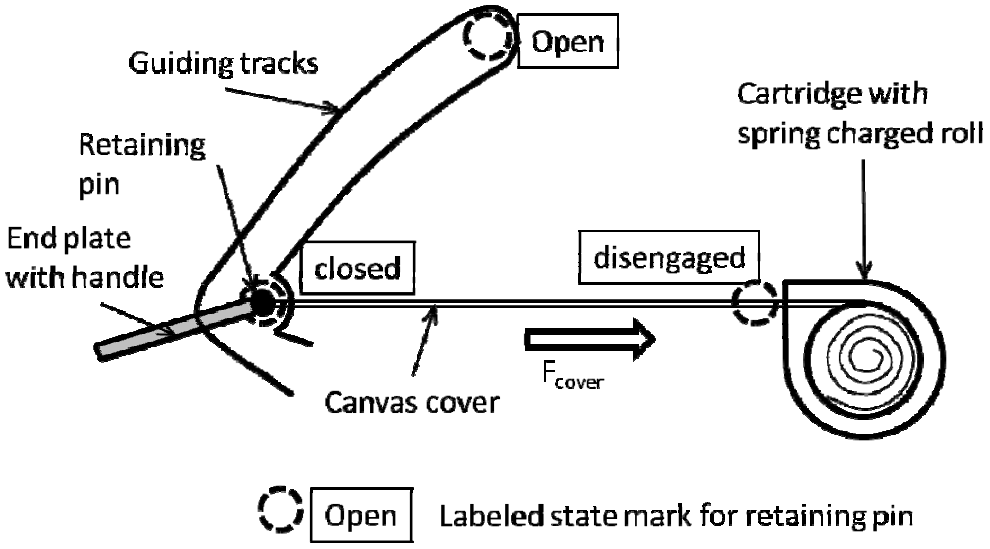

Figure 11: Sketch of a luggage compartment cover

To close the cover, a vehicle occupant must grab the handle, situated at the end board, pull the

end plate down along the guiding tracks and secure it to a hook-like part at the end of the track. When the back door is closed the end plate fills the gap between the back door and the retaining pins. The canvas stretches between the endplate and the cassette and covers the main area of the luggage compartment.

3.2 Validation study: 3D-printer

The validation study is more complex to show the generality of the model libraries developed and the scalability of the modeling approach. Scalability is the property of a model to hold when the modeled system or product is bigger or more complex than the initial system considered.

3.2.1 The RepRap project

The validation study is a three dimensional printer that applies the fused deposition modeling (FDM) method. The information about the 3D printer is taken from the RepRap project. The idea of the project is to evaluate the impact of a self replicating machine on society. All mechanical parts, except machine elements, can be made by the machine itself. Since almost every interested person should be able to participate in the project and make his/her own printer, all information is available under GNU Public License (GPL). Detailed information can be found on the web-page www.reprap.org or in the publication of the authors [JONES et al. 2011]. Figure 12 shows the image of the first model of the RepRap project 3D-printer called Darwin.

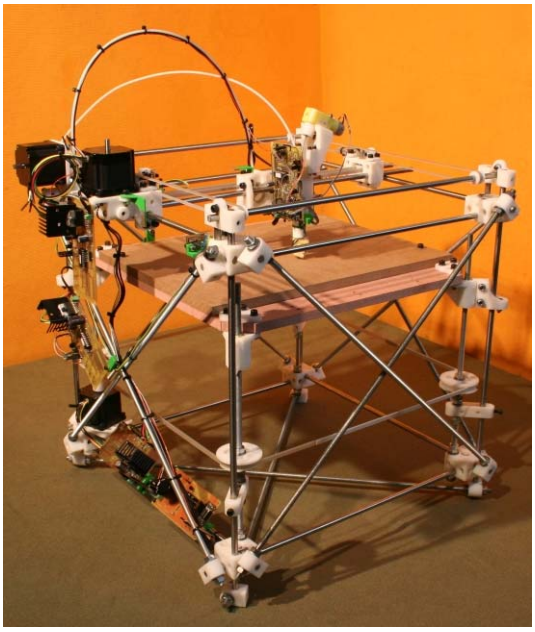

Figure 12: Darwin, first 3D printer model of RepRap project [JONES et al. 2011].

The example for modeling was selected for several reasons. First, the product is complex enough to show functional modeling and decomposition. Second, much information is available in the internet and finally the printer is further developed in an open community and improved several times. Although the project is finished, a community that makes adoptions and improvements still exists.

Since SysML is a modeling language capable of business process modeling (BPM- the activity to present the processes of an enterprise in systems engineering) there are many possibilities of modeling in the RepRap project. For instance the process of information distribution and how to purchase the parts for a 3D-printer as well as how to contribute additional knowledge to the community could be modeled. For simplicity, only the conceptual design of the printer hardware is modeled.

3.2.2 The RepRap3D-Printer

Using FDM, a three dimensional object is created by a number of layers of fused plastic. Layers are created by the extrusion of strands that are placed to fill the cross section of that layer. The plastic is provided as a filament and extruded through a thin nozzle that is moved in two directions over a print bed, see Figure 13. The extrusion of plastic is actuated and stopped depending on whether solid material is needed at a certain place. Once the layer is finished, the nozzle is lifted by the dimension of the thickness of the layer and the next layer is created. In the print head, a mechanism feeds the filament into the heated nozzle. The feeding is proportional to the relative velocity of the print head against the object to print. A temperature sensor measures the temperature at the nozzle, i.e. the hot end. The heating of the nozzle is actuated according to the temperature measured relative to the target temperature. The target temperature depends on the melting temperature of the used plastic.

The positioning unit is a machine that allows the movement of the print head relative to the print bed in a three dimensional Cartesian space, which is the printable volume. The print bed is the support where the first layer is created. The positioning unit therefore consists of a frame and linear drives. A linear drive consists of a set of linear supports (shafts, rails, etc.), a carriage allowing linear movement on the support and a driving device (belt-pulley, rack-pinion, lead screw-nut, etc.) transforming rotation into translation. The linear drives are driven by standard type electric actuators and are arranged to form a kinematic chain between the print bed and the print head. The frame gives the required rigidity, contact to the ground (e.g. a table) and is the non-moving part of the printer.

The kit to build up a printer includes a printed circuit board for the electronic components that control the actuators and process signals. Actuators are the stepper motors to move each axis and the feed mechanism and a resistor for heating the nozzle. Sensors are the switches that detect the end of an axis movement and the temperature sensor at the nozzle. The circuit board can be connected to a computer via a USB bus like an ordinary printer. The mechanical part of the printer consists of machine elements and special parts. The special parts have a shape that is difficult to manufacture with traditional milling but easy with rapid prototyping machines. The special parts can be produced by the printer itself. The other parts of the printer are machine elements like threaded rods, nuts, washers and shafts and a board.

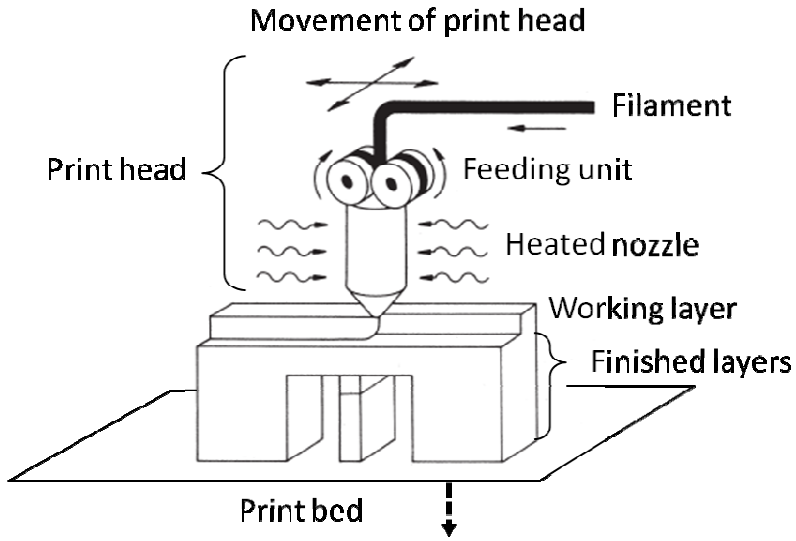

Figure 13: Principle of fused deposition modeling (FDM) adopted from DUBBEL [2007]

The community provides open source software that converts 3D models into G-code. It contains the path information for the print head. Open source driver software (referred to as firmware) is used to test the functionality and to calibrate the printer. The RepRap community provides all information to set up one's own RepRap machine including the data for printing the special parts of the printer.

In the validation study, details like single screws and washers are not considered. The modeling is stopped at a level where essential components can be identified from a model library. Specially shaped parts and machine elements like screws are out of scope. Further not all diagrams that have been modeled are relevant for the validation of the thesis. Diagrams that show only different details but do not expose additional contributions are left out.

4 Modeling approach for conceptual design using SysML

Within this Chapter, first an approach to supporting conceptual design using SysML is shown. Next, the most relevant topics, modeling function structures and component topology are presented. The purpose of functional modeling and the principles of modeling are explained. Then the appropriate presentation form in SysML is derived from the SysML specification and requirements for conceptual design. Functional modeling, required principles for functional modeling and SysML syntax are compared. In a similar way, the model for components is presented. The creation of model libraries for the support of modeling function structures is defined.

4.1 Conceptual design approach supported with SysML

This section presents an overview of the modeling approach for conceptual design with SysML exemplified through the modeling example given in Section 3.1. In previous work, a similar study is presented [WÖLKL & SHEA 2009]. The basis of that study is the VDI guideline for conceptual design [VDI 2221 1993]. A description of the tasks is given in Table 4-1. The computational aid consists of partial models (PM), editors (ED), operational methods (OM) and data and knowledge based (DKB). At least stages one and two can be covered completely with CASE tools and SysML. In sections 4.1.1 to 4.1.3 the stages will be shortly explained to give an overview of how SysML can be used in conceptual design.

Table 4-1: Overview of tasks in conceptual design from VDI 2222, also shown in Figure 1

	Computational aid	Tasks
Stage 1: Define and clarify design task		
PM	Requirements list	- Formulate design task
ED	Forms, Text, Symbols and Graphics	- Classify task
OM	Analysis, Search and Planning methods	- Search for similar requirements lists an
DKB	Requirements lists, Design task plans, Standards and Normative	- Create new specification
Stage 2: Determine functions and their structures		
PM	Functions, Function structure	- Recognize functions and describe them - Describe and allocate flows - Allocate requirements - Structure and link functional blocks
ED	Text and symbols for functions, flows and Structures	
OM	Analysis, Search, Combination and Selection methods	
DKB	Functions and Function structures (Standards or previous Developments)	
Stage 3: Search for working principles and their structures		
PM	Working principle, Working structure	- Search, allocate and select working prir - Represent principle solutions - Augment requirements torequirement - Refine design plan
ED	Sketches, Symbols, Graphs and Drawings	
OM	Methods of analysis , search , variation, simulation, pre-calculation, evaluation and selection; Rulebased systems; Agents	
	Conceptual solutions (Phys. effects, Working principles, Working	

4.1.1 Define and clarify design task with SysML

Clarify the design task requires the designer to obtain task and requirements from the client. SysML offers either the possibility to write some text into an overview diagram or create a block definition diagram that only shows the topmost block containing the name of the artifact to design and then add some description to it. The description of the task should be short and precise, further details can be shown as requirements.

SysML provides the requirements diagram for requirements modeling. Modeling elements are requirements (derived from UML class) and relations. Each requirement contains a name, an id and the description text. Requirements can be organized in a tee like structure via containment relations. A containment relation is indicated by a crosshair symbol at the upper end of the relation. In Figure 14, depicting requirements for the luggage compartment cover example, the relation reads “R1” contains “R1_1”, “R1_2” and R1_3”. Refinement relations are used to express the requirement as a state diagram or activity diagram. In Figure 14, the activity “CloseLCC_TL” is an example for the use of a refinement relation. It describes the requirement to close the cover with an activity diagram i.e. a sequence of actions. Satisfy relations point from blocks to requirements that are satisfied by the block. The blocks “RetainingPins” and “Rest2Close” are needed to hold the cover in closed position. These blocks are not known when the requirements are collected the first time, but they are added during modeling the concept. Verify relationships relate test cases to the requirements that are verified by the test case. Test cases are behavior diagrams that describe the test procedure that is applicable to verify the requirements. The chief interest of engineering design is to document the requirements. The requirement to relate requirements to other model elements stems from systems engineering. CASE tools provide analysis tools that allow analyzing dependencies with matrix methods, e.g. a verification matrix for requirements and test cases.

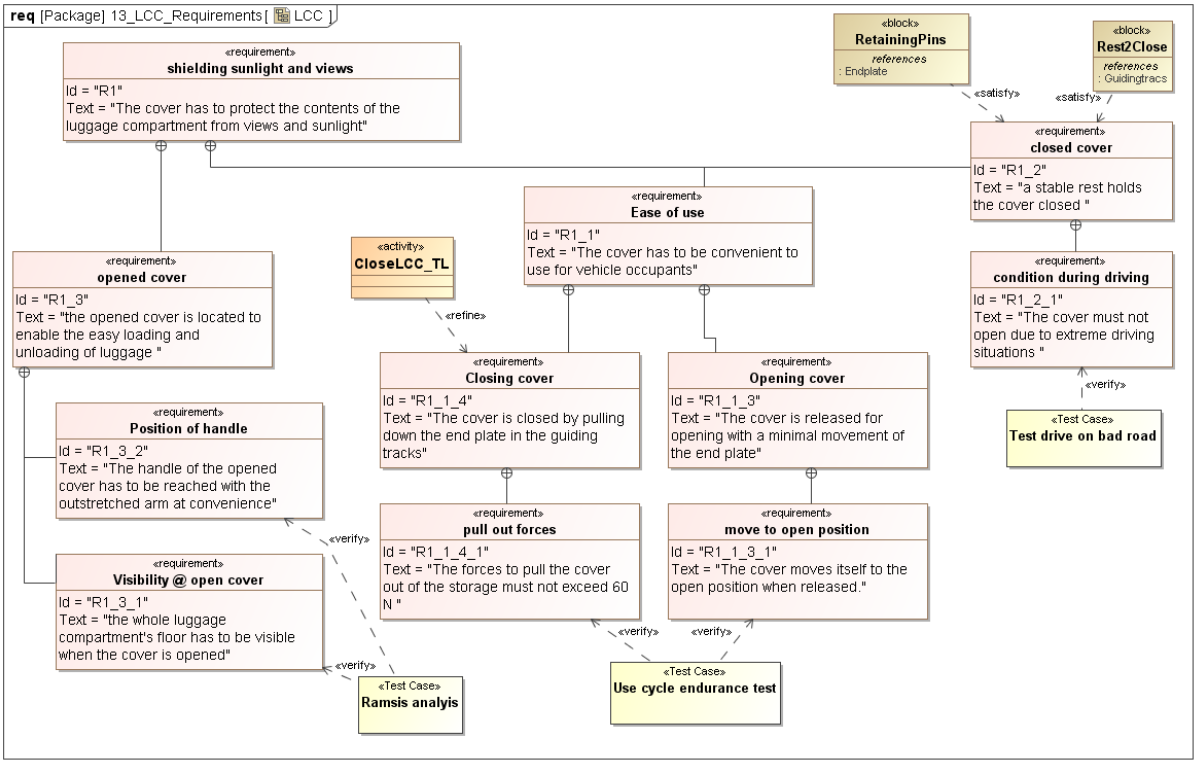

Figure 14: Requirement diagram for the luggage compartment cover

Another task of conceptual design is to scope the task. In SysML, the use case diagram is taken over from UML. It shows the (intended) behavior of a system from the perspective of a user, client, operator etc. The modeling elements are actors, use cases and relations. [BOOCH et al. 1998] A use case diagram shows what the systems does and what actors are involved in what tasks. Use case diagrams help identify all stakeholders and specific use cases that may require a specific consideration. In the study, use case diagrams will be used to focus functional modeling of the product function. Figure 15 shows a use case diagram for the luggage compartment cover. The use case “close cover” is detailed with an activity diagram in Figure 15. The stick man represents the actor and the ellipses the use cases. Included use cases can be invoked at any time. Extensions to use cases are activated on a certain condition, for instance when the vehicle occupant wants to load a bigger item that will inhibit the closing of the cover, he or she can disengage or take out the cover to use the space that is usually blocked by the cover.

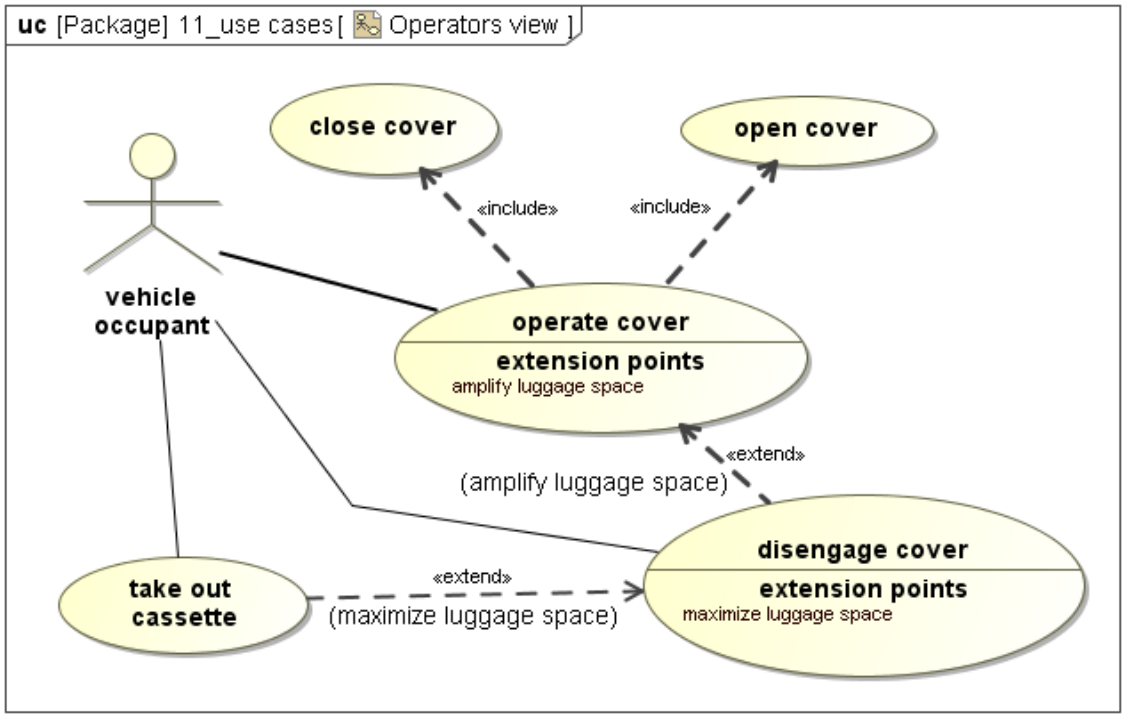

Figure 15: Use case diagram for the luggage compartment cover

Although some checklists can be found for requirements [ROTH 2000], no model library for requirements is proposed in this work. However, working in a special domain like passenger car design there is a set of requirements related to material that does not often change and is reused whenever needed. The integration of these requirements can be done in a modeling template and is worthwhile when a SysML model is used to create documents (e.g. supplier's inquiry).

4.1.2 Determine functions and their structure

Determining functions and their structure is a core activity in modeling a conceptual design. In the previous studies [WÖLKL & SHEA 2009, WÖLKL & SHEA 2011] the activity diagrams are used for functional modeling. This was done following best practice of user guides, modeling examples and has been intensively discussed with modeling experts. The justification for using a SysML behavior diagram was given in 2.3; a detailed examination of the relevant specification and a comparison with modeling principles is provided in Section 4.2. Activity diagrams are used for the presentation of function structures. Again CASE tools take the role of the editor.

Activity diagrams provide object nodes, actions, fork nodes, join nodes and relations called control flow and object flow. The function library has the role of a data- and knowledgebase. Both set up of the library and modeling of function structures are core contributions of the

thesis and presented in detail in Section 4.2. Defining flows for connecting functions is also possible in activity diagrams. Flows are represented by object flow paths that connect the pins of actions. Object flows consist of material, energy and information flows in contrast to the control flow that describes the sequence of actions executed within an activity, which was the original use in UML.

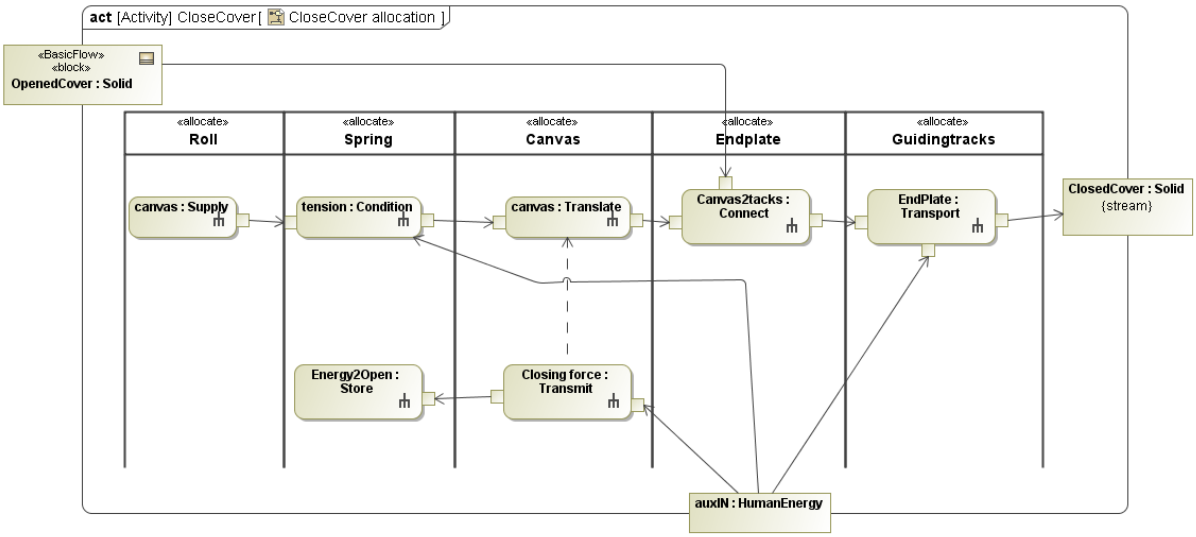

Figure 16: Activity diagram as functional model

In Fig. 16, the use case “close cover” from Fig. 15 is broken down into basic functions. When the cover is pulled out of the cartridge, the rolled up stored material is transformed into a planar shape. At the same time energy is stored in the clockwork spring attached to the roll. It applies force to the cover in order to tension and roll it up into the cartridge. An object flow from outside the activity is introduced. It represents an energy flow from beyond the borders of the use case, in this case human energy that is needed to pull the cover to the closed position, to load the spring for opening and to tension the cover. The compartments with headlines are “so-called” swim lanes or allocation partition lanes. They are used to allocate functions to other model elements. In the case of Figure 16, the activities are allocated to blocks representing components. The allocation is done after the components are determined.

4.1.3 Search for components and component structures

Instead of searching for working principles, which are artifacts that deploy physical phenomena, the approach directly maps components to functions. For the presentation of components and the internal structure of the product, SysML block definition diagrams (BDD) and internal block diagrams (IBD) are used. Modeling elements are blocks and associations for BDDs, part properties, ports and links for the IBD. The required operational methods for that phase are the search for components, methods for their structural and parametric variation, analysis and simulation. The integration of SysML with analysis and

simulation is not within the scope of this thesis.

The DKB used for component modeling is a model library that holds a large amount of preexisting components presented as blocks. The selection of components out of E-Cl@ss is presented in Section 4.3.

Once the needed components have been identified, they have to be linked to one another in an IBD. The links embody the flows defined in the function structure e.g. electrical energy needs a cable for being transmitted. Properties are the primary structural elements for internal block diagrams. Part properties describe the elements needed or the decomposition of a BDD. The IBD shows how these part properties are related. It can only be derived from a block; therefore at least the parent block has to be defined upfront.

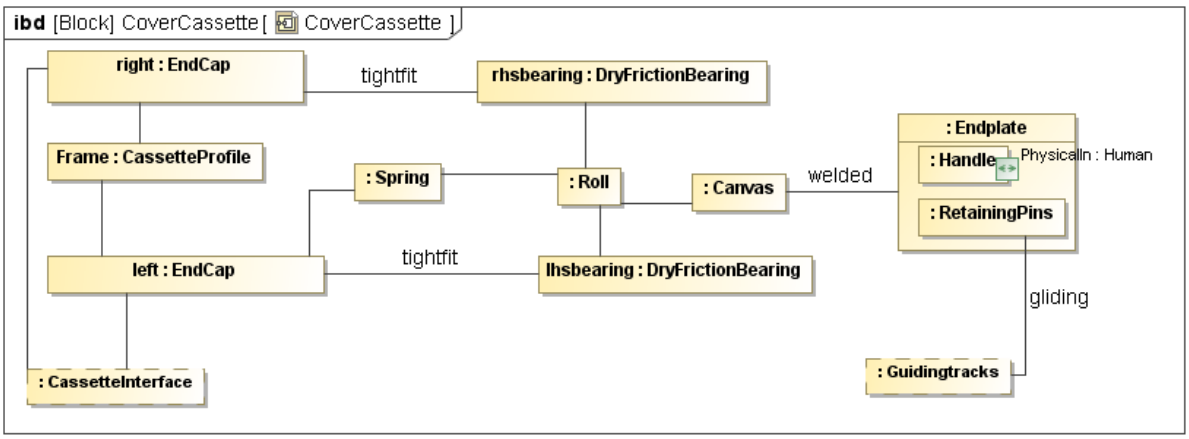

Figure 17: Internal block diagram of cover cassette

Figure 17 shows the internal block diagram of the cover cassette. The interfacing parts can be seen as referenced parts and presented as boxes with dashed lines. All part properties have either to be taken out of the model library or defined upfront. The dry friction bearing, the handle and the spring are the part properties taken from the model library. The other elements are depending on the specific shape of the vehicle and have to be defined from scratch. Only semi finished parts to make the new parts, e.g. plastic foils or alloy profile can be searched for in the model library. This, however, needed further decomposition for all existing parts and an extension on the existing model library.

Another viewpoint on the composition of the cover is provided by the block definition diagram. In Figure 18 a BDD is shown. It is the basis for the IBD shown in Figure 17.

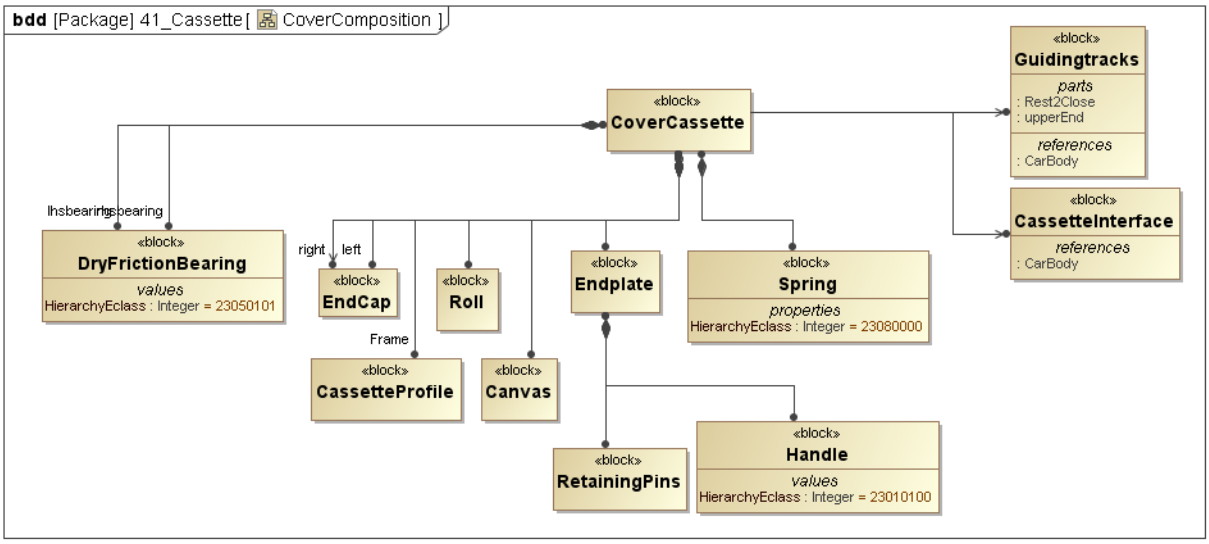

Figure 18: BDD of the cover cassette

The BDD can show blocks to a greater detail than an IDB. A block can have several compartments like values, parts, references, operations, etc. On the other hand a BDD cannot show the structure of how the single blocks are interrelated. In Figure 18, the blocks taken out of the model library can easily be identified by the "HierarchyEClass" value. The relations between the blocks are either compositions (with black diamonds at the upper ends) or links (with arrows at the lower ends). A composition means that a block at the lower end is necessary for the block at the upper end. A link says that the blocks “Guidingtracks” and “CassetteInterface” are referenced from the cover cassette but not necessary for the existence of the parent block.

4.1.4 Summary of modeling approach with SysML

In the last three sections, the approach for conceptual design has been presented together with the SysML diagrams that can be used to support the approach. The model language is generally applicable. Yet, the data- and knowledge bases are not defined for modeling. Data and knowledge bases are necessary to support the designers to create comparable models. They will prevent that people can model any way they like and save time that without model libraries is needed for the definition of modeling elements. A core activity of conceptual modeling is functional modeling. It is a key activity to identify solution candidates for components and a means to convey design objectives. At any modeling level there are means to link to other modeling elements, satisfy and verify links in requirement diagrams and the allocation of activities to components. To make clear how a SysML model works, a metaphor can be used. A SysML model is like a building where the diagrams are the windows to look into the building and discover the interrelations that keep the building together [FRIEDENTHAL et al. 2008]. It is impossible to see all dependencies at once. With the given overview, the

specific details of functional modeling and component modeling for conceptual design with SysML will be presented.

4.2 Functional modeling and presentations of function structures

In section 4.2.1 the principles for functional modeling from standard approaches in mechanical engineering are shown. The identified principles set the requirements for modeling function structures with SysML. In 4.2.2 the presentation forms of several function structures exposing the principles are shown. They are examples of common functional modeling in current product design practice. The extracted principles are the basis for the function structure presentation using SysML.

4.2.1 Principles for function model presentations

The model presentation applied in this thesis for functional modeling is a network type graph. Graphs consist of nodes and edges. For the function model, the nodes represent transformations and the edges represent flows. The three major flows *material, energy* and *signal* may be illustrated by different line types of the edges as shown in Fig. 19. Nodes can be connected to other nodes by at least one, mostly many edges. This is the core of a function structure presentation. Figure 19 shows some examples of functional modeling to identify the principles of modeling.

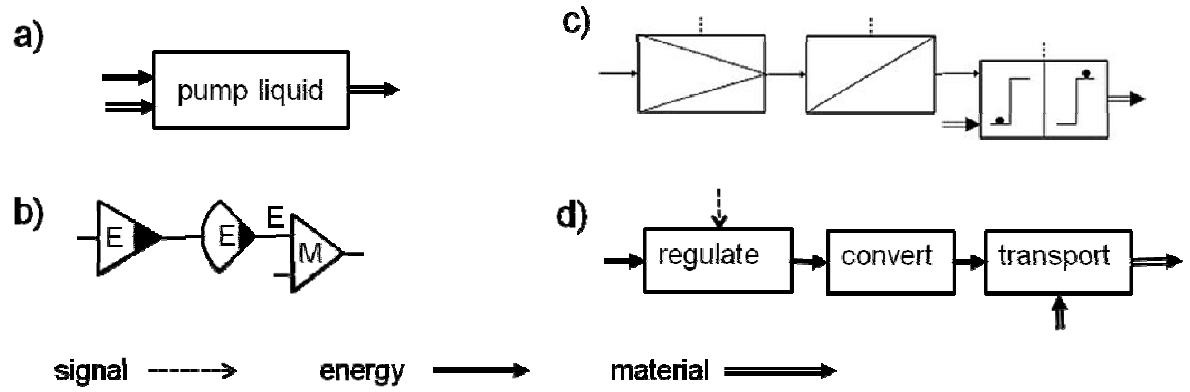

Figure 19: Different function model presentations: a) super function of a modulated pump; b) decomposition of a) with ROTH's symbolic presentation [ROTH 2000]; c) Koller's presentation[KOLLER 1998] d) presentation using NIST-FB terms, modeling according to [PAHL et al. 2007]

The presentations b), c) and d) in Figure 19 look different although they are decompositions of the same top level function shown in a). ROTH's model presentation uses symbols for "*change magnitude of energy*", "*convert energy*" and "*connect energy to material*". The letters "E" and "M" indicate that the processed flows are *energy* and *material*. There is no information about the direction of the flow, suggesting all flows can be reversed and functions

are reversible. The meaning of the sequence in Figure 19c) is the same but instead of labels the flow information presented by the line types is used to connect the elementary functions. Additional information is given with the direction of the flow indicated by arrow heads. The box diagram of fig. 19 d) can use any term to describe the flow. In this case, terms used are from the NIST functional basis. Since there is no entry for charging matter with energy (kinetic, thermal or other) it is difficult to select an appropriate function primitive. Instead of *transport*, *convert* could have been used.

The following principles can be applied to functional modeling to enhance it.

The **black box** principle (Fig.19 a is a black box representation) is proposed by Rodenacker [RODENACKER 1970] in the context of functional modeling and internationally published by Pahl [PAHL et al. 2006]. With the black box principle, the designer focuses on the incoming and outgoing flows, without looking at what is happening inside. The designer questions the interaction with the environment or rest of the system. He or she has to reason about given or needed flows. The original presentation does not even have a name in the box. However, more recent publications place the name of the main top level function in the box [ROTH 2000]. The opposite of the "black box" is the "white box". It shows the internal structure of a black box. In Figure 19, graphs b, c and d are "white boxes" of the "black box" shown in Fig 19 a.

Functional decomposition is the principle to break down a top level function into lower level functions. The incoming and outgoing flows at the top level can be derived from input-output requirements. During decomposition a higher level function is broken down into a sequence of lower level functions.

To help functional decomposition the following terms are used:

- **Super function (process, behavior)** is used for a high level function, for instance as ordered by the customer. It cannot be realized by one system element or component and has to be decomposed to smaller portions. A super function can be designed deliberately to form a module of some highly connected underlying functions. Some authors [NAGEL et al. 2009] refer to super function as process. In systems engineering, it is referred to as behavior.
- **Sub function** is one lower level function of many to decompose a super function. The terms super function and sub function are used to indicate the hierarchical relation of functions.
- **Elementary function** is a function that cannot be decomposed any more. An elementary function is a candidate for a function primitive that is the smallest unit of function that can be used for building function structures. The term elementary function was coined by KOLLER [1975] as "complete, qualitative description of an elementary operation". Where elementary refers to: "not decomposable into smaller entities".

There are two ways for presenting decompositions. First, the functions can be arranged in a hierarchical tree as a graph. The other way is to model the white box of a function (node, black-box) and present the internal structure of it. In this case the interacting flows of the

super function are the incoming and outgoing flows of the newly depicted white box. The SysML specification provides means for deriving a hierarchy from black-box to white-box decomposition.

The **principle of control volume** is used in many symbolic modeling techniques, e.g. in thermodynamics. This is sometimes referred to as a system border to define the region of interest. When decomposing a top level function and keeping the outline of the box, the control volume is already drawn. In the area of functional modeling, the principle of control volume is hardly discussed in books and in recent publications. However, applying the control volume principle makes it easy to validate flows coming from outside and leaving the system of interest and to draw a balance of them. To denote incoming and outgoing flows of function structures, NIST functional basis [HIRTZ et al. 2002] uses the functions "export" and "import". This is similar to ROTH [2000], who uses sources and sinks to start and end traces of flows. Although the control volume principle is not very common in functional modeling, it is a consequence of black-box white-box decomposition and inevitable when working with SysML.

The **principle of directed edges** is indicating the direction of a flow and is common to the most functional modeling presentations. Still there are some left that do not use directed flows. ROTH [2000] for instance allows explicitly to inverse flows in his function representation. The main direction to read functions based on the schema of ROTH was from left to right. Only when a flow is needed in only one direction to fulfill the system purpose, he does label the flow to be irreversible. The main reason for the non-directedness of flows lays in the strong bias on energy converting functions of Roth. KOLLER [1975] and most of the authors of recent publications indicate the direction of a flow with an arrowhead and allow only connecting functions with conjugate directions, e.g. an energy output of one function with an energy input of another function. The principle of directed flows simplifies checking the consistency of function structures at the cost of more function primitives. Then every function that has been formulated as direction neutral has to be split in a two functions for one direction and the reverse direction. For instance the function "change magnitude" splits up into "increase" and "decrease" when working with directed flows.

The **use of semantic ordering schemas**: Starting with Rodenacker, there were discussions in product design as to what terms should be used for functions and what is an elementary function. ROTH and Koller created different lexicons for functions and flows. They added concise descriptions and graphical representations for each function. Independently from each other they developed design catalogs. Common to both is creating a function structure with the entities described in the lexicons and then search a matrix for appropriate principle solutions. Recent international publications often use the NIST functional basis [HIRTZ et al. 2002], a taxonomy reconciled of the NIST taxonomy [SZYKMAN et al. 1999] and the functional basis [STONE & WOOD 2000]. Compared to Roth and Koller, the terms of the NIST functional basis are defined less strictly, a result of the reconciliation of existing taxonomies.

The advantage of semantic ordering schemas is that the meaning of the functions and their relations are clearer. They allow the processing of the results for the search in design catalogs or automated design methods. There are different kinds of ordering schemas like taxonomy and ontology. Taxonomies relate similar terms in a hierarchy e.g. the taxonomy of flows,

ontology in contrast relate terms in a semantic context e.g. a function transfers an incoming flow into an outgoing flow. The term dictionary is used in this thesis when the taxonomy is augmented by synonyms, descriptions and examples. A function is described by what the operation does and what flows it can accept. Examples and synonyms help to explain functions in a concise way. Some of the dictionaries constrain the flows processed by operations. This can be formulated as the interaction point principle. The use of a model library that mirrors this dictionary is a core contribution of this thesis

The **interaction point principle** specifies the number, directions and types of qualitatively different flows that can interact with an operation. In a further specification, the flows can be divided into main and auxiliary flows. Main flows are mandatory for the operation where as auxiliary flows can be applied to the operation. As an example, consider the function change magnitude, which can be designed as fixed ratio as well as modulated. Having a control signal applied, this would imply that the function is modulated, otherwise fixed ratio. In this thesis, the interaction point principle is used to define the number of different flow types. In some functions the types are restricted to a primary level flow like the function transfer that is limited to energy only.

Some principles can be validated by formal rules and presented by model elements. The graphs in Figure 19 use some of these principles described above. Basically the differences of the presentations lie in the amount and way of encoded information. For modeling elements the information either is encoded in a symbolic way or by text. Some presentations use a mixed form of symbols and written text as function terms or additional labels.

Computer integrated representations are different from that. Functions can be presented in a symbolic way with symbols like in ROTH [2000] or KOLLER [1975]. Yet, the benefit is not in mimicking an existing symbolic presentation but in establishing and maintaining formality and processing the models for analysis or manipulation tasks.

Model languages for systems engineering require formality. It is necessary to avoid inconsistencies while modeling big systems. General purpose modeling languages come with a concrete syntax that contains the rules of how to use the modeling elements. This complies with modeling Layer M2 in the four layer architecture [OMG 2011]. The concrete syntax can be adopted with the definition of additional stereotypes and constraints, i.e. rules for modeling elements. This provides the flexibility to adopt SysML to the tasks of conceptual design modeling when needed. The motivation to realize an integrated chain of modeling in conceptual design has lead to the selection of SysML. The following section explains the diagram types for functional modeling inside SysML.

4.2.2 SysML syntax for function model representation

In section 2.4.3 SysML is presented as a modeling language derived from UML. With the reconciliation of the terms of product design and systems engineering, a functional model is a kind of a behavior model. In this section the SysML specification is analyzed and compared with the modeling principles of the previous section. This results in a justification for using the appropriate diagram type for functional modeling and model library creation.

SysML offers three branches of diagrams named structure diagram, behavior diagram and

requirement diagram (see Fig. 20). In the branch of behavior diagram, the sequence diagram depicts sequences of messages that are passed between blocks. State machine diagrams depict the states and state transitions of modeled entities. Use case diagrams describe the effect of a product for different users. Finally it is the activity diagram that can be used for functional modeling.

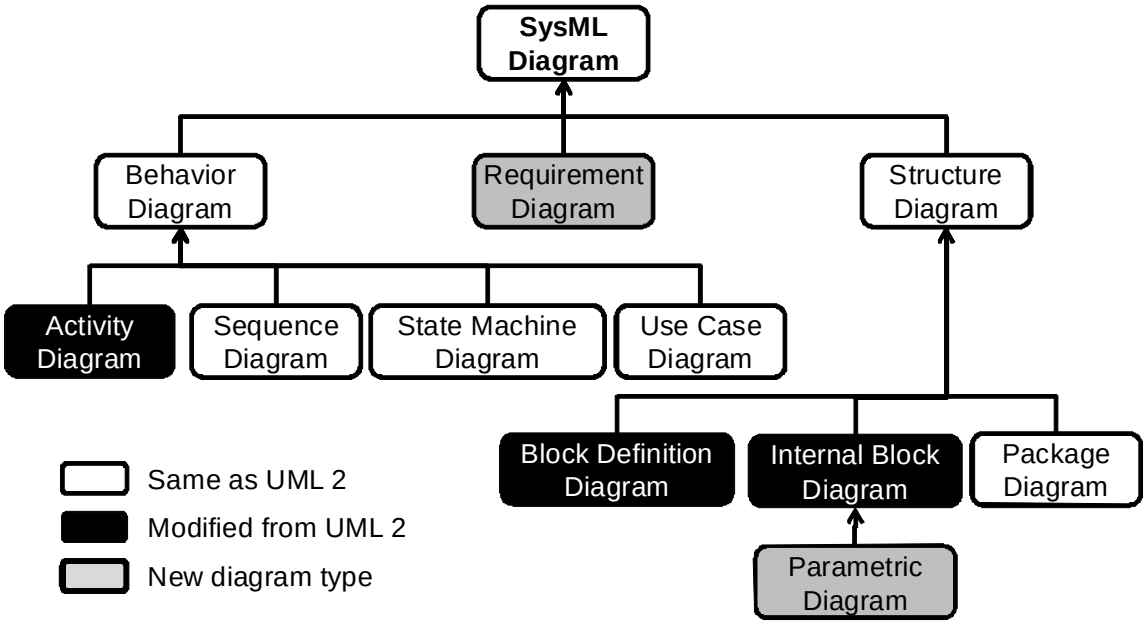

Figure 20: SysML diagram type overview, redrawn from [OMG 2010b]

Considering the SysML specification [OMG 2010b] the following is stated about activity diagrams: *"The activity diagram is used to describe the flow of controls and the flow of inputs and outputs among actions"*. It has been double checked if other diagram forms would be suitable as well. Behavior is often referred to as function structure, so structure diagrams might be a possible presentation for function structures. SysML blocks are described as "*modular units of system description. Each block defines a collection of features to describe a system or other element of interest. The Internal Block Diagram (IBD) captures the internal structure of a block in terms of properties and connecters between properties.*" Principally the concrete syntax of both activity diagram and IBD would hold for a suiting presentation of a function structure. The SysML specification, however, describes blocks and block diagram as part of the system. A function is the abstract qualitative description of what output flow a system or a part of it has to provide at a given input flow. This matches with activity diagrams. SysML is designed to meet the requirements of ISO 10303, Application Protocol 233 Systems Engineering [ISO 10303-233 2009]. Thus, the definition of function and behavior from Section 2.3.3 holds.

An example of an object flow orientated function structure is also demonstrated by FRIEDENTHAL et al. [2008] with the water distiller example in Chapter 15 of the cited textbook. They model functions in the style of enhanced functional flow block diagram

(EFFBD) and reach a result that is in its appearance similar to the example presented in Figure 16.

4.2.3 SysML specification for functional modeling

Now that the diagram type for functional modeling is identified, the relevant content of the specification for that diagram type is investigated. Since SysML activity diagrams are modified from UML activity diagrams, the UML superstructure [OMG 2010c] is the relevant specification for activity modeling. It specifies: *"Activity modeling emphasizes the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because events occur external to the flow."*

The object flow oriented modeling paradigm is applied. This means that a flow that is passed over to the next function triggers the execution of that function. For functional modeling as presented in Figure 16, control flow is not relevant. Therefore, only the elements for object flow modeling of the concrete syntax of activity diagrams are needed. SysML offers different syntax level for activity modeling. The modeling of function structures corresponds to the level of complete structured activities of UML. The following statements in italics are cited form the UML superstructure since SysML partially reuses UML activities:

"An **activity** represents a behavior that is composed of individual elements that are actions."[OMG 2010c] For modeling of function structures, behavior corresponds to a super function. Consequently activity diagram is the appropriate means to depict a function structure. A function is represented by a SysML action.

"An **action** represents a single step within an activity, that is, one that is not further decomposed within the activity. However, a **call behavior action** may reference an activity definition, in which case the execution of the call action involves the execution of the referenced activity". [OMG 2010c] Since an action is not a packable element, it cannot be stored in a model library. The mechanism of the call behavior action allows actions referring to a predefined activity. It is the function description stored as an activity definition in a model library. Whenever that action is activated by an incoming flow it references the activity in the model library.

"**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." [OMG 2010c] For the library element they correspond to the definition of flows either entering a function or leaving a function. For function structures they define flows crossing the control volume. Activity parameter nodes have a defined type and direction to represent input and output parameters. They are realizing the interaction point principle.

*"A **pin** is a typed element and multiplicity element that provides values to actions and accepts result values from them."* [OMG 2010c] Pins specify the flow that is interacting with the function in a function structure. Flow type and direction are specified with pins. What is presented as an input pin on a call behavior action corresponds to an input activity parameter node at the invoked action. This means that activity parameters for the activities in the model

library have to be defined to generate pins during the use of an action as function (call behavior action) in a function structure.

*"An **object flow** is an activity edge that can have objects or data passing along it*." [OMG 2010c] The object flow is used to represent the flow in the sense of functional modeling.

Some constraints originated from the UML superstructure or the SysML specifications should be mentioned here:

- *Object flows have to be the same type at upper and lower ends*. [OMG 2010b]This complies with the fact that the flow cannot have different objects (flows in the sense of functional modeling) at its ends. An object (flow) only can be changed by a function
- The direction of the flow has to be conjugate at the ends. [OMG 2010b] That allows only connecting an output pin of the upper end action with the input pin of a lower end activity.
- *"Object flows may not have actions at either end".* [OMG 2010b] This enforces the use of pins on actions or the use of Object nodes to describe the flow in the sense of functional modeling.

In the description of the semantics of object flows there is a stated fact about multiple edges leaving from an object node or activity pin: "*Two object flows may have the same object node as source. In this case the edges will compete for objects. Once an edge takes an object from an object node, the other edges do not have access to it".[OMG 2010c]* This is an important fact for functional modeling since it is not possible to duplicate a material or an energy flow. For a continuous flow, this means that, given a pin with two leaving object flows, a portion of the flow will go to one edge and the leftovers of it to the other edge. If flows consist of discrete pieces, the single piece can follow either one or the other edge. The use of decision nodes or fork nodes is possible within SysML but does not make sense to use with the proposed way of functional modeling.

SysML nodes are identified by name. The name is unique within one namespace, meaning there can be only one modeling element with the same name within one package. One element can be used several times with different roles. As an example, a block can be defined for a wheel, which has the role of a front or rear wheel. To break down a system, the concept of packages is used to divide the model into different namespaces. This implies that there cannot be a function with the same name and different meanings but one function can be used in different contexts. If specific functions are necessary they have to be renamed.

4.2.4 Presentation of a function structure in SysML

The specification is implemented in the language description of a CASE tool. All points presented above can be found as concrete syntax (modeling elements) for depicting diagrams or evoke a certain behavior of the tool during modeling. The diagram in Figure 21 shows an example of a diagram created with the CASE tool Magic Draw. The same diagrams are used as for the modeling example to explain the modeling elements of SysML.

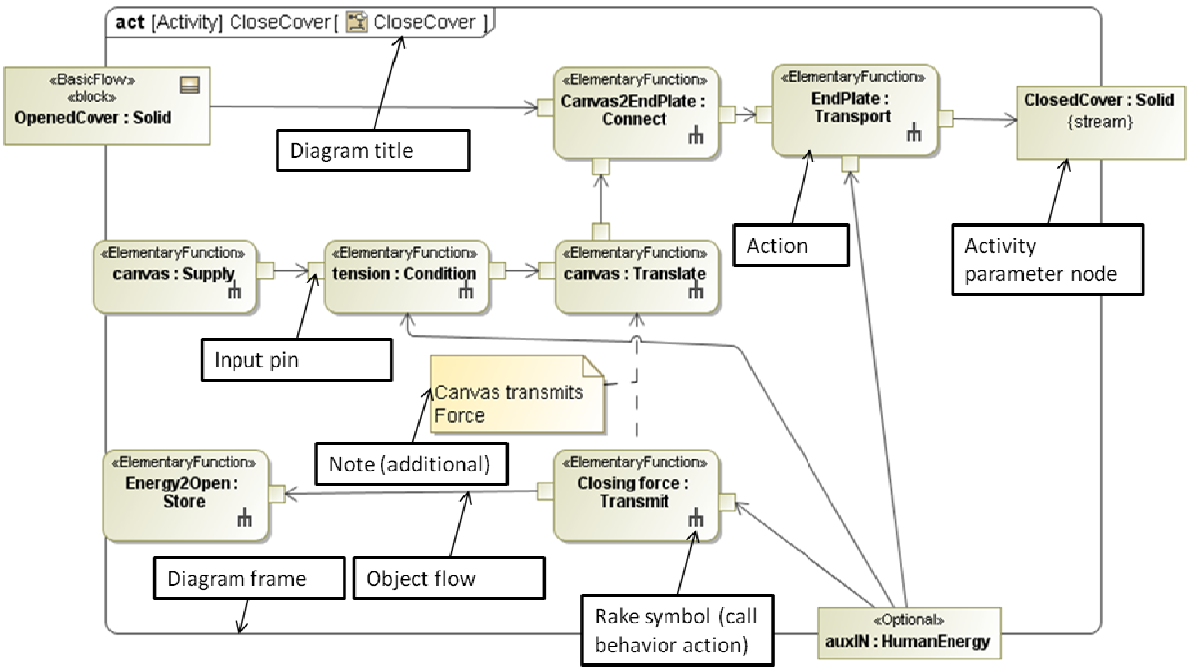

Figure 21: Presentation of a function structure as an activity diagram, example: close cover

At the left corner of the diagram, the diagram title is placed. It is composed of the SysML diagram type [Activity], the diagram name “CloseCover” and the diagram context “[CloseCover]”. The shaded rectangles at the borders of the diagram frame present the activity parameter nodes as interaction points for the flows crossing the control volume of the function structure. The type of the activity parameter node can be any block. If not specified in advance, an element has to be specified during modeling. In the case of the example in Figure 21, the types of the activity parameter nodes are predefined blocks from the model library representing flows (See Appendix, Table 9.1).

A function represented by an action is shown a shaded rounded rectangle. The stereotype is <<ElementaryFunction>> to identify it as an element for functional modeling. The rake symbol at the left lower corner of action nodes indicates that it is a call behavior action. The colon notation means “action name: referenced behavior”. In the case of the example the action references an activity in the model library for functions. Pins are used to define the direction of object flows that are interacting with actions. They are typed elements. The type of a pin can be a SysML block or a UML class.

In the diagram, a note is shown. Notes are comments that do not have a meaning to the computer internal representation but can be used for explaining the diagram for the viewer. In Figure 21, a note to explain the dependency of the transmittal of the force and the translation of the canvas is used. The force is needed to load the spring to retract the canvass into the cassette when the cover is opened.

The comparison of the principles for functional modeling to SysML activity modeling shows

that SysML provides an appropriate concrete syntax to support functional modeling. Table 4-2 summarizes and compares functional modeling, modeling principles and appropriate SysML presentation.

Table 4-2: Comparison of needed principles and given SysML presentation

functional modeling	modeling principle	SysML presentation
function structure	network type graph	activity diagram
function	node	action
flow	edge	object flow
Differentiated flows for material, energy and signal	typed edge visually: (double, plain and dashed)	flow type
decomposition	sub graph decomposition	call behavior action / referenced activity diagram
black box	Black box principle	single action with pins
predefined flow	interaction point	parameter nodes or pins
scope of function structure	control volume	diagram frame
flow direction	directed edge	direction property of parameter and pin

The SysML presentation covers all points that are identified as modeling principles. This justifies the use of activity diagrams for functional modeling. To optimally support functional modeling and produce comparable function structures, a modeling library is also needed to define a fixed taxonomy of functions and flows.

4.3 Set up of a model library for functional modeling

The way of modeling functions is now clarified. Next, the content of a model library is presented. Several sets of function primitives are presented in Chapter 2. The NIST-FB has been selected as a candidate for transformation into a SysML-based model library. The objective is to use function structures for the building of a domain independent component structure. Other function primitives provide limited coverage of one or two domains and are intended for the mapping with working principles, which is not considered here.

The purpose of a model library is to organize and provide model elements for reuse. The modeler’s questions address two aspects: What is the information content for reuse? How can it be modeled in the library?

The first question is targeted at identifying the static content of information that the modeler uses to dynamically create models. In terms of graphs, only different types of nodes or relations can be used to model the library contents. For functional modeling, this means that only elementary functions or function primitives and associated flows are candidates for a model library. The function structure is the dynamically created model supported by the library elements. If there is problem specific information that changes often it does not make

great sense to store it in a library. Then it becomes a challenge to find a balance between the effort to create the model library and the benefits of using a model library, i.e. its reuse. In addition to effort and modeling speed, or usability, there is a strong reason for a modeling library for functional modeling in the creation of comparable function structures.

The scope of this work is a general purpose model library for function primitives in product design and systems engineering. It can serve as a basis for modeling general and specific functions using SysML

4.3.1 Information content of NIST Functional Basis (NIST-FB)

The NIST-FB is widely known in the research community and now is reviewed and cited in research papers. The methodical set up of the taxonomies assures that the terms for functions and flows are commonly accepted within a certain domain, organization or project. The aim is to determine how the pieces of information can be transformed from this NIST-FB to a SysML model library. At some points, the modeling language requires more formality than the paper-based NIST-FB. Therefore, it has to be verified how the NIST-FB can be tailored to fit with functional modeling with SysML.

The following gives an overview of the information content of the NIST functional basis:

- Function: three level hierarchy of terms
- Function description: describes what the function does to an incoming or outgoing flow. At the class level there is an explicit listing of what types of flow the function processes. This listing is not consequently applied in the secondary and class level functions. Examples at the secondary and tertiary level show in what context the function is used. The number of necessary interaction points can be inferred from the description, but is not explicitly given.
- Flows: three level hierarchy of terms
- Flow description: describes the flow with a small example
- Power conjugate complements (only with energy): describe the flow variable and effort variable of a technical flow

There are at least two different possibilities to set up the library. The NIST-FB is a dictionary that comes with explanations and synonyms. Flows and functions are two different taxonomies. Thus, they can be modeled in two separate model libraries. NIST functional basis presents the taxonomies in the same way. Flows are arranged in a hierarchy (inheritance, is a) and they are needed to express the function together with the function itself.

Functions have to be assembled with the appropriate flow type when modeling a function structure. The overall number of library elements is limited to 45 flow definitions and 52 operation definitions at three hierarchy levels. This makes it straight forward to update the library, delete elements and maintain consistency.

The other way is to multiply functions by flows. This makes sense when the meaning of a

function changes depending on the involved flows. With a high number of functions and flows the number of entries in the model library grows. Consequently updating of the library gets difficult. For instance, if a flow changes that is used in a certain number of functions, all the functions using that flows have to be changed also. With separate taxonomies this is not the case.

4.3.2 Transformation of NIST-FB flows in the model library

For the implementation of the two morphemes of the functional basis there are several steps to take. The first step is the definition of the right sequence of modeling the library elements. The modeling language requires that every element that is used has to be defined upfront. A function is represented as an activity with activity parameter nodes. The types of the activity parameter nodes are flows. Therefore, the flow taxonomy has to be transferred to SysML before the function taxonomy

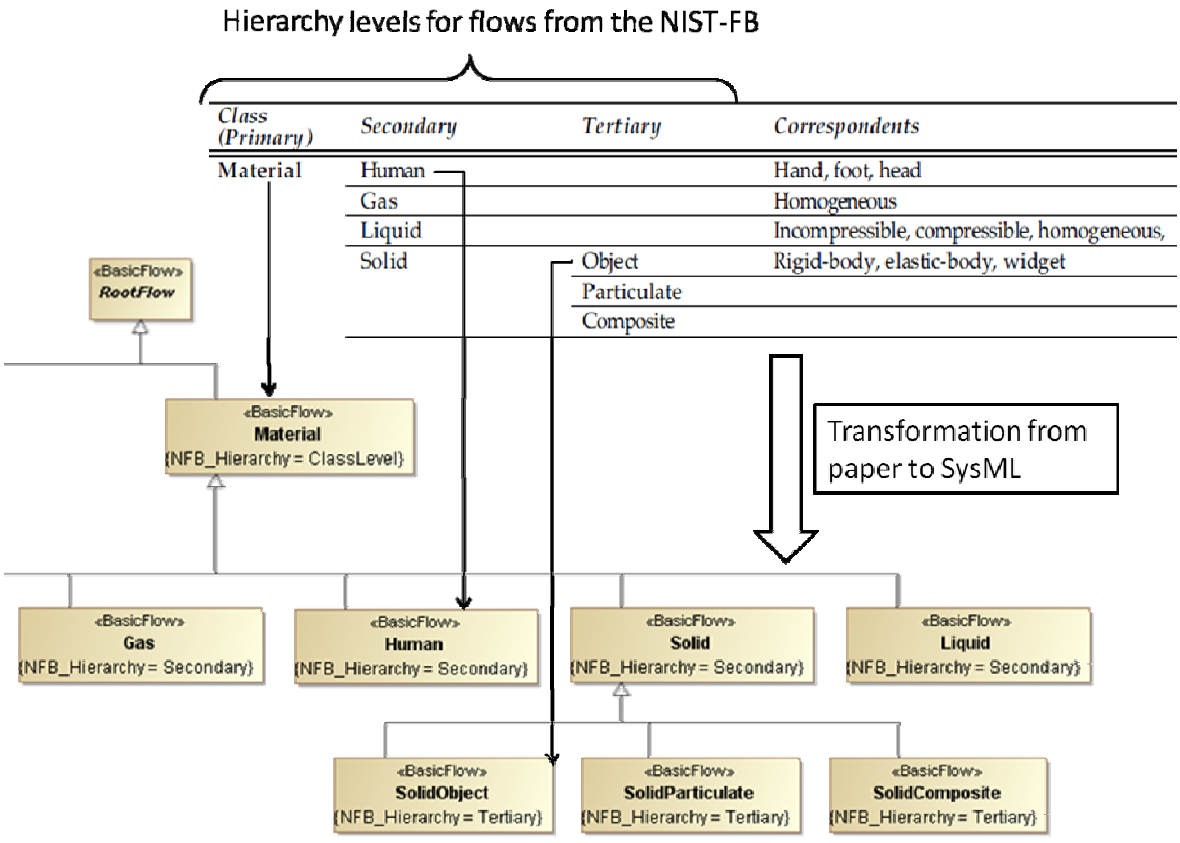

Figure 22: Transformation of NIST FB-flows [HIRTZ et al. 2002] to SysML model library; portion of the library

In Figure 22, a part of the table of flows from the NIST technical report [HIRTZ et al. 2002] can be seen. Underneath, the corresponding part of the model library is shown as a package diagram. Corresponding to every flow entry in the table, an element stereotyped

<<BasicFlow>> is created in the SysML model library package. This stereotype is a specialization of a SysML block that carries an additional tagged value to indicate the hierarchy level of the flow in the NIST-FB. In the model library the elements are arranged as hierarchical tree whose levels mirror the hierarchy levels indicated in the source document. In the tree, the only kinds of relations are generalizations. It reads “Material” is the generalization of “Solid”. The arrow head points to the general classifier.

In the model library, there is one additional element called “*RootFlow*”. The NIST FB only provides three hierarchy levels; however, an additional level is needed. The first reason is to keep together all flows. Without this mechanism, the model library of flows contained three hierarchical trees rooting in “*Material*”, “*Energy*” and “*Signal*”. The other reason comes from the definition of functions. A lot of functions work with any of the three flows. Therefore a place holder is needed for a subsequent specification of the flows of functions.

4.3.3 Definition of functions for the model library

The transformation of the functions to a model library is similar to the transformation of flows. However, there are some differences. The model element is not a block but a stereotype called <<ElementaryFunction>> extended from the UML metaclass "activity". The activities are arranged as a hierarchy structure with the same four levels starting from the single “RootOperation” node down to tertiary level. The specific part begins with the definition of activity parameter nodes for every activity. These are necessary to create corresponding pins when actions are created from the library elements.

For the creation of activity parameter nodes, information is needed about number of flows, flow directions and flow types. The NIST-FB does not explicitly provide this information. Implicitly the information is given in the function description. In Figure 23, an example of how to set up the specifications for functions as SysML activities is presented. The table on the top of Fig. 23 is a cutout of the hierarchy table given in the NIST technical report [HIRTZ et al. 2002]. The text below is the description with examples taken from the same document. “*Transfer*” is a function where the flow is shifted from one place to another, thus the flow at input and output are of the same type. All three flows at class level can be applied. This results in setting the type <<RootFlow>> from the flow library. Additional to these two flows, an auxiliary flow is needed. The auxiliary flow is typed with “*Energy*” and is needed for the transfer of material due to Newton’s first and second law. The “Base Classifier”-value of “*Transfer*” indicates the function “*Channel*” as the generalization of “*Transfer*”.

The flows for “*Transmit*” (*energy*) and “*Transport*” (*material*) are taken from the description in the same manner and transferred into the specifications of corresponding SysML activities. The example in Figure 23 shows how implicit information from the NIST-FB is converted into explicit information in the model library.

Primary	Secondary	Tertiary	Correspondents
Channel	Import		Form entrance, *allow*, input, *capture*
	Export		Dispose, eject, *emit*, empty, *remove*, destroy, eliminate
	Transfer		Carry, deliver
		Transport	Advance, lift, move
		Transmit	Conduct, convey

c) **Transfer.** To shift, or convey, a flow (material, energy, signal) from one place to another.
i) **Transport.** To move a material from one place to another. Example: A coffee maker *transports liquid (water) from its reservoir through its heating chamber and* then to the filter basket.
ii) **Transmit.** To move an energy from one place to another. Example: In a hand held power sander, the housing of the sander *transmits human force to the object being* sanded.

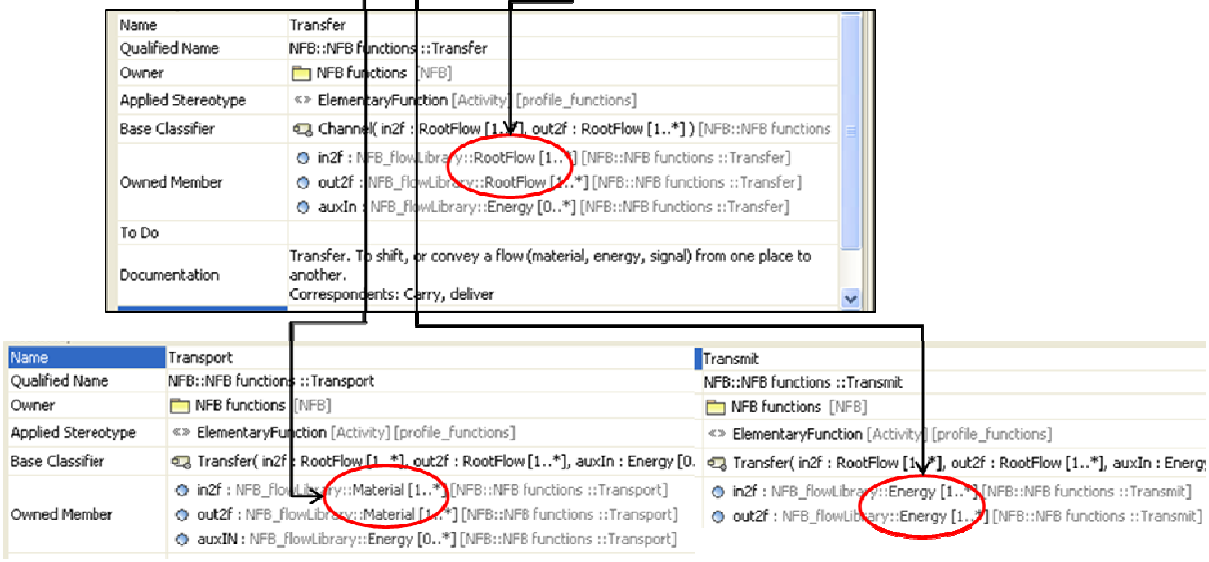

Figure 23: Definition of functions for the model library according to the NIST technical report [HIRTZ et al. 2002]

The creation of library elements for flows and functions in the model library is now given. The library now is ready for use.

4.4 Implications of model library for functions

During the implementation of the model libraries some incidents connected to either the NIST-FB or the SysML have been resolved.

The flow model library is the first part of the library that is transferred to the CASE tool. It results in a set of 45 SysML blocks stereotyped as <<BasicFlow>>. The term "*Human*" is used twice in the flow definition. Once as a *material* flow at the secondary level and a second time at secondary level as *energy* flow. Within one package, SysML does not allow to use a name twice. Therefore, all energy flows at the secondary level have been extended with "*Energy*". For example, "Human" in the model library is written as "*HumanEnergy*". Compound words with uppercase are called camel case words since the uppercase letters

remind one of the humps of a camel. Writing in camel case is common in some programming languages and CASE tools. An overview of all the flows brought to SysML can be found in the Appendix, Table 9-1. The ambiguity inside the NIST-FB extends to the function taxonomy. "*Signal*" is a flow type and a function at class level. Besides, "*Signal*" is a SysML metaclass. To eliminate this ambiguity "*SignalFlow*" and "*SignalOperation*" are used instead as names of the model elements.

The function library is a package with 52 activities, each of them having some activity parameters. The names of the activities are derived from the original NIST-FB. Information about what flows an operation can take is not explicitly given there. This information has been inferred from the description of the functions. Since the descriptions are only short and often do not disambiguate the three class level flows, it is difficult to figure out what flows are needed. Only some function descriptions come with a clear statement that a certain flow is needed. For instance, to "*transmit*" means to "*transfer"* energy from one place to another. Most functions are described as applicable to *material*, *energy* and *signal*. A special case is "*Provision*", it is the base classifier for all functions that provide or store material and energy. To manage this, an additional block is introduced named "*MaterialEnergy*" that generalizes both energy and material flow. In total 115 object ports are added to all 52 function definitions. The complete information transferred from the NIST-FB to the function model library is presented in Appendix, Tables 9-2 and 9-3.

During the creation of the library it has been discovered that the third hierarchy level seldom provides additional constraints or information. Having a hierarchy, it is expected that the elements of lower level inherit form the upper level elements. At the same time, lower level elements are more specific than upper level elements. In Figure 24 some deviations from a strict inheritance are visible. First, the number of outgoing flows of both *"Separate"* and *"Distribute"* is different. They inherit from "*Branch".* Thus, it is expected that they have the same number of pins but more specific ones. But they have a different number of pins and are not more specific. That *"Distribute"* has one outgoing pin is justified by the description that the output flows are similar to each other. For modeling, this will be resolved by multiple object flows from one pin. In contrast, the flows of "Separate" and its' children are different since the output flows are distinguishable. Therefore, more (at least two) pins are needed. Although distributed over three hierarchy levels, five functions shown in Figure 24 have the same incoming and outgoing pins. Further, the tertiary level does not bring any additional information or constraint on the flows, e.g. that "*Remove"* is only applicable with material flow as form the "correspondents" of the NIST-FB can be inferred. If the search for components is based on the information contained in the pins (type and direction) it is not possible to differentiate the five functions unless they are specified in context of a function structure. Consequently, the tertiary level is not necessary, since it does not add additional information in terms of differentiation of pins.

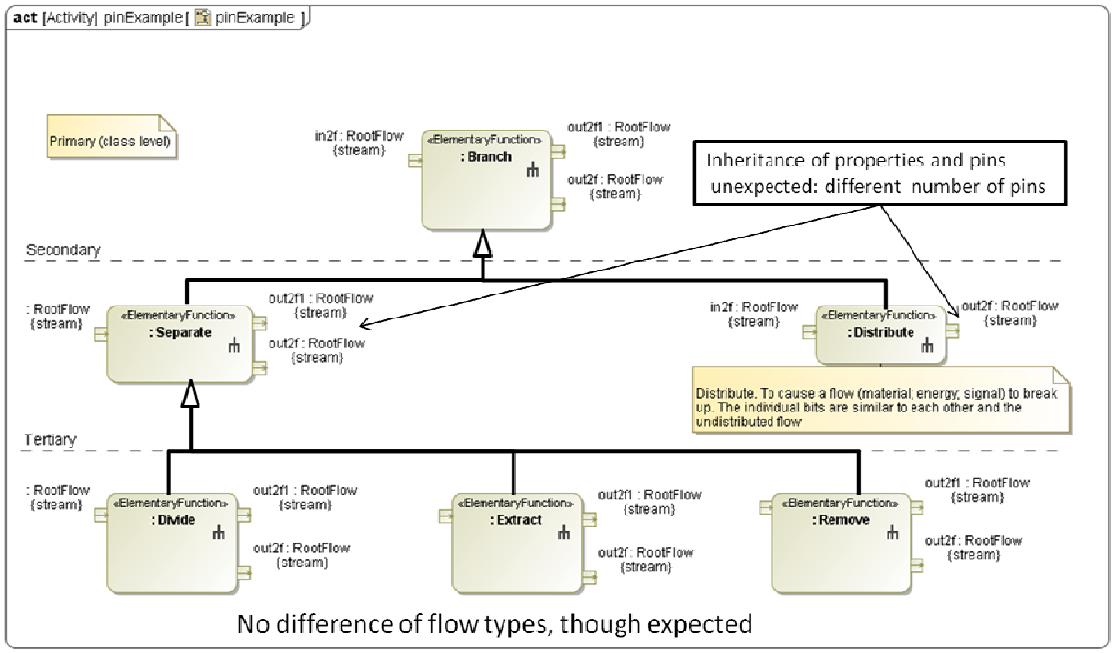

Figure 24: Hierarchy levels of functions vs. details of pin definitions

The example above shows that the tertiary level is of little use for conceptual design. A similar finding was made in context of automated design synthesis where the researchers stated that the secondary level is sufficient for synthesis [KURTOGLU et al. 2009].

Some functions of the NIST-FB are not elementary. An elementary function could not be expressed by a sequence of sub functions. Take the example of the function "*Link*", where two flows are connected with an intermediary flow. Connecting an eraser by a metal sleeve with the pencil shaft is an example for the function "*Link*". This can be expressed by two or more "couple" functions, where a flow is first coupled with the intermediary one and then the resulting flow is coupled with the other flow. It can be replaced by coupling a metal sleeve with the pencil shaft and joining the eraser with the prepared pencil shaft. If the conceptual task is to bring together the two flows, it would be sufficient to use an eraser with a bore that directly couples with the pencil shaft.

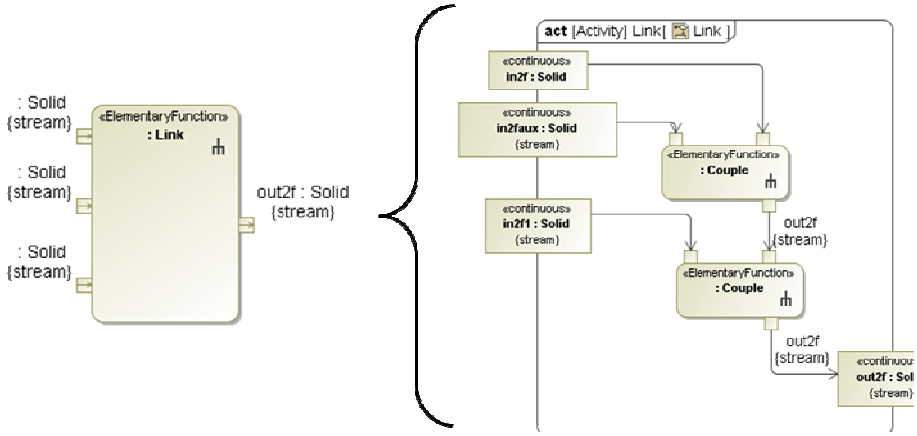

Figure 25: Decomposition of the function "Link"

Figure 25 shows how the function "*Link*" can be decomposed into two "*Couple*" functions. This shows that "*Link*" is not an elementary function.

The class level function "*Convert*" is the only function that does not have any children at the secondary or tertiary level. It can be applied with any kind of flow. The difficulty here is to find out the number of necessary interaction points. To convert a type of energy it needs one type of input and one type of output pin, e.g., to convert mechanical energy to electrical energy two types are needed. The conversion of material is more difficult. To convert water into steam, thermal energy is needed. Therefore, three different pin types are required. Consequently the operation convert comes with three different pin types. The direction of the third interaction point is not clear, however. To convert the steam back to water, it has to be condensed. This means energy is taken from the steam. Therefore, the direction of convert material function's auxiliary object port cannot be determined. The mapping of a function to a physical property of water, namely to evaporate or condense is an example for the difficulty to determine flows for functions. This is especially true for functions that connect energy flows to the material flows. It reveals that the function "*Convert*" needed some specialization to at least separate energy conversion from material conversion. Most of the functions do not need any auxiliary flow when the main flow is energy, otherwise need an auxiliary flow when the main flow is material, e.g. *"transport material"* needs an auxiliary energy and *"transmit energy"* does not.

4.5 Modeling of the product structure in conceptual design

A product or system is composed of many interrelated elements to fulfill the product function. The component structure shows how the single components are related. It is created from the component models that represent the embodiments of the functions given in the function structure. In this section the concepts of modeling for the component structure and the matching of the functions' incoming and outgoing flows to input and output ports is presented.

4.5.1 Definition of conceptual structure modeling

In the concept stage, the objective is it to build up a first topology of the product or system. For this reason, component models need to be simple but provide enough information for the model evaluation and provide modeling concepts that allow model evolvement. In the first instance, a generic component serves this purpose. In the concept phase we search for a variety of solutions not only for one. Therefore, conceptual component models at this stage should have a high granularity and show only the information needed for conceptual design.

The granularity of the structure can be defined using the system complexity level hierarchy. The relevant levels of complexity as defined in Table 2-1 are related in the concept meta-model given in Figure 26 on the left hand side. It shows the composition of a product down to a part as a SysML BDD.

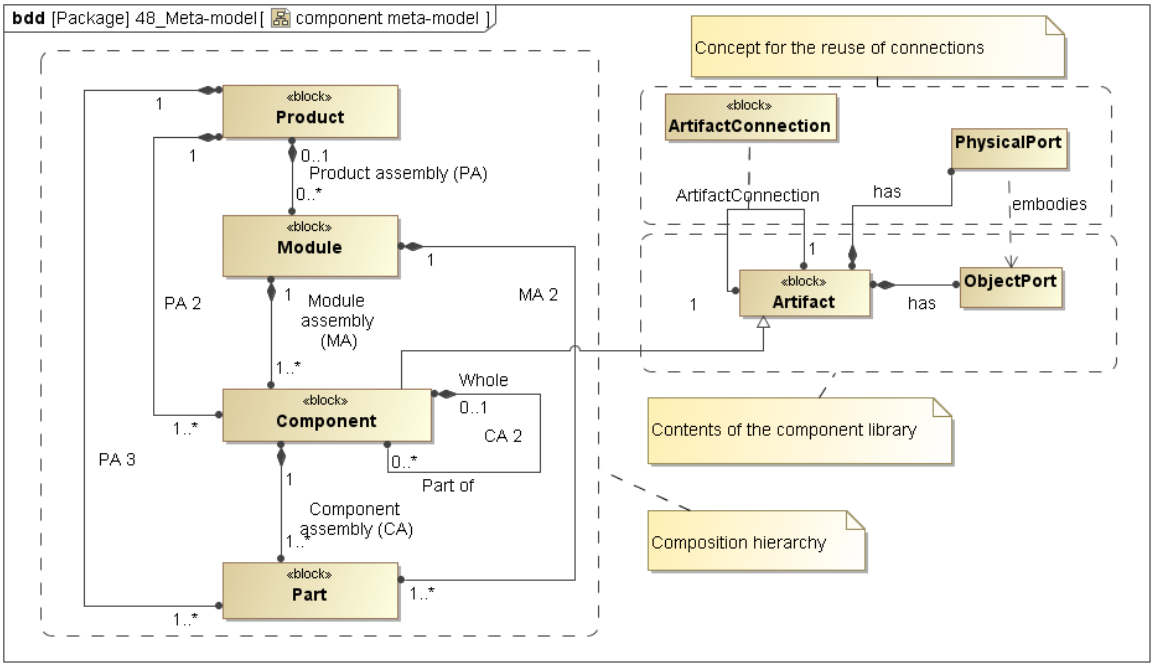

Figure 26: The component metamodel for system element hierarchy, components library and connection concept

The relations denote compositions, where the node at the upper end, marked with a black diamond, is composed or assembled from nodes of the lower end. The multiplicity of the relations is shown at the ends of the relations, e.g. “Component assembly” linking “Component” to “Part” with “1” at the upper end and “1..*” at the lower end. The meaning is that exactly one “Component” is composed of one or many “Parts”. The second composition with “Component” at the upper end is “CA 2”. It relates “Component” to itself. Consequently any “Component” can be composed of “Parts” or other “Components”. In this thesis, the embodiments for functions are modeled at component level. The part level is used for the definition of some connectors in IDBs. A component structure forms either a module or a product, depending on the size and structure of the product. Below “Part”-level there are only features like surfaces, holes or threaded holes, these are considered as too detailed and left out

for modeling. In this thesis, the hierarchy is used to define the level of detail of the modeled diagrams.

4.5.1.1 Definition of the component model as library element

On the lower right hand side of Figure 26 the contents of component library is shown. The "Component" has a generalization link to "Artifact". The latter has the two kinds of ports, named "ObjectPort" and "PhysicalPort" the meaning is, that any Artifact has object ports and physical ports. "Component" inherits form "Artifact"; thus, components inherit the ports from "Artifact". For the identification of components of the model library only the object port is relevant. The physical port is necessary for the assembly of elements. Therefore it is part of the concept for the reuse of connections.

When searching for a device that decreases rotational speed, the interest is to indentify components off the shelf (COTS) that can do this with the desired ratio. From function the point of view, it is indifferent if a gear pair, a worm gear pair or a belt drive is used for it. In this context the load and ratio of the transmission device may lead to an exclusion of a certain component, e.g. the worm gear pair for its ratio is out of requirement's range. Other requirements are the amount of energy that has to be transmitted and the final speed that has to be reached. Therefore, in the concept stage, modeling of requirements is essential to narrow down the selection of components from identified solution candidates. The focus is on the selection of abstract parts or solution candidates. Solution candidates are components that embody the function. The final selected component is called the solution element.

Components have various properties. To name some they are weight, geometry, used material, physical properties, interfaces, supplier information, cost and performance. To identify a solution candidate out of the model library it is important to define the object ports for library elements. They often can be inferred from attributes specifying the component. For instance, a DC electrical motor has a rated voltage and current on the electrical side and rated torque and angular velocity on the mechanical terminal. From that the object ports for **electrical energy** and **rotational mechanical energy** can be derived. For energy, the power conjugate complements indicate the object flow of the component. Material flows can be identified by the kind of interface, e.g. fittings for liquids and gaseous materials, and signal output by its value specification, e.g. $mV/\Delta T$ for the temperature-voltage relation of a thermostat.

For an abstract component model, the object ports are sufficient but reality is different. The object port of electrical energy is an abstraction of the cable connected to the electrical motor. The cable itself consists of two wires. Both have to be connected to bring electric current to the armature and make the DC electric motor turn. The object port can be mapped to physical ports (plus terminal, minus terminal) needed to make the component work. In the same way, the object port for rotational energy can be mapped to a physical interface. One physical port is the shaft or pinion. The other is the housing. The rotational movement of the shaft is relative to the housing. Without fixing the housing, it could move by reaction forces. In this example two object ports are mapped to four ports at part level. However this does not hold for every component. There are components with just two physical ports, e.g. Cardan shafts or piston rods, or three ports e.g. gear boxes having an input shaft, an output shaft and housing.

A similar concept is described in KURTOGLU et al [2009]. He defines **object ports** containing flows. **Medium ports** are ports to enable the medium of the flows to enter into the device, e.g. hydraulic fluid entering a hydraulic cylinder. Finally, **assembly ports** describe the condition of assembly. Medium and assembly ports can be realized through the same concepts, e.g. a screw fit used for a flange (medium port) or the assembly of a cover (assembly port). Therefore, the differentiation into assembly ports and medium ports is considered as not necessary.

For the modeling of components it is important to distinguish between object ports that contain flows, like functions, and physical ports that embody object ports. Object ports are used to identify the component out of the catalog and physical ports are used to connect components.

4.5.1.2 Concept for the reuse of connections

In Figure 26 right hand side on top, the physical port is presented together with the association block "ArtifactConnection". Association blocks allow specifying and decomposing a connection. It is described and exemplified in the SysML specification [OMG 2010b]. Considering that some of the object ports have to be mapped to a different number of physical ports and that the connections have to be specified in detail for the evaluation and the embodiment design, this concept has been selected. It allows the adoption and reuse independently form the component library. In general, an association block refers to an association that connects two SysML blocks. The details of the association can be defined in an internal block diagram showing the inner structure of the association .

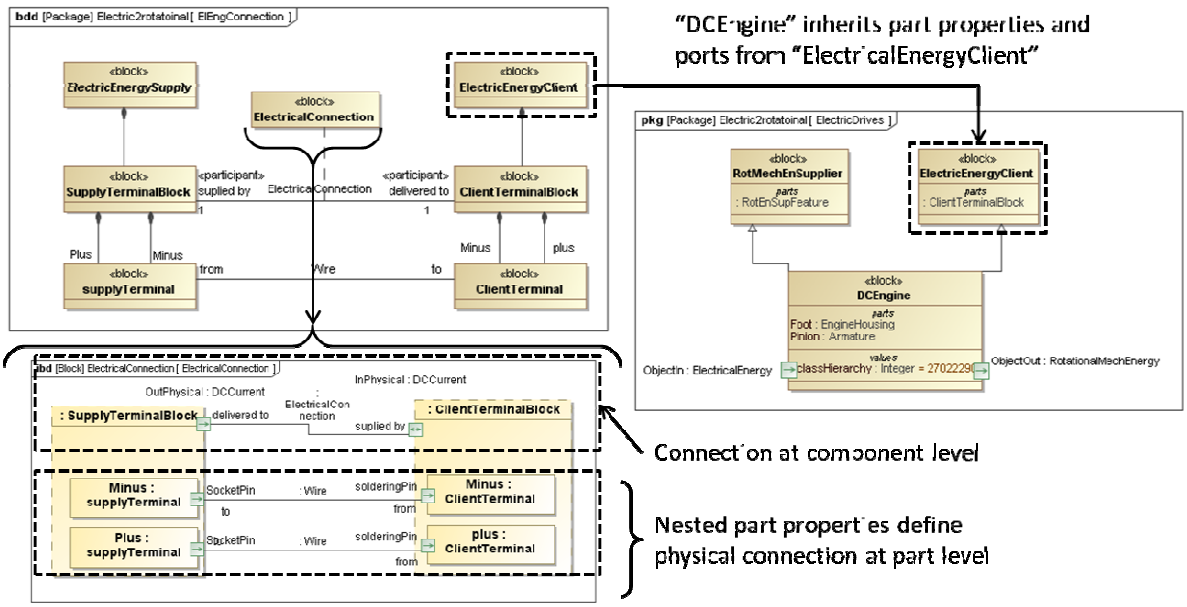

Figure 27: Definition of a detailed component connection using an association block

In Figure 27 left hand side upper, the association block named “ElectricalConnection” connects the blocks “SupplyTerminalBlock” and “ClientTerminalBlock”. Both blocks are compositions of a super ordinate block representing a proxy, i.e. “ElectricalEnergyClient” for a device that needs electrical energy. The associated block itself is at component level and is composed of its parts “ClientTerminal” that have the roles “Minus” and “Plus”. In the IDB (Fig. 27 left hand side lower) of the association block the associated blocks are shown with a dashed line, i.e. a participant property. The part level is shown as nested part properties inside the participant property. The connecters of the IDB are typed with the associations in the BBD. With the definition of the inner structure of the association block, a connection is defined. To use this predefined connection a component has to inherit from the proxy. In Figure 27 right hand side this is exemplified for a DC electric engine that specializes from both “ElectricalEnergyClient” and “RotMechEnSupplier”, shown with the generalization links (white arrowhead). Consequently all properties, i.e. inner structure and physical ports are available in the component model for “DCEngine”.

For conceptual design it is sufficient to define the component level, e.g. a terminal block, a plug or a socket. If it is necessary to verify the connectivity of two elements it is also important to know the number and type of different terminals contained in the terminal block. Therefore, the part level is shown as nested part properties in the IBD in Fig. 27. More details, e.g. the shape of the single contacts of a socket, are too specific for conceptual design but belong to embodiment design, where geometric details are defined.

Now that the model elements for the components are defined, the next section presents information sources for components.

4.5.2 Identification and selection of component information sources

Since more than a century, selecting components from catalogs is common in engineering. In engineering design there are catalogs for machine elements and other components. In mechanical engineering it is common to work with component catalogs like Roloff-Matek. Here components are presented in a structured way including suppliers and the interval of values to critical properties e.g. supplier GKN/Walterscheid can provide shafts with length compensation for a maximum of 1000rpm-2000rpm, sustaining torsion loads from 9Nm to 10600Nm. This specification is incomplete, but the catalog indicates that there is a supplier for shafts to transmit rotational energy. The Roloff-Matek Catalog itself makes use of E-Cl@ss, the world fastest growing consistent standard for classification and description of COTS [GRÜNDER 2009]. E-Cl@ss fits to communication and data exchange for E-commerce.

For a model library, the objective is to obtain a wide overview about solution candidates. Therefore, it is decided to rely on product and service classification standards (PSCS). They provide a comprehensive overview about available solutions on the market. Once a solution candidate is identified, the component is one possible solution that has to be integrated in the component structure. Then, the next step is to find suppliers that can deliver suitable solution elements as well as additional information, e.g. simulation or CAD models.

PSCS provide taxonomies of products and services and is fueled by electronic procurement rather than by design synthesis. This means that terms are not used exclusively; auxiliary

agents like lubricants are included as well as complete production machinery. Many attributes are defined, but only some are of interest for design synthesis. Information about object ports or physical ports is missing. Some ports can be derived from performance related attributes but this is not generally given.

In order to use this as an information source for solution candidates, certain filters on such classifications have to be applied. First, all services have to be sorted, then some of the machine elements that cannot realize a function by themselves, like washers, bolts, nuts and nails, have to be removed.

E-Cl@ss is used an example for how to transform a PSCS to a library of solution candidates. It is structured in a four-level hierarchy and contains more than 30 000 entities. The levels are called **segments, main groups, groups and commodity classes** (see Figure 28). At the segments level the focus is on segment 23- "machine elements" and segment 27 – "electric engineering process and control engineering". These two segments provide a great foundation for a model library relevant to mechatronic systems. Segment 23 contains 37 main groups, 257 groups and 1661 commodity classes. Segments group all entities that are needed mainly in a certain business branch, e.g. segment 23 addresses mainly mechanical engineering and segment 27 addresses electrical engineering. Main groups form families e.g. shafts, are further detailed into groups, e.g. Cardan shafts, shaft-hub connections etc. The procurable artifact finally is defined as an instance of **commodity class**. Below commodity class, the properties and sets of properties are defined. Any supplier can use that structure and set the corresponding values to the properties for every item she or he offers in his/her vendor's catalog.

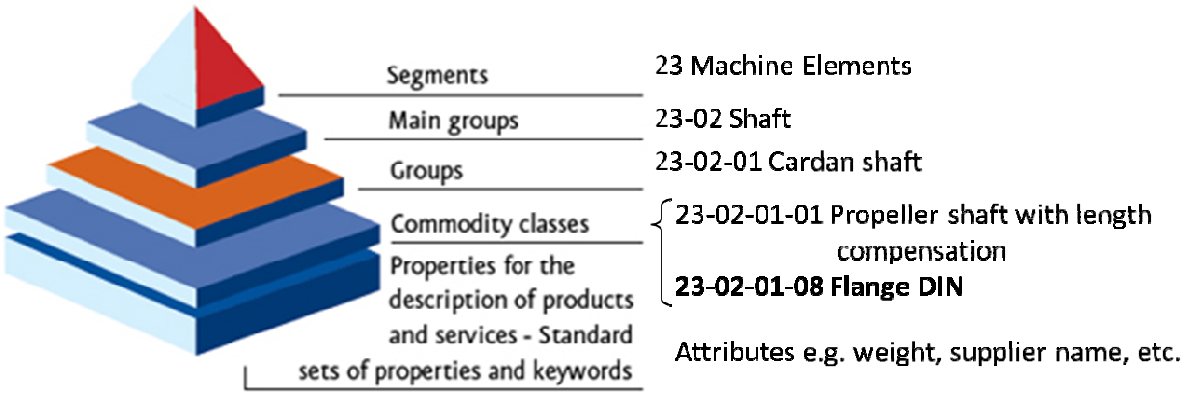

Figure 28: Structure of e-Class with examples on the left side adopted from [KOZIEL et al. 2006]

Figure 28 exemplifies the structure of E-Cl@ss with two items at commodity class levels. The propeller shaft is at component level. It is an item that can embody the function "transmit rotational energy". An example for an item at part level is "Flange DIN". It is a specific part of the propeller shaft; therefore it can be used for the definition of a component connection and have a physical port. In this work the resolution stops at the propeller shaft, which corresponds to the component level.

E-Cl@ss presents the necessary properties for every commodity class but no values for them. The suppliers of the parts add the values to each model they add to their component catalog. Due to this, no values are provided in the component model library. Once a solution candidate is identified, specific parts with specific values have to be selected from supplier catalogs for a detailed investigation.

4.5.3 Identification of solution candidates in the model library

The objective of building a model library of components is to support the sequence of tasks in conceptual design. The main point is to identify components out of a model library. Principally, there are three possibilities described in the literature: mapping, identification by inputs and outputs and naïve identification by name. Naïve identification by name uses the mental lexicon and experience of the designer and therefore is excluded. Figure 29 shows an overview of identification methods.

Identification by mapping can be divided into structural mapping and direct mapping. The first means components can be found in a library structure parallel to the functional basis. Such a method has been proposed by KURTOGLU at al. [2009]. Direct mapping requires a predefined link from a function to an artifact embodying the function. For artifacts embodying more than one function, the two mapping types reach their limits. With structural mapping, a component embodying two functions has to be in two locations in the library structure. This contradicts the idea of having unique, redundant-free models in the library. For direct mapping, every component has to be linked to functions that can be embodied by the component. Adding and modifying components becomes difficult. Every new component would have been analyzed as to what functions it can embody and the links would have to be identified and set one by one. Changes of one of the model libraries would trigger changes in the many direct link definitions.

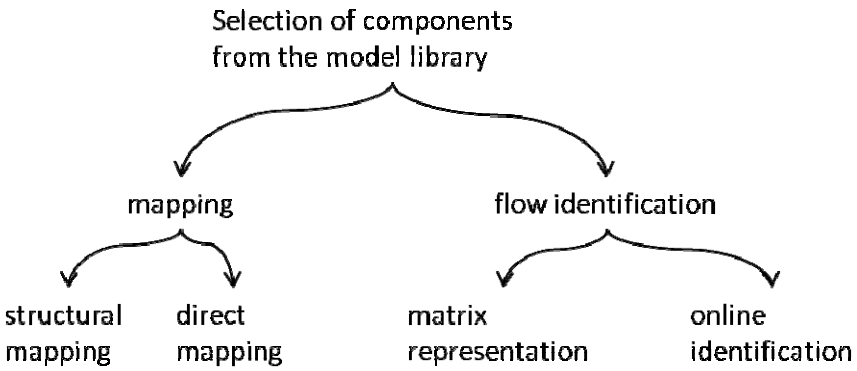

Figure 29: Component identification methods

Identification by inputs and outputs is used in the design catalogs of ROTH and Koller. Since they worked with paper-based methods, they had to present inputs and outputs as a mapping matrix to access the catalog. They create a specific presentation for finding components

before use. In a global market place with a huge number of components, this is not viable. In a computer internal representation, it is not necessary to create a mapping model. Instead, the model library can be searched directly for input and output types. The challenge lies in the definition of appropriate inputs and outputs for the model library contents.

The search for components is based on the flow information of the function to embody. In the function models, the flows are represented as typed pins. In the component library every component model has typed object ports. The types of ports are taken from the NIST-FB taxonomy of flows. Embodiments for functions are then identified by searching for components that have the same port properties, i.e. direction and type, as the pins of the nodes in the function structure. The object ports of the model library items have the purpose to identify the components. They can be derived either form attributes of the component or from the real interfaces of the component. The latter are modeled as physical ports on the component model. They are not modeled directly in the component library, but are part of the component connection models realized with association classes.

The next section explains how the relevant components are identified form E-Class and transferred into a model library.

4.6 Setting up the model library for components

The transfer of component information from an e-Cl@ss XML file to a Magic Draw file is done in several process steps, Segment 23, machine elements, is processed manually without any specific parsing tool. First, the relevant segment is parsed to a spread sheet. At the main group level several entities were found without functional meaning. This happens when an entity cannot realize a function by itself or refers to service, e.g. e-Cl@ss entity 27029990: Electrical drive (repair, unclassified). All these items, e.g. process materials, lubricants and adhesives, all door and window fittings and all wood connections are sorted out at the main group level. At group level, all entities with the terms repair, maintenance and unclassified were taken out. If it were possible at group level, the object ports were assigned to the entities and the commodity classes were skipped. If not, the commodity classes are taken into account. These steps have been carried out to create an example model library for reusable components. Figure 30 gives an overview of the information extraction for the model library. Finally, the object ports according to the physical ports that the device contains are set.

After pre-processing of the data in a spread sheet, it is mapped into a SysML model. In it, the information is structured into several packages to keep the structure of the classification standard. In this condition, the library contains more than 237 entities in several packages. During modeling, it will be tested how model elements can be found and identified out of the model library.

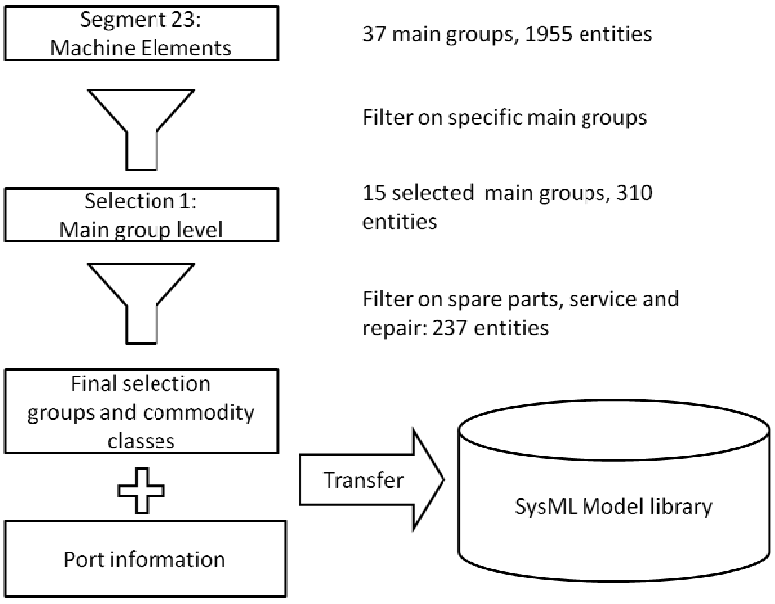

Figure 30: Information transfer from e-Cl@ss to model library

With the solution candidates, only abstract parts are provided. It is an attribute of abstract parts that the specification is incomplete. For model evolvement in terms of simulation data or CAD data, the suppliers can be asked for these data. Another possibility is to combine the information from E-Cl@ss with different catalogs. The hierarchy number, given as default value of the attribute “EclassHierarchy” of the component model in the library, can be used for searching catalogs.

4.7 Description of the component model library

A solution candidate is defined as a SysML Block. It has object ports stereotyped as flow ports. These are named as "ObjectIn" or "ObjectOut", where the “In” and “Out” complies with the port direction. The NIST-FB flow library is used to set the types of the ports. These ports are defined for the selection of the component, i.e. matching the incoming and outgoing flows of the functions. The example library contains more than 230 entities. The first challenge is the structuring of the contents of the library, determining the appropriate grade of detail and gaining an overview about the contents.

The example library is structured using SysML packages. This comes closest to the structure in E-Cl@ss from which the contents are derived. The structure is a four level hierarchy. The relation between the parent member and the child member is a “belongs-to-group” relation. Establishing an inheritance is not considered in the first row. The objective is to identity the components by ports and attributes. Inheritance is not a focus and can be done in future work.

Also defining the detail grade came with some difficulty. Normally, a component is found in

commodity class level, the lowest of the four-level hierarchy. Hierarchy level is neither an indicator for simplicity nor for providing a contribution to the realization of a function. At the lowest level there can be found parts, e.g. bolts as well as modules e.g. linear drive modules. Bolts are used for fixing together two solid objects, they comply more to an intermediary flow for the function “Link” than they realize a function. An Acme screw drive, in contrast, is a linear drive that is designed to fulfill the main function convert rotational mechanical energy to translational mechanical energy. Further, it can support forces or solid objects, transport solid objects and guide material. A linear drive is a complex component that itself is assembled as a guiding device such as a shaft or profiled rod and a carriage moving on it. The carriage itself is connected to the device that is providing linear motion and needs rotation as input. For the solution candidates model library all main groups of segment 23 are shown as blocks. Where group and commodity class level is identified as relevant for the conceptual design, example packages at the main group and group levels are introduced.

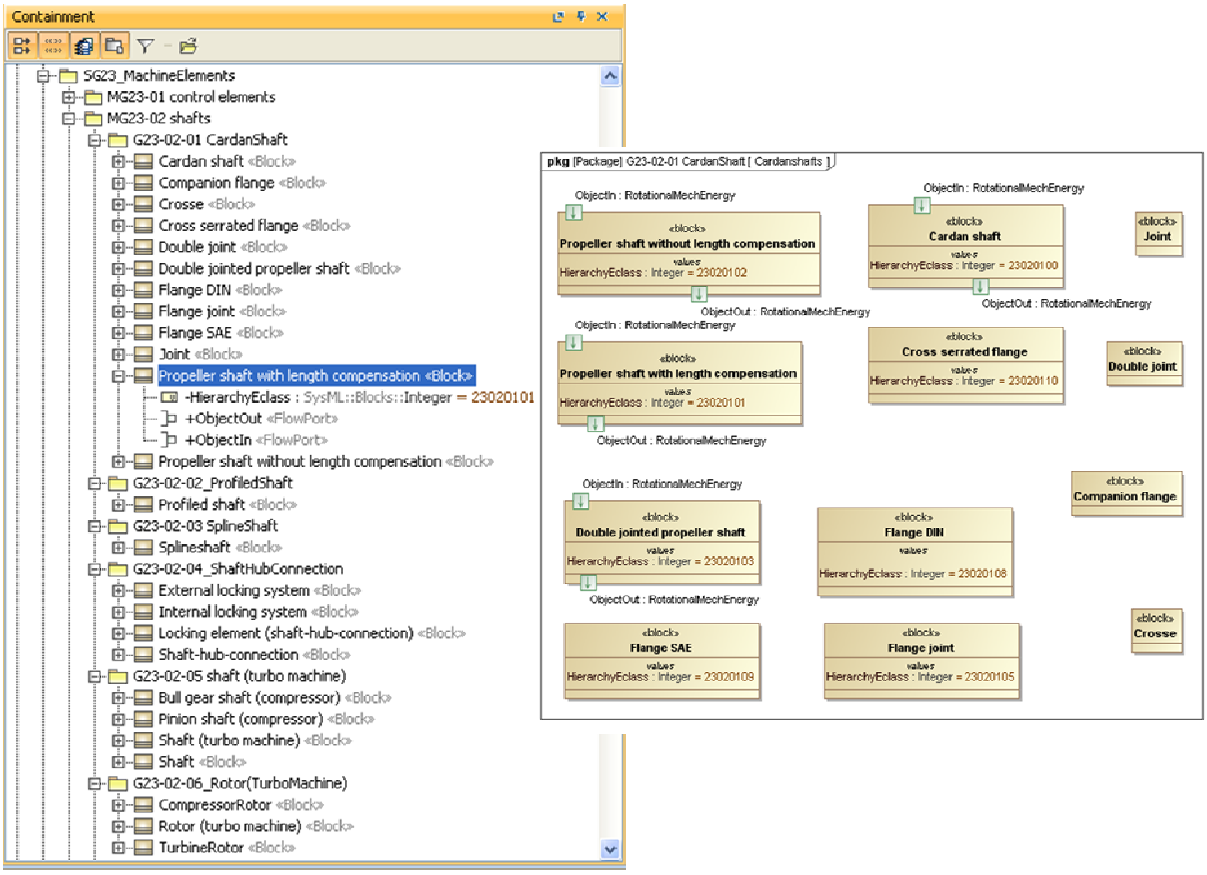

Figure 31: Component library and package of Cardan shafts (screenshots from CASE tool)

Figure 31 presents a screenshot of the model library content. On the left side, the containment tree shows how segments, main groups and groups are transferred into a package structure in the model library. In CASE tools, a containment tree is an index of all model elements including nodes, relations and diagrams. The right hand side shows the package diagram of

Cardan shafts. For every shaft, the object ports are shown. Blocks without ports, e.g. “Flange” and “Joint” indicate that these are parts of Cardan shafts. There are no relations between blocks, neither in the containment nor in the diagrams. Figure 31 shows an example of instances according to the metamodel for component library content as defined in Figure 26. Every “Artifact” having an “ObjectPort” is represented by a component with one or more ports e.g. “Propeller shaft with length compensation”.

A portion of the component library has been presented. Presenting only a portion in a package is a possibility to show details like ports in a graphical way. However, it is difficult to present the whole library at once either on the screen or on a poster due to the number of entities in the model library structure.

5 Results

In this Chapter, further details of modeling are explained with the modeling example. Then, the usability aspects of modeling with model libraries are presented. Finally, the validation study shows an additional step of decomposition for functional and component modeling.

5.1 Use of libraries by modeling example

While in section 4.1 it has been shown how SysML can be used generally for supporting the systematic design approach, in this section the focus is on modeling with library support.

5.1.1 Modeling with library support – the luggage compartment cover

The first diagrams of the study have been presented without describing how the support with model libraries could work. In this section, a short overview over the modeling steps and focus on presenting the best practice of library use is given. Figure 32 shows the sequence of model creation. Each step stands for an activity that can be decomposed into modeling several elements and depicting several diagrams.

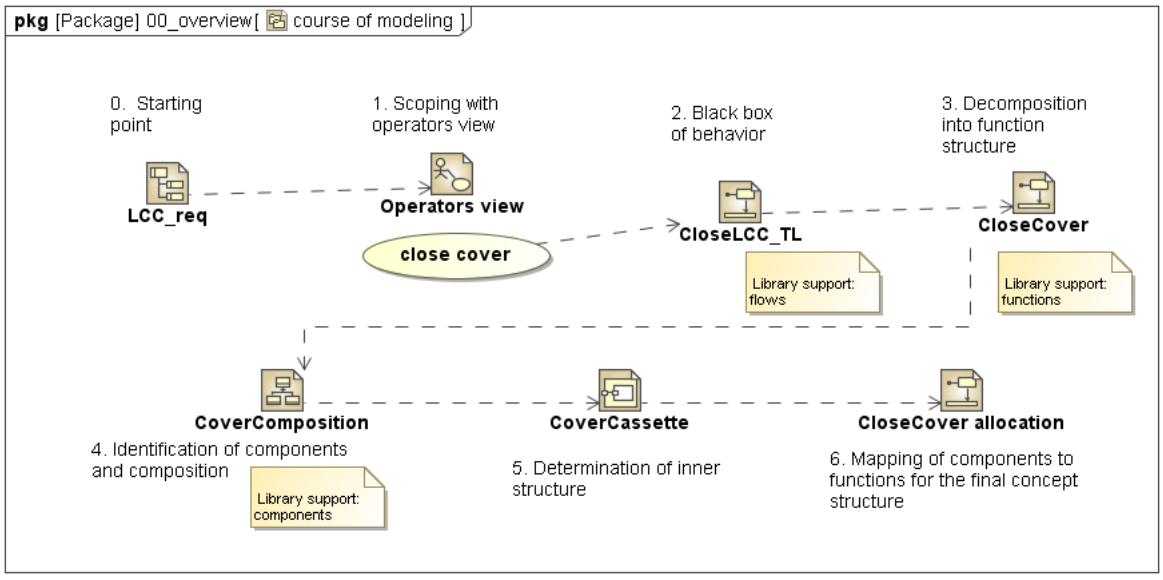

Figure 32: Sequence of modeling augmented with library support

The first diagram is the requirement diagram “LCC_req”. Then a use case diagram scopes the product from the users view. In step two, the use case “close cover” is refined by a black box diagram that in turn is decomposed into a function structure. At this step, the flow library is

used for the first time, as the note in Figure 32 indicates. In step four, functions are found from the library and used to build a function structure. In the next step components are mapped to the functions and taken out of the components library. They are used either directly in the block definition diagram “CoverComposition” or for the determination of the inner structure in an internal block diagram. Finally the selected components are allocated to the previously created function structure.

5.1.2 Library supported functional modeling

The first step that is supported by a model library is the creation of black box behaviors or top level functions. When modeling the input and output parameters, the types are selected from the model library of flows. This is done in the specification dialog box of the action. Specification of elements is a standard task for model elements presented in CASE tools. The benefit for the modeler is that he or she does not need to specify a flow upfront. The top level function however is the basis for the decomposition into a function structure.

Figure 33 shows the top level function for close cover. The comments are inserted to make clear from which element the abstract object flow is derived. No specific function term is used at this level. Input and output pins are typed with flow objects from the NIST-FB library. The use of flows makes sure that the black box action can be converted into a white box diagram without readjusting the flows. In the second level of decomposition these pins are the activity parameter nodes at the frame of the diagram. For the decomposition of the top level function into a function structure further steps have to be taken.

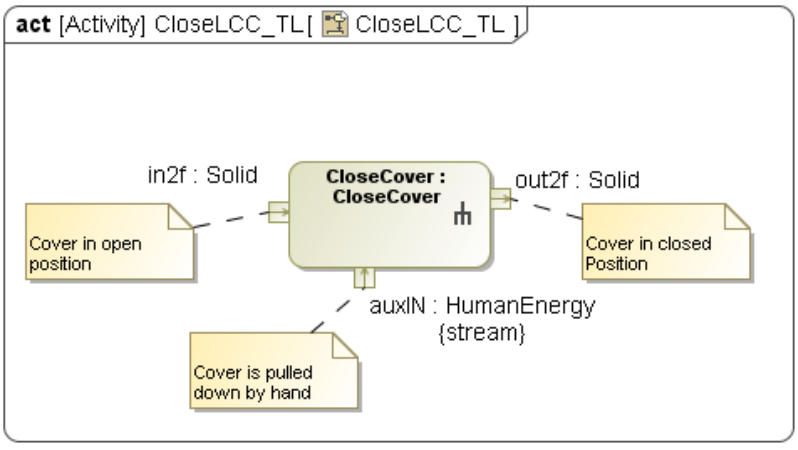

Figure 33: Commented Top level function

The first step is creating a diagram with the activity parameter nodes derived from the black box action. Then the first functions can be inserted using the model library. Figure 34 shows the steps of working with the library for functional modeling. It is a portion of the screenshot of the CASE tool during modeling. At this moment the diagram is not complete. On the side is the function library in the containment tree presentation; on the right side is the diagram

pane where elements from the containment tree can be dragged. A function can be inserted (Fig 34 step(1)) by dragging an activity directly from the containment tree (Fig 34 left) to the canvas of the activity diagram (Fig 34 right). In the diagram, a call behavior action with all predefined pins is created. In Figure 34, the pin type for the function "Store" is "MaterialEnergy". This is a special type that indicates that the modeler can choose between the flows material and energy. An alternative way to draw a function node in the diagram is placing an action node on the diagram canvas and selecting a behavior from the context menu. It results in the same call behavior action if the same library element is selected as a behavior. In step (2), the pins of the actions have to be specified to a specific flow. This is done by selecting "type" in the context menu or the specification dialog of the action. Then the type is selected from the model library of flows. After all pins have been specified, the pins can be connected by object flows (3). The SysML specification allows only connecting pins with the same type and conjugate flow direction. With the first fully defined action in the activity diagram at least one object flow can be set between one of the activity parameter nodes, i.e. the nodes at the diagram frame and the pin of the action. The adding of call behavior action nodes and connecting object flows is repeated until the function structure is complete.

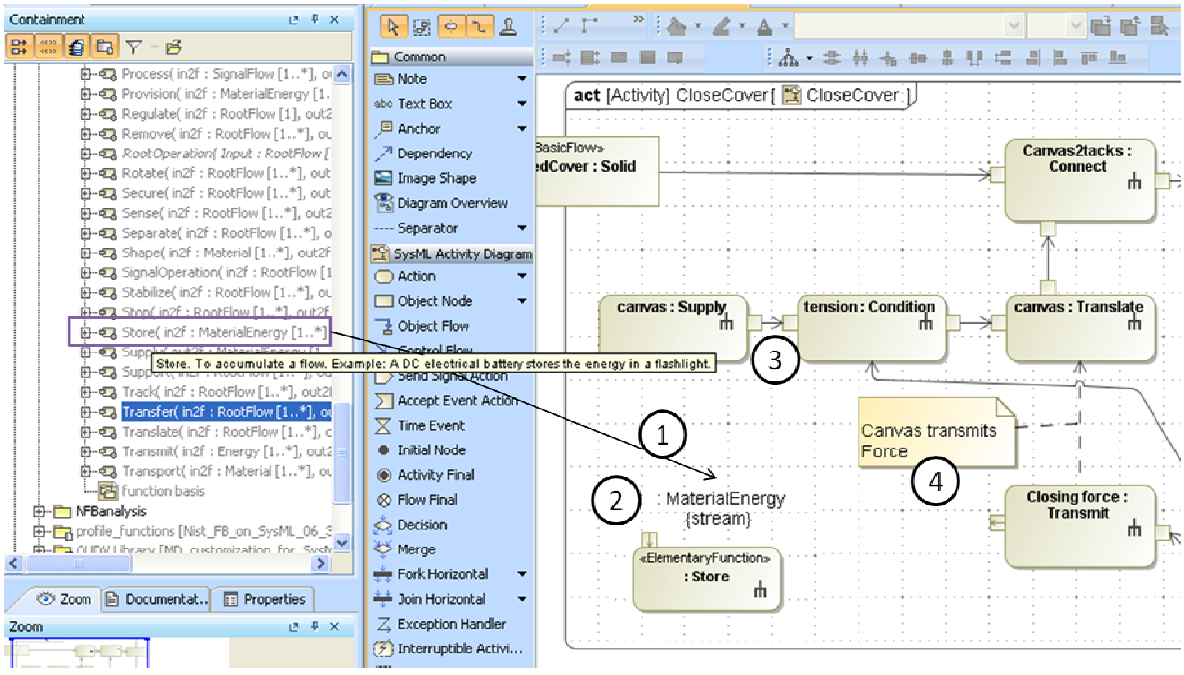

Figure 34: Steps of using function library (diagram in creation)

As a last step, comments (4) can be inserted e.g. the hint that two functions are not independent from another. An indicator for a completed diagram is when all activity parameter nodes are connected. When the diagram is complete, there are several checks to run. First, check if all necessary pins have been connected. This has to be done by hand. Other checks are whether all pin types are set correctly and all object flows have the same type at source and target end. When all the checks are made, the activity diagram is ready for the next

step. Figure 34 shows a partial diagram. The finished diagram is shown in Figure 35.

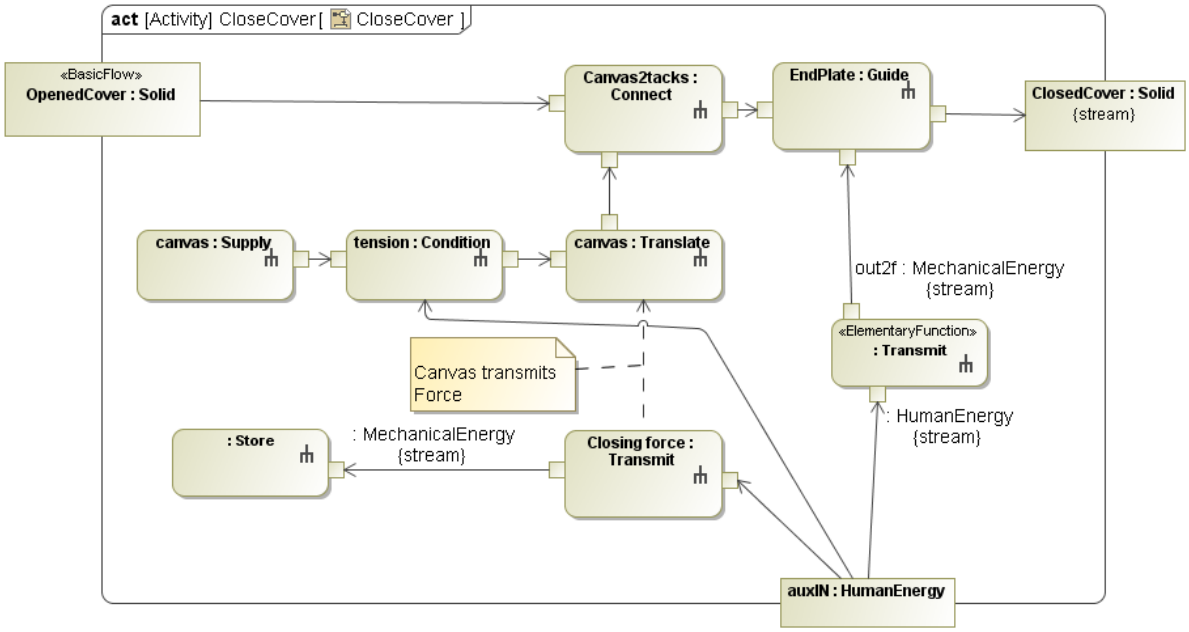

Figure 35: Completed function structure "CloseCover"

In the SysML specification, the representation of activities as an association hierarchy in a block definition diagram is stated in Section 11.3.1.1 of the SysML specification [OMG 2010b]. This is implemented in the modeling tool as a wizard that rearranges the network type activity diagrams into a tree structure. Without respecting the parameters (flows), the resulting diagram ends up with the seven activities that are referenced due to call behavior actions. This feature presents a fast conversion of the network graph to a function tree structure. It helps to gain an overview of the functional decomposition. The result is shown in Figure 36.

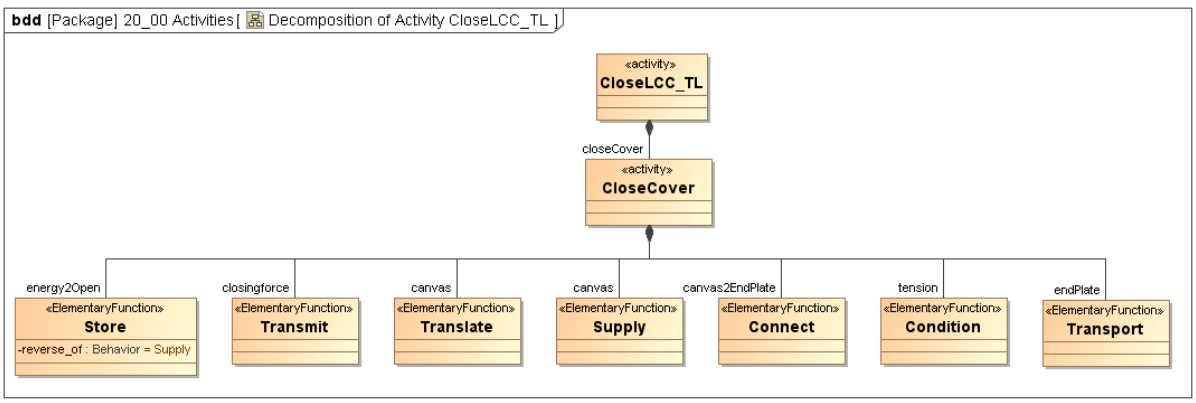

Figure 36: Hierarchical decomposition of functions realized with activity decomposition

With a complete function structure, the search for components can be started. First a package for the solution candidates is created. Inside that, nested packages for the embodiments for the functions are created. Then the model library is searched for blocks that have object ports with the same direction and the same type as the functions' pins. With Figure 37, the search for components with Magic Draw's quick search that enables searching for port name and port type at once is presented. This allows identifying elements where at least one port type matches and directly points to the results in the containment tree. Every model element with a corresponding port structure is then copied into the solution candidate package. Several situations can occur during that search: No hits, when no predefined elements are in the model library. Several hits with an exact matching and several hits where the components show more ports than the functions have pins.

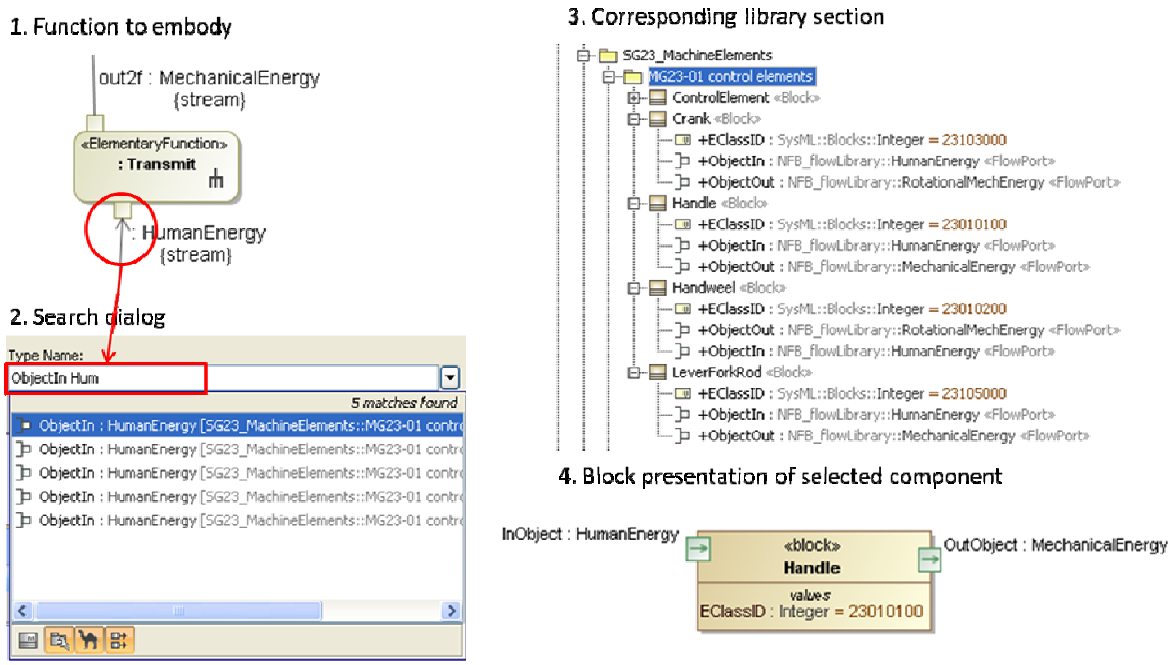

Figure 37: Search for components in the component library

When all the blocks are collected in packages, an Internal Block diagram can be drawn. For an internal block diagram it is necessary to have an owning block. This block represents the artifact under design. In the modeling example of the luggage compartment cover, components from the library have been identified as well as specific components have been created where no entry in the component library existed. For each of those, an additional block is created. The first resulting diagram can be something like the internal block diagram in section 4.1. It can be further refined by specifying the connections as shown in Figure 38.

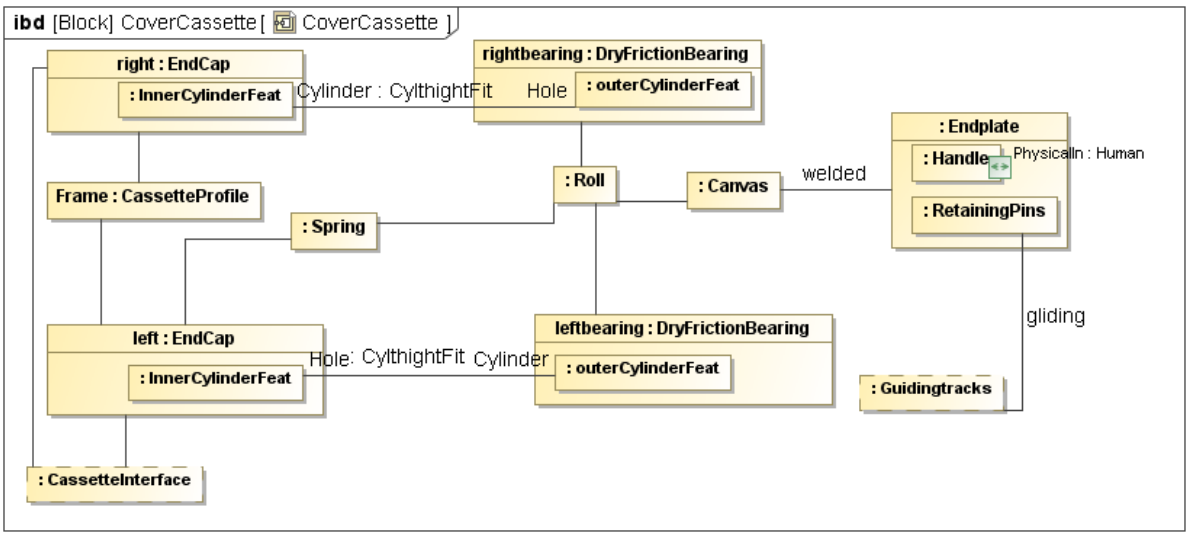

Figure 38: IBD with specific connections

Two bearings are fit in the end caps with the link called “CyltightFit” that represents a cylindrical tight fit of the dry friction bearing and the end cap. The connection in this case is defined in the model, not in the model library. An example of a physical port is given at the handle where the human hand can be placed.

5.2 Validation Study

In Section 5.1 a modeling study with a simple mechanical system is presented. For the validation of the approach the 3D printer is modeled. It is a more complex mechatronic product. The differences between the two products are shown in a comparison table (Table 5-1).

Table 5-1: Comparison of modeling example and validation study

	Luggage compartment cover	3D-Printer
Domain	mechanical / automotive	mechanical/ electrical / software
Components off the shelf	few	many
Information	restricted access	available (Apache Public License)
Functional decompostion	two levels	three levels

The difference between the modeling example and the validation study aims to show the general applicability of SysML modeling and library support in conceptual design. With more involved domains in the validation study, the challenge of concise functional modeling and

partitioning of functions to select components increases. However, the selection of components should be easier, since more off the shelf components are used. Due to the increased complexity, it is necessary to break down functional modeling with an additional decomposition step. For functional modeling and component modeling the example is one linear motion unit. These units are used to move the print head relative to the print support. At the module level of the product structure, some results on structural configuration of the 3D printer are presented.

5.2.1 Requirements modeling of the 3D-Printer

The approach does not change with larger projects; however, the presentation of the whole model becomes more difficult. The requirements diagram needs a bigger format (e.g. DIN/A3) to be legibly shown (see also Fig. 39). However, there are reasons to model requirements in some diagrams. Several branches of requirements exist like general requirements that apply on the whole product or requirements that are related to a special domain such as software or electronics. It makes sense to keep these requirements together and to appreciate the possibility to add “satisfy” and “verify” links in the diagrams. To keep an overview over a big number of requirements, the SysML specification and the CASE tool offers the possibility to create a requirements table from all requirements, no matter if they are presented in a diagram or exist just in the containment tree. Table 5-2 shows the requirements for the 3D printer in the table presentation. This form of presentation can be expanded to a far larger number of requirements. The unique numbering of requirements is specified in the SysML specification. It makes sure that every requirement can be traced through the modeling process.

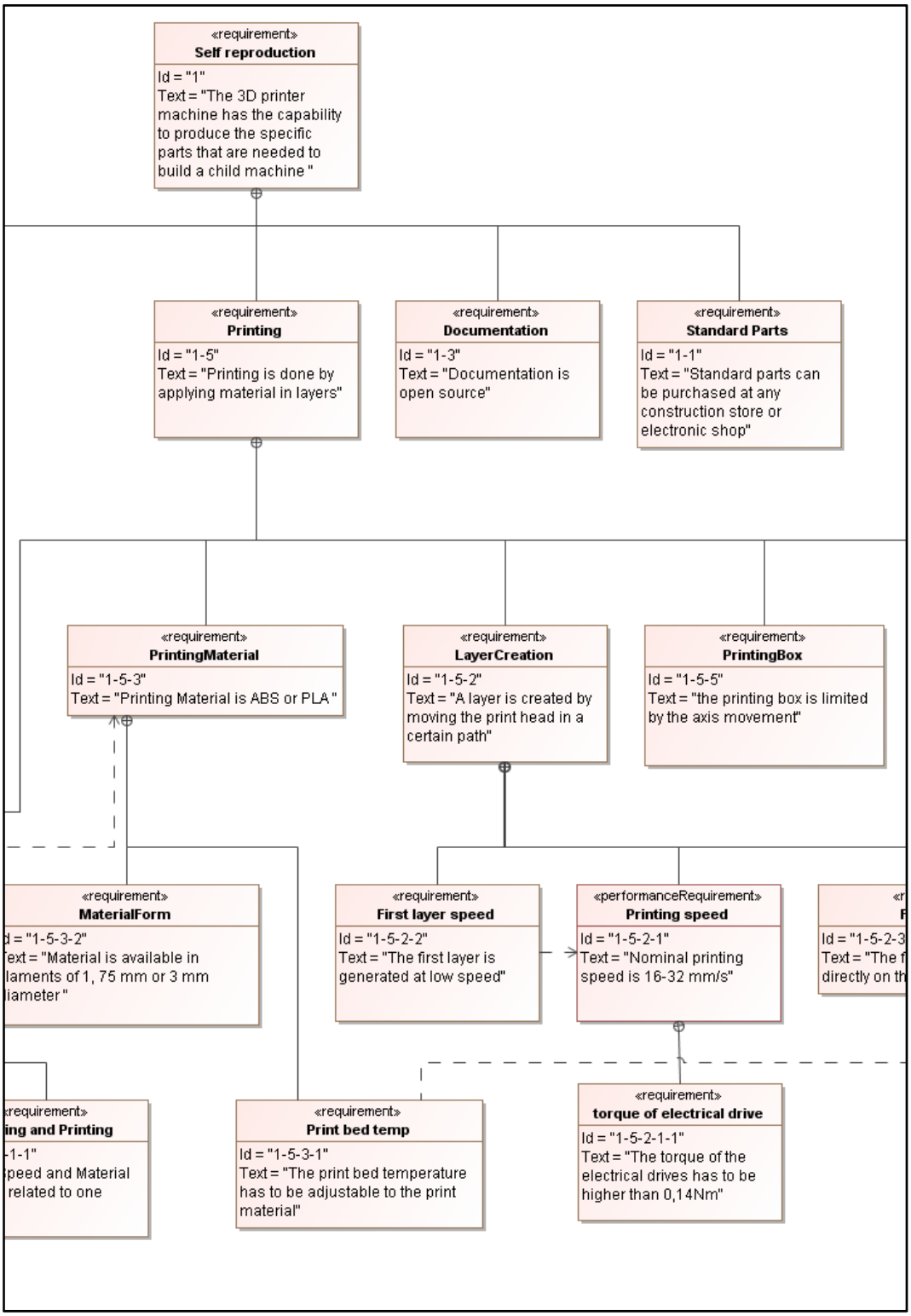

Figure 39: Cut out of requirements diagram (approx. one third of whole content)

The contents cut out of Figure 39 can be found in the table below in another form.

Table 5-2: Requirements for the 3D-printer

ID	Name	Text
1	Self reproduction	The 3D printer machine has the capability to produce the specific parts that are needed to build a child machine
1-1	Standard parts	Standard parts can be purchased at a highway store or electronic shop
1-2	Special parts	Special parts are made by printing with a parent machine
1-2-1	CAD data	Solid Models are available open source for special parts
1-3	Documentation	Documentation is open source
1-4	Software	Software is open source
1-4-1	CAD Software	CAD software is used for creating solid models for printing
1-4-2	CAM Software	CAM Software creates path and layer information from CAD models
1-4-3	Firmware	Firmware for the control unit is necessary
1-5	Printing	Printing is done by applying material in layers
1-5-1	Printing Process	The material is applied by melting portions of solid material
1-5-1-1	Moving and Printing	Speed and Material feed are related to one another
1-5-1-2	Moving only	The printing process has to be stopped for moving only
1-5-2	Layer Creation	A layer is created by moving the print head in a certain path
1-5-2-1	Printing speed	Nominal printing speed is 16-32 mm/s
1-5-2-1-1	torque of electrical drive	The torque of the electrical drives has to be higher than 0,14Nm
1-5-2-2	First layer speed	The first layer is generated at low speed
1-5-2-3	First layer	The first layer is applied directly on the print bed
1-5-2-4	Position of print head	The position of the print head has to be determined during printing
1-5-2-4-1	Position accuracy	The reproducible position accuracy of the print head is about 0,1 mm
1-5-3	Printing Material	Printing Material is Acrylonitrile butadiene styrene (ABS) or polylactic acid (PLA)
1-5-3-1	Print bed temp	The print bed temperature has to be adjustable to the print material
1-5-3-2	Material Form	Material is available in filaments of 1, 75 mm or 3 mm diameter
1-5-4	New layer	a new layer is created by moving the print head vertically from the print bed
1-5-5	Printing box	The printable volume is limited by the axis movement
1-5-6	Accuracy	Printing accuracy is smaller than 0,3 mm

The next step in modeling is creating the use case diagram. Since the whole diagram is too big to legibly present, the relevant part for the subsequent modeling of functions is presented in Figure 40. Nevertheless, the use case diagram helps to scope the problem and take different viewpoints. For the printer validation study, the actors “Machine user”, “Machine builder” and “Printer control” are modeled. It is common practice in systems engineering that artifacts of the system can be defined as actors. The focus of the study lies in the use case “Printing”.

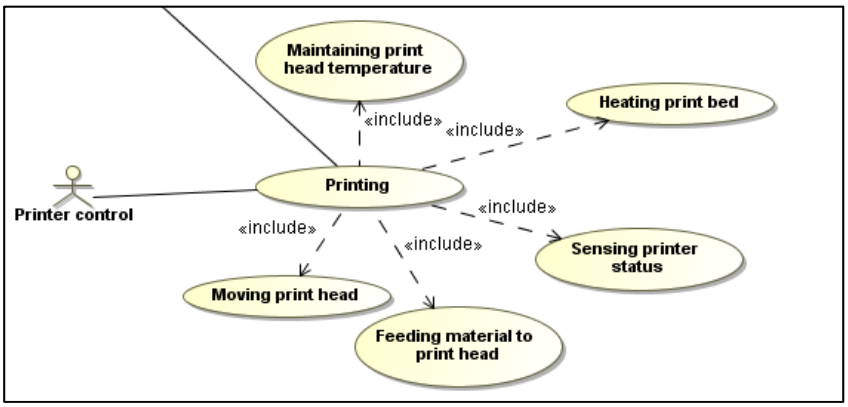

Figure 40: Cut out of 3D-printer's use case diagram

The 3D printer has to provide all means necessary for printing. In the following steps, the use case is refined with functional modeling.

5.2.2 Validation of functional modeling

The difference to the modeling study is an additional layer in the decomposition structure. For the validation study, six diagrams are modeled: one black box diagram at product level, one diagram to decompose the black box into function modules and four diagrams to decompose the function modules. A function module is a higher level behavior that still can be decomposed. Figure 41 presents an overview of the diagrams used for the decomposition of the main function. In contrast to that, the luggage compartment cover example only used one step of decomposition.

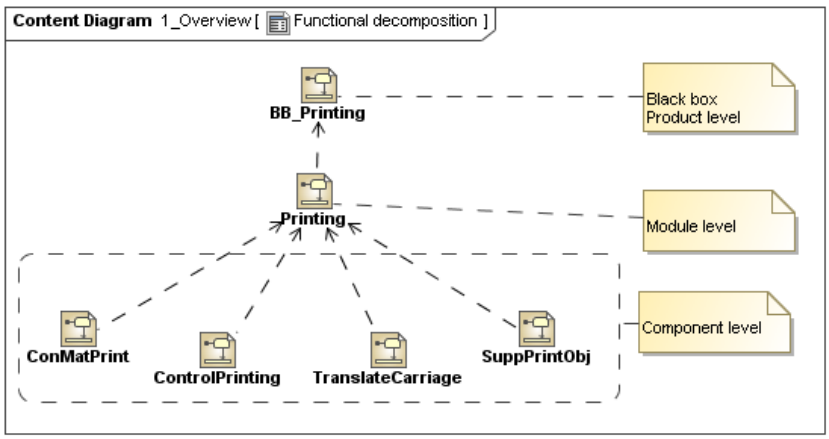

Figure 41: Overview of the functional decomposition of the 3D-printer

In the following the black box “BB_Printing”, the module level function “Printing” and the component level “TranslateCarriage” are presented.

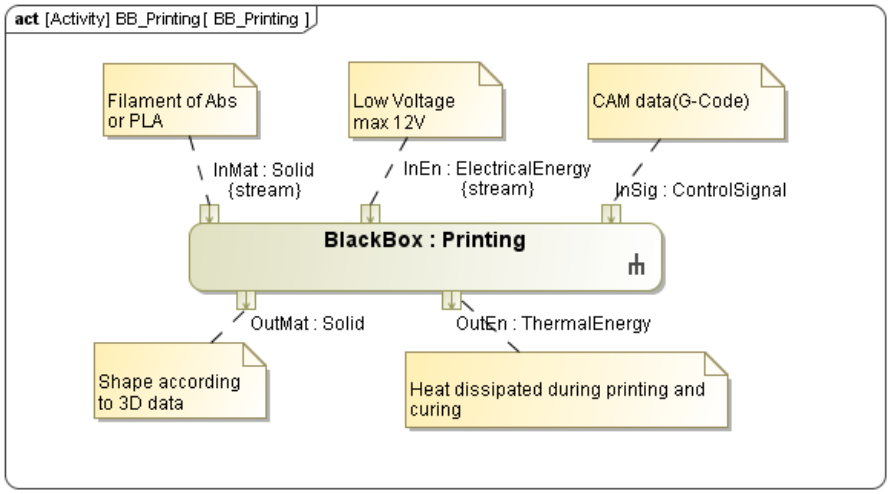

Figure 42: FDM-printer function as a black box diagram

The first step functional model is the black box diagram of the function at top level. It is shown in Figure 42.

The black box depicts the inputs and outputs of the printing of the 3D-printer. The pins are commented with notes for a clearer understanding. It is decomposed with an activity diagram into module level that provides an overview of functions of the whole printer. Like in the black box model of the printer’s function, the terms are not taken from the function library. They are complete verb noun pairs, like “Fuse Material”, and still decomposable to library elements. Types for parameter nodes and pins are taken out of the flow library to maintain the consistency of the model. At this level an action node can be seen as a function module. The comments on the action nodes augment information as to what module of the printer embodies the function. The function structure created at the second level of functional modeling is not necessarily mirroring the partitioning of the product or system from the structural viewpoint. Figure 43 shows the function structure at the module level. Material flows are depicted as a bold line. Some flows have a name to clarify the meaning of the object flow. The “TranslateCarriage” function is modeled three times to represent the three Cartesian directions the print head has to move. It is the same for any of the three axes. But the three linear motion units that embody that function are carrying different loads. The Z-axis, attached to the frame, carries the linear module of the X-axis that in turn carries the print head. The print bed is assembled to the linear module of the Y-axis. This configuration mirrors the function structure of the printer model “Mendel” in the RepRap project. Each action in Figure 43 is a call behavior action that references an activity diagram.

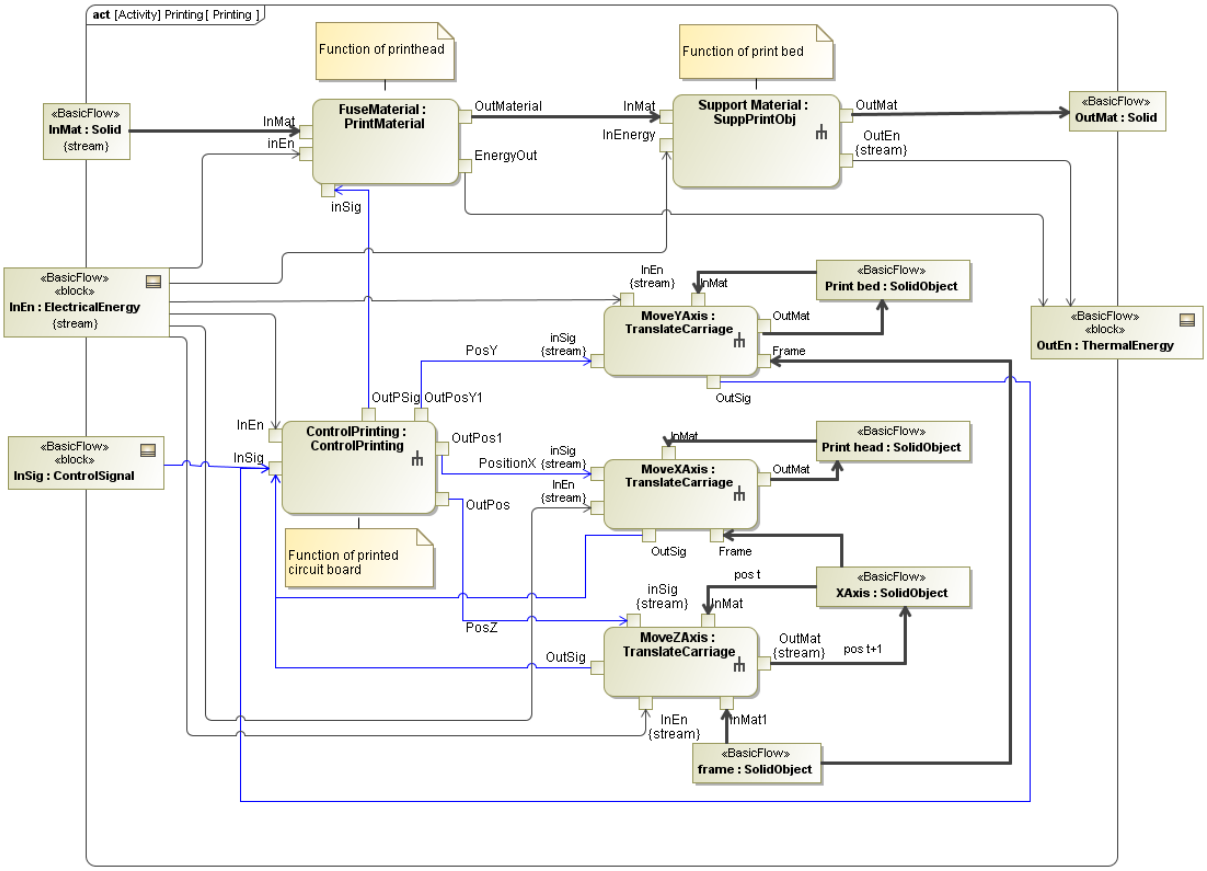

Figure 43: Functional decomposition at module level

With the third level or component level, the functions are fully decomposed. This means that in the third level the action nodes are taken from the function library. Figure 44 shows the decomposition of “TranslateCarriage”. The pin types are hidden from the diagram in order not to overload it. All actions are connected via object flows that only use the pins. Some pins have more than in or out one object flows. The function “LoadVsGravity: Support” has two object flows entering in one pin. This is allowed when the flow type is the same. In this case, one solid object, e.g. the print bed is supported with the linear unit by another solid object, e.g. the frame. The diagram in Figure 44 shows the function structure of one linear module. It is referenced three times, i.e. as X-, Y-, and Z-Axis from the diagram in Figure 43. The function “LoadVsGravity: Support” describes, that the linear module can carry a load. “Carriage:Translate” is the function that the Carriage can be moved in one direction. These two functions can be embodied by a linear bearing. “Carriage:Transport” describes that the carriage is driven.

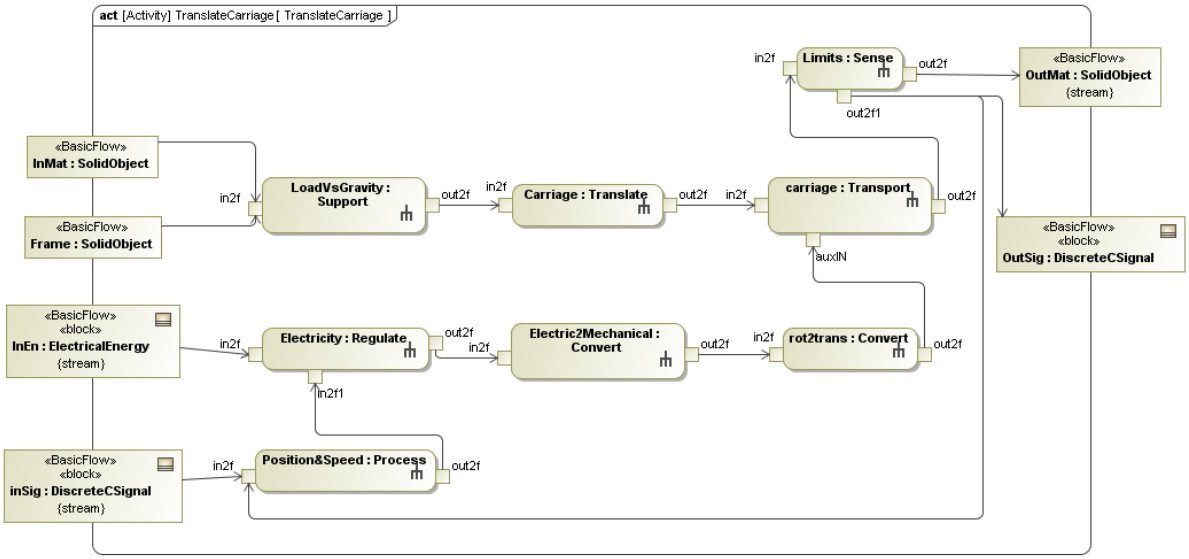

Figure 44: Function structure at component layer

These are the results for functional modeling. In the next step these functions are mapped to components by searching the component model library for solution candidates.

5.2.3 Validation of component modeling

In modeling the component structure, the difference to the modeling example is in the number of elements out of the model library that can be used for the embodiment of functions. The search for solution candidates and their grouping becomes more important. It is the same as in the results of the modeling example. Additionally the solution candidates are grouped into a generalization set with the generic component as the general member and each component as a specialized member. A generalization set is a group of inheritance relations where the child elements share the same parent element.

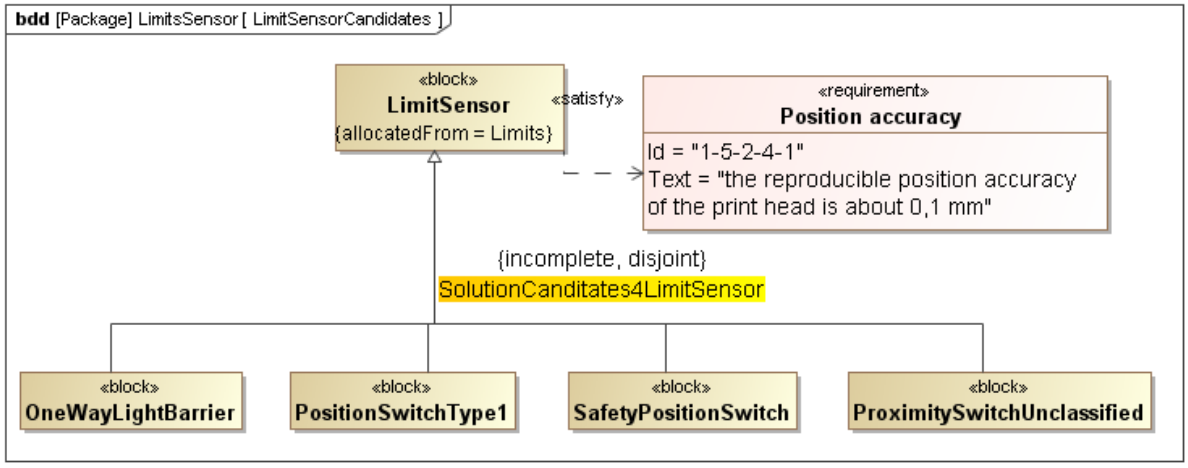

Figure 45: Solution candidates for the limit sensor

In Figure 45 the generalization set “SolutionCandidates4LimitSensor” is shown. The generalized member is the block “LimitSensor”, a generic component that is used as a substitute to embody functions. The specialized members are blocks taken out of the component model library. A generalization set has two properties that describe coverage and overlap between their members. The solution candidates have the coverage to “incomplete”. A complete coverage is given if all choices are known, e.g. am and pm for daytime. The overlap of the generalization set is disjoint. This means that only one of the specific solution candidates can be the solution for the limit sensor. The requirement “Position accuracy” is placed into the limit sensor candidate diagram. The limit sensors are used to initialize the linear drives. Therefore, they have to support the position accuracy.

There are also components that embody more functions than just a single one. This means functions link to one component out of the library. SysML offers the technique of allocation that is applied on the function structure shown in Fig. 44. Allocation is a relation where a SysML block, representing the embodiment is related to an action representing a function. Figure 46 shows the allocation via swim lanes. Every lane is related to a block that represents a generic component as substitute for a number of solution candidates.

In Figure 46 there is one swim lane that is partitioned into two sub lanes. The function process is allocated to the software that runs on the microcontroller of the circuit board. The function regulate is allocated to the hardware side of the circuit board. For regulating the energy of electric drives power semiconductors called h-bridges are used. Microcontrollers cannot provide enough power for electric drives. They are part of electronic and software design.

The functions support, translate, transport and convert rotational to translational energy are allocated to a generic linear drive. A linear drive is defined as a device that can take a load and move it a certain translational distance. Therefore, these four functions are allocated to one generic component. Linear drive, electrical drive and limit sensor form a linear motion unit. It is used as an example for an internal block diagram. While allocating actions to

blocks, the partitioning of the system is also carried out. Actions that are allocated to the circuit board are separated from the mechanical part although they are necessary in terms of function to make the linear motion unit work.

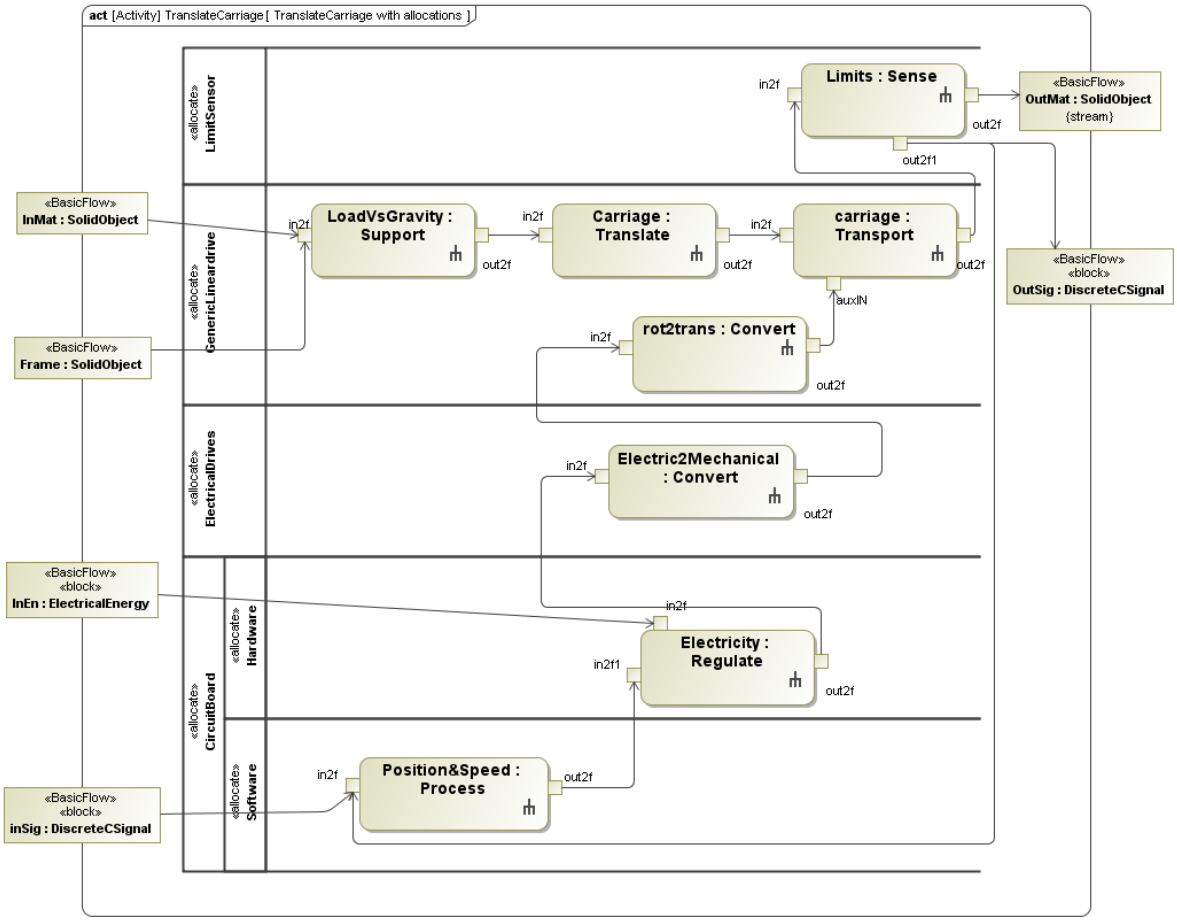

Figure 46: Function structure allocated to blocks with swim lanes

The allocation results in a structure of generic components. It is beneficial to define the physical interfaces as ports on the generic components. The generalization sets that have been created for the solution candidates let the specific components inherit the port definitions from the generic component. After the definition of the physical ports, it is possible to draw internal block diagrams for the structure of the system. This is done for the horizontal linear motion unit in Fig. 47. It contains three specific solution elements. Since there are more than one solution candidate for each of the three solution elements, the number of solutions can be calculated from the combination of all solutions candidates.

- Generic linear drive → seven solution candidates
- Generic electrical drive → five solution candidates
- Limits Sensor → four solution candidates (see Fig.45)

These solution candidates represent groups of COTS, each of which can be offered from several suppliers. This results in 140 technically different solutions and a large number of different solutions when going into more detail.

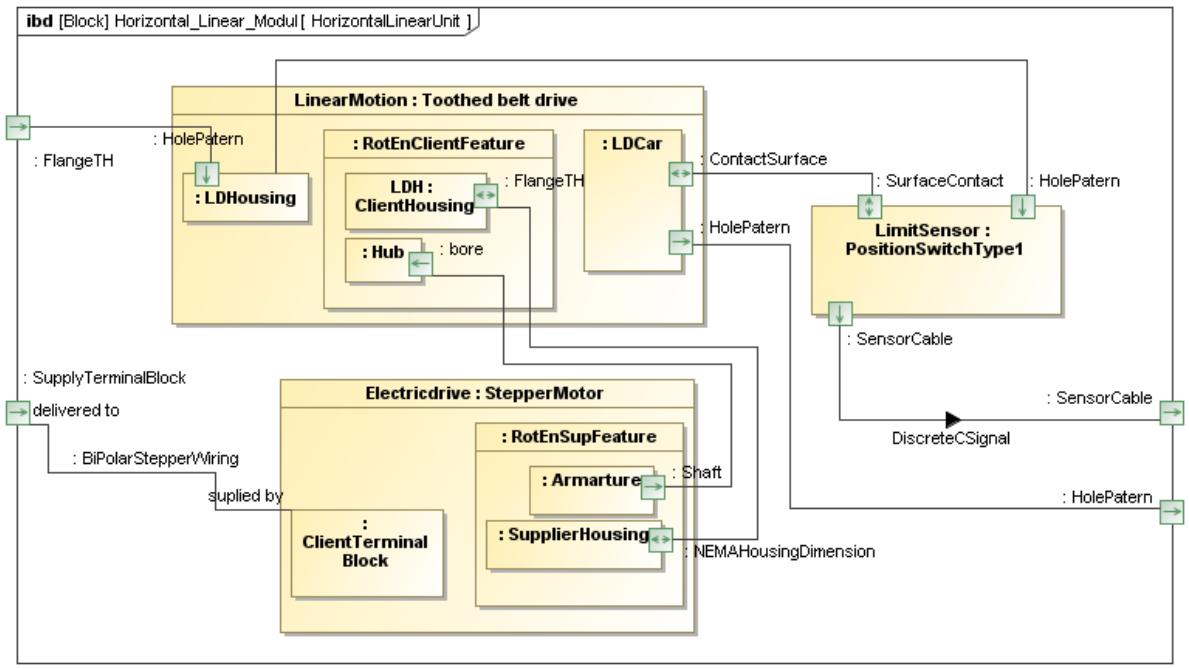

Figure 47: Internal block diagram showing specific components of a horizontal linear motion unit

Figure 47 shows the internal block diagram with the embodiment of the linear drive of Figure 46, except printed circuit board. The definition of modules, components and parts is according to Table 2-1. In this IBD, there are shown only physical ports. The toothed belt drive is shown with the rotational energy client feature (“RotEnClientFeature”), the housing (“LDHousing”) and linear drive carriage (“LDCar”). Each of these elements offers physical ports for connecting with other elements. All of them were defined for the generic linear drive and inherited to the toothed belt drive. Both the electrical and the mechanical connection of the stepper motor are defined with association blocks as presented in Chapter 4, Fig. 27. The mechanical connection is shown in part level detail and separates the connection of the shaft and the housing. The inner structure of the limit sensor is not shown. In the same way, all alternative solutions for linear motion units can be presented. The requirement for positional accuracy of 0.1mm of the print head to the print bed led to the combination of stepper motor and toothed belt drive. This combination is also cheap and realizable for people with limited knowledge in electronics.

For creating a structure at a higher level modules like the horizontal linear unit are allocated to the module level function structure presented in Figure 43. Up to now only one linear drive for horizontal movement is presented. The requirements state that the movement of the z-axis has to carry more loads due to gravity and can be slower than the x- and y- axis. It is for the movement of one layer into the next layer. The requirement of being built at home and the

low loads rule out the roller and ball screw drives, which are for heavy loads. These two requirements allow selecting one linear drive for the z axis: an Acme screw drive. It replaces the toothed belt drive for the vertical linear unit. For the other modules the allocation of functions to components is done likewise. Then the modules are assembled to the complete product structure. Figure 48 presents the structure diagram of the whole 3D- printer.

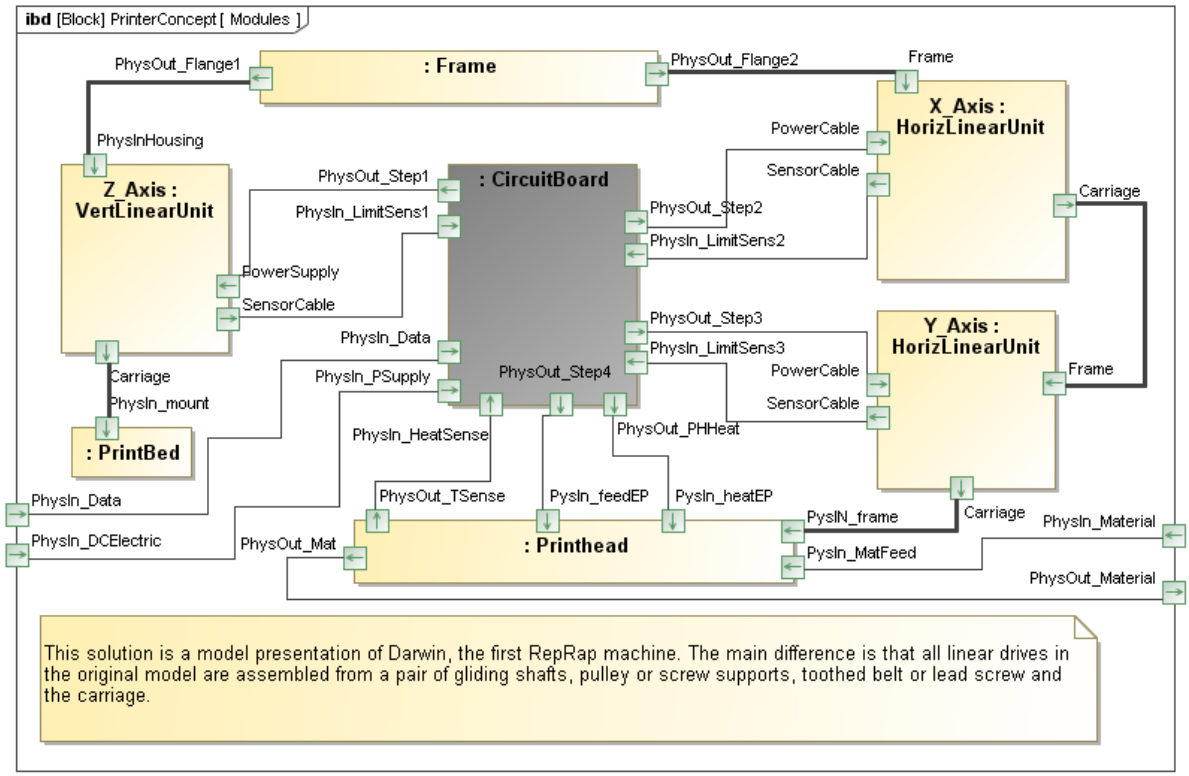

Figure 48: Product level structure of the whole printer

The IBD shows a high level structure with less detail than the IBD in the previous figure. Every linear unit in this figure shows the four ports that in the previous figure are on the diagram frame. But here no internal structure is shown.

To optically separate electronic modules from mechanical modules, the circuit board has a different color. Mechanically, in this model, the modules are connected to a chain. This is the effect of the partitioning of the printer structure into modules. In Figure 48 two of the linear motion modules are connected to the frame. The bold connectors symbolize mechanical connections. The original RepRap-printers assemble parts of the linear motion units, namely the guiding shafts and the pulley supports directly to the frame. Housings for linear motion units do not exist. If this is modeled at the component level the structure results in a network structure with a different structural partitioning. The partitioning into modules has benefits in the reuse of modules and in the integration of artifacts that embody functions of high cohesion e.g. linear motion units or print heads.

At the detail level, variants exist due to different components that are assembled. For this level, different structures can be created by variation of the structure. ROTH [2000] presents several options for this, namely change of order, replacement of elements, adding and deleting of elements and parallel and serial repetition of elements. The printer structure can be varied at the module level as long as three directions of movement of the print head relative to the print bed can be realized. All elements that are presented in the IBD in Figure 48 are necessary to realize that. Thus, the order of the modules in the mechanical chain can be varied or components can be added. In Figure 49, some of the possible variations by changing the order of elements are shown. For giving a compressed overview of the variants, Figure 49 is not drawn with the CASE tool, since this comparison needed six IBDs if done with a CASE tool.

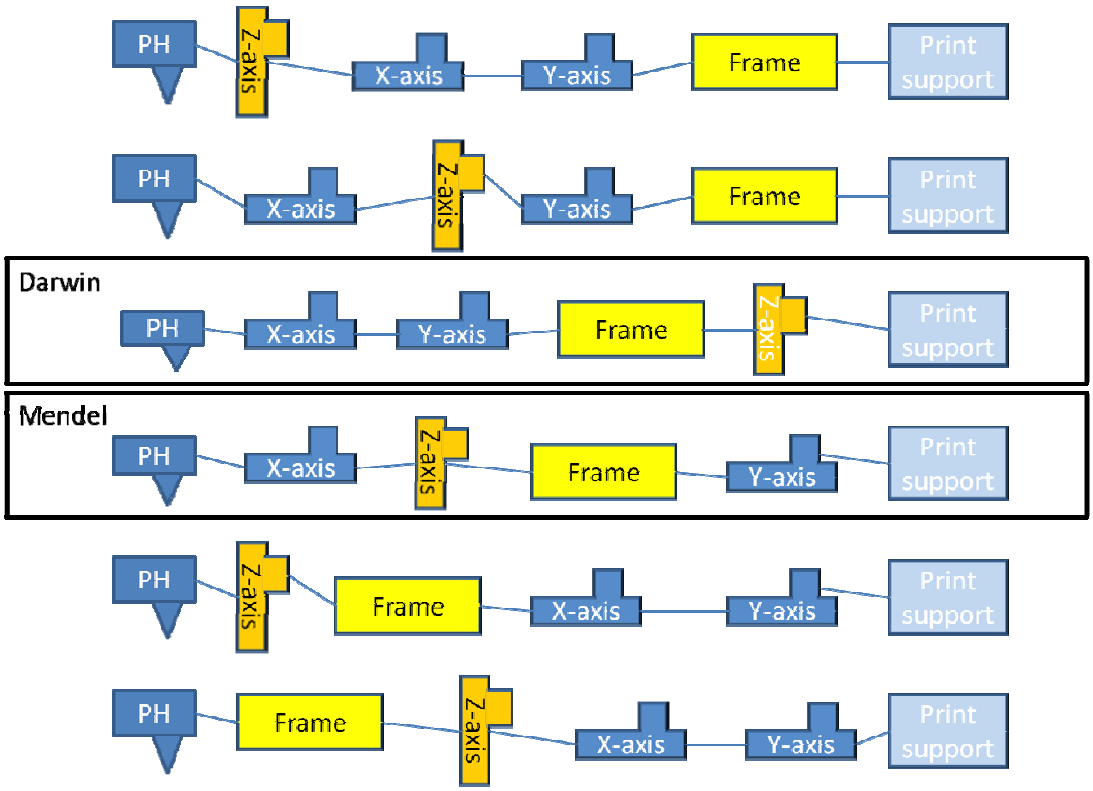

Figure 49: Structure variation at module level (PH = print head)

The two structures that are surrounded with a box represent the structures of the printer models “Darwin” and “Mendel”. Darwin copies the structure of commercially available rapid prototyping machines. Its modularized structure model is presented in Fig. 48. Mendel has been improved in comparison to Darwin. The print support is a rectangular plate. Darwin uses four synchronized lead screws for realizing its Z-movement. Mendel evolved from Darwin and is cheaper and easier to build. There the print support is assembled on the carriage of a horizontal linear unit. The second horizontal linear unit is assembled on the Z-motion unit. This can be realized with two synchronized lead screws only. Thus the number of components

is less than in the Darwin model. The module level applied to the validation study allows varying the structure at high level. At component level this would be hardly visible. The modules can be decomposed to components and integrated into other modules, like the frame, if needed. However structural variation is a task that belongs to conceptual design and can bring benefits as this example showed.

5.2.4 Summary of validation study results

Generally, modeling with library support for conceptual and functional modeling is applicable for modeling products with a large number of functions as well as for a small number of functions. The main difference is the introduction of an intermediate level. It is necessary due to the legibility of diagrams. The large number of elements forces one to split up the model into different diagrams or to use alternative presentation forms, like the table for the requirements.

Greater differences exist in component modeling. There the search of solution candidates becomes more important with a bigger number of components. Different components that are found for the embodiment of a function need to be structured. It gets insufficient just to place them into a package. The introduction of a generic component and specification of a generalization set are means to manage variants. Ports and parts that are defined at the generic component level are inherited by the specialized solution candidates.

6 Discussion

Here the main points related to the contributions of this work are discussed. The first area is link between MBSE and engineering design. Then the function library and functional modeling are discussed. In Section 6.4, the PSCS and their transformation to a model library are discussed. Usability aspects discovered during the empirical study are then presented in the following section. The discussion of the validation study ends the Chapter.

6.1 Application of MBSE methods for engineering design

The similarities at a high level in the concept phase of the design processes and systems engineering processes are presented in Chapter 2. The task of requirement analysis and functional modeling are similar. Even parts of the system synthesis can be seen as common. For these tasks, methods of systems engineering often stem from software engineering. In many cases for the abovementioned tasks, these are also valid for engineering design. With modeling examples this has been shown for requirements, use cases, functions and structure. General methods like decomposition, object orientation and inheritance have been applied likewise. However, there are differences between the two domains that cannot be neglected. Systems engineering has a clear focus on the whole system and on the interplay between subsystems whereas engineering design often has a limited scope to only a product and how it is manufactured. Consequently the methods and models cannot be used without checking their validity and purpose.

Difficulties arise in the different meaning of the terms function, behavior and physical property that have evolved from different schools of thought in product design and systems engineering. Differently coined terms lead to misunderstandings in daily communication. The standardization efforts of models for systems engineering [ISO/DIS 10303-233 2009] has led to the development of a consistent dictionary of terms for functional modeling. This is the basis for consistent modeling and the development of modeling languages and tools. In Chapter 2, it has been shown that the terms can be mapped to the commonly used terms of engineering design. Compliance to this mapping allows applying the modeling methods described for systems engineering. The modeling language SysML is compliant and provides a syntax supporting modeling according to the standard. This inhibits the integration of modeling methods that build their own ontology with a different meaning, like the KIEF ontology, using completely different meanings for function and behavior.

6.2 Uncertainty of NIST functional basis

The NIST technical report states that the NIST functional basis is reconciled from different sources. The authors of these sources had a different purpose for the use of their function taxonomies. The functional basis is verified in empirical studies where products have been disassembled and their parts assigned to functions. This does not guarantee that the functional

basis can be used for synthesis or is transferred into a computational representation without drawbacks. The intention, though, has been to have a general purpose functional basis to make functional models comparable throughout different products and modelers.

One of the critical points of the library set up is the introduction of explicit input and output flows. In the NIST technical report [HIRTZ et al. 2002], most of the functions are not limited in number and type of flow that they interact with. For the function library, it is necessary to describe the number and types of interacting flows. In the functional basis, there are functions with a different number of flows. Table 6-1 shows how many flows are needed to realize some of the functions of NIST-FB.

Table 6-1: Number of flows for functions

No. of flows	functions
one	store, provide, import export, stop
two	transmit, transfer, guide, change magnitude, distribute
three	sense, regulate, join, actuate, separate, branch, convert
more	join

In the NIST functional basis information about the number of involved flows is only given indirectly in the description of each function. To have a better idea of how many interacting flows a function can have, they have been compared with the function definitions of Koller and Hundal. Where the terms coincided in the semantics with Koller's definitions, these were used to define how many interaction points are necessary. Some functions are not clear whether auxiliary ports are necessary e.g. auxiliary energy for converting ice into water. For those cases the auxiliary pin or its respective parameter node has been modeled in the model library.

Another finding connected to the NIST functional basis is that there are some functions that are not elementary. This means that there are more functions in the library than actually needed to represent all possible functions. This can lead to the selection of more specific components and limit the number of possible solutions. Further it increases the number of functions in the functional basis. On the other hand, the functional basis is ordered in a three level hierarchy. Modeling with secondary level elements reduces the number of modeling elements if there are too many.

6.3 Transfer of the NIST functional basis into a model library

While investigating an approach for functional libraries in SysML other possibilities were checked. Due to the structure of the original functional basis and the concrete syntax of SysML, two model libraries for functions and flows are implemented. Another possibility is to multiply every operation with every flow. Operation is a function where no flows are assigned. Consequently the number of library entries is the number of flows multiplied by the number of functions. For the NIST FB there are would be more than 2385 functions that

theoretically have to be defined for model library. The actual number would be even larger since functions with multiple inputs and outputs exist.

$N_{flows} \times M_{operations} = N_{functions}$

Even if the exceptions are respected where flows are not applicable to an operation, the number will not decrease drastically. The benefit of having only one library is contrasted by a huge number of entries. Consequently, the difficulty of updating the library and finding the right function while modeling rises. Remembering that in SysML the name of the model element is unique inside one package, different names have to be created like transmit mechanical energy, transmit electrical energy, etc. It is not possible to manage the large number of similar terms by using the name as an identifier.

For realizing independent libraries of flows and functions, two additional flows are introduced. “RootFlow” represents all flows. “MaterialEnergy” supplements the flows material and energy and its children. These two flow types are only for the model library set up and should not be used in the function structures. They are a means to use the inheritance structure of the flow library. This mechanism allows exchanging the flows in the activity diagrams representing function structures with any flow that inherits from two supplement flows. When a new flow is defined, it simply has to be added to the existing inheritance tree of flows. Then it can be used for modeling. This solution can be abused in a way that the modeler does not specify the flow types and ignores the validation results of the function structure.

Both methods of building model libraries support the objective of modeling flows in systems or products. The critical point is the model evolvement where one may add some specific flows. In case of a multiplied library, at least one additional operation has to be added, and the number of functions with the same operation in the library increases. The effects of changes of flows are different. While with one library the change of a flow implies exchanging all functions that use that flow from the library, with two independent libraries the modeler at any time can choose a more specific flow type.

Like a lexicon, the libraries of flows and functions provide the vocabulary for building function structures in general. Specific problems can be treated by creating sub graphs. Modeling specific problems, it may occur that some of the elementary functions forming a sub graph for a special reoccurring problem can also be reused. For instance, the transport problem, that always involves a transported load, energy to create displacement and a specific path that the load is transported. In this case, a specific transport function is implemented as an activity that uses several library elements. It then is referenced like the functions of the library. However, it is not considered to extend the library with sub-graphs.

6.4 Discussion of component model library set up

Although only a small portion of the whole PSCS is transferred to a SysML library, some issues can be identified: First, the large amount of data. Processing all of it is not the best choice since only meaningful function relevant items are needed. A selection of relevant segments or main groups may sort out potential solution candidates from other domains. This

easily can happen since individuals rely on their own experience first. Things from other domains are usually out of the horizon of experience of an individual working within a certain domain [TERNINKO et al. 1998].

During the transfer of the information to the model library, elements connected with service, repair and maintenance filtered out. These elements are found in commodity class level next to relevant components. The fact that elements of different meaning are stored inside the PSCS was already discovered in an ontology project on E-Cl@ss [HEPP 2004, HEPP & ABRAMOWICZ 2007]. Applying the right filter for conceptual design is the challenge.

After applying a filter on all commodity class levels of a segment there are still a large number of function relevant components left. For each, object ports and physical ports have to be added. This is possible in the same way that e-Cl@ss is built up- by a community and perpetual revisions of the terms. However for the test amount of about 260 items, this has been done manually. Blocks inside SysML are identified by name. As a consequence, the number of freely available names for modeling will decrease. The names of some commodity classes start with the same letters like “electrical…” and causes problems in search. The number of matches is too high for a fast identification of the desired item. One of the stated aims is to present the right modeling element at the right time. For a rich repository of components, other solutions have to be found. One idea is an independent database that produces SysML blocks on demand and imports them to the modeling tool. This can be considered as sufficient since only at discrete points of time during the design process is the search for solution candidates run. In the validation study, the component model library is deleted from the modeling project after the selection of the solution candidates.

The direct search in the model library for components was worked around by filtering the source spread sheet for input and output ports. This worked out as a quick possibility to identify all solution candidates. One idea is to have a query that allows searching the model for blocks owning specific ports like with the structured query language (SQL). A query statement can be:

SEARCH *block* FROM *model library* WHERE (*port* HAS direction = in AND type = electrical energy) AND (port has direction = out AND type = rotational energy)

The output will be a list with all blocks that have one input port typed with electrical energy and one output port typed with rotational energy.

For the modeling study, only one component library has been taken into account. The modeling of association blocks for defining component to component connections shows potential for another model library. The definitions can be reused several times. They provide a detailed definition of how components can be connected and they use parts from PSCS e.g. screws, plugs, sockets, flanges, fittings, etc. The effort to define one component to component connection is high compared to the component itself. For the example shown in Chapter 4, Fig. 27, seven blocks must be defined to define a component to component connection. A simpler definition at least needs five blocks for a definition. The component blocks then can inherit from the client and supplier definition at each end of an association block.

The use of object ports for the identification of solution candidates is another point of discussion. The ports that are relevant for the connection of components are physical ports.

Are object ports the right means to identify solution candidates? Initially, matching input and output structure of functions and components is convincing but the search does not work as intended. From the SysML point of view, ports are treated as inner elements of blocks. Any other inner element can provide the same modeling function as ports. There could be an attribute that indicates the embodied input and output flows. Better however would be a possibility to recognize the embodiments for functions by attributes that are specific for the components, e.g. rated voltage and rated current of an electric drive is recognized as electrical energy input. This would save the definition of additional elements on component blocks.

Despite the challenges due to the amount of items, catalogs and PSCS will always be a great source of information. In case of the modeling example it has proved that several solutions can be found inside the classification standard. This justifies a further investigation of the integration of that information into a model library. Structuring a model library from scratch or relying on domain specific databases does not make sense for domain spanning modeling. Finally, each specific component can be specialized from a block representing a commodity class item.

6.5 Usability Aspects of modeling with library support

While working with students, who had the task to model a system with SysML there was always the complaint that the CASE tool was not easy to use. A lot of complaints were connected to the fact that the modeler had to define every element before he or she can use it in diagrams. The students were used to simulation tools like Matlab or Modelica that come with model libraries. They expected the same convenience with a CASE tool. This is not the only aspect of usability. Table 6-2 summarizes relevant usability aspects that are explored in the following sub sections.

Table 6-2: Overview of usability aspects

Aspect	Refers to	Comment	Section
Model library		Provides a standard convention for modeling functions	6.5.1
Drawing diagrams	CASE tool	Speed and convienice of modeling	6.5.2
Syntax rules	SysML	Guide formal modeling	6.5.3
Self defined rules		Improves functional modleing	6.5.4

6.5.1 Usability improvements for model libraries

The first aspect is the use of a common basis that is provided by a model library. Every element defined in advance avoids an interpretation of how to use the element by the individual designer. The model library improves modeling in several areas compared to paper-based functional modeling and modeling without libraries. First, it replaces the text book or look up table for functions and flows. Descriptions explaining the library element can

be integrated in the model library. In Figure 34, this is demonstrated with the pop-up description that appears when the mouse arrow is placed over the element. Then, it is not needed to define every modeling element according to a template since the template itself is stored in the model library. Finally, additional elements in conjunction with the model element can be defined. In case of the function library, the additional elements are the activity parameter nodes with their types and directions. The rules for placing inputs and outputs are shifted from a description to the model library due to the definition of activity parameter nodes in the function library. When referencing the activities of the function library, pins are created according to the actions in the diagram. Thus, number, direction and general type of a pin is predefined

Write protection of the model library is a means to prevent that the model library is modified. Consistent terms are necessary for a more compatible modeling. The library elements stay the same or only can be changed in a controlled way contrasted to a repeated redefinition of a function set for every new model.

The component library is different, though. With a huge number of elements and a changing offer structure from the suppliers' side, changes will be inevitable. The structure, following the structure of E-Cl@ss will not change that fast and guarantees that components can be identified by the unique ID provided from E-Cl@ass.

Another area is the ease of modeling; the efforts are shifted towards the model library. Creating a function structure without library use, e.g. as in Fig. 16 involves the creation of one activity, one activity diagram, three parameter nodes, seven actions, 12 pins and 10 relations. The modeler working without a library and reaching the same formal model has to define all the blocks needed as pin types and parameter node types and has to define all seven actions according to the NIST-FB. After modeling, she or he cannot reuse the actions, because they can only be used once in the context of the activity.

A library user models differently. Of course the diagram, the three parameter node and ten relations have to be created as well. But, for all actions, the modeler can just drag predefined activities out of the library and drop them on the diagram pane. They come with all pins predefined in the library. The pins are not specified yet. However, they easily can be specified from the context menu. All the necessary flows are specified in the flow library. This obvious improvement for the working process can be expressed in figures: The library user does not define a block, references seven activities instead of creating actions and does not need to create a single pin from scratch. The definition of a pin is a four step process, where the pin itself, its stereotype, direction and type has to be defined. In the case of using libraries, just the pin type has to be set.

6.5.2 Usability aspects of drawing SysML diagrams

This usability aspect is related to how CASE tools work. Normally, every element that is used in a diagram has to be defined upfront. There is a clear separation between definition and usage. When modeling an activity diagram to show a function structure, every action, every activity parameter, every pin and every flow has to be defined upfront. Without a model library this is tedious work. With a model library, the elements from the model library can be

used. The preferred way is to drag functions directly from the library to the diagram. This makes sure that the element is really from the library and not a similar spelled definition from another package. This saves modeling time. Still there are several steps, like to set the needed object flow and to give a name to the actions. But this needs only a third of the time than working without model libraries.

In general, CASE tools support drawing diagrams and generate an internal representation at the same time. This is a difference to using a presentation tool like Microsoft PowerPoint or Visio that allows drawing diagrams but do not offer formal control to the diagrams. CASE tools in contrast, allow presenting or hiding inner elements of the nodes on demand, e.g. flow ports or values on block nodes.

6.5.3 Usability aspects of SysML syntax

The SysML syntax sets the formal rules that have to be obeyed while modeling. In Section 4.2.4 the relevant syntax for functional modeling has been presented. Here it is shown how the syntax rules are implemented and affect the modeling. In the SysML specification [2010b] or in the UML Superstructure [OMG 2010c] constraints are described for every UML/SysML syntax element. They are described as java snippets or as clauses in the Object Constraint Language (OCL). CASE tools that comply with the specification implement a validation rule for maintaining constraints. Some of the constraints have concrete effects on functional modeling with library support

The **parameter synchronization** rule is a SysML validation rule that checks if the call behavior action and the called behavior use the same types for corresponding pins and activity parameters. If the pin type of an action in the diagram is changed to a kind of flow not in the line of inheritance of the activity parameter in the model library, the SysML validation engine issues a warning message. For instance, when changing the pin type of the function “Store” in Figure 35 to “TasteSignal”, it is not inherited from “MaterialEngergy” any more. After the validation check, the call behavior action is highlighted in the diagram as well as in the containment tree and a warning message is displayed. The CASE tool offers some options to repair the broken parameter synchronization.

Another SysML validation rule checks the types at the source and target end of an object flow relation. An object flow connects the pins of two actions. In case of different types at the lower and upper end, a validation engine displays an error message. A context menu allows selecting whether the source or target type should be changed to correct that error.

With these two examples it is shown, that the SysML syntax provides for additional formality for functional modeling. Further the concept of the function library is supported in a way that the absence of a flow needed for the execution of a function forces the modeler to select another function out of the library. For instance, intending the function “Transport” for energy flows does not work due to the predefined flow type “Material”. The modeler has to change to another function e.g. “Transfer” that works with any flow. This can be formulated in reverse, so that the existence of a certain flow can force the modeler to use a certain function.

6.5.4 Self-defined rules for usability improvement

The definition of the function library, however, has opened a gap that allows for inconsistent modeling. The definition of functional models allows connecting functions with the flows material, energy and signal. To provide a means for selecting any of the flows or only material or energy, the flows “RootFlow” and “MaterialEnergy” are introduced. The SysML syntax does not prohibit the use of these flows. This can be solved by a specific rule imposed on the pins of actions representing functions. It is implemented as a constraint using the OCL.

The OCL specification [2010a] describes the Object Constraint Language as a language that allows writing unambiguous constraints. Some of the validation rules of Magic Draw’s SysML profile are written in OCL, while others are binary. Binary validation rules are of complied java code. The CASE tool Magic Draw offers two types of validation. Active validation is carried out during modeling; passive validation is carried out at a discrete point of time e.g. when the model is stored or a button is pressed. In general, constraints ensure the correct use of the modeling language and formality of the model.

In the function model library the activity parameter nodes of functions are typed with “RootFlow”, ”MaterialEnergy” or one of the three primary level flows. After inserting an action representing a function the modeler has to set the type of flow to the actually desired flow. The use “RootFlow” and “MaterialEnergy” results in an undefined function structure due to a not specified flow. To avoid this case, the validation rule checks that there is no “RootFlow” or “MaterialEnergy” used for typing pins in function structures. Figure 50 shows the effect if the validation engine detects a rule that evaluates to false. The flow should be set when the library element is referenced. To remind the modeler immediately when the activity is applied, the validation type is set to active validation.

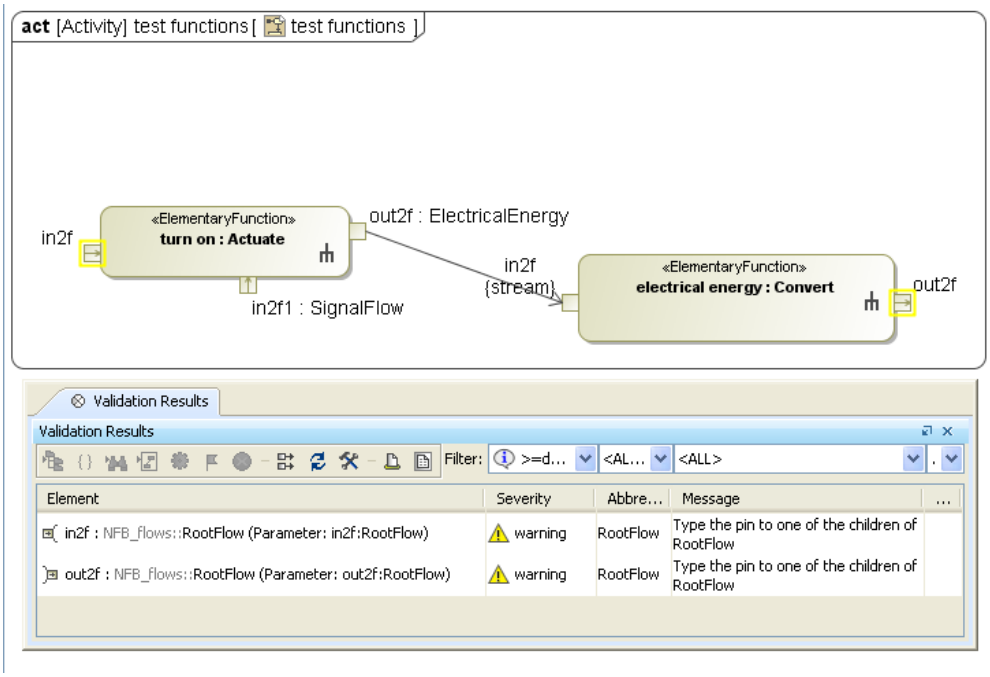

Figure 50: Validation of pins in function structure

To avoid any changes in the model, the creator of a validation rule can rely on OCL. It can be used for querying the model but not to evoke changes in the model. The result of a validation can be true or false. In case of false, a message is issued. The validation engine only evaluates constraints of the stereotype <<validation rule>>. Therefore, a constraint with that stereotype has to be created. The OCL clause specifies the context the type of condition and the rule. For the simple rule that highlights all pins typed with “RootFlow” the OCL clause is:

Context pin inv
Self.type.name <> ‘RootFlow’ or ‘MaterialEnergy’

The word after “Context” specifies to which elements the constraint clause applies. The key word “inv” is the abbreviation for invariant, which means that the rule is valid at any time. Self refers to the context pin where the name attribute of pin type is queried. The clause evaluates to true if there is no pin with the type “RootFlow” or “MaterialEnergy”. In the diagram of Fig. 50, there are two pins typed “RootFlow”, highlighted by the validation mechanism of the CASE tool.

6.6 Discussion of modeling study and validation study

The modeling example and validation study showed that many modeling tasks can be performed in an integrated manner in a CASE tool working with the SysML language. With the validation study, the applicability to a more complex system is shown. This is termed the scalability of modeling since the modeling method can be applied to problems of any scale. The core however is the use of the flow and function model library. It proves that the SysML syntax supports using a library for functional modeling.

In functional modeling, SysML provides support through the mechanism of call behavior actions. This allows specifying functions in a top down approach. The decomposition of functions turns the function into a behavior that is described by more detailed functions until the model library can be used. The model library supports functional modeling by providing functions and predefined flows in the library elements for functions. The SysML syntax of call behavior actions requires the synchronization of activity parameters and pins. Thus, the modification of flows is limited by the predefined type of the library element. This leads to more formality. Finally, faster modeling of function structures than without the library and a validation rule determining non-specified pins contribute to improved usability of the function libraries and lead to more consistent models.

In the modeling example with the luggage cover it was possible to find only a few components out of the component library. This was due to the very specific domain of passenger car interiors. In contrast, for the validation study, many components have been taken out of the component library. The 3D printer example was designed for using many off the shelf components. This shows that it is possible to identify components with the matching of pins to object ports. Consequently, the high number of solution candidates influenced the modeling. It has been necessary to introduce generic components as supplements for all solution candidates. This increases the flexibility to create different solutions by exchanging the generic components with specific solutions candidates. The introduction of generalization

sets for the solution candidates comply with the rows of morphologic boxes that are used to order solution candidates in paper-based methods. The generalization sets are a possibility to present the variants of solution candidates.

The added complexity of the validation study led to the introduction of an additional decomposition step. This can easily be managed with a CASE Tool and SysML. The module level first has been applied to functional modeling and then to the modeling of the component structure. The larger number of modeling elements is also a driver for further decomposition since the diagrams with a larger number of nodes and relations easily get cluttered and are not legible when printed out on a DIN/A4 sheet.

For the definition of component to component connections there are two possibilities: definition of ports for every component or definition of reusable associations via association blocks. The definition of ports is simpler and faster. But association blocks allow a more detailed definition at several levels, e.g. a power supply cable (=component level) can be decomposed to the single wires (=part level). Association blocks show their strength with complex connections. Both definitions can be used for component modeling.

An additional aspect is the modeling of variants. With the use of generic components as supplements, a general structure is created. By the replacement of generic components with specific ones, variants are created due to the combination of different solution candidates. Structural variation is different. A structural variant can be created by rearranging existing components. Due to component variants and structural variation, a large number of valid solutions can be created. At module level, it has been possible to show a structural variation, a method that should be applied in conceptual design. This variation is done without an influence on the functional model.

The existence of many solutions quickly leads to the question which of the solutions should be selected. Here, evaluation and simulation tools come into play. The CASE tools only can support the simulation of activity diagrams to show if all actions can be executed or if there are some actions that are not executed at all. For the simulation of physical properties the model has to be linked or converted into a specific simulation model, e.g. a Modelica model. The coupling of SysML and Modelica is described by JOHNSON et al. [2008].

7 Conclusions

The similarity of product design and systems engineering has much potential for synergies. Especially in conceptual design and model based systems engineering the synergies can be related to single engineering tasks like requirement analysis, functional modeling and component modeling. VDI guideline 2221 already takes into account computational support but does not offer methods for model integration, like modeling languages. Requirements for modeling requirements, functions or component structures have been defined already for paper-based methods. These requirements can now be supported by a formal modeling language like SysML. Existing representations for functional modeling have been investigated, formulated as modeling principles and matched to the SysML syntax. Better communication, because of the use of diagrams, the reuse of models and an improved comparability are reached due to the support of model libraries. The key to realize this potential is the clear definition of terms in a meta-model as well as for modeling itself. Where it is possible to reach a common understanding due to the adoption of core terms from either side, the possibility for the use of common modeling methods for both product design and systems engineering is given here. For modeling conceptual design tasks with SysML, it is successfully presented in Chapters 4 and 5 after the terms have been clarified in Chapter 2. For other approaches, where the deviation of the meaning of core terms is too big, it does not make great sense to force modeling into an incompatible syntax of a modeling language.

Table 7-1: Comparison of different levels of modeling

Paper-based	**With SysML**	**With SysML and library**
single models for each aspect	Integrated models	integrated models
no links to other models	Links to models of other aspects, i.e. requriememts to behaviors	links to other model aspects, e.g. functions and flows
no meta model	modeling language metamodel, i.e. concrete syntax	domain specific meta model i.e. functional model
semantically not necessarily clear	clear semantic of language	clear semantic of modeling elements
manual comparison of models	tool supported comparison	tool supported comparision
drifting semantics (induvidual interpretation)	individual definition of model elements	prefined modeling elements
inconsistencies of partial models	inconsistencies can be avoided (by linking to partial models)	mapping of partial models is defined
difficult to reuse (copy & paste)	reuseable models	model libraries designed to reuse
function decompostion by redrawing	decompostion due concrete syntax e.g. call behavior actions, parameter synchronization	additional support by flow library, e.g. by using same flow types at different decomposition levels

The aspects of improved modeling due to the use of SysML and model libraries are presented in a comparison shown in Table 7-1. The use of SysML allows creating an integrated model where various aspects can be linked to one another, e.g. a requirement can be liked to a test case or a block that satisfies that requirement. Working paper-based only, there is no explicit metamodel. The modeling language provides as a metamodel the modeling elements forming the concrete syntax, e.g. an action. Thus, the meaning of an action is defined. Model libraries,

in contrast, contain modeling elements of the same semantic meaning for the modeling at the instance level (M1, see Fig. 9), e.g. functions as a transformations of inputs to outputs.

Model libraries prevent the drifting of the semantic of library elements. This is a consequence of growing experience of the designers that cause a change in the interpretation of a term and affect the usage of it with paper-based models. The modeling element in the library, however, does not change. The other aspects relate to the reuse of elements. With paper-based modeling, this is only possible by re-drawing a diagram while with model libraries, modeling elements are designed for reuse. The matching between functions and components avoids inconsistencies in the inter-model connection of functions and components. And the decomposition of functions and component structures is supported by SysML modeling tools and model libraries.

7.1 Feasible library support for functional modeling

For the definition of a model library, the SysML specification has been investigated. For the flow library, inheritance and the generalized “RootFlow” as a root node are exploited. Flows are predefined as blocks. Functions are activities that use the flows as types for activity parameter nodes. The elements of the SysML model library are definitions ready for use. The effort of definition is saved whenever an element of the model library is referenced. The specific design of the function model library as a package of activities makes sure that in activity diagrams used for functional modeling only elements with predefined flows can be used. During modeling, the modeling example and the validation showed that the use of the model library improved modeling. Qualitatively, reduction of modeling time is noticed due to the modeling steps for definition of the modeling elements that are no longer needed. These are the steps that are needed for the definition of flows and for the creation of all pins needed for the representation of flows. All functions are well defined using the right flows. The single function structures are comparable with one another, since the function nodes have the same meaning. In case of doubts about the meaning of function, the model library contains all necessary documentation that explains a flow or a function. Summarizing all points, the implementation and use of the function library leads to a convenient, formally correct and fast way to model function structures. The use of model libraries for any abstract concept similar to functional modeling is recommended. It is the means to support the interpretability and the comparability of conceptual functional models. Kruse et al. [2012] presented another example for functional modeling, that shows the applicability of functional modeling with SysML.

7.2 Library support for component modeling

The purpose of the component library is to provide a large number of solution candidates for the embodiment of functions. The choice for the structure of a component model library is to take over the structure of E-Cl@ss, a four level hierarchy. This choice is justified since E-Cl@ss is a product and service classification standard that holds a great number of commercial products and is updated permanently by a community of component suppliers. In a pragmatic way all services and auxiliary agents are sorted out and object ports added to the

selected machine elements. The challenge still is to identify whether a component embodies one or more functions and is function relevant or an element is a part, like a washer or a screw. The selection of function relevant components and the assignment of object ports have been done by hand. Transferred to the CASE tool the great number of model library items disturbs modeling block definition diagrams or internal block diagrams. Whenever a block or a part property is drawn in SysML the case tool suggests a name from the existing blocks. The list of potentially suggested modeling elements grows with every library element. Splitting the model library into different domains excludes solution candidates and is not seen as a solution. Therefore, a large model library (more than 300 elements) should only temporarily be connected to the modeling project and disconnected after the relevant model elements are retrieved.

In order to find all embodiments for a function, there has to be a search mechanism that searches for inputs and outputs of the components represented by input and output ports. The implementation of a search, as presented in the discussion, failed due to implementation issues of the CASE tool. The number of components is large compared to the number of functions. This prevents all components from being seen at once. Browsing the model library as with functions does not work. Another aspect is that components are not selected by name as with functions but by the specific properties of their object ports. This mechanism is used to avoid that the modeler gets trapped in a certain region of the component library while in another region a valid solution stays undiscovered. Further, designers do not know all the specific terms of the components, therefore the identification by object ports is helpful.

It can be concluded that for an excessive library, a CASE tool needs a library management that offers different levels of model library access, implements sophisticated search and a possibility to compare the library with other kind of libraries, e.g. E-Cl@ss.

7.3 Conclusions from modeling results

Modeling with CASE tools and SysML is a convenient way to model. It allows definition of elements, without a graphical representation, where elements are just added to the containment tree and to work graphically by drawing diagrams. The graphical presentation is always linked to the computational representation, which is the great strength of CASE tools. Due to this, models can be formally validated and inter-element links analyzed. This cannot be done with separated spread sheets or sheets of paper. Task description, requirement formulation, functional modeling and the search for embodiments are possible in SysML. The simulation of physical properties is possible trough the integration of other tools. Indirectly the integration with a simulation tool can be supported by modeling libraries mirroring the model libraries of the tool to integrate. The component model library then has to be extended by a simulation model library that parallels the component structure.

Modeling the example with the luggage compartment cover has already shown some potential for providing a structured process that involves drawing a number of diagrams. The validation study, however, shows that the leverage of library supported modeling grows with the number of involved elements. More elements are taken out of the function library and more variants are found in the component library. It is important that the SysML syntax supports the

decomposition of function structures as well as the decomposition of component structures. This makes SysML applicable for large systems and projects. The strength for decomposition is a disadvantage when models should not be decomposed. In this case, the diagrams rapidly grow large, confusing and illegible when plotted out in a two page format. The diagrams showing the structure of the functional modeling library and the hierarchical decomposition already reach a size that is not printable on the page of a book without being illegible. The management of variants and the variation of structures are two points that are worthwhile for further investigation. The validation example proved that structural variants matter since the Mendel printer is simpler to build and cheaper than the Darwin printer.

7.4 Outlook

Modeling with model libraries in SysML showed its potential. For functional modeling it shows maturity ready for industry where it needs some work on the component definition for the library. Further research in context of SysML is necessary to quantify the increase of modeling speed that can be reached with library support that is indicated qualitatively in this work. This can be done with a comparative study with different levels of tool support. Another field of research is how to present oversize diagrams and still convey relevant information although details cannot be shown any more. The management and search mechanism for large libraries needs to be investigated. Further there are open questions regarding the integration with simulation tools, methods for variation, validation and analysis.

With the publication of the standard for product data exchange for systems engineering (ISO 10303-AP233) it is a question of time until SysML will be used productively in industry projects. SysML and the community of users is evolving and growing. Until now CASE tools are driven from the requirements from software engineering, yet the requirements from product design are different, therefore issues for improvements of the tools have to be investigated and addressed by the tool vendors.

8 References

[ALTSHULLER 1994]
Altshuller, G.: And Suddendly the Inventor Appeared. 1st ed. Wrocester, MA: Technical Innovation Center, Inc. 1994. ISBN: 0-9640740-2-8.

[ALVAREZ CABRERA ET AL. 2009]
Alvarez Cabrera, A. A.; Erden, M. S.; Tomiyama, T.: On the Potential of Function-Behavior-State (FBS) Methodology for the Integration of Modeling Tools, CIRP Conference. Cranfield, 2009.

[ANDERL & TRIPPNER 2000]
Anderl, R.; Trippner, D. (Eds.): STEP Standard for the Exchange of Product Model Data. Stuttgart, Leipzig: B.G. Teubner 2000. ISBN: 3 519 06377-8.

[ANDREASEN 1991]
Andreasen, M. M.: Design Methodology. Journal of Engineering Design 2 (1991) 4, p. 13.

[BAHILL & DANIELS 2003]
Bahill, T.; Daniels, J.: Using Object-Orienated and UML Tools for Hardware Design: A Case Study Systems Engineering 6 (2003) 1, p. 21.

[BALMELLI ET AL. 2006]
Balmelli, L.; Brown, D.; Cantor, M.; Mott, M.: 2006, p. 18.

[BOOCH ET AL. 1998]
Booch, G.; Rumbaugh, J.; Jacobson, I.: The Unified Modeling Language User Guide. Reading, Massachusetts 01867: Addision Wesley Longman, Inc. 1998. ISBN: ISBN.

[BRIMBLE & SELLINI 2000]
Brimble, R.; Sellini, F.: The MOKA Modelling Language. In: Knowledge Engineering and Knowledge Management: Methods, Models, and Tools: 12th International Conference, Ekaw 2000, Juan-Les-Pins, France, Year. Springer

[BRUNETTI & GOLOB 2000]
Brunetti, G.; Golob, B.: A feature-based approach towards an integrated product model including conceptual design information. Computer-aided design 32 (2000) 14, p. 877-887.

[BUUR 1990]
Buur, J.: A Theoretical Approach to Mechatronics in Design. PhD, Technical University of Denmark, Lyngby (1990).

[CHANDRASEKARAN 2005]
Chandrasekaran, B.: Representing function: Relating functional representation and functional modeling research streams. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM) 19 (2005) special Issue, p. 10.

[CHANDRASEKARAN & JOSEPHSON 2000]
Chandrasekaran, B.; Josephson, J. R.: Function in Device representation. Computers in Engineering 16 (2000) 3-4, p. 162-177.

[CHILDERS & LONG 2004]
Childers, S. R.; Long, J. E.-. A Concurrent Methodology for the Systems Engineering Design Process. 2004, p. 6.

[DOD 2001]
DOD: Systems Engineering Fundamentals. Fort Belvoir, Virginia: Defence Acquisition University Press 2001.

[DOD/CIO 2012]
DoD/CIO: DoDAF - DoD Architecture Framework Version 2.02 DoD Deputy Chief Information Officer <http://dodcio.defense.gov/dodaf20.aspx> 2012 July, 11th 2012.

[DUBBEL 2007]
Dubbel, H.: Taschenbuch für den Maschinenbau. Berlin: Springer Verlag 2007. ISBN: 3-540-22142-5.

[EACOE 2011]
EACOE: The Enterprise Framework
<http://eacoe.org/pdf/EACOE_Enterprise_Framework.pdf> 2011 31.01.2011.

[ERDEN ET AL. 2008]
Erden, M.; Komoto, H.; van Beek, T.; D'Amelio, V.; Echavarria, E.; Tomiyama, T.: A review of function modeling: Approaches and applications. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM) 22 (2008) 02, p. 147-169.

[ESTEFAN 2008]
Estefan, J. (Ed.): Survey of model-based systems engineering (MBSE) methodologies. Technical report, INCOSE MBSE Focus Group, 25 Mai 2007 (2008).

[FENVES ET AL. 2005]
Fenves, S.; Sriram, R.; Subrahmanian, E.; Rachuri, S.: Product Information Exchange: Practices and Standards. Journal of Computing and Information Science in Engineering 5 (2005) 1, p. 238-247.

[FENVES ET AL. 2004]
Fenves, S. J.; Foufou, S.; Bock, C.; Sudarsan, R.; Bouillon, N.; Sriram, R. D. (Eds.): CPM 2: A Revised Core Product Model for Representing Design Information. US Dept. of Commerce, National Institute of Standards and Technology (2004).

[FRANK 2006]
Frank, U.: Spezifikatioinstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme. Dissertation, Universität Paderborn, HNI Verlagsschriftenreihe, Band 175, Paderborn (2006).

[FRIEDENTHAL ET AL. 2008]
Friedenthal, S.; Moore, A.; Steiner, R.: A Practical Guide to SysML: The Systems Modeling Language. 1 ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 2008.

[FUCHS 2004]
Fuchs, D. K.: Prinzipien für die Problemanalyse in der Produktentwicklung. Technische Universität München, München (2004).

[GAUSEMEIER ET AL. 2009]
Gausemeier, J.; Frank, U.; Donoth, J.; Kahl, S.: Specification technique for the description of self-optimizing mechatronic systems. Research in Engineering Design 20 (2009) p. 201-223.

[GENESERETH & FIKES 1992]
Genesereth, M. R.; Fikes, R. E. (Eds.): Knowlege Interchange Fromat Reference Manual, Version 3.0. Stanford University, Stanford, CA (1992).

[GERO 1990]
Gero, J.: Design Prototypes: A Knowledge Representation Schema for Design. AI Magazine 11 (1990) 4, p. 26-36.

[GRABOWSKI & ANDERL 1991]
Grabowski, H.; Anderl, R.: Advanced Modeling for CAD /CAM Systems. (1991)

[GRABOWSKI ET AL. 1993]
Grabowski, H.; Anderl, R.; Polly, A.: Integriertes Produktmodell - Entwicklungen zur Normung von CIM. Berlin: Beuth Verlag GmbH 1993. ISBN: 3-410-12920-0.

[GRÜNDER 2009]
Gründer, W.: Roloff/Matek Bauteilekatalog. 2009,

[HEPP 2004]
Hepp, M. (Ed.): A Methodology for Deriving OWL Ontologies from Products and Services Categorization Standards. Florida (2004). (Available at SSRN: http://ssrn.com/abstract=689184 or http://dx.doi.org/10.2139/ssrn.689184)

[HEPP & ABRAMOWICZ 2007]
Hepp, M.; Abramowicz, W.: ProdLight: A Lightweight Ontology for Product Description Based on Datatype Properties Business Information Systems. In: Springer Berlin / Heidelberg 2007, p. 260-272. ISBN: 978-3-540-72034-8. (Lecture Notes in Computer Science).

[HIRTZ ET AL. 2002]
Hirtz, J.; Stone, R.; McAdams, D.; Szykman, S.; Wood, K.: A functional basis for engineering design: Reconciling and evolving previous efforts. Research in Engineering Design 13 (2002) 2, p. 65-82.

[HOFFMAN 2006]
Hoffman, H.-P.: SysML based Systems Engineering Using a Model-Driven Development Approach. In: INCOSE Internationa Symposium, Orlando, FL, Jul. 12 Year.

[HOLT & PERRY 2008]
Holt, J.; Perry, S.: SysML for systems engineering. London: 2008. ISBN: 978-0-86341-825-9. (Professional applications of computing series).

[HUBKA & EDER 1988]
Hubka, V.; Eder, W. E.: Theory of technical systems. 1st ed. London: Springer 1988.

[HUNDAL 1990]
Hundal, M. s.: A Systematic Method for Developing Function Structures, Solutions and Concept Variants. Mechanisms and Machine Theory 25 (1990) 3, p. 243-256.

[IEEE1471 2000]
IEEE1471, Systems and software engineering — Recommended practice for architectural description of software-intensive systems: IEEE Std 1471-2000. New York: 2000.

[ISO13584 2010]
ISO13584, Part 42: Description methodology: Methodology for structuring parts families Industrial automation systems and integration - Parts library: . Geneva: 2010.

[ISO 10303-233 2009]
ISO 10303-233, Application protocol: Systems engineering: Industrial automation systems and integration — Product data representation and exchange Genava: ISO 2009.

[ISO/10303 1994]
ISO/10303, Product data representation and exchange: Industrial automation systems and integration. Genava: ISO 1994.

[ISO/DIS 10303-233 2009]
ISO/DIS 10303-233, Application protocol: Systems engineering: Industrial automation systems and integration — Product data representation and exchange Genava: ISO 2009.

[ISO/IEC26702 2007]
ISO/IEC26702: ISO/IEC 26702 Systems Engineering - Application and Mangement of the Systmems Engineering Process. 2007.

[ISO/TS10303-1453 2009]
ISO/TS10303-1453, Application module: Function based behavior: Industrial automation systems and integration — Product data representation and exchange. Geneva, Swiss: 2009.

[JOBE ET AL. 2008]
Jobe, J.; Johnson, T.; Paredis, C.: Multi-Aspect Component Models: A Framework for Model Reuse in SysML, ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2008). New York, 2008.

[JOHNSON ET AL. 2008]
Johnson, T.; Paredis, C.; Burkhart, R.: Integrating Models and Simulations of Continuous Dynamics into SysML. In: 6th International Modelica Conference, Bielefeld, Germany, March, 3rd 2008 Year. p. 11. (Modelica Conference)

[JONES ET AL. 2010]
Jones, G. J.; Wagner, D.; Dvorak, D.; Mishkin, A.: Human-Rated Automation and Robotics. National Aeronautics and Space Administration 2010. (A Design Handbook for Robust Operations, Controll and Fault Management).

[JONES ET AL. 2011]
Jones, R.; Haufe, P.; Sells, E.; Iravani, P.; Olliver, V.; Palmer, C.; Bowyer, A.: RepRaP - the relicating rapid prototyper. Robotica 29 (2011) p. 177-191.

[KEMMERER 1999]
Kemmerer, S. J. (Ed.): STEP, The Grand experience. Washingtion, D.C.: U.S. Government Printing Office 1999. (Nist Special Publication).

[KERZHNER & PAREDIS 2009]
Kerzhner, A. A.; Paredis, C.: Using Domain Specific Languages to Capture Design Synthesis Knowledge for Model-Based Systems Engineering International Design Engineering Technical Conferences & Computers and Information in Engineering Conference San Diego, 2009.

[KLEIN 2000]
Klein, R.: Knowledge modelling in design – the MOKA framework. In: International AI in Design conference, Year. Kluver p. 77-102.

[KOLLER 1975]
Koller, R.: Konstruktionslehre für den Maschinenbau: Grundlagen zur Neu-und Weiterentwicklung technischer Produkte mit Beispielen. 1 ed. Berlin: Springer 1975.

[KOLLER 1998]
Koller, R.: Konstruktionslehre für den Maschinenbau: Grundlagen zur Neu-und Weiterentwicklung technischer Produkte mit Beispielen. 4 ed. Berlin: Springer 1998. ISBN: 3-540-07444-9

[KOLLER & KASTRUP 1998]
Koller, R.; Kastrup, N.: Prinziplösungen zur Konstruktion technischer Produkte. Berlin: Springer 1998.

[KOMOTO & TOMIYAMA 2010]
Komoto, H.; Tomiyama, T.: Computational support for system architecting, International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. Montreal, Quebec, Canada, 2010.

[KOZIEL ET AL. 2006]
Koziel, G.; Höhnle, T.; Dehne, H.: Grundsatzleitlinie des E-Cl@ss e.V. <www.eclass.de> 2011 28.01.2011.

[KRAUSE ET AL. 1993]
Krause, F.-L.; Kimura, F.; Kjellberg, T.; Lu, S. C.-Y.: Product Modelling. Annals of the CIRP 42 (1993) 2, p. 11.

[KREIMEYER 2010]
Kreimeyer, M.: A Structural Measurement System for Engineering Design Processes. München: Verlag Dr. Hut 2010. ISBN: 978-3-86853-399-6.

[KRUCHTEN 2003]
Kruchten, P.: The Rational Unified Process: An Introduction. 3rd ed. Reading, MA: Addison Wesley Professional 2003.

[KRUSE ET AL. 2012]
Kruse, B.; Münzer, C.; Wölkl, S.; Canedo, A. M.; Shea, K.: Workflow and Modeling Conventions for FBS Modeling of Mechatronic Systems in SysML using Libraries, IDETC. 2012.

[KURTOGLU ET AL. 2009]
Kurtoglu, T.; Campbell, M. I.; Arnold, C. B.; Stone, R. B.; Macadams, D. A.: A Componet Taxonomy as a Framework for Comutational Design Synthesis. Journal of Computing and Information Science in Engineering 9 (2009) Journal of Computing and Information Science in Engineering,

[LINDNER 2012]
Lindner, K.: Medias (R) professional <http://medias.ina.de/medias/de!hp/;bwwWEeNg5u6d> 2012 07.06.2012.

[MAIER & RECHTIN 2009]
Maier, M. W.; Rechtin, E.: The Art of Systems Architecting. 3 ed. Boca Raton: CRC Press - Tailor & Francis Group 2009. ISBN: 978-1-4200-7913-8.

[MOD 2010]
MoD: Ministry of Defense Architecture Framework <http://www.mod.uk/DefenceInternet/AboutDefence/WhatWeDo/InformationManagement/MODAF/> 2011

[MODARRES & CHEON 1999]
Modarres, M.; Cheon, s. W.: Function-centered modeling of engineering systems using the goal tree–success tree technique and functional primitives. Reliability Engineering and Systems Safety 64 (1999) p. 43.

[NAGEL ET AL. 2008]
Nagel, R. L.; Hutchinson, R. S.; Stone, R. B.; Macadams, D. A.; Donndelinger, J.: Function Design Framework (FDF): Integrated Process and Function Modeling for Complex System Design, ASME International Design Engineering Technical Conference. New York, 2008. (DETC2008-49369)

[NAGEL ET AL. 2009]
Nagel, R. L.; Stone, R. B.; Perry, K. L.; McAdams, D. A.: FunctionCAD: A Functional Modeling Application Based on the Funtion Design Framework, IDETC/CIE 2009. San Diego, 2009.

[OMG 2010A]
OMG: Object Constraint Language. OMG 2010a.

[OMG 2010B]
OMG: OMG Systems Modeling Language specification 1.2 <http://www.omg.org/spec/SysML/1.2/ > 2010 19.07.2010.

[OMG 2010C]
OMG, Version 2.3: OMG Unified Modeling Language(TM) Superstructure. 2010c.

[OMG 2011]
OMG: OMG Unified Modeling LanguageTM (OMG UML), Infrastructure Version 2.4.1. 2011.

[OMG 2012]
OMG: OMG Systems Modeling Language <www.omgsysml.org> 2012 01.06.2012.

[PAHL ET AL. 2006]
Pahl, G.; Beitz, W.; Feldhusen, J.; Grote, K.-H.: Konstruktionslehre. Berlin: Springer 2006. ISBN: 13: 978-3540340607

[PAHL ET AL. 2007]
Pahl, G.; Feldhusen, J.; Grote, K.-H.; Beitz, W.: Engineering Design: A Systematic Approach. 3rd ed. London: Springer Verlag 2007. ISBN: 978-1-84628-318-5.

[RAULEAUX 1876]
Rauleaux, F.: The Kinematics of Machinery. 1st ed. New York: 1876.

[RODENACKER 1970]
Rodenacker, W. G.: Methodisches Konstruieren. 1. Ed ed. Berlin: Springer 1970. ISBN: 3-540-05171-6. (Konstruktionsbücher).

[ROTH 1994]
Roth, K.: Konstruieren mit Konstruktionskatalogen: Band 2: Konstruktionskataloge. 2nd ed. Berlin: Springer 1994.

[ROTH 2000]
Roth, K.: Konstruieren mit Konstruktionskatalogen: Band 1: Konstruktionslehre. Berlin: Springer 2000.

[SAINTER ET AL. 2000]
Sainter, P.; Oldham, K.; Larkin, A.; Murton, A.; Brimble, R.: Product knowledge management within knowledge-based engineering systems. In: ASME Design Automation Conference, Year.

[SEN ET AL. 2009]
Sen, C.; Summers, J. D.; Caldwell, B. W.; Mocko, G. M.: Topological Information Content and Expressiveness of Function Models in Mechanical Design, International Design Engineering Technical Conferenecs & Computers and Information in Engineering Conference. San Diego, CA, 2009.

[SHU 1998]
Shu, N. P.: Axiomatic Design Theory for Systems. Research in Engineering Design 10 (1998) p. 189-209.

[STACHOWIAK 1973]
Stachowiak, H.: Allgemeine Modelltheorie. Wien: Springer 1973. ISBN: 3-211-81106-0.

[STONE & WOOD 2000]
Stone, R. B.; Wood, K.: Development of a Functional Basis for Design. Journal of Mechanical Design 122 (2000) 4, p. 359-370.

[STRAHINGER 1998]
Strahinger, S.: Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerungdurch das Konzept des Metaisierungsprinzips, CEUR Workshop Modellierung '98. Münster, Germany, 11.-13- März 1998 1998.

[SUDARSAN ET AL. 2005]
Sudarsan, R.; Fenves, S. J.; Sriram, R. D.; Wang, F.: A product information modeling framework for product life cycle management. Computer-Aided Design 37 (2005) p. 13.

[SZYKMAN ET AL. 2000]
Szykman, S.; Sriram, R. D.; Bochenek, C.; Racz, J. W.; Senfaute, J.: Design Repostiories: Engineering Design's New Knowlege Base. IEEE Intelligent Systems 15 (2000) 3, p. 48-55.

[SZYKMAN ET AL. 1999]
Szykman, S.; Sriram, R. D.; Racz, J. W.: The Representation of Function in Computer-Based Design, ASME Design Engineering Technical Conferences. Las Vegas, Nevada, Sept. 12-15 1999 1999.

[TERNINKO ET AL. 1998]
Terninko, J.; Zusman, A.; Zlotin, B.: Systematic Innovation: An Introduction to TRIZ. Boca Raton: St. Lucie Press 1998. ISBN: 1-57444-111-6.

[THOMAS ET AL. 2009]
Thomas, J.; Sen, C.; Mocko, G. M.; Summers, J. D.; Fadel, G. M.: Investigation of the Interpretability of Three Function Structure Representations: A User Study, International Design Engineering Technical Conferenecs & Computers and Information in Engineering Conference. San Diego, CA, 2009.

[ULLMAN 2002]
Ullman, D.: The mechanical design process. 3 ed. New York: McGraw-Hill Companies 2002. ISBN: 978-0072373387.

[ULRICH & EPPINGER 1994]
Ulrich, K.; Eppinger, S.: Product design and development. New York: McGraw-Hill 1994.

[UMEDA & TOMIYAMA 1995]
Umeda, Y.; Tomiyama, T.: FBS modeling: Modeling scheme of function for conceptual design. Workshop on Qualitative Reasoning about Phys.'Systems (1995) p. 271–278.

[VDI2206 2006]
VDI2206, VDI-Richtlinie 2206: Entwicklungsmethodik für Mechatronische Systeme. Berlin: Beuth Verlag 2006.

[VDI 2221 1987]
VDI 2221 (Ed.): Systematic Approach to the Design of technnical Systems and Products. VDI Handbook of Design, VDI, Düsseldorf (1987). (Original in German Nov. 1986. Latest edition of 1993)

[VDI 2221 1993]
VDI 2221 (Ed.): VDI Richtlinie 2221- Methodik zum Entwickeln und Konsturieren technischer Systeme und Produkte Handbuch Konstruktion, VDI Auschuß Konstruktionsmethodik, Düsseldorf (1993).

[VDI 2222 1997]
VDI 2222 (Ed.): VDI Gudieline 2222, Sheet 1 - Methodic development of solution principles. VDI Handbuch Konstruktion, VDI, Düsseldorf (1997).

[VDI /VDE 3681 2005]
VDI /VDE 3681: Classification and evaluation of description methods in automation and control technology. 2005.

[WEILKIENS 2006]
Weilkiens, T.: Systems Engineering with SysML/UML. Heidelberg: dpunkt 2006. ISBN: 389864409X. (Basis für die Arbeit mit SysML, Beispielhafte Erklärungen)

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

[WÖLKL & SHEA 2011]
Wölkl, S.; Shea, K.: Modellbibliotheken für die Funktions- und Komponentenmodel-lierung in der Konzeptentwicklung mit SysML Workshop Mechatronisch Systeme. Paderborn, 2011.

[WYMORE 1993]
Wymore, A.: Model-Based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design. Boca Raton: CRC Press 1993.

[YOSHIOKA ET AL. 2004]

Yoshioka, M.; Umeda, Y.; Takeda, H.; Shimomura, Y.; Nomaguchi, Y.; Tomiyama, T.: Physical concept ontology for the knowledge intensive engineering framework. Advanced Engineering Informatics 18 (2004) 2, p. 95-113.

9 Appendix

Table 9-1: Flow definitions for the model library

Class	Secondary	Tertiary		Class	Secondary	Tertiary
Material	Human			Energy		
	Gas				**Human**Energy	
	Liquid				**Acoustic**Energy	
	Solid				**Biological**Energy	
		Solid**Object**			**Chemical**Energy	
		Solid**Particulate**			**Electrical**Energy	
		Solid**Composite**			**Electromagnetic**Energy	
	Plasma					**Optical**Energy
	Mixture					**Solar**Energy
		GasGasMix			**Hydraulic**Energy	
		LiquidLiquid Mix			**Magnetic**Energy	
		SolidSolidMix			**Mechanical**Energy	
		SolidLiquidMix				**Rotational** MechEnergy
		LiquidGasMix				**Translational** MechEnergy
		SolidGasMix			**Pneumatic**Energy	
		SolidLiquidGasMix			**Nuclear**Energy	
		Colloidal			**Thermal**Energy	
SignalFlow	**Status**Signal					
		AuditorySignal				
		OlfactorySignal				
		TactileSignal				
		TasteSignal				
		VisualSignal				
	ControlSignal					
		AnalogCSignal				
		DiscreteCSignal				

Terms printed in normal type are taken over from NIST functional basis. Terms with bold and normal type: the bold part is taken form NIST functional basis

Table 9-2: Definition of functions according to [Hirtz et al. 2002] and definition of interacting flows (1/2)

Primary	Secondary	Tertiary	Description
Branch			Branch. To cause a flow (material; energy; signal) to no longer be joined or mixed.
	Separate		Separate. To isolate a flow (material; energy; signal) into distinct components. The separated components are distinct from the flow before separation; as well as each other. Example: A glass prism separates light into different wavelength components to produce a rainbow.
		Divide	Divide. To separate a flow. Example: A vending machine divides the solid form of coins into appropriate denominations.
		Extract	Extract. To draw; or forcibly pull out; a flow. Example: A vacuum cleaner extracts debris from the imported mixture and exports clean air to the environment.
		Remove	Remove. To take away a part of a flow from its prefixed place. Example: A sander removes small pieces of the wood suRootFlowace to smooth the wood.
	Distribute		Distribute. To cause a flow (material; energy; signal) to break up. The individual bits are similar to each other and the undistributed flow. Example: An atomizer distributes (or sprays) hair-styling liquids over the head to hold the hair in the desired style.
Channel			To cause a flow (material; energy; signal) to move from one location to another location.
	Import		Import. To bring in a flow (material; energy; signal) from outside the system boundary. Example: A physical opening at the top of a blender pitcher imports a solid (food) into the system. Also; a handle on the blender pitcher imports a human hand.
	Export		Export. To send a flow (material; energy; signal) outside the system boundary. Example: Pouring blended food out of a standard blender pitcher exports liquid from the system. The opening at the top of the blender is a solution to the export subfunction
	Transfer		Transfer. To shift; or convey; a flow (material; energy; signal) from one place to another.
		Transport	Transport. To move a material from one place to another. Example: A coffee maker transports liquid (water) from its reservoir through its heating chamber and then to the filter basket.
		Transmit	Transmit. To move an energy from one place to another. Example: In a hand held power sander; the housing of the sander transmits human force to the object being sanded.
	Guide		Guide. To direct the course of a flow (material; energy; signal) along a specific path. Example: A domestic HVAC system guides gas (air) around the house to the correct locations via a set of ducts.
		Translate	Translate. To fix the movement of a flow by a device into one linear direction. Example: In an assembly line; a conveyor belt translates partially completed products from one assembly station to another.
		Rotate	Rotate. To fix the movement of a flow by a device around one axis. Example: A computer disk drive rotates the magnetic disks around an axis so that the head can read data.
		Allow DOF	Allow degree of freedom (DOF). To control the movement of a flow by a force external to the device into one or more directions. Example: To provide easy trunk access and close appropriately; trunk lids need to move along a specific degree of freedom. A four bar linkage allows a rotational DOF for the trunk lid.
Connect			Connect. To bring two or more flows (material; energy; signal) together.
	Couple		Couple. To join or bring together flows (material; energy; signal) such that the members are still distinguishable from each other. Example: A standard pencil couples an eraser and a writing shaft. The coupling is peRootFlowormed using a metal sleeve that is crimped to the eraser and the shaft.
		Join	Join. To couple flows together in a predetermined manner. Example: A ratchet joins a socket on its square shaft inteRootFlowace.
		Link	Link. To couple flows together by means of an intermediary flow. Example: A turnbuckle links two ends of a steering cable together.
	Mix		Mix. To combine two flows (material; energy; signal) into a single; uniform homogeneous mass. Example: A shaker mixes a paint base and its dyes to form a homogeneous liquid.
Control Magnitude			Control Magnitude. To alter or govern the size or amplitude of a flow (material; energy; signal).
	Actuate		Actuate. To commence the flow of energy; signal; or material in response to an imported control signal. Example: A circuit switch actuates the flow of electrical energy and turns on a light bulb.
	Regulate		Regulate. To adjust the flow of energy; signal; or material in response to a control signal; such as a characteristic of a flow. Example: Turning the valves regulates the flow rate of the liquid flowing from a faucet.
		Increase	Increase. To enlarge a flow in response to a control signal. Example: Opening the valve of a faucet further increases the flow of water.
		Decrease	Decrease. To reduce a flow in response to a control signal. Example: Closing the value further decreases the flow of propane to the gas grill.

Flow 1	Flow2	Flow N	Flow condition	Remark
in / RootFlow	out / RootFlow	out / RootFlow		
in / RootFlow	out / RootFlow	out / RootFlow	different flow types	
in / RootFlow	out / RootFlow	out / RootFlow	different flow types	
in / RootFlow	out / RootFlow	out / RootFlow	different flow types	
in / RootFlow	out / RootFlow	out / RootFlow	different flow types	
in / RootFlow	out / RootFlow		same flow types	more flows are leaving from pin
in / RootFlow	out / RootFlow			change of location
out/ RootFlow				Not recommended
in / RootFlow				Not recommended
in / RootFlow	out / RootFlow			
in / material	out / material	inout /energy		energy needed or dissipated
in/energy	our/energy			
in / RootFlow	out / RootFlow			
in /RootFlow	out / RootFlow			
in / RootFlow	out / RootFlow			
in /RootFlow	out / RootFlow			
in /RootFlow	in/RootFlow	out / RootFlow		
in /RootFlow	in/RootFlow	out / RootFlow		
in /RootFlow	in/RootFlow	out / RootFlow		
in /RootFlow	in/RootFlow	in / RootFlow out /RootFlow		four different flows!
in /RootFlow	in/RootFlow	out / RootFlow		
in /RootFlow	out/RootFlow		different property of flow	
in /RootFlow	out/RootFlow	in / signal		
in /RootFlow	out/RootFlow	in / signal		
in /RootFlow	out/RootFlow	in / signal		
in /RootFlow	out/RootFlow	in / signal		

Table 9-3: Definition of functions according to Hirtz et al. [2002] (2/2)

Primary	Secondary	Tertiary	Description
	Change		Change. To adjust the flow of energy; signal; or material in a predetermined and fixed manner. Example: In a hand held drill; a variable resistor changes the electrical energy flow to the motor thus changing the speed the drill turns.
		Increment	Increment. To enlarge a flow in a predetermined and fixed manner. Example: A magnifying glass increments he visual signal (i.e. the print) from a paper document.
		Decrement	Decrement. To reduce a flow in a predetermined and fixed manner. Example: The gear train of a power screwdriver decrements the flow of rotational energy.
		Shape	Shape. To mold or form a flow. Example: In the auto industry; large presses shape sheet metal into contoured suRootFlowaces that become fenders; hoods and trunks.
		Condition	Condition. To render a flow appropriate for the desired use. Example: To prevent damage to electrical equipment; a surge protector conditions electrical energy by excluding spikes and noise (usually through capacitors) from the energy path.
	Stop		Stop. To cease; or prevent; the transfer of a flow (material; energy; signal). Example: A reflective coating on a window stops the transmission of UV radiation through a window.
		Prevent	Prevent. To keep a flow from happening. Example: A submerged gate on a dam wall prevents water from flowing to the other side.
		Inhibit	Inhibit. To significantly restrain a flow; though a portion of the flow continues to be transferred. Example: The structures of space vehicles inhibits the flow of radiation to protect crew and cargo.
Convert			Convert. To change from one form of a flow (material, energy, signal) to another. For completeness; any type of flow conversion is valid. In practice; conversions such as convert electricity to torque will be more common than convert solid to optical energy. Example: An electrical motor converts electricity to rotational energy.
Provision			Provision. To accumulate or provide a material or energy flow.
	Store		Store. To accumulate a flow. Example: A DC electrical battery stores the energy in a flashlight.
		Contain	Contain. To keep a flow within limits. Example: A vacuum bag contains debris vacuumed from a house
		Collect	Collect. To bring a flow together into one place. Example: Solar panels collect ultraviolet sun rays to power small mechanisms.
	Supply		Supply. To provide a flow from storage. Example: In a flashlight; the battery supplies energy to the bulb.
Signal			Signal. To provide information on a material; energy or signal flow as an output signal flow. The information providing flow passes through the function unchanged.
	Sense		Sense. To perceive; or become aware; of a flow. Example: An audiocassette machine senses if the end of the tape has been reached.
		Detect	Detect. To discover information about a flow. Example: A gauge on the top of a gas cylinder detects proper pressure ranges.
		Measure	Measure. To determine the magnitude of a flow. Example: An analog thermostat measures temperature through a bimetallic strip.
	Indicate		Indicate. To make something known to the user about a flow. Example: A small window in the water container of a coffee maker indicates the level of water in the machine.
		Track	Track. To observe and record data from a flow. Example: By tracking the performance of batteries; the low efficiency point can be determined.
		Display	Display. To reveal something about a flow to the mind or eye. Example: The xyz coordinate display on a vertical milling machine displays the precise location of the cutting tool.
	Process		Process. To submit information to a particular treatment or method having a set number of operations or steps. Example: A computer processes a login request signal before allowing a user access to its facilities.
Support			Support. To firmly fix a material into a defined location; or secure an energy or signal into a specific course.
	Stabilize		Stabilize. To prevent a flow from changing course or location. Example: On a typical canister vacuum; the center of gravity is placed at a low elevation to stabilize the vacuum when it is pulled by the hose.
	Secure		Secure. To firmly fix a flow path. Example: On a bicycling glove; a Velcro strap secures the human hand in the correct place.
	Position		Position. To place a flow (material; energy; signal) into a specific location or orientation. Example: The coin slot on a soda machine positions the coin to begin the coin evaluation and transportation procedure.

Flow 1	Flow2	Flow N	Flow condition	Remark
in /RootFlow	out/RootFlow			
in /RootFlow	out/RootFlow			
in /RootFlow	out/RootFlow			
in /RootFlow	out/RootFlow			
in /RootFlow	out/RootFlow			
in /RootFlow				
in /RootFlow				
in/RootFlow	out/RootFlow			not very precise
in/RootFlow	out/RootFlow	in /energy		material conversion needs energy
in/Material Energy	out/Material Energy			
in/Material Energy				
in/Material Energy				
in/Material Energy				
	out/Material Energy			
in/ RootFlow	out / RootFlow	Out/ Signal		unchanged carrier flows
in/ RootFlow	out / RootFlow	Out/ Signal		
in/ RootFlow	out / RootFlow	Out/ Signal		
in/ RootFlow	out / RootFlow	Out/ Signal		
in/ RootFlow	Out/ Signal			
in/ RootFlow	out / RootFlow	Out/ Signal		
in/ signal	out / RootFlow			
in/ RootFlow	out / RootFlow	Out/ Signal		
in/ RootFlow				
in/ RootFlow				
in/ RootFlow				
in/ RootFlow				