Variability Management in Design of Mechatronic Systems

zur Erlangung des akademischen Grades eines
DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)
der Fakultät Maschinenbau
der Universität Paderborn

genehmigte
DISSERTATION

von
M.Sc. Farisoroosh Abrishamchian
aus Teheran, Iran

Tag des Kolloquiums: 30. April 2026
Referent: Prof. Dr.-Ing. habil. Ansgar Trächtler
Korreferent: Prof. Dr.-Ing. Roman Dumitrescu

Acknowledgements

I would like to express my sincere gratitude to Prof. Dr.-Ing. habil. Ansgar Trächtler for his guidance, support, and trust throughout my doctoral research. His expertise, valuable advice, and encouragement have significantly contributed to the completion of this dissertation.

I am also grateful to Prof. Dr.-Ing. Roman Dumitrescu for serving as the second reviewer of this thesis and for his interest in my research. Furthermore, I would like to thank the members of the doctoral examination committee, Prof. Dr. Iryna Mozgova and Dr.-Ing. Sandra Gausemeier, for their time, commitment, and contributions to the evaluation of this dissertation.

My special thanks go to Dr.-Ing. Felix Oestersötebier for his valuable advice, insightful discussions, and the careful review of the manuscript. His constructive comments and suggestions greatly improved the quality of this work. I am likewise grateful to Dr.-Ing. Arathi Pai for her careful review of the manuscript and her helpful feedback. I also thank Michaela Wiemers for her support and assistance with the administrative aspects of the doctoral process, particularly during its final stages.

Furthermore, I would like to express my gratitude to my former colleagues at the Chair for the inspiring and enjoyable atmosphere that accompanied my doctoral studies. I will always cherish the memories of the doctoral seminars, annual retreats, and the many discussions, collaborations, friendships, and shared life experiences that made my time at the university both professionally rewarding and personally memorable.

Finally, I would like to thank my family, friends, and those closest to me for their patience, encouragement, and unwavering support throughout this long journey. In particular, I am deeply grateful to Yasaman, whose persistent encouragement, honest feedback, and belief in me provided the motivation to bring this long-standing project to completion when it might otherwise have remained unfinished. This dissertation would not have been completed without that support.

This dissertation marks the completion of a project that began many years ago, and I am thankful to everyone who contributed to it in various ways.

Cologne, June 2026

Farisoroosh Abrishamchian

Zusammenfassung

In dieser Arbeit wird ein Ansatz für das integrierte modellbasierte Design intelligenter dynamischer Systeme vorgestellt. Ein Konzept zur Etablierung einer domänenübergreifenden Lösung ist das grundlegende Ziel dieses Systemdesigns. Um die Entwurfskomplexität aufgrund der systemübergreifenden Verwendung verschiedener Modellierungswerkzeuge zu verringern, werden die Konsistenz zwischen mehreren Domänen und Abhängigkeiten zwischen Subsystemen berücksichtigt. In dieser Arbeit wird eine Richtlinie mit einem flexibleren Verfahrensmodell vorgeschlagen, um einem festen Zeitplan für die Entwurfsverarbeitung zu folgen. Zu diesem Zweck wird das ursprüngliche V-Modell der VDI-Richtlinie weiter in Unterfunktionen unterteilt. Außerdem werden dem entwickelten Systemdesignprozess von Oestersötebier [Oes17] einige zusätzliche Unterfunktionen hinzugefügt. In Anbetracht dessen wird eine Methode des Variabilitätsmanagements angewendet, um Gemeinsamkeiten und Unterschiede der verschiedenen Implementierungsfragmente des resultierenden Produkts zu verwenden. Um es zu realisieren, wird eine feature-orientierte Softwareentwicklung angewendet, um Konfigurationsoptionen bereitzustellen und die Generierung eines modellbasierten Systemdesigns basierend auf einer Auswahl von Features zu erleichtern.

Abstract

In this thesis an approach for the integrated model-based design of intelligent dynamic systems is presented. A concept for the establishing a cross-domain solution is the basic target of this system design. To reduce the design complexity because of involving different modeling tools across the system, consistency between multi domains and dependencies between subsystems are considered. In this work, a guideline with more flexible procedural model to follow a fixed schedule form for design processing is proposed. For this purpose, the original V-model of the VDI guideline is further broken down into subfunctions. Also, some additional subfunctions are added to the developed system design process of Oestersötebier [Oes17]. Considering that, a method of variability management is adapted to use commonalities and differences of the various implementation fragments of the resulting product. To handle this, feature-oriented software development is applied to provide configuration options and to facilitate the generation of model-based system design based on a selection of features.

Publication

[1] Abrishamchian, Farisoroosh;Trächtler, Ansgar: Configuration of Mechatronic Systems Using Feature Models. In: *2nd International Conference on System-Integrated Intelligence*, S. 27-34, Bremen, Germany, Jul. 2014, Universität Bremen

[2] Abrishamchian, Farisoroosh; Oestersötebier, Felix; Trächtler, Ansgar: Feature Model Approach for Managing Variability of Dynamic Behavior Models in Mechatronic Systems. In: *Proceedings of the ASME 2015 International Mechanical Engineering Congress & Exposition IMECE 2015*, Houston, Texas, Nov. 2015

[3] Abrishamchian, Farisoroosh; Trächtler, Ansgar (Hrsg.): Feature Model Based Interface Design for Development of Mechatronic Systems. In: *IEEE International Symposium on Systems Engineering*, 3. - 5. Okt. 2016 IEEE, IEEE (Details)

[4] Oestersötebier, Felix; Abrishamchian, Farisoroosh; Lankeit, Christopher; Just, Viktor; Trächtler, Ansgar: Approach for an Integrated Model-Based System Design of Intelligent Dynamic Systems Using Solution and System Knowledge. In: *Proceedings of the 3rd International Conference on System-Integrated Intelligence*, Paderborn, 13. - 15. Jun. 2016, Elsevier (Details)

Contents

Abbreviations

A-Box Assertional Box

CAD Computer-Aided Design

CAE Computer-Aided Engineering

COM Component Object Model

CONSENS CONceptual design Specification technique for the ENgineering of complex Systems

DBM Dynamic Behavior Model

DLL Dynamic Link Library

DMM Domain Mapping Matrix

DSM Design Structure Matrix

ECAD Electrical-CAD

EDLC Ensemble Development Life Cycle

FAST Family-Oriented Abstraction, Specification, and Translation

FDF Functional Design Framework

FMI Functional Mock-Up Interface

FMP Feature Modeling Plug-In

FMU Functional Mock-Up Unit

FODA Feature-Oriented Domain Analysis

FOSD Feature-Oriented Software Development

GUI Graphical User Interface

HiL Hardware-in-the-Loop

HMI Human Machine Interface

IDE Integrated Development Environments

MBS Multibody Simulation

MBSE Model-Based System Engineering

MCAD Mechanical-CAD

MiL Model-in-the-Loop

MMC Multifunctional Model Client

OEM Original Equipment Manufacturer

OWL Web Ontology Language

PIDO Process Integration and Design Optimization

PLC Programmable Logic Controller

RDF Ressource Description Framework

S-Function System-Functions

SFSL Semi-Formal Specification Language

SiL Software-in-the-Loop

SPARQL SPARQL Protocol And RDF Query Language

SPL Software Product Lines

SPLOT Software Product Line Online Tools

SQL Structured Query Language

STARS Software Technology for Adaptable Reliable Systems

SysML Systems Modeling Language

T-Box Terminological Box

TcCOM TwinCAT Component Object Model

TCP Tool-Center-Point

UML Unified Modeling Language

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformation

1 Introduction

Engineers in the context of system design strive to faultlessly describe a real system, whereby transparency of the reality is not relinquished. The tension between free spaces and boundary conditions, the estimation of feasible, impossible opportunities and behavioral predictions for future real complex system with scientific methods makes the system design a great adventure [Jan10]. This is especially prominent in mechatronic systems.

A mechatronic system is a result of cooperation and collaboration of mechanical, informatics, electronics, electrical and control engineering. This involvement of different disciplines in a system increases the complexity of technical products as well as their development process. This means that development process has to take place, with the goal of designing an overall integrated system. The complexity of mechatronic systems is increasing through the raising number of various intellectual domains. For handling this complexity, attaching the existing part of various intellectual domains together to make an integrated system is not enough. *Synergetic effects are aimed for, comprising more than the mere addition of the disciplines [Ise08].*

In the past years, the proportion of classical, purely mechanical systems has declined significantly in mechatronics systems compared to the software proportion. For example, a camera in the 1970s was purely mechanical. The photographer was responsible for adjusting the focal point, aperture opening etc. and the picture taken was stored on a film reel. Today, cameras are equipped with various sensors. It can set the focusing point of the lens on the subject and vary the exposure and other settings of the camera according to the users' requirements automatically through software. Furthermore, the images are stored digitally. Automatic compensation of one parameter against another is allowed by software. Therefore, in the advent of the digital age, software development processes also need to be considered in the design of mechatronic systems. Additionally, the variance of products with different functionality has also increased over the past years.

Consequently the use of computer aided design methods has also evolved. Considering once again the camera, whereas in the past, Computer-Aided Design (CAD) models were sufficient for the system design, today, it is necessary to use a plethora of different computer-supported modeling languages from various domains to describe the complex mechatronic system that the camera has become.

The expanding complexity of products including expanding product variability can often no longer be met with the present design methods. This is because of

many challenges that emerge from the absence of methods, processes, information or tools. There is a need of a supporting system for modeling and analyzing the entire mechatronic systems [AFT+10]. To achieve an integrated design, the mechatronic system design process should be optimized.

Questions like how to overcome these challenges and proposing solutions for mechatronics systems, are discussed in this work.

1.1 Development methodology of mechatronic systems

There are numerous development guidelines and methodologies for design of mechatronic systems, which are mostly focused on individual design task, such as VDI Guideline 2206 [VDI04]. In the Figure 1-1 the V model approach is illustrated in details. The design process of mechatronic systems is described as follow [VDI04]:

- Requirements: The defined starting point is specified precisely and described in the form of requirements. They are the base of a later product.
- System design: To establish a cross-domain solution concept the overall function of a system is broken down into main subfunctions. They describe the main physical and logical operating characteristics of the future product.
- Discipline specific design: More detailed interpretations and calculations usually takes place separately in the domains involved. It is necessary to ensure the performance of the function.
- System integration: To form an overall system and to allow the interaction to be investigated the results from the individual domains are integrated.
- Assurance of properties: Means of the specified solution concept and the requirements check continuously the design progress. The actual system characteristics match with those wanted has to be assured.
- Modeling and model analysis: The modeling and analysis of the system characteristics with the aid of models and computer-aided tools for simulation flank the described phases.
- Product: A product is the result of a continuous macro-cycle. It is actually the specimen of the future product.
- Assurance of properties: The specified solution concept and the requirements has to be checked continuously during the design process. The actual system characteristics has to be matched with those wanted.

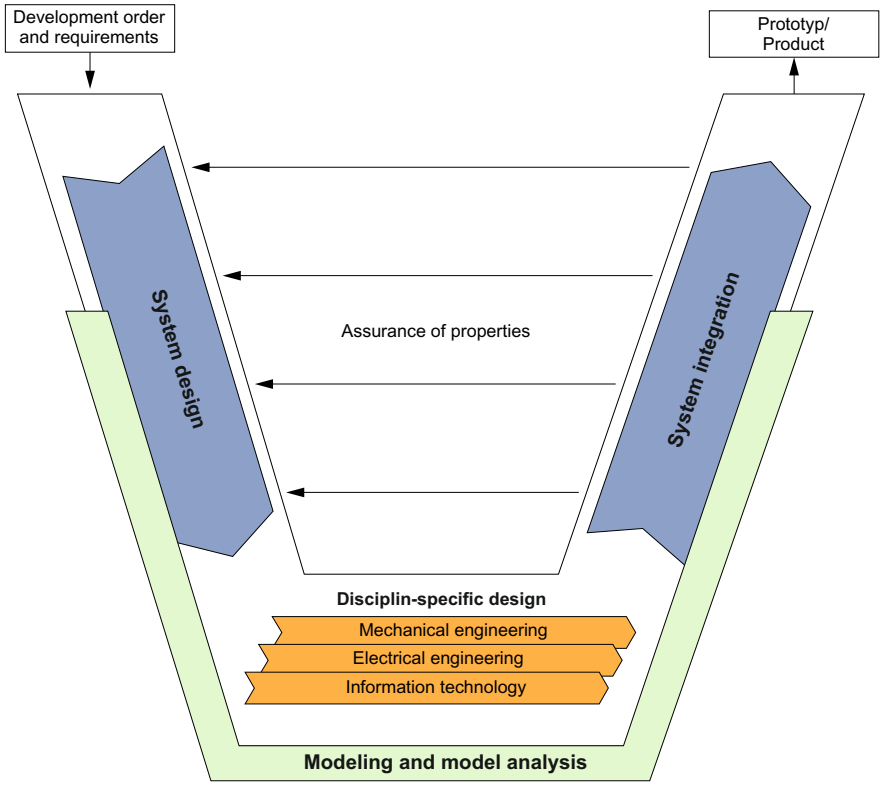


Figure 1-1: V model as a macro-cycle [VDI04]

1.2 Problem and Motivation

The V model provides a methodology for design of mechatronic systems, starting from requirements to the end product. However, the problem of modeling mechatronic products becomes more challenging as the number of variants, components and the functional layers increases both, in hardware and software. Further, there is a need of a tooling system for supporting the modeling and analyzing of entire mechatronic systems [AFT+10].

A mechatronic product by nature is complex, because it contains multiple domains of engineering. Furthermore mechanics, electronics and software components could be involved in each function of a product [GPW09]. Due to the highly coupled nature of mechatronic products, typically change in one subsystem induces change in the other subsystems.

A multi-disciplinary and holistic development of mechatronic systems must be supported by a particular design process [ZBLD+14]. It is complicated to find the dependencies between design steps, when several domain-specific tools are required and the design process is partitioned into domain-specific problems.

Many properties of the product interconnect to each other [TDU+07] and affect each other in a heterogeneous way. These interactions make the design process complicated.

Traditionally engineers are specialized in different departments or specific activities. Communication across domains is difficult without having a basis. They need to learn about other's specialization and their requirements from the system to handle the interdependencies and data exchange along the design process. Moreover engineers need to have knowledge about different involved domain-specific tools to manage the dependencies and to handle the complexity of their own particular engineering problems.

In order to design an interconnected mechatronic system, an integrated and a comprehensive system and simulations are required that encompasses multiple domains. For efficient modeling and simulation in the mechatronics field, there is a need for tools that can be used for configuring the submodels which are simulated in different simulation tools. Moreover, the proposed model should support varying models with varying modeling details. According to that, the identification of several commonalities and variabilities of subsystem models along the design of entire mechatronic systems is possible.

The methods of variability management, traditionally from the software domain, use commonly models to represent the differences and commonalities of the various resulting products or implementation fragments [BPSP04]. These methods will be used to handle variability in a mechatronic design process.

In this work, following problems and strategies are addressed:

Problem statement:

- Current V model is not sufficient to handle expanding complexity of mechatronic system design
- The expanding variability of mechatronic products are not considered in mechatronic system design
- There is a lack of tools and methods supporting multi-disciplinary design

Based on the problem statement above, following issues in particular are expanded and facilitated during the design of complex mechatronic systems:

- Designing:
  - How can the design of a complex mechatronic system be simplified?
  - How can dependencies be managed between subsystems?
- Variability management:

  - How can the Software Product Line techniques be adapted in design of mechatronic systems?
  - How can the subsystems be reused for next developments?
  - How can consistency be supported during the system design and between multi domains?
- Tooling:
  - How can different modeling tools be applied across the system design?

1.3 Guide of Chapters

Chapter 2:
In this chapter searching relevant literature showing evidence of a problem, and proposed solutions to tackle a particular problem. Three aspects are considered:

- Concept of mechatronic systems design
- Variability management with the focus on feature modeling
- Existing tools for simulation, coupling, integration and variability management tools regarding to mechatronic systems

My work:
2.4: The scientific gaps at the end of the chapter is addressed to explain the gaps of existing solution for designing mechatronic systems and how to cover them.

Chapter3:
A guide for the design of mechatronic systems is the V model, which is adopted from software development and adapted to the requirements of mechatronics. Here, the V model procedure is detailed and supplemented with procedural steps.

My work:
2 Synthesis of system design; 2 Synthesis of discipline-specific design; 3.1.4; 3.1.5: In this chapter is shown, how to adapt feature modeling along design process in system design and discipline-specific design for ensuring reuse of information and consistency between different disciplines.

3.2: A concept of the mapping interfaces based on feature modeling is presented to improve the effort of maintaining consistency between all artifacts and to avoid inadequate definition of interfaces.

Chapter 4 (thesis concept):
In this chapter the management of variability in design of mechatronics system is investigated to provide the grammar for structuring a variability model.

My work:
4.2: Each individual subsystem design is considered as variation with the consideration of technical side of the design. This subsystems have to be designed individually and have their own life-cycles. The strategy is, first, to focus on common design character of a component and then what is different. To facilitate system design customization, the artifacts used in different system components have to be sufficiently adaptable to fit into the different systems design in the product line, as depicted in Figure 4-3 and Table 4-1.

Chapter 5 (concept implementation):
In this chapter an implementation prototype of the Multifunctional Model Client according to a classical waterfall approach is introduced.

Highlights of my work:
5.3: The implementation of design concept is outlined according to the Figure 5-2.

5.3.1: The detail of design concept and its functionality is defined step by step by the use of activity diagram in the Figure 5-3, 5-4 and 5-5.

5.4.2: GUI of three screen operations of the MMC, conceptual design (Figure 5-14), specific design (Figure 5-21) and integration (Figure 5-24) are described according to the V model.

Chapter 6 (case study according to the concept):
In this chapter, an application example of two cooperating delta robots that juggle a ball by passing it to each other (Figure 6-1) is used to illustrate the proposed approach for an integrated model-based design process.

Highlights of my work:
6.2.2; 6.3.2: In synthesis phase, active structure, feature model, shape and behavior and activity diagram of delta robots are illustrated to select a specific and/or more concrete solution patterns. By using feature models, the idealized dynamics model of the system, suitable parameter configurations are determined and tested.

6.2.3: In analysis different scenarios are studied and measured. According to this basis, the modeling objectives for behavioral models are prepared or detailed. They contain target sizes for the model, degree of detail and complexity, which are dependent on the test concept developed.

6.4: In the course of System integration, SiL and HiL for the delta robots are executed (Figure 6-25).

6.5: At the end, developed design method is evaluated by analysis of the methods and approaches towards the traditional V model. Here, the effectiveness of feature model is calculated to estimate total effort of design method in the case study with the existing parameters.

2 Literature Review

This chapter aims to describe the literature relevant to the thesis. The overall aim is to understand the current problems within the design of mechatronic products, and identify the factors that have led to this situation. In this chapter searching relevant literature showing evidence of a problem, and proposed solutions to tackle a particular problem are presented. The gaps that this research aims to fill are highlighted through the state of the art description.

The first section gives an overview of different concept of the mechatronic systems design to show the lack of common language to represent an entire system. On that point, the transfer of models and information between domains is discussed. In the next section, variability management with the focus on feature modeling and Feature-Oriented Software Development (FOSD) is investigated. The purpose here is to show the adaption of this concept to configure mechatronic systems in the form of multi product line. Third section represents the existing tools for simulation, coupling, integration and variability management tools regard to mechatronic systems. This chapter is finished with the scientific gaps in the existing solution for designing mechatronic systems.

2.1 Modeling and Design of Mechatronic Systems

There are many different definitions for the term *system*. According to Helmut Bode, a system is defined as follows:

A system is to be understood as the totality of objects (elements) which are in a holistic context. Through their interrelationships, they are bounded by their surroundings. The couplings of the individual objects are much stronger than the couplings or interactions with the environment [Bod07].

As can be seen in the Figure 2-1, a system can be divided into three components: The input variables consist of the energy, material and information flow from the environment. The main component of the system processes the data and ultimately outputs it as output variables in the form of energy, material and information flows [Trä14].

In order to describe systems, often models are used. These models can be defined according to Kowalk [Tra13] as follows: A model is a summary of characteristics of a real (or empirical) artificial system, as well as a determination of the relationship between these features, since a model can never include all the features of a system, a model is an abstraction of a real system. A signal must be recorded, stored and processed using technical elements in order to reproduce the information

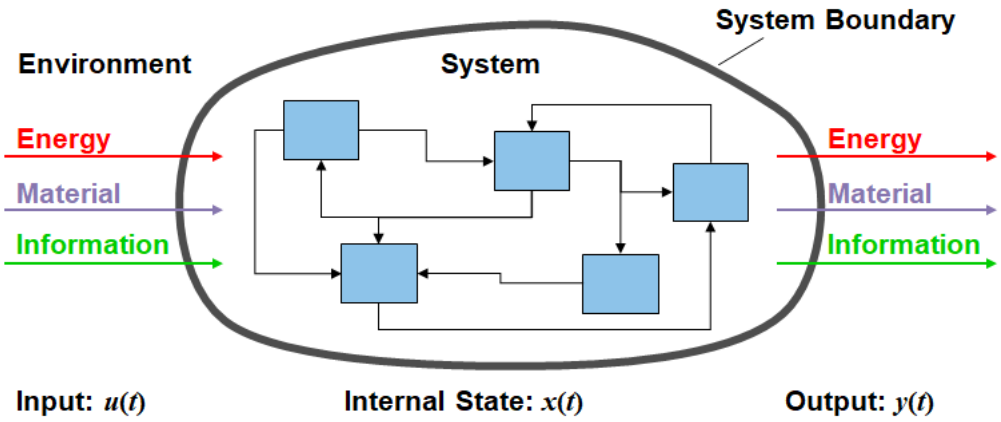


Figure 2-1: System structure, adopted from [Trä14]

contained in the signal. Signals are divided into useful and interfering signals. Useful signals are integrated as an input or output signal, also they serve to control a process. Since in the real world every process is affected by disturbances of the environment, e.g. temperature changes etc., the interfering signal is also included in the process to reproduce the real procedure of a process [EGK08].

The advantage of modeling is that they are cheaper than the original system because the production of the original system can be very expensive. Models also reduce the risks of testing a system, such as nuclear safety equipment or other tests that could damage the entire system. The model can also help the development, if the system does not yet exist or no longer exists. Since the systems to be modeled are becoming more and more complex and comprehensive, not all information and data can be given. Too much data makes the model more complex and some data is often not known at all. Therefore, models are simplified with adapted assumptions, and complicated effects are ignored or greatly simplified. Thus, it is important in modeling to balance between the required accuracy and the effort to be made in order to decide which effects can be ignored [Bod07].

Other approaches for assessing of a design concept are informal approaches such as A3 overviews [Bor10], semi-formal description such as CONceptual design Specification technique for the ENgineering of complex Systems (CONSENS) [GLL12] and formal description towards modeling based, such as Systems Modeling Language (SysML) [FMS14].

A3 overview does not support the specific mechatronic aspects, such as the implication of the different allocation of functions within domains. For this purpose semi-formal and formal description offer better solution. Their modeling skills gain a systems' simulation and analysis understanding, such as modelica [FE98]. Indeed to access the study of the overall mechatronic system an alternative

concept is needed for capturing the properties of models for reusing and coupling them.

Different groups of researchers represent a holistic approach towards modeling based on informal and formal representation as a solution to the common language issue. The A3 architecture overviews is an example of informal description. This approach provides an overview of the complete system architecture in terms of different system aspects, such as functional and physical aspects. The designer should represent all these aspects on an A3 size paper, thereby providing limited but useful information, not only to represent their design concept, but also to be able to communicate with other domain experts during a multi-disciplinary design process [Bor10]. A3 overviews has the same potential of becoming a common language as functional thinking as an approach, but they carry the same drawbacks.

2.1.1 Alternative Concepts of Design

Clearly, there are often different ways of solving a design problem. For mechatronics due to the designers' potential lack of awareness of the cross-disciplinary impact, it becomes more difficult because of selecting one alternative over another [Qam13].

To help assess the consequences of a design concept, the reviewed literature proposes several ways. Relationship management techniques is one of the different solutions like the Design Structure Matrix (DSM)/Domain Mapping Matrix (DMM)[EB12], QFD [HC$^+$88], and FunKey Architecting [Bon08]. Modeling relationships between the functions, and components of a product, along with taking user preferences into account are tackled in these approaches. However, it often becomes too cumbersome to analyze the consequences of a design choice through multiple relations across domains, due to the effort required and efficacy of the method, despite being useful to gain understanding of them [Qam13].

2.1.2 Common Approach to Represent a Concept

A multi-disciplinary team of domain experts performs usually mechatronic design. They process unlikely to possess inter-disciplinary knowledge to get a detailed understanding of the whole design problem. Therefore, to establish a common mechatronic view is difficult - instead, different domain-specific views are established and the dependencies in between are unclear. Especially during the conceptual phase of development a medium is required that permits designers to effectively communicate with each other to establish a common understanding, as discussed in [WACT$^+$10]; [TDU$^+$07]; [Ada04]; [GFM01]; [Buu90]; [SV89].

One of the proposed solutions for a common language is to utilize following methods based on functional thinking, such as [Qam13]:

- Functional approach by [Buu90].
- Function-Behavior-State modeling by [UKS+05].
- Function-Behavior-Structure framework by [Ger90]; [GK04] and [UTT+90].
- Functional Design Framework (FDF) [NSH+08].
- Axiomatic design by [Suh01].

A good review of different functional modeling approaches and their applications is provided by [EKB+08]. However, only a part of the picture of design activities is considered by functional thinking, which does not support other factors such as structural aspects and how component choice and configuration affects system properties. Functional modeling is abstract when considering where the description of the product concept is performed. Furthermore, after functional modeling the development process typically becomes domain-specific[Qam13].

Another possible solution to the common language issue is managing design activities through requirements such as Systems Engineering process [SR09], and the work performed by [WACT+10]. However, although they can be used for goal specifications (of the product to be), or result specifications (documenting the final product), requirements are largely used to control a design concept rather than represent one, which is the sole purpose of the common language discussed here [Qam13].

The Semi-Formal Specification Language (SFSL), CONSENS [GLL12] is a language more specifically related to mechatronics as a holistic design approach. Thereby, a mechatronic concept is specified in terms of a number of aspects, such as function and active structure. If a mechatronic system needs to be described through different views SFSL is more appropriate. However, during the initial design phases, such a representation will also lead to domain-specific concretization. Different semantics related to each mechatronic aspect is proposed by SFSL, so that engineers with different background can relate to their respective aspects. In that sense, this not a pure common language, but can be treated as a language targeting different mechatronic aspects during the conceptual design phase [Qam13].

SysML is an approach to support Model-Based System Engineering (MBSE), that represents a system completely in terms of its structure and behavior through different diagrams [FMS14]. In this way a holistic approach towards system design based on a formal language is established. The usefulness of SysML in the conceptual design phase of product development is explained with a similar approach to the SFSL, [WS09] and [FHP+10]. However, SysML is not suitable for

representing a mechatronic concept during the conceptual phase since it does not cover different mechatronic aspects is suggested by [GSG+09], hence leading to the creation of SFSL. As a result, SysML and SFSL differ in the way aspects are treated. The communication problems between people with different backgrounds is not usually solved by formal modeling approaches such as SysML, nor produced models are intuitive and easy to understand.

In eliminating a reasonable subset of the problems currently faced in conceptual design of mechatronics all presented approaches can be beneficial such a partial model of CONSENS [GTS14a], and ENTIME project [GTS14b]. ENTIME focuses particularly on consistently model-based development as well as early analyzes of dynamic system behavior. In this project a procedural model was developed according to a concrete development task. In this method individual general development steps and usage of the development methods and software tools are described. The process model leads developers systematically through the development of mechatronic systems. Among the others, a consistent transition from the CONSENS to the control and software design model was described. CONSENS is a specific approach for the early interdisciplinary system design. It is the starting point for the subsequent elaboration for the communication and cooperation of all developers.

Additionally, the formalization of documents and models (e.g. requirements, functions, behavioral models, etc.) are described along the development process with a target-oriented search in the Semantic Web. Methodically, system design area is separated from control and software design to focus on the cross-disciplinary creation of a product concept. The design process is followed by detailed elaboration with focus on the integrated control and software. However, to manage the dependency and consistency between different areas defined in ENTIME needs to be enhanced. Different properties of each alternative model need to be grouped into an overall design study, although this is not easy due to only implicit information (about dependencies) being available to the designers.

2.1.3 Transfer of Models and Information between Domains

In designing mechatronic products, the involved multi-disciplinary teams carry out domain-specific development in order to move forward in the design process, leading to the creation of different domain-specific design models, or domain specific views. The communication probabilities between two design domains in terms of: person-person, method-method, model-model, and analysis tool-analysis tool is classified by [FOR+09]. Due to lack of analogy between two completely different methods within different domains, a person-person communication is regarded as error prone, and method-method communication is not always possible.

Communication between analysis tools are possible, because it is not based on their development but on execution (e.g. co-simulation) of developed models. The required information is available explicitly and correctly through models for a model-model communication between two domains [Qam13].

If a universally common methodology is welcomed for mechatronic design, the issue of communication may not seem so significant. The VDI2206 guideline [VDI04] has been suggested but it does not offer any solutions for managing correlations of mechatronic domains, or a proper approach to deal with abstraction between design models and their evolution during design iterations. Nevertheless, the need for model communication approaches is explained as well.

Data exchange standards might be a proper solution for supporting communication. For example, to empower data exchange some international standards have been developed within the CAD community. The most notable one is probably the ISO10303 or STEP standard [Pra01]. Nonetheless, as explained by [Gie08], the industrial uptake of STEP and other popularized standards have not had a decent functionality. No solutions have been found for some basic problems, such as errors in exchanged data and loss of data while using a standardized neutral model [Gie08]. Furthermore, exercising tool-specific standards might not be precisely compatible, leading to errors while transferring data between heterogeneous tools. Therefore, relying on a single tool supplier (too expensive) or depending on documents and meetings for data exchange (too time consuming) would be preferred for organizations.

Beside the STEP Standard, the Web Ontology Language (OWL) [DSB+04] is a decent recommendation for defining vocabularies, basically for acquiring knowledge about a domain. Nonetheless, it is interpreted in several ontologies encompassing the conceptual space of various fields. Moreover, covering the conceptual space of the whole product and its life cycle to have a single ontology is really demanding [Gie08].

Many publications have reported employing the model transformation technology to conform heterogeneous models. Using domain-specific Modelica libraries to perform multi-domain dynamic analysis and controller design tasks on a mechanical system represented by CAD models has been offered by [EBP+03]. An example of integration between CATIA [CAT] and Modelica has been shown by [BWM+06]. [CS07] proposed the integration between Electrical-CAD (ECAD) and Mechanical-CAD (MCAD) models. Most approaches aim towards integrating a system model built through languages such as SysML or Unified Modeling Language (UML) with a corresponding analysis model. Moving from a domain-specific model towards a system model may quite possibly add to the extent of abstraction. To establish relationships between domain-specific models, occasionally the system model is used, such as the following instances:

- Integration of UML and Simulink [HMP04]; [SST+07]; [VD06]; [BON+07]; [Bol07].
- Integration of SysML/UML and Modelica [JKP+12]; [SHF+10].
- Integration of SysML and ECAD [SSP09].

SysML extensions to support modeling of discrete/- continuous hybrid dynamic behavior of mechatronic systems are studies by [CLP11]. A multi-view modeling approach based on the integration of a structural view in SysML with an electrical design (ECAD) view and with the analysis view in Modelica is explored by [SKS+10]. A general framework for expressing a design problem in SysML, and for relating it to analysis and test cases through graph transformations was suggested by [KP10].

In spite of the fact that integrating heterogeneous models in the above-mentioned methods might be an answer to information exchange, yet it is insufficient for supporting consistency during the entire system design, because they are bounded just to establish the relationships between a part of design process with the specific tools. Modeling and dependency management to support model transformations is only feasible in the event that they are built to support consistency. Employing an integration framework in a systematic fashion to initiate communication between tools is necessary. For instance, iFEST [TEkB+12] is a proposal covering an integration framework (principles, interface specifications, guidelines and an integration technology base) as well as supporting tools, framework compliant platforms and adapters for commonly used embedded systems tools. Its main notion is that distribution of design activities is done over multiple tool sets and each tool acts as both provider and consumer of services and data to\from other tools. No predominant platform exists between tools, instead while running, tools directly communicate with each other. It is worth noting that this approach is different from ad-hoc, point-to-point integration of tools, owing to the fact that a predetermined specification defines each tool's services and data, empowering the utilization of the same interface in any other tool chain configuration. Moreover, by employing a language (TIL) [BEKL+14] and Cyphy [SLN+12] proposed for such languages [Qam13], new tool chains can be built.

2.2 Variability management

This section gives a guide for the development of a Software Product Lines (SPL), as well as, for the development and maintenance environment of the SPL which is provided by supporting tools.

Variability modeling supports the development and the reuse of variable development artifacts [PBL05]. In a software product line, variability is an essential

property of domain artifacts. Hence, variability modeling is used to capture the variability of domain requirements, architecture, components, and tests. Variability is introduced during the product management subprocess when common and variable features of the software product line applications are identified [PBL05].

According to Software Engineering Institute, a software product line can be defined as: *A SPL is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [AFr].*

A set of common assets which includes requirements, design, test plans, test cases, reusable software components and other artifacts is the main attractive part of SPL. Instead of developing for each product separately, the common assets are shared in all the derived products. To increase the productivity, quality and decrease the cost and time to market can be applied the development of individual products from a set of common assets. Sharing of core assets reduces the development effort which can ultimately decrease the time to market and cost. SPL consists of two major development parts [MS10]:

- Domain engineering: *the development of core assets is based on the commonality (common features) and variability (varying features) in the products.*
- Application engineering: *the development of individual products by reusing the core assets and adding functionality which is specific to each product.*

Combining platform and mass customizing in order to provide customized products has many implications for the development process. The basis for deriving the individual products is provided explicitly by the variability modeled in it. Basically, an accompanying project is set up, when a new product is developed. Then requirements are categorized as commonality or variability (part of product line or product-specific). An initial product version is instantiated to make products with various assets (e.g. architecture, implementation, etc.). In this stage of development process, just up to 10% must be developed in further steps; i.e. up to 90% of the product may be available from reuse [LSR07].

Following points causes hurdles and obstacles in the development of SPL[MS10]:

- Lack of investment in SPL because product line as a software development technique is still under development phase
- Lack of case studies from small industry opposite to big industry (aerospace, automobile, military, mobile phone)
- *Open source community should be involved in the development of product line tools and techniques except few bigger organizations*
- Management and organizational risks

- Unequal attention from both academics and industry

The following survey is a summary of the most relevant pieces of variability modeling.

Six variability mechanisms for providing variation points in a software product line are described by Jacobson: inheritance, uses, extensions, parameterization, configuration, and generation [Gri97]. Five variability mechanisms are described by Bosch: inheritance, extensions, configuration, template instantiation, and generation [Bos00]. There is a PRAISE project from Alcatel/Thomson-CSF Common Research Laboratory (LCAT) with focusing on the design and representation of a product-line architecture with UML [CEKJ+00].

CelciusTech [BC96] shows parameterization as a variability mechanism to manage the parameters and understand the dependencies between them. NASA Goddard Space Flight Center Flight Software Branch (FSB), additionally to this result, created a technique to represent variability [MLS+00, MTS00], which created a new symbol to identify the variable elements. Feature-Oriented Domain Analysis (FODA) and Ensemble Development Life Cycle (EDLC) logic is used in both Software Technology for Adaptable Reliable Systems (STARS) [Dav95, Tas95] and Boeing's avionics software [BCC+99, Sha98] by providing the reuse with a list of variants to choose from rather than the ability to create a unique variant of their own. There are many processes specifically for building product lines, such as Product Line Software Engineering (PuLSE) [BFK+99], KobrA [ABB+01], Wheels [CJB00], Bosch's Product-Line Design Process [Bos00], and Family-Oriented Abstraction, Specification, and Translation (FAST) [W+99].

2.2.1 Feature Modeling and Configuration

An important concept of an SPL is the feature model [PCF14]. To represent the common and variable features in SPL feature models are deployed [KCH+90]. An increment in functionality or a system property relevant to some stakeholders is represented by a feature [KCH+90], that may refer to functional or non-functional requirements, architecture decisions, or design patterns [BCD02].

To represent different viewpoints, subsystems, or concerns of the software products, developing an SPL involves feature modeling in practice [BSSP02]. Therefore, to have tool support to aid the companies during the SPL variability management is necessary [PCF14].

In the following used phrase of feature modeling are briefly described:

Feature description: consists of a feature definition and a rationale [BPSP04].

Feature definition: explains characteristic of the domain by the feature to demonstrate what this feature is about. The definition is presented in from of an

informal text only or in from of a defined structure with predefined fields and values for some information such as the binding of the feature, i.e. a feature is introduced for the time in the system (configuration time, compile time etc.) [BPSP04].

Rationale: explains when to choose a feature, or when not to choose it [BPSP04].

Feature value: is an attached type/value pair of a feature to describe non-boolean features more easily [BPSP04].

Feature relations: describes valid selections of features in a domain. The feature diagram is the main representation of these relations. Additional relations can be attached to a feature [BPSP04].

Feature model: is valid combinations of features in a domain, because not all combinations of features are valid and lead to useful software systems [KCH+90]. Feature model is structured hierarchical, whereas each feature can have subfeatures [CE00]. The connections between features indicate whether they are AND, OR or ALTERNATIVE. *The children of and-groups can be either mandatory or optional [Bat05]. It is a common notion for variability and their semantics is as follows: the selection of a feature implies the selection of its parent feature. Furthermore, if a feature is selected, all mandatory subfeatures of an and-group must be selected. In or-groups, at least one subfeature must be selected and in alternative-groups, exactly one subfeature has to be selected. Finally, all cross-tree constraints must be fulfilled [TKB+14].*

Feature diagram: shows graphical representation of a feature model, as shown in Figure 2-2. It is a directed acyclic graph where the nodes are features. An explanation of features and their connection and its representation in feature diagrams are given in Table 2-1 [BPSP04]. *A feature is abstract, if it is not mapped to implementation artifacts and concrete otherwise [TKE+11]. In Figure 2-2 the features Hello and World are mandatory and simply print the features name. The features Wonderful and Beautiful are alternatives, but not required.*

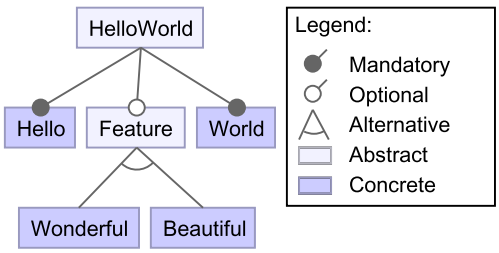
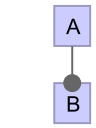
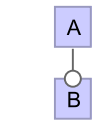
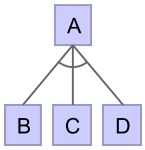
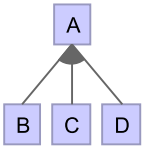


Figure 2-2: A simple feature model modeling an SPL of Hello World programs [TKB+14]

Table 2-1: Explanation of feature diagram elements [BPSP04]

Feature type	Graphical representation
Mandatory: Mandatory feature B has to be included if its parent feature A is selected	
Optional Optional feature B may be included if its parent feature A is selected.	
Alternative Alternative features are organized in alternative groups. Exactly one feature of such a group B, C, D has to be selected if the group's parent feature A is selected.	
Or Or features are organized in or groups. At least one feature of such a group B, C, D has to be selected if the group's parent feature A is selected.	

Cross-tree constraint: is dependencies which cannot be expressed otherwise. Usually below the feature diagram is shown a propositional formula over the set of features [TKB+14].

Configuration: is a subset of all features defined in the feature model. A valid configuration is the combination of features, which is allowed by the feature model *(i.e., if it fulfills the semantics of groups and all cross-tree constraints). Otherwise, the configuration is called invalid [TKB+ 14].*

Feature-Oriented Software Development

FOSD provides a paradigm for the construction, customization, and synthesis of software systems [AK09]. *The basic idea of FOSD is to decompose a software system in terms of the features it provides. The goal of the decomposition is to construct well-structured software that can be tailored to the needs of the user and the application scenario [TKB+ 14].* According to this idea, to plan and implement SPLs (referred to as domain engineering) as well as to select features and generate customized programs (application engineering) FOSD can be used. To support FOSD process an Eclipse-framework based is developed and named FeatureIDE. During this process four following phases are distinguished [TKB+14]:

- Domain analysis: *to capture the variabilities and commonalities of a software-system domain, which results in a feature model.*
- Domain implementation: implementation of all software-systems of the domain at the same time, while mapping code assets to features
- Requirements analysis: requirements mapped to the features for a customized software system, resulting in a configuration
- Software generation (or composition): generation of a configuration and the domain implementation

Among the above phases domain implementation and software generation highly depend on each other. FeatureIDE is a SPL implementation technique, which describes how features are mapped to implementation artifacts and how to generate customized software systems [TKB+14].

2.2.2 Configuration of Mechatronic Multi Product Lines

To extend the concept of product lines to integrate software development artefacts like requirements, classes or components, many approaches were proposed [AK09], [Buc10], [Str04]. A way to connect classes as well es refinements of classes with features. is introduced by Apel et. al [ASW+11]. Those classes were modified based on the previous mentioned refinements depending on the selected configuration of a product line. Finally, generated artefacts provides the software. Although this approach is provided for SPL not specially adequate for mechatronic systems because neither hardware artefacts nor dependencies between software and hardware artefacts can be considered.

The concept of product line is extended by Streitferdt and an association is considered between features and requirements [tSH+05]. Therefore different types of dependencies are identified which mainly they can only be applied to features within one feature model. Additionally deployment dependencies are not taken into account.

Clements et. al. [CN02] introduces another refinement of the concept of product lines, where a process for the development of software product lines is presented. This approach is applied to multi product lines for mechatronic systems.

A process for a combined hardware and SPL is outlined in [LAL+09] which would meet the scenario of a product line for mechatronic systems. Nevertheless, associations between features and dependencies are not considered.

Streitferdt et. al. [tSH+05] suggests one approach where the configuration of software and hardware features is considered within one product line. However, [tSH+05] shows mapping requirements and using one single feature model, which

limits the differentiation between software and hardware features. Furthermore [tSH+05] integrates no information about the deployment and execution environment.

Rosenmüller et. al. [RS10] focuses on the handling of multi product lines, where Rosenmüller et. al. introduced composition models to associate multiple feature models within one multi product line to generate a configurator based on the composition models.

Brink et. al. [BPS12] provides the mapping of one abstract feature to many features of different feature models. Additionally this approach enables the generation of a user specific view with for example hidden options, regardless of whether these are mandatory or just not available for a specific customer. This work considers hardware parts, a deployment constraint or an abstract configuration view, which is necessary for modeling mechatronic systems. Furthermore a direct configurable view is provided for customers through the use of abstract feature models and a preselection of features could be realized by the use of Original Equipment Manufacturer (OEM). Therefore the configuration of multiple product lines is enabled without any expertise about underlying models.

In [RW06] an approach for modeling multi level feature trees is introduced by Reiser et. al.. This approach organizes feature models hierarchically but does not consider different development artifacts.

In [DSB+11] combination of different modeling methods for variability by the use of web services is presented.

Czarnecki et. al [CK05] introduce an approach where the configuration is tiled into different steps based on different levels of abstraction to simplify the process of configuration. Further variation points are affected and the available options are limited by a previously selected configuration. This configuration is allowed through various stakeholders where an expert starts with the configuration and finally the customer gets a restricted view with a highly limited choice.

A progressive approach for the configuration of product lines is provided by Benavides et. al. [WDS+09]. Therefor, a formal model is used and mapped to a constraint satisfaction problem which than can automatically be resolved by a constraint resolver. It is a good way to handle the complexity of a product line not good enough for multi product lines [BPS12].

2.3 Tooling

In this section, simulation and their tools regards to mechatronic system is discussed. Here, is showed how they can be coupled to analyze and simulate

the entire system with different simulation tools. Also different engineering frameworks are reviewed to point out wide variety of tools and methods to encapsulate individual analysis. Lastly, different variability management tools shows how feature modeling with graphical and textual notation can be presented and which tool is used in this work.

2.3.1 Simulation

The simulation is a depiction and an execution of a technological process from a system. A simulation calculates the model, in which input variables are converted into output variables. The simulation can be used to examine whether the requirements for the real process are met or not. However, a simulation requires a physical, mathematical or abstract model. According to VDI 3633, simulation is defined as follows:

Simulation is the reproduction of a system with its processes in an experimental model, in order to reach knowledge that can be transferred to reality. In particular, the processes are developed over time[Ing13]. From the definition it can be deduced that a simulation is dependent on a model and a system. *Without system analysis, no mathematical model! Without mathematical model, no computer simulation [Bod07]!* System simulation is characterized by the fact that a product is presented with its environment in a virtual environment. This allows real hardware to be integrated into the simulation. The simulation procedure is as follows [mus]:

1. *Formulation of the problem*
2. *Development of the model*
3. *Generation of data*
4. *Implement the model*
5. *Verification and validation of the model*
6. *Implementation and evaluation of simulation sequences*
7. *Interpretation of the simulation results*
8. *Implementation of the simulation results into reality.*

In the product development, there are different application areas of the programs to meet the requirements of the customers while at the same time increasing the product quality at lower development times and costs. In order to meet ever-increasing requirements in the market, product development cycles must become shorter and more efficient. This development can be observed mainly in the fields of automotive production or aerospace, where the costs and time are reduced by means of prototypes of vehicles and vehicle components. A

simulation helps to speed up the process by linking different components to a whole system so that early predictions and corresponding early concept decisions can be made. Simulations of possible application scenarios in early development phases avoid errors caused by material renewals, special design specifications or novel mechatronic components, and provide answers to possible questions that would otherwise have to be tested at additional costs in prototypes. Since, in the case of modeling which in turn interact in a complex mechatronic system, such as, a vehicle in which hydraulic and electrical components interact, it is no longer enough to work with only one software. In addition, since the various components have different mathematical requirements, problem-oriented modeling and simulation software must be used accordingly. In order to form a complete system, all components which must be coupled both from different domains or tools, as well as in numerous mathematical descriptions, each requiring different numerical solvers, must be coupled. In order to be able to create simulation in a realistic manner, several physical phenomena must interact with one another [VIN].

Since, the mechatronic system consists of more and more sensors, actuators and different physical phenomena, the entire system must be split into different subsystems, which are depicted in different system tools. In order to ensure a seamless combination of the subsystems and various data, interfaces are required with which the programs can exchange data and models. For model exchange or co-simulation, programs can offer special interfaces which are compatible with certain programs or support the interface standard Functional Mock-Up Interface (FMI)[FMI14]. The difficulty with co-simulation is to couple models with different time constants to a whole system. Due to the number of different application areas in the subsystems, it can quickly lead to errors during the data exchange. In order to avoid it, only short time steps have to be selected during the data exchange of the models.

2.3.2 Coupling of Simulation Tools

The industry is increasingly looking for a standard approach that can be used to analyze and simulate the mechatornic systems with different simulation tools from different engineering disciplines, e.g. simulation of mechanical multi-body systems together with electronics. This combination of reality is modeled and simulated with a tooling in the virtual world. Therefore a combination of multi-body modeling software and control software is needed for simulation of entire system with all involved simulation tools. An example of these application systems are simulation tools from different engineering disciplines, which are integrated in a common simulation environment.

Complex mechatronic systems can be simulated either with a simulation tool or with several tools. The first solution uses a tool whose main focus is on a discipline, while the components from other disciplines with simplified models are considered. In this way, modeling costs are saved at the expense of the modeling depth. In the second variant, a tool is used for each discipline. This results in models with a high level of detail, but the user is confronted with the coupling problem. The aim of the tool-dependent coupling of simulation tools is the development of a user-friendly interface for the user. In this work, on existing simulation programs and their interfaces will be built on. Thereby interfaces become problem-oriented. It is adapted to the technical requirements of the system and to user requirements. This avoids the expense of a comprehensive total solution and can be concentrated on already existing interface architecture. A comprehensive study will then provide information on how this interface is to be expanded to ensure the reliability of the simulation.

Because a simulation tool is based on a description method with a certain level of detail, a pairing of multiple programs is generally similar to leveling their models on a common abstraction level or accessing a data interface served by both programs (see Figure 2-3).

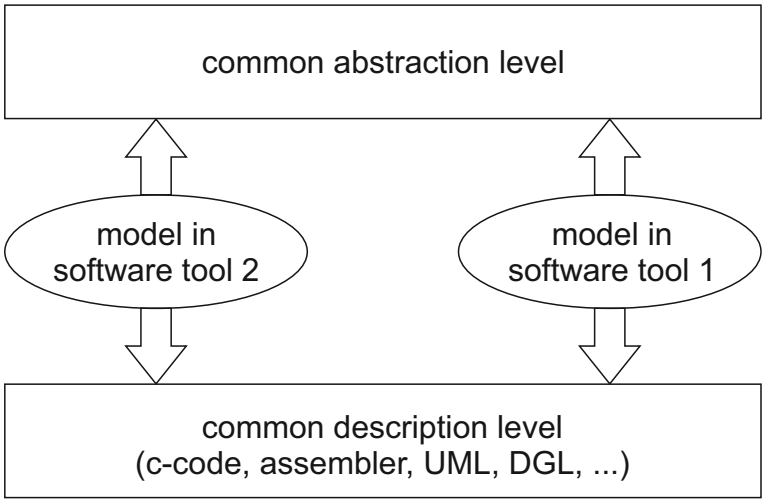


Figure 2-3: Coupling principle of two simulation tools

The wide range of application areas and the high number of different development tasks require different tools. The tools have to be specially adapted to the disciplines used in order to meet the requirements. The more tools are used, the more problematic is the assembly of a complete system from the various models. The large number of areas and variables make co-simulation difficult, since the integration of different tools can often alter signals or limits, which

ultimately leads to incorrect results. There are difficulties with a co-simulation in the connections of simulation tools, the correct data exchange, the coupling of systems with different dynamic behavior as well as in the guarantee of the correctness of the simulation results. Since the increasingly complex systems require a coupling of several models from different tools, an interface that solves exactly this problem is required.

GUSMA[HRG11] stands for coupled company-wide simulation of mobile work machines and was a project from the years 2008 to 2011, in which the Karlsruhe Institute of Technology (KIT) with other partners developed a platform in MATLAB/Simulink with which various simulation tools can be co-simulated. The goal of the project was to standardize the developed platform to enable co-simulations based on MATLAB/Simulink. As a result, the user would only be able to work with MATLAB/Simulink. Therefore, an interface to the MATLAB workspace is required for the coupling of a simulation software with GUSMA [HRG11, VHG10].

In order not to have to reset the variables and limits after each integration, the FMI was created, which allows various tools to exchange and co-simulate data via an Functional Mock-Up Unit (FMU). After GUSMA could not be implemented in the market, the FMI has established itself as an international interface standard and is supported by numerous tools. Using this interface, simulation tools can be coupled without losing data or falsifying results.

FMI/FMU:

The FMI is a tool-independent interface standard, which has spread in the industry. Daimler AG has initiated and organized the development of the FMI, with the aim of improving the exchange of simulation models between suppliers and OEM. The FMI is in constant development and has already been extended by FMI 2.0. With the FFMI interface, one is able to couple physical models from different domains and to standardize model components in industrial simulation tools (FFMI for model exchange) and co-simulation interfaces in the non-linear system dynamics (FMI for Co-Simulation), as shown in Figure 2-4.

Dynamic files can be compiled using a data archive, the FMU. The FMU models are provided as a .zip file. This data includes an Extensible Markup Language (XML) file for describing occurring variables and a C code for a script that lists all of the formulas that are used in the model. The generated C code must conform to the FMI standard and therefore be converted into a Dynamic Link Library (DLL) [1] [dll19] so that the FMU can be simulated. All required model equations

[1] Data and models can also be exchanged via offered libraries, such as via a DLL. A DLL is a file that allows various programs to access program data. Any program that uses the DLL file can use and implement the codes and data. The Shell.dll file provides routines for various

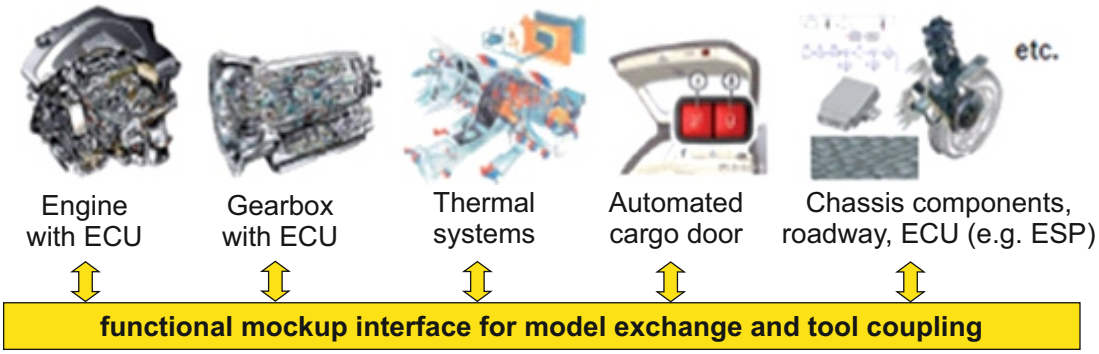


Figure 2-4: Improving model-based design between OEM and supplier with FMI [FMI14]

or access to co-simulation tools are provided via System-Functions (S-Function). An S-Function is a computer language description of a Simulink block [Wha]. The S-functions can be provided in both binary and source formats. The XML file can be read by any programming language such as C/C++, Python or Java, so that no additional effort is required with different tools [FMI14].

With the establishment of the FMI for Model Exchange and Co-Simulation as the standard, the exchange of simulation models across tool boundaries has greatly simplified, and users benefit from more efficient processes and completely new simulation methods[EUP], explains Torsten Blochwitz, Research & Development Manager at ITI GmbH and FMI Project Manager at the Modelica Association.

The co-simulation is divided into the master/slave application. The master controls the co-simulation by analyzing the connection graphs, selecting the appropriate simulation algorithm, and controlling the simulation accordingly. Slaves are simulation tools that simulate subsystems. The slaves can exchange data, execute control commands, and display status information.

In the co-simulation with the FMI, all information about the slaves that are relevant for communication in the co-simulation environment is made available in a slave-specific XML file. This helps in characterizing the capabilities of the slave to support extended master algorithms, such as the use of variable communication step sizes or higher-order signal extrapolation. The parameter constellations and call sequences depend on the capabilities of the slave. Depending on the internal state of the slave and the function parameters, the slave can decide which action to perform before the calculation [FMI14], [Arn10].

functions that make it possible to drag and move data. Software development companies offer various CAE, CAD or FEM tools that can communicate via in-house interfaces and exchange data [dll19].

Figure 2-5 shows a distributed co-simulation in which the master and the slave run on different computers. The slave can be any simulation tool. The data exchange is handled by a communication layer implemented in a particular FMI wrapper. The master and the slave do not perceive the communication layer, but only use the FMI for co-simulation [BOA+11].

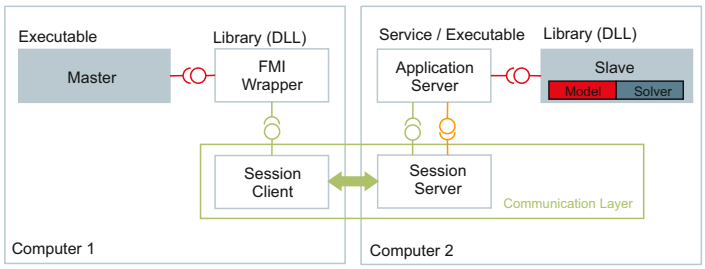


Figure 2-5: Distributed co-simulation scenario

2.3.3 Integration Tools

An integration tool is *an engineering framework that provides wide variety of tools and methods to encapsulate individual analysis or simulation models. Almost any software application can be included in a workflow: simulation models, mathematical models, databases, as well as CAD and Computer-Aided Engineering (CAE) models.*

ModelCenter[Modc], ModeFRONTIER[Modb], SCADE Suite[Sca19] and such tools are a product line of Process Integration and Design Optimization (PIDO) tools, that empowers users with a Model-Based Development Environment. *For instance, through ModelCenter, a work flow process can be created by integrating disparate models. ModelCenter provides out-of-the-box connectors which enables data exchange with different tools. As a result, dependencies between properties existing in a range of models can be modeled [Qam13].* SCADE Suite is suitable for the implementation of security-relevant embedded systems for critical applications spanning requirements management, model-based design and simulation. Due to the large range of functions, however, getting started is difficult, i.e. the existence of this function is not readily apparent. The usability of the SCADE Suite is difficult due to the partial absence of operating concepts, which are accustomed to other Integrated Development Environments (IDE) [Sca19].

"IQUAVIS" developed by Japanese companies ISID and iTiD Consulting is a system that supports conception design work in complex product development in the automobile industry. In the concept design corresponding to the early stage of product development, functions are installed to examine hardware and control

software, to create a design proposal that keeps the logical consistency of both [iQU].

ModelCenter and ModeFRONTIER offer optimization tools in addition to the possibility of integration. ModeFRONTIER also provides a connection option for Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL) simulations in addition to various interfaces to simulation tools. ModelCenter, on the other hand, is specialized in the optimization of models. Abaqus[Aba] is a computational program for the analysis and evaluation of models and Rational Rhapsody is a development environment for analyzing models and simulations. All programs contribute an important part to their field of expertise, but the combination of the different tools helps to achieve optimum and realistic product development.

2.3.4 Variability Management Tools

There are many different tools for feature modeling with graphical or textual notations and for creating a valid configuration. Pure systems[pur] and Gears [Big] are the two main commercial products. KConfig [ker] and CDL [eco] are especially well known in open source development. Thereby a textual modeling notation and a configuration editor are provided that checks for valid configuration and, to some degree, helps resolving conflicts. Whereas CDL is primarily known from the operating system eCos [BSL+10], most prominently KConfig is used for Linux. Also, there are several academic tools, such as GUIDSL. It is a tool for product-line development to support feature modularization. A generated form is used to create and save configurations [Bat05]. GuiDSL focuses on providing explanations, why a certain feature cannot or has to be selected. *Software Product Line Online Tools (SPLOT) is a collection of interactive online tools for providing a feature model editor as a tree view, a configuration editor with decision propagation, automated analysis on feature models, and example feature models [MBC09, TKB+14].* In contrast, the focus of FAMA framework is on the comparison of different solvers for the automated analysis of feature models [BST+07]. Also numerical constraints and optimal configurations can be derived in FAMA. In the Feature Modeling Plug-In (FMP) numerical constraints are also allowed [AC04]. Thereby tree views for creating feature models, configurations, and partial configurations in the process of staged configuration are provided. S2T2 Configurator presents a tool supporting users in creating valid configurations by providing visual explanations derived from a satisfiability solver [BJS09]. Indeed similar facilities is provided by FeatureIDE.

Feature-oriented programming:
The AHEAD tool suit is provided for feature-oriented programming tools [Bat06]. New keywords extends the syntax of the host languages Java 1.4, XML, and JavaCC

to support feature-oriented programming. Different composers as command-line tools are provided by AHEAD. Similarly, the syntax of C++ is extended by FeatureC++. Using command-line tools can be composed by new keywords and feature-oriented C++ files [ALR+05]. To create and compose feature-oriented files FeatureHouse as a language-independent approach can be used [AK09]. Currently, Java 1.5, C#, C, Haskell, JavaCC, Alloy, and UML can be composed by FeatureHouse. All three tools do not have any support to decide whether the configuration is valid or not. They just list all features as a configuration.

All these tools are integrated by FeatureIDE in Eclipse and connects feature-oriented programming to feature model and configurations [TKB+14].

FeatureIDE

FeatureIDE is a supporter of several implementation techniques for FOSD with low costs. Also, for teaching and comparing SPL implementation techniques with respect to their applicability, FeatureIDE is qualified for the development of SPLs. As mentioned above, all phases of FOSD is supported by FeatureIDE.

In FeatureIDE, by adding and removing features in a graphical editor a feature model can be constructed graphically. Whereby feature models can heavily be changed over time and thus, a feature including its subfeatures can be moved to a new parent feature, with FeatureIDE the feature models are stored in an XML format and the feature model could be edited graphically and textually simultaneously.

In the next step, to configure the features of a feature model a graphical editor is needed. A configuration editor supports requirements analysis within FeatureIDE. The feature model from domain analysis is given as input to the editor and configuration choices are offered. Required features are selected and saved in a configuration file. The user can create multiple configurations and mark one of them to be the current configuration for which FeatureIDE composes and compiles source code. Developers are supported by configuration editor in creating valid configurations. According to the feature model only configuration choices exist(i.e., it propagates decisions) [MBC09]. For instance, a user can not eliminate a mandatory feature in the feature model can not select two alternative features at the same time. Features for which the user can not change in the current state are grayed out. Additionally, selection or elimination of marked features turns an invalid configuration into a valid one (left editor in Figure 2-6), to support the user in creating valid configurations. Also, right editor in Figure 2-6 is an extended configuration editor. In the simple configuration editor, there are four states for each feature: manually selected, automatically selected, automatically eliminated, and undecided. A manual elimination causes automatic selection and elimination

of many subfeatures and other features, which depend on the eliminated feature [MBC09]. For instance, through eliminating of a feature all its subfeatures can not be selected anymore [TKB+14].

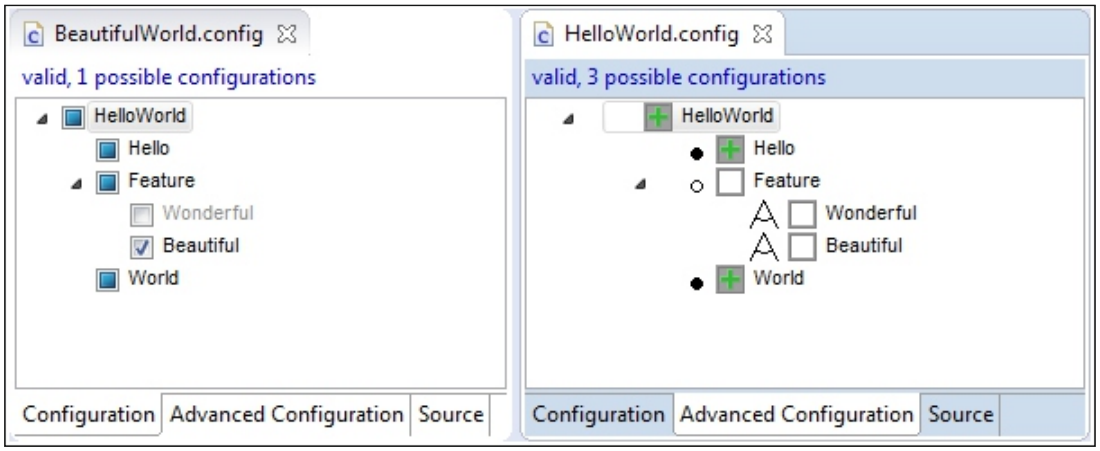


Figure 2-6: On the left side, an invalid configuration editor and some highlighted features On the right side, the advanced configuration editor, in which features can be eliminated to reduce the remaining configuration options [TKB+14]

Also a feature model can contain cross-tree constraints. For this purpose, a constraint editor is provided in which constraints can be created or edited. With a content assist for convenient handling as well as syntactic and semantic validity checks is enriched by the editor (see Figure 2-6). For example, mismatching brackets, dead features, false optional features, unsatisfiable constraints, and redundant constraints could be detected by these checks[TKB+14].

2.4 Scientific Gaps

The comprehensive of literature review leads to the following conclusion:

- Designing: in aid of mechatronic design different solution are proposed in literature. The challenged issues as follows are highlighted in the chapter 2.1:
  - Lack of a common language to represent a concept due to different design phases
  - Lack of consistency between design phase
  - Transfer of models and information between domains

- Variability management: one approach to configure of mechatronic multi product lines has been proposed, but it is not completed. Variability management can simplify the configuration of mechatronic systems.
- Tooling: in aid of mechatronic design following challenges are highlighted in chapter 2.3:
  - Lack of common tools for coupling of different common simulation tools
  - Challenge of integration tools
  - Challenge of integration of variability management tools within traditional mechanical tools

By considering the above-mentioned conclusions from the literature review, focusing research efforts in the potential areas during the variants of mechatronic system design was made possible. Each product variant has its own variant of simulation models. Different software is a way to easily implement the individual product variants that originally required individual model variants. In many systems, this amount of variability is now many times larger than before variability of models came into play. Therefore, presently a strong need for adapting product line engineering can be observed in the mechatronic domain, especially when size and complexity of models exceed the limits of what is feasible with traditional approaches.

Lack of models and information delegation between domains is identified as a problem in the current literature. Also assessing consequences of selecting between alternative modeling concepts is an issue. This thesis tries to cover this missing challenges by a concept of Multifunctional Model Client (MMC). Basically this concept collects, store and administrate clue information required for fulfilling some tasks, like assisting activities related to definition of variability, supervising variable artefacts and contributing to activities related to sorting out variability.

Also, a comprehensive literature review ensues that there have been various solutions in literature to support mechatronic design, but lack of variability management is a common factor in each challenge. For applying local design changes, dependencies within mechanical design should be considered, though historically it is difficult to have integration with mechanical design tools, making dependency management for mechanical design even more challenging. Many incomplete approaches have been proposed to ensure consistency between models in which dependency modeling has an essential role.

Last but not least, the provision of a feature modeling language and a supporting tool is the basis enabling modeling and management of dependencies during the design process. Furthermore, methods for enabling communications between models will also be implemented in a tool. The predicted influences of the developed support on the success factors are included in the implementation.

3 Developing Mechatronic Systems

The conception of mechatronics is made from the integration of knowledge from different areas of physics and technical disciplines. This integration is used to achieve a synergic effect. To obtain a product with highest possible technical parameters, mechatronics can be consider as a new and innovative technology. A guide for the design of mechatronic systems is the V model, which is adopted from software development and adapted to the requirements of mechatronics. In the thesis of Oestersoetebier [Oes17] the V model procedure is detailed and supplemented with procedural steps. In this chapter, this methodology is illustrated and explained in details, how the feature modeling helps to manage the variabilities during the design of mechatronic system. Additionally, some steps and activities from previous work [Oes17] is just shortly addressed.

3.1 Design Concept of Mechatronic Systems

There are numerous development guidelines and methodologies for design of mechatronic systems, which are mostly focused on individual disciplines, such as VDI Guideline 2206 [VDI04]. In this work, the design concept extends the V-model by providing additional details and demonstrating how the reuse of information can automate certain design steps. Figure 3-1[Oes17] illustrates the V-model approach in detail and describes the design process of mechatronic systems step by step. Furthermore, the V-model includes interdisciplinary coordination to ensure the exchange of information and consistency among the different disciplines involved. The ENTIME project [GTS14b] uses market-based solution elements to dynamically allocate tasks, which is difficult to achieve using traditional design techniques. In addition, the project enables communication between independent systems for the evaluation of software development methods.

3.1.1 Design Sequence

System Design

According to Figure 3-2, interdisciplinary system design is the first phase to specify and formalize the first idea of a system development. The result of a possible solution selection and analyze them is the principle solution. To elaborate and define the discipline-specific tasks and responsibilities, a principle solution is needed [Oes17].

In the following design process of mechatronic systems is described in details according to the V model, as shown in Figure 3-1.

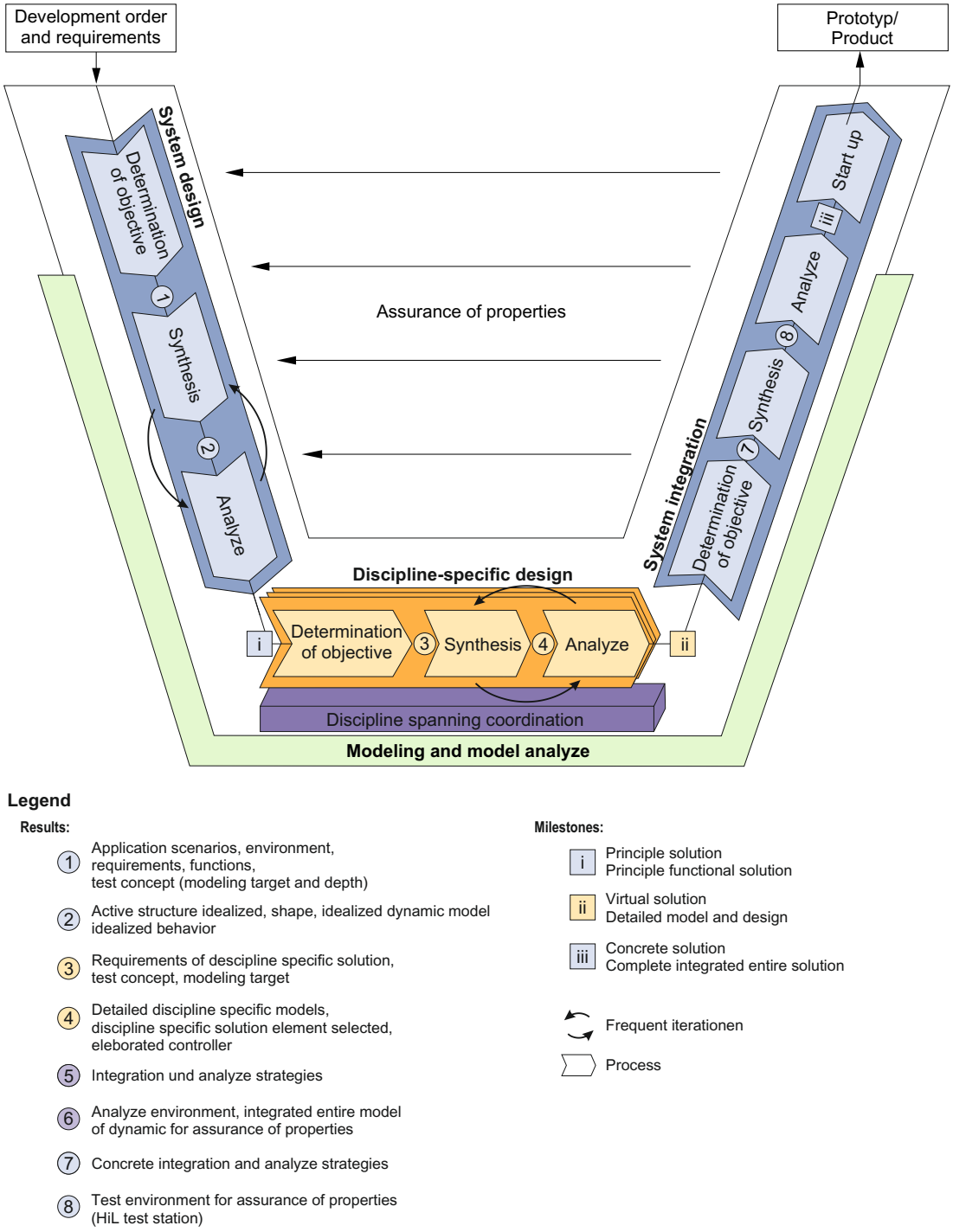


Figure 3-1: V-model of VDI guideline with details [Oes17]

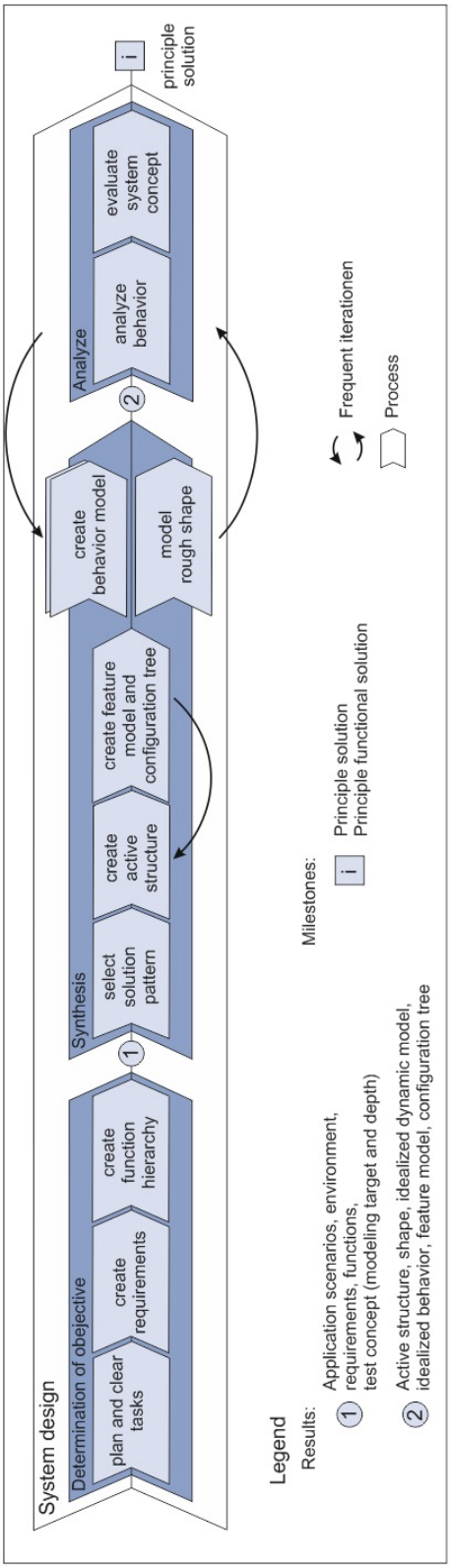


Figure 3-2: Discipline Spanning Conceptual Design or System Design [Oes17] extended by feature model

(i) **Determination of objective:**
The first step in the system design phase is to specify the tasks and boundary conditions clearly. For this purpose, first requirements, functional and structural models are first prepared in the course of the planning and analysis of the task (for example environment and an active structure). The application scenarios are the basis for this. It describes operating situations and the desired behavior of the system [Oes17].

Requirements:
This is the computer-internal representation of the requirements. The basis for this is the requirement list. It provides a structured collection of all requirements (for example, size, performance data) to the product to be developed. These requirements are valid throughout the product development as a benchmark for the product to be developed, which must be sufficient. A distinction is made between desired requirements and fix requirements. Each requirement is described verbally and, if possible, defined by attributes and their expressions [GFD+08].

Environment:
This model describes the environment of the system to be developed and how the system is embedded within this environment. Relevant influencing factors (e.g., subordinate systems) as well as the influences themselves (e.g., information) are identified. In addition, external influences on the system can be characterized as disturbances. Furthermore, the interactions between these influences are analyzed. A consistent set of common operating situations is considered to define the conditions under which the technical system must function. Influences that trigger a transition of the system are characterized as events. Catalogs of influence areas and influencing factors support the creation of environmental models [GFD+08].

Application scenarios:
Application scenarios are the first concretizations of the system. They specify the way in which the system is to behave in a state and a particular situation, and in what manner and due to which events state transitions should take place. Application scenarios characterize the problem to be solved for a given case and also roughly describe the possible solution [GFD+08].

Functions:
Functions are derived which must be fulfilled. For this purpose, the main function is subdivided into subfunctions by means of a function hierarchy. The formulation of the functions is independent of possible technical solutions. This requires abstraction and concentration on the essential problems that need to be solved [PBF+13]. At the same time, the problem is divided into different parts until they can be solved individually [GDP+10].

The results of determination of objective step, as shown in Figure 3-2, are application scenarios, environment, requirements, function, test concept (modeling target and depth).

(ii) **Synthesis:**
In this step, specific concrete solution pattern also solution elements associated with the solution patterns can be selected based on the concretized requirements by using parameters of idealized models [Oes17].

Solution pattern selection:
In the step of searching for a solution pattern, solution principles are searched which fulfill one or more subfunctions. The goal here is to find as many as possible solution patterns per partial function in order to expand the solutions. If a partial solution is prepared in the form of solution patterns, the developer can be supported by solution knowledge, as explained in 3.1.3. Otherwise, she/he has to rely on experience or to search manually. Effective re-use also requires that solution patterns be made available in a knowledge base [Oes17].

Active structure:
An important partial model of CONSENS [GTS14a], as shown in Figure 3-3, is the structure of active principles (or active structure) [GLL12], which displays the system elements and their interactions [GDP+10].

An active structure is created for each resultant solution variant. It contains all the system elements and the flow relations between the elements. For this purpose, each system element is used by an active structure that is associated with a solution pattern. A new solution has to be developed for a partial function that has no solution pattern. In this case, the interfaces of the system element are defined by active structure. For implementing the requested system it results one or more solution possibilities. Each one is described in active structure [Oes17].

Feature model

According to the active structure a feature model is created, which offers a selection between variants of system elements while supporting their interactions. The approach in this thesis is that every component is introduced as a feature to be demonstrated within a feature model. Here, every alternative model is considered, particularly different variant of model design to represent the system design.

The feature model contains the name of every possible model/design according to the tooling name and modeling depth. It is documented as a parent/child relationships that denote features and subfeatures. Regarding to the definition of configuration editor in the subsection 2.3.4 a configuration

tree according to the feature model can be created. A tree of models from different components is depicted. To create an initial system knowledge (see 3.1.2) a list of existing models is provided and showed in the configuration editor. According to this configuration, an initial dynamic model is created, as described in the following section. Every change in a dynamic model causes changes in partial models and inverse. Along that all of this changes are mapped to the system and solution knowledge.

Initial dynamic model

To analyze the dynamic behavior of the system at this point an idealized dynamic behavior model, based on the feature model is created. Dynamic models are an important success factor in the design process of mechatronic systems to reduce errors. The detailing of dynamic model will be increased during the future process steps. If a suitable solution pattern has been identified using the methodologies of the solution based on application scenarios, functions and requirements, a pre-selection is made from the set of available models, which are part of the solution pattern and are therefore suitable for describing the pattern [GTS14b].

Nevertheless, different models with different assumptions, abstractions and limits can still be selected. Subsequently, a dynamic model according to the selection of user is created. If the selected model exists in the library, they are created completely, otherwise partly. Involved components are connected together to make a runnable model. If components are not compatible with each other, interfaces must be restructured to make them compatible. An abstraction (modeling depth [Loc]) can be roughly assigned to the different phases of the design process. At this level of design an elementary design and simple objectives analysis is sufficient, therefore a low modeling depth with low complexity can be used, as shown in Figure 3-4.

Shape model:
In the conceptional phase the shape of the system is determined and modeled. This applies in particular to operative face, knots, hatch surfaces and support structures. The computer-assisted modeling is done with the help of the usual 3D CAD system [GFD+08].

The synthesis step results active structure, shape model, idealized dynamic model and idealized behavior model, as shown in Figure 3-2.

(iii) **Analysis:**

In this step, all modules of the idealized function are required. The dynamic behavior of system is analyzed and integrated modules are evaluated to create a principle solution. Analysis and synthesis usually take place in many iterations.

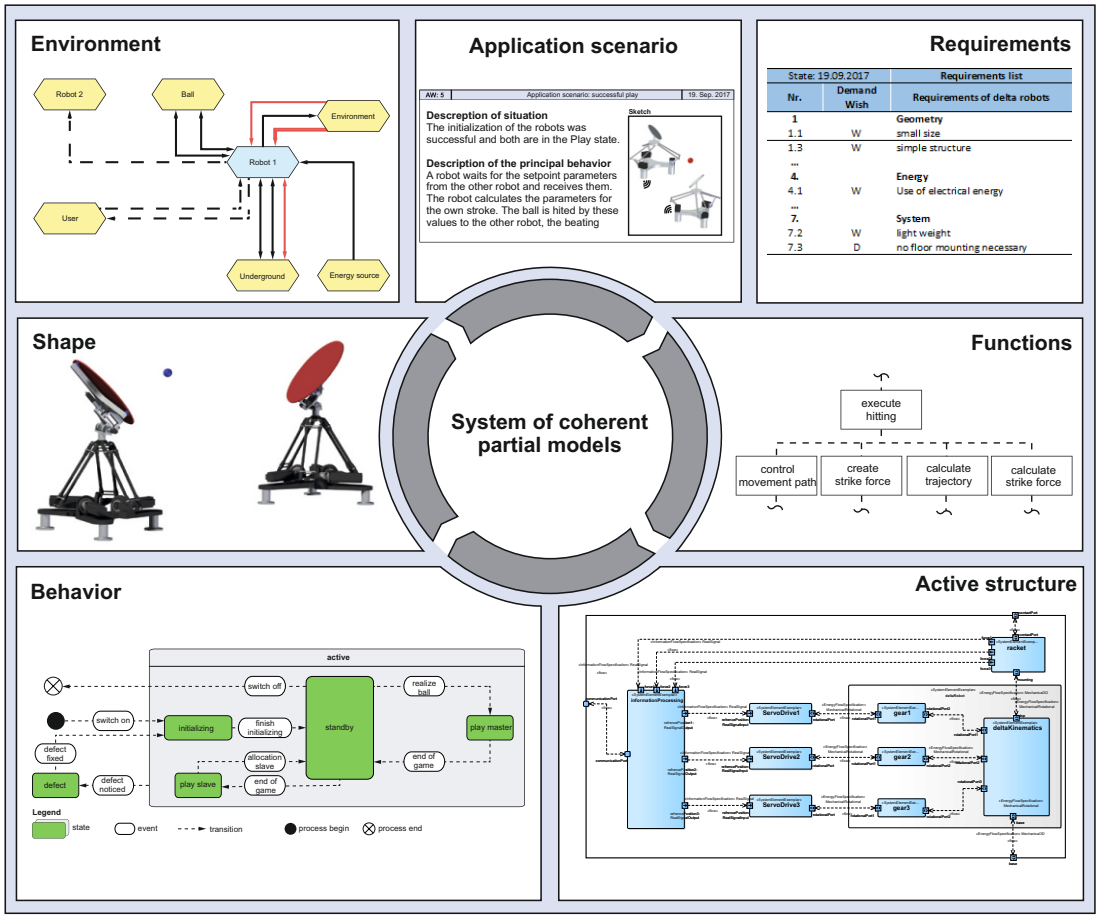


Figure 3-3: Extended Specification Of Solution Pattern (partially from [GTS14b])

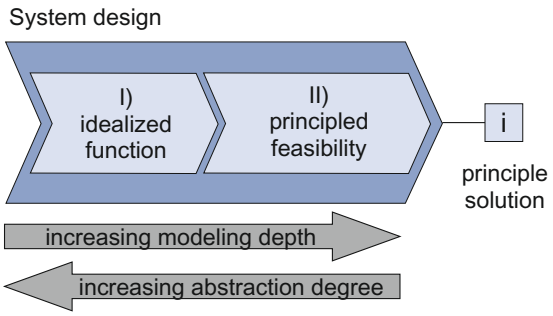


Figure 3-4: levels of modeling depth during discipline spanning conceptual design[LSB+12, LOT14]

System dynamic is designed according to the method of mechatronic composition that is a holistic view of system dynamics based on the mathematical model. In the first step to interpret the behavior of the controller in mechanical system the required kinematic and dynamic functions are fulfilled. In the next step an appropriate control strategies are designed for the synthesis of a first information processing. Then for the holistic interpretation of the principle solution are performed[Oes17].

All this information will lead to the already described principle solution, which is output of System design-phase, as shown in Figure 3-2.

Behavioral modeling:
This partial model includes different types of behavior. The system states with the associated process flow processes and the state transitions can be modeled with representation of several systems interplay [GFD+08].

Discipline-Specific Design

The parallel elaboration of the system takes place in the individual disciplines after the milestone principle solution. The cohesive CONSENS partial models (see Figure 3-3) is kept up-to-date. The different disciplines have specific methods and procedures, that they are divided into the subordinate steps of the determination of objective, synthesis and analysis [GTS14b]. The procedure is shown in Figure 3-5, and detailed below.

(i) **Determination of objective:**

Discipline-specific requirements for elaboration are collected and supplemented. These merely concern the competency of the descriptive discipline. For example, concrete objectives and requirements for the sensing/observation of a pneumatic cylinder position are defined. In the test concept activity is to evaluate on the Model-in-the-Loop (MiL), or are further tests required in SiL or HiL.According to the concrete requirements and the defined test concept, the modeling objectives for behavioral models can be prepared or detailed [Oes17].

Here, the target of determination of objective is to get requirements of discipline specific solution, test concept and model target, as shown in Figure 3-5.

(ii) **Synthesis:**

In the synthesis step, appropriate solution elements or more specific solution patterns are selected from discipline-specific solution knowledge in order to realize particular system functions [Oes17]. For example, suitable solution

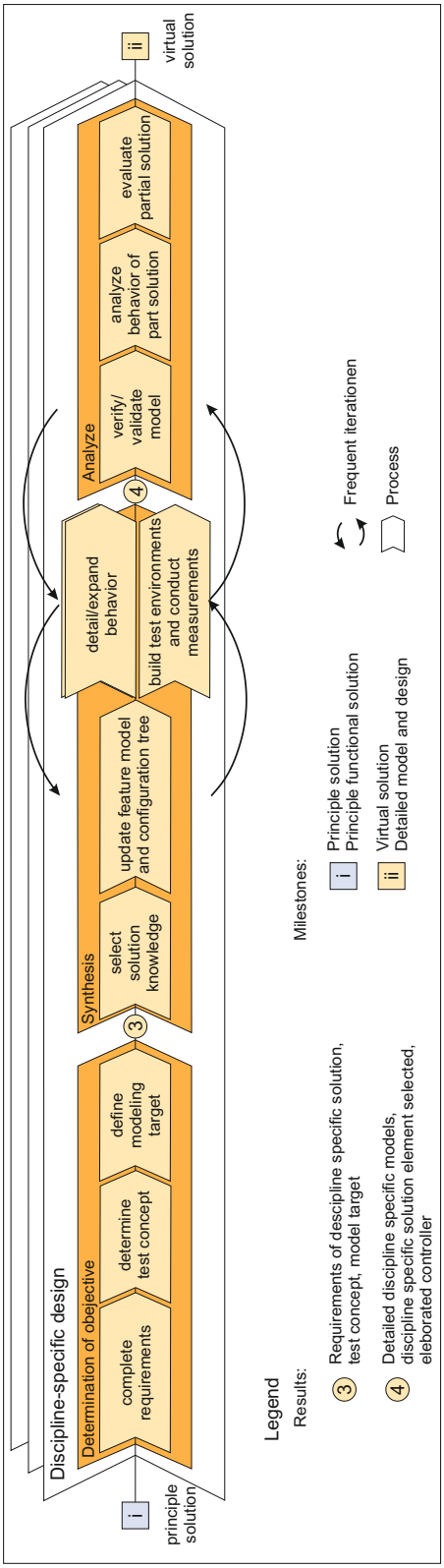


Figure 3-5: Discipline Specific Design [Oes17] extended by feature model

elements for the cylinder and valve are selected by the responsible engineering discipline.

Large number of solution knowledge can be limited by the parameters obtained in the idealized models. To support such as inconsistent classification systems and terminologies the semantic technologies [Oes17] are used to search for reusable knowledge. The relevant system elements have to be reworked, if no solution element are found [GTS14b].

The reusable solution knowledge also includes detailed solution element models. Due to uniform interfaces, available solution element models can be used instead of the solution pattern models. This reduces the effort involved in the subsequent activity behavioral model [Oes17].

Based on the selection of the solution elements, which is carried out by the synthesis of the system design phase, the initial system model (active structure) is more detailed. It causes changes in the dynamic model. Base on the updated system model, the existing feature model must be updated. The feature model receives information from system knowledge and updated system model. In feature model the information is set up, e.g. changing the type of a pneumatic cylinder in the dynamic model means selection of another branch of the feature model and subsequently causes change in system model.

Refine initial dynamic model

For models used in later design stages, interfaces must remain consistent across different levels of modeling detail. During the progressive refinement of the design, the modeling depth may change while interface compatibility is preserved.

Update feature model

To insert a new design feature base to the completed dynamic part or entire model, an instantiation of the feature object will be performed. A set of default parameter values are parsed to this instance. In the case of a dynamic model modification, the initial feature model is updated and the new features are identified from the dynamic model. The semantic of feature model consists of the complete consistency. Parameter of components are identified from the dynamic model, i.e. the features represent parameters. For example, parametric data for a pneumatic cylinder like radius, location, length and material must be supplied. Updating components are followed by updating interfaces and connectivity. These data will be checked through requirements analysis and adaptability of the new properties. Also these data are all collected in the feature model. If they cause a semantic error, the process returns for a correction of the input data or is aborted.

Update system model

According to the updated feature model following elements will be updated:

- existing components (add / update characteristics)
- interfaces of an existing component (add / update characteristics)
- existing connections

Update system knowledge

According to the updated system model following element will be updated:

- existing component (add/update characteristics)
- interface of an existing component (add/update characteristics)

The rest of the activity in the synthesis phase is just like the synthesis of system design. Grounded on the new information and updated feature model the Dynamic Behavior Model is created. By completing and modifying the Dynamic Behavior Model (DBM) after updating feature model, the system model must be updated and afterward system knowledge.

At the end of this step, the result is detailed discipline specific models, discipline specific solution element and elaborated controller, as shown in Figure 3-5.

(iii) **Analysis:**

Analysis and elaboration is done iteratively for individual synthesized components. As shown in the Figure 3-5, *validation is concerned with determining whether the conceptual simulation model (as opposed to the computer program) is an accurate representation of the system under study [LK99].*

By measurements or validation of exact reference model and identify the parameters the entire model of mechatronic principle can be verified and validated after validation of individual submodels or solution elements. For this purpose a test sample is needed for analysis and evaluation of part solution in the last step.

As shown in Figure 3-5, virtual solution, detailed model and design are achievement of discipline-specific design.

System Integration

Once the interdisciplinary model-based analysis has been completed, the system integration phase can begin. The simulation models developed during the previous design stages continue to support integration activities and validation tasks throughout this phase. Figure 3-6 [Oes17] illustrates the workflow of model-based

system integration within the overall integration process. The process starts with the Virtual Solution and progressively advances toward an integrated and validated Concrete Solution. Similar to the earlier design phases, the integration process is structured into three main steps.

(i) **Determination of objective:**

According to the virtual solution from the the interdisciplinary coordination integration strategies, concrete test and analysis concept are defined. Here is to describe which component and how should be tested by HiL[Oes17].

While in the system design MiL as well as in the discipline specific design SiL was used, the model-based system integration is tested by HiL. A distinction is made between simulations in which the information processing in the model is simulated and parts of the basic system are built up (component HiL) from the inverse case (control device HiL). In the field of automation, this is the term used for virtual commissioning, meaning the analysis of the real control or the control code by means of a model of the remaining system [SLB+13].

A part model is simulated and remaining system are built up as components of HiL. In the cost-intensive design and testing of real systems it is possible to reduce iterations through the stepwise transition from the model to the reality [Oes17]. Determination of objective step results concrete integration and analyze strategies, as shown in Figure 3-6.

(ii) **Synthesis:**

On the basis of decision between component HiL and control device HiL, the necessary components can be procured and constructed in the synthesis step. At the end of this step test environment for assurance properties (HiL-test station) are achieved, as shown in Figure 3-6. Here, only the critical system elements are detailed, whereas real-time requirements beside large models must be considered.

(iii) **Analysis:**

At the end of the model-based system integration, the integrated (partial) solution is analyzed and evaluated. Analysis begins with partial solution and ends with concrete solution. To evaluate all components completely they can be integrated in a prototype and take it into operation. As shown in Figure 3-6, concrete solution and complete integrated entire solution are the product of model-based system integration [OAL+16].

In the following subsections system knowledge, solution knowledge and solution element are described. They are stored in a knowledge base and used as library of system models and simulation models in the design process.

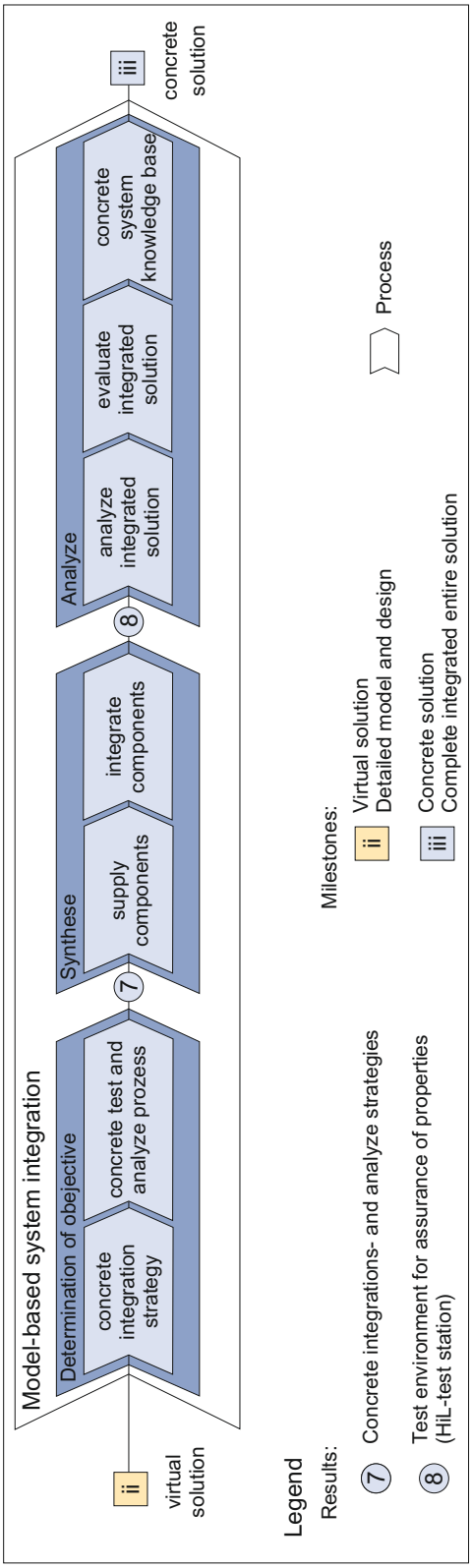


Figure 3-6: System Integration [Oes17]

3.1.2 System Knowledge

System knowledge can be semantically described, made available and used in the development process [Oes17]. To integrate correspondingly available knowledge of the solution and to use the resulting system or its system elements after the development as a new semantically prepared solution knowledge is possible. The system interconnections as a central semantically defined image can be maintained in the development process as a reliable sources [Oes17].

In the interdisciplinary coordination a system knowledge is important to extends the scope and expressiveness of previous system models, e.g. SysML. The consistency is ensured by using the information of the system knowledge model through updating specific models and development artifacts to process partial aspects of the systems in specific tools [Oes17]. Characteristic features of the system, and system-relevant features of the system elements are all collected in system knowledge model. Also to link models with elements and model parameters with system properties the system knowledge is used [Oes17].

Which discipline-specific and interdisciplinary models exist must be known to describe the shape and the (dynamic) behavior [OAL+16]. The required information listed in Table 3-1. The information are divided in two sections: Characteristics and Interface/Parameter, which are defined in regard to the model. Model and their interface and parameter are identified with ID and Name. An ID contains a unique value, which identifies each model in a database. A name should give a good declaration of a model and rest of properties describe a model in detail. For naming of models having a uniform conventions like a high level programming languages can avoid mistakes, e.g. using a capital letter for a new word master instead of underscore to separate words [OAL+16]. The names should be in a continuous language and same or very similar names should be avoided [OAL+16].

3.1.3 Solution Knowledge

Solution knowledge is the starting point for the disciplines-specific elaboration, including responsibilities and sequences [Oes17]. It is resulted from the positive evaluated system and validated functionality of roughly defined principle solution.

3.1.4 Solution Pattern

Behavior models are used for the idealized design of system dynamics. Detailed information of development is not normally available in this early phase of development, detailed behavior of a solution element. Here, the dynamic properties of the solution elements which are associated with the solution pattern are abstracted. Thereby boundary conditions should be considered. The result models are solution

Table 3-1: Model documentation

Model	Properties	Description
Characteristics	ID	ID of model
	Name	Name of model
	Date of create	Create date of model
	Creator	Creator of model
	Updated	Modified date of model
	Modeling depth	Detail of model
	Model type	CAD, CAE, …
	Solver	Used algorithm of model
	Validated	Valid model
Interface / Parameter	ID	ID of interface / parameter
	Name	Name of interface / parameter
	Type	Input, output or parameter
	Unit	Unit of interface / parameter
	Value	Value of interface / parameter
	Default value	Default value of interface / parameter
	Maximum / Minimum	Minimum value or maximum value
	Connected interface	Interfaces, which connected to this interface.

variants that during the system analysis and assurance of properties can be created. Here, the target modeling depth is greater than 1 and smaller than 3 to be the range of the principle feasibility [Oes17].

Idealized model for cylinder:
an idealized solution pattern model for pneumatic and hydraulic cylinder is an example that sets in very different performance classes, but the solution pattern model can still be used at a more abstract level. As shown in the Figure 3-7, if the cylinder has 3 models with different modeling depth the idealized model is *CylinderMD1* that has just minimal interfaces inherited from *CylinderInterface*. The pneumatic and hydraulic cylinder can be inherited from the a solution pattern model.

3.1.5 Solution Element

Specific, ready-to-use (partial) solutions are selected in the specific design, which can be re-used in new systems without major adaptation. These solutions are included the supplier's solution and also internal solution from previous projects as solution elements. It represents concrete expressions of solution patterns, which are defined in modeling depth 3 [Loc]. A solution pattern is replaced by

a selected solution element. Thereby, it is necessary to match the interfaces by changing the detail level without change of the model structure. Solution elements have different levels of detail to consider variation of application purpose and the associated accuracy requirements. In order the minimum interfaces defined by the interface model are transmitted to corresponding solution element models to simplify an exchange in every cases [OAL+16]. As is depicted in Figure 3-7, if the cylinder has 3 models with different modeling depth the minimal interface is *CylinderInterface*. The idealized model is *CylinderMD1* that has just minimal interfaces inherited from *CylinderInterface*. In the modeling depth 2 (*CylinderMD2*) and 3 (*CylinderMD3*) there are more interfaces additional to the interface model.

Definition of interfaces in Figure 3-7 [Mat]:
Pressure A is absolute pressure in the thermal liquid chamber A.
Pressure B is absolute pressure in the thermal liquid chamber B.
Displacement is a displacement of the piston at the start of simulation.
Thermal conserving port is associated with the gas in the pneumatic chamber.
Ambient temperature is a single-precision scalar value.

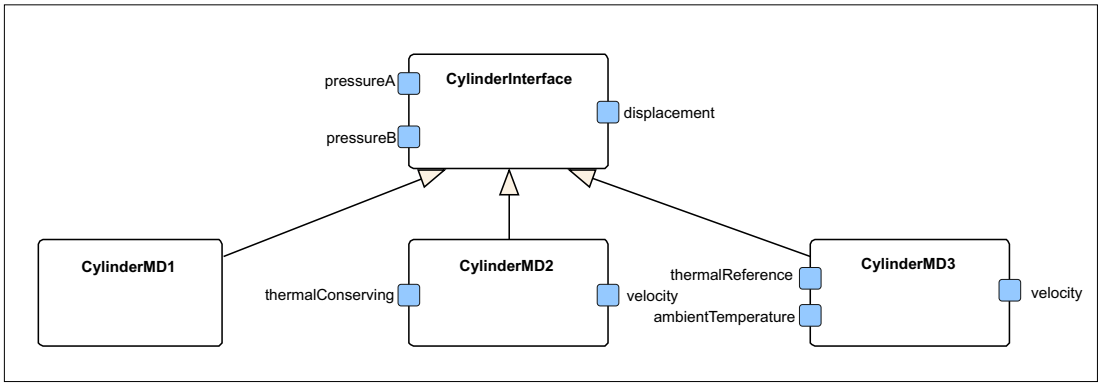


Figure 3-7: Interface model to ensure the interchangeability of various detailed models of a cylinder

To make the solution elements reusable, they require effort and detailed knowledge. As an example, frequently used CAD models can be used as solution elements to provide users knowledge about product [Kuf11].

For the purpose of tracking and property protection, it is necessary that the each element has an id, as shown in the Table 3-2, i.e. each entry represents an element of system model or dynamic behavior model, which is linked in a suitable form through IDs.

Table 3-2: The structural of id

SE (solution element)	name	date + time
SP (solution pattern)		
SYSEL (system element)		
MODEL (dynamic behavior model)		
SEM (dynamic model of solution element)		
SPM (dynamic model of solution pattern)		

3.2 Interface Design

To become explicit about the interface definition, in Dynamic Behavior Model it must be mentioned that an interface refers to any logical or physical relationship required to integrate the boundaries between systems and their environment [RT12]. According to that an interface defines the system boundary that is presented by a system for interaction with other systems [FD13].

In this section, the mapping of interfaces and parameters during the generic procedure of the mechatronic systems design, is tackled. Inadequate definition of interfaces results in many problems along the design process. These problems appear especially during the integration of subsystems. Accordingly, to avoid the aspects of the following subset are addressed:

- clarification of interface classes,
- considering the consistency of interface information in different disciplines from system model to DBM,
- information exchange among the design engineers.

In the following subsections according to SysML profile, a flow specification diagram is presented to define a concept of the mapping interfaces based on feature modeling. To formalize a system knowledge in a semantic system model and to enable efficient integration of approved solution knowledge, here is explained how to prepare DBM for reusing.

3.2.1 Interface Definition in SysML

To improve the effort of maintaining consistency between all the artifacts, here, a concept of the mapping interfaces based on feature modeling is created. Every interface is defined by a set of attributes independent of their type. All properties of an interface are derived from a system or a dynamic model, and are collected in a corresponding feature-element of a feature model. In this way, connectivities of these interfaces are depicted and managed in the feature model. The interfaces

are matched together through this concept among the subsystems connectivity. This methodology ensures the consistency of interface definitions among different disciplines.

On the basis of the SysML profile [MMC+10], interface relationships can be presented in a flow specification diagram. An excerpt of it is shown in Figure 3-8. Flow specification defines a set of flow properties that correspond to individual pieces of a common interaction point [MMC+10].

Flow specification diagrams are made up of two basic elements, interfaces and relationships. It contains also different kinds of interface properties, together with flow items. Flow specifications describe the types of interfaces that exist in a system, whereas relationships describe, what the relationships are between various interfaces. For example, generalization describes an element as a specialized descendant of another element, containing additional properties and behavior [MMC+10]. Each interface is typed by a *Flow Specification* and can be nested with zero or more to other interface. Interfaces that have flow properties contain a small arrow showing the direction of the flow (whether into the interface, out of the interface, or both). Also the interfaces are classified in three flows, energy, material and information [Kai14]. A flow port can be specialized further through three subtypes:

- description flow port typed by a flow specification
- flow specification contains flow properties
- flow properties specify types of items that can flow in and out of block

Here, three major types of interfaces are defined: inputs, outputs and ports. Inputs and outputs have just one flow property, alongside ports could have several properties. Energy- and material-flow is from port-type and information-flow presents input- or output-type. Each flow property is mapped to an interface, which is a part of user defined library in DBM. Some classes generalize another class. For example, mechanical-class generalized rotational- and translational-class. Also mechanical rotational is a subclass of mechanical rotational torque and mechanical rotational velocity, i.e. a mechanical rotational-class additional to angle- and acceleration-flow property has torque- and velocity-flow property. Like this, mechanical 3D-class has all flow properties of rotational- and translational-class. The Table 3-3 summarizes of used elements in Figure 3-8.

3.2.2 Modeling Interfaces in Dynamic Models

To fulfill the requirements, system knowledge is formalized in a semantic system model. Furthermore, it enables efficient integration of approved solution knowledge, e.g. DBM are prepared for reuse.

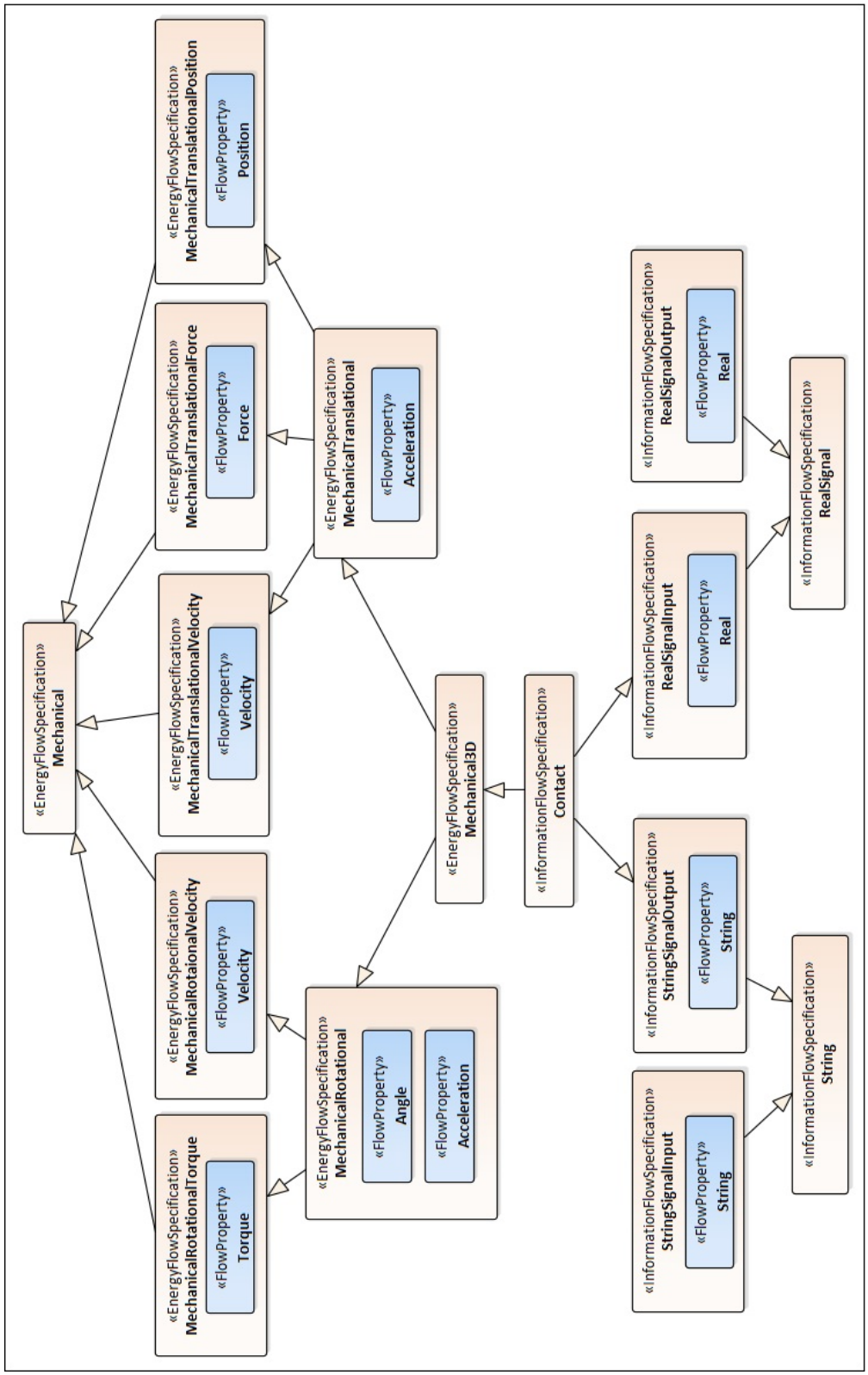


Figure 3-8: Flow specification diagram in SysML

Table 3-3: Flow specification definition

	Input	Output	Port
Flow specification	one directional interface		bi-directional interface
Flow property	data type		physical port type

Here, ports are divided to different data types to connect one component to another for coupling of multi-discipline simulation tools. For coupling several simulation tools is needed to know of which type of data is a port composed.

To achieve seamless integration of the presented approach in everyday work, focus is on coupling it with widely used tools like Enterprise Architect [Ent], MATLAB/Simulink [Mat] or Dymola [Dym].

The tool Matlab/Simulink/Simscape supports limited data types, which do not cover all the data types shown in Figure 3-8 from one hand. On the other hand, the ports do not assign any type of physical connections.

For mapping interfaces of dynamic models to each flow specification of system model, a customized interface library in Matlab/Simulink is created. An excerpt of customized interfaces is represented in Figure 3-9. Each customized interface according to Figure 3-9 is depicted each flow specification of class diagram shown in Figure 3-8. According to it a mask is created for every interface in Matlab/Simulink to include the flow properties as parameters, e.g. Figure 3-10.

Here, customized interfaces are defined to simplify the discipline specific elaboration. It is used in the development process when solution patterns or solution elements must be connected to make a part of a system available for simulation. For this purpose a customized interface simplify the communication between different disciplines. They can be used directly for the development as a prepared adapter between different participated disciplines. Interfaces are defined to delimit a system from its environment or a subsystem from another one. To use them in the system design, it is useful to refer to existing interface definitions or define new customized interfaces. For example, by concretization during the design process submodel detail is changed too, in this case the modeling depth can be varied without changing compatibility of the interfaces. One purpose of customizing them is to support the compatibility of submodel to their environment due to different variants. Overall, customized interface could be used for:

- exchange of solution pattern models against solution element models
- exchange of different solution patterns
- exchange the submodels as smoothly as possible

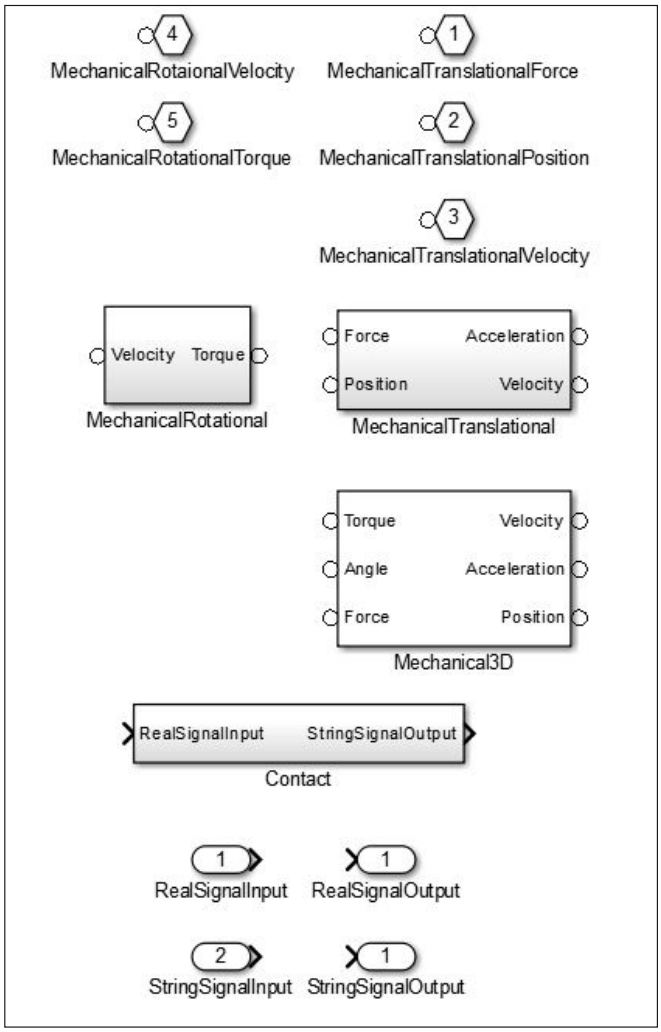


Figure 3-9: Excerpt of customized interfaces in Matlab/Simulink

- ensure the consistency of different models' interfaces
- using advantages of object-oriented modeling languages such as Modelica in Matlab

According to the customized interfaces in Matlab, their inheritance of interfaces to the different model can be directly used. In the Figure 3-7, the interface model is inherited from three differently detailed models of a cylinder. By increasing the modeling depth, interfaces can be added to the current one, For example, an additional heat port or a further input. By keeping current interfaces the interface consistency of different models is ensured.

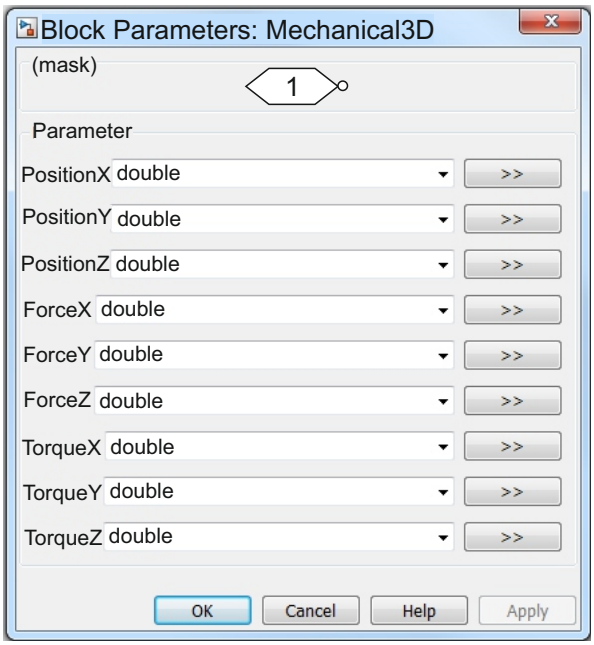


Figure 3-10: Mask of a mechanical 3D interface in Matlab/Simulink

3.3 Summary

In this chapter, designing the complexity of mechatronics systems from the early phase is explained, how to attach the existing part of various intellectual domains together to make an integrated system. For this purpose, the design concept of V-model is extended by details to show how to automate some design steps. This concept is extended by modeling library/database definitions as followed:

- System knowledge: semantically described the system models.
- Solution knowledge: sum of solution pattern and solution element
- Solution pattern: idealized design of system dynamic model.
- Solution element: design solution, which are used in specific projects.

Also, here is shown, how to adapt feature modeling along design process for ensuring reuse of information and consistency between different disciplines. Additionally, a concept of the mapping interfaces based on feature modeling is presented to improve the effort of maintaining consistency between all artifacts and to avoid inadequate definition of interfaces.

4 Variability Management and Feature Modeling

This chapter explains the investigations and researches performed on management of variability during the design of mechatronic systems.

The change of focus from the individual system and project to the product line is the main difference between traditional single system development and software product line engineering. This is mainly a shift in strategy. To take this advantage there are some principles for a successful design of mechatronic systems according to software product line engineering, as described below [LSR07]:

- Variability management: each individual system is considered as variations of a common theme. This variability is straightforward and it needs systematic management.
- Architecture-centric: taking advantage of similarities among the individual systems is the technical side of the design which needs consideration.
- Two-life-cycle approach: the individual systems are developed on the basis of a software platform. These products, together with the platform, have to be engineered and have their own life-cycles.

The way that goods are produced has changed significantly in the course of time. Formerly goods were handcrafted for individual customers. By and by, the number of people who could afford to buy various kinds of products increased. In the domain of automobiles this led to Ford's invention of the production line, which enabled production for a mass market much more cheaply than individual product creation on a handcrafted basis. However, the production line reduced the possibilities for diversification [PBL05].

The strategy followed here is, first, to focus on what is common to all components, and next, to focus on what is different. In the first step, artifacts are provided that can be reused for all products. These artifacts are built from components or derived from another platform or earlier systems [PBL05].

To facilitate product customization, the artifacts used in different products have to be sufficiently adaptable to fit into the different systems produced in the product line. This means that throughout the development process, identification and description of where the products of the product line may differ in terms of the features they provide, the requirements they fulfill, or even in terms of the underlying architecture etc has to take place. Therefore, flexibility has to be provided in all those artifacts to support products customization [PBL05].

Different products of the same product line may for instance have different components. For example the cars are designed in a way that allows a common

approach to support the different motors with their different sizes, etc. Such flexibility comes with a set of constraints. [PBL05].

4.1 Adapting of Variability Management

A Software Product Lines is a family of software products that share a common set of features and differs in others [ASW+11]. Modeling variability and commonality are the key elements in developing product families and product lines. *The objective of the analysis of commonality and variability is to identify strategic reuse [CI01]. These techniques provide an easy, understandable, and generic way of representing the variability information, independent of a specific application domain [The14]; as cited in [OAL+16].* Feature, representative of a SPL, are end-user-visible behavior of a software product that is of interest for some stakeholder [ASW+11]. *A feature model represents the information of all possible products of a software product line in terms of features and relationships among them [SAM12]; as cited in [OAL+16].* A feature interaction is a situation in which the composition of multiple features leads to emergent behavior that does not occur when one of them is absent [ASW+11].

This flexibility is a precondition for product customization; it also means that it could help to predefine what possible realizations can be developed. In addition, a definition of exactly the places where the products can differ, so that they can have as much in common as possible, exists. The flexibility described here is called *variability* in the software product line context. This variability is the basis for product customization. These products that share the same platform and exhibit similar features were called a product line. e.g. a manageable set of cars belonged to one product line that was based on a single platform.

For managing variability in a product line, three main types must be distinguished [LSR07]:

- Commonality: a characteristic (functional or non-functional) can be common to all products in the product line. It is called a commonality. Later it would be implemented as part of the platform.
- Variability: Some products may have a common characteristic, but not all of them. It must be clearly modeled as a possible variability and implemented in such a way that only the selected products can have it.
- Product-specific: a characteristic might be found only in one product - at least for the predictable future. These specifications are often not required by the market as such, but they are requested by individual customers. Although these variabilities will not be merged with the platform, the platform must have the potential to support them.

Although handling commodities and variabilities is carried out mostly in domain engineering, product-specific parts are managed exclusively in application engineering, as shown in Figure 4-1.

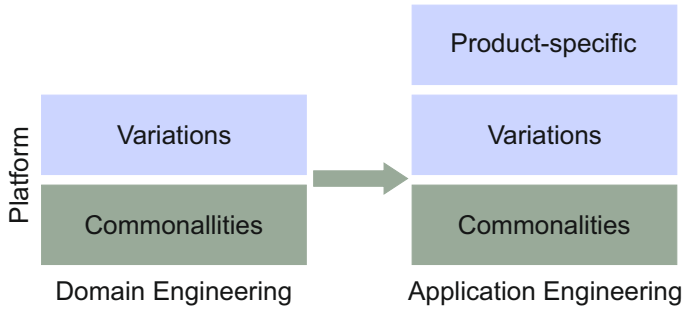


Figure 4-1: The relation of different types of variability [LSR07]

Many different approaches have been developed to represent variability. Here, the modern approaches such as using features as basic concepts for variability representation are used. The key factor is that the core of representation is the products that differentiate them from other products. Describing those variabilities must be done in order to derive a specific product line instance. The variability model will have an impact on traditional views on requirements, design, etc. [LSR07]. An example of this relationship is given for the case of a UML class model in Figure 4-2. In the class diagram a parent-child relationship in a delta robot example is shown. The composition link shows that Delat Robot owns Servo Drive and Racket and in the next level Servo Drive has Servo Inverter and Servo Motor. In the right side of the Figure 4-2 feature model according to class diagram is depicted. Each feature points to just one component. Therefore if there are two servo drives, for each one is provided a feature. Each servo drive has one servo motor and one servo inverter. From the design aspect different combination of components can be seen as all products of the Software Product Line. According to desired behavior analysis of the system or subsystem all components can be optional selected.

In next section the basic structure of feature model is represented to show how to describe the variabilities in mechatronic systems. In other words, here is to see which impact the variability modeling has on traditional view of mechatronic systems.

4.2 Adapting of Feature Model

In this work a grammatical feature model is developed to define the relationships between interfaces of modeling tools and modeling depth. In order to illustrate the

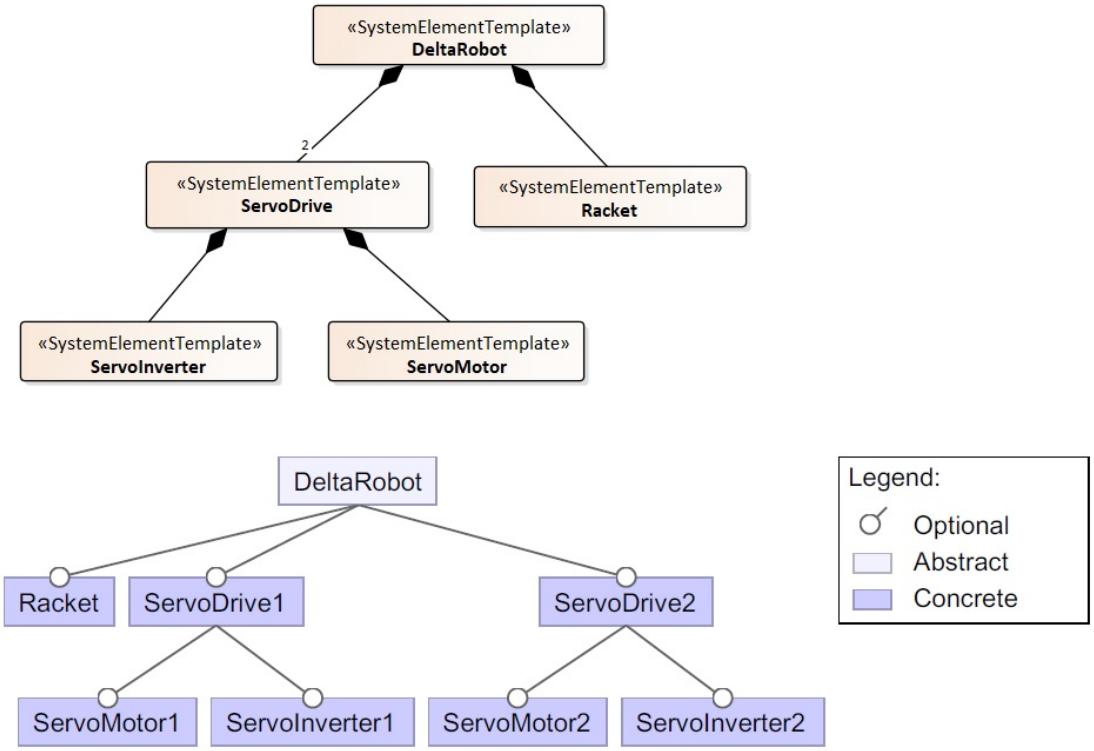


Figure 4-2: Relation between class diagram and variability model

issue, the semantics of feature modeling for presenting the dependencies between submodels of a mechatronic product is extended. The simulation models that describe a product should be connected to each other in order to address the interactions that exist among various subsystems of the product.

A feature model represents a hierarchically arranged set of system elements so-called components of a mechatronic system. Relationships between a parent feature and its child features (or subfeatures) are categorized as:

- Mechatronic system
- Component
- Modeling tool
- Modeling depth
- Interface
- Input
- Output
- Port

- Parameter

The first level of category is a mechatronic system, which is modeled. The last level of category is parameters, inputs, outputs and ports.

First, the model interfaces of each component are considered. It is necessary to introduce every component as a feature to demonstrate the minimal interfaces of the components within a feature model. Each component has a minimal modeling depth, which is represented in an abstract feature [LSB+12]. In addition, every component has parameters and variables such as inputs and outputs. Each interface must include at least one input and one output.

In the Figure 4-3 are all these categories depicted in a feature model. They represent different model aspect components, which exist in a mechatronic system. From the Figure 4-3 following statements are to understand:

- A system (CurrentSystem) as the root can have more than one component that each component is choosable.
- Component has interfaces(Interface).
- Every interface is a type of output, input or port, which are mandatory to choose. This interface is independent.
- Component is modeled in different tools that one of them can be selected from ComponentTools.
- Every component is modeled in four modeling depth(MatlabModelingDepth1), which are depends on each other
- Every modeling depth beside to the next modeling depth has interface (MatlabMD1Interface) and parameter (MatlabMD1Parameter)
- Ever interface is a type of input, output or port, which are choosable. This interface depends on the tool.
- Parameters (parameter) are optional to select.

Following points describe developed feature modeling language for design of mechatronic systems, which are not possible to be shown in a figure.

- A feature model can consist of several components. To model a combination of components, the desired components to make a part model or model the entire system can be selected.
- A component has common interfaces, which in both system and simulation model appear. These interfaces must be selected in every model of the component and independent of the selected model.
- An Interface is from type input, output and port.

- A component could be modeled in various modeling tools.
- A component could be modeled just once in the simulation model of entire system.
- A component could be modeled more than once in one simulation tool with diversity of details, so-called modeling depth. They are modeled in feature model hierarchically from top to down, i.e. from very simple model to more complicated model. Here, variability dependency is optional. This grammar gives the possibility if one level is selected the additional interfaces are selected too. For example, if modeling depth 3 is selected, also the interfaces of modeling depth 2 could be selected too, since quite often some interfaces or parameters are common in different modeling depth.
- A model of a component has two abstract characters, modeling tool and modeling depth. Parameters and interfaces differentiate this models from other models of this component. In a model configuration developer can decide which interfaces and which parameters must be used in the model.

In the Table 4-1 a catalog of commonly used derivations of feature models to mechatronic systems is presented. The different categories of variabilities in DBM of mechatronic system is separated and each of these categories is treated in different way. Further, each category is depicted in the context of feature modeling.

In next subsections each category of variabilities in DBM from the Table 4-1 are described in details.

4.2.1 Interaction between Submodels

The proposed approach shows the interactions between different modeling tools at varying modeling depth for different components of a product. The feature model helps in the system integration step to form an overall system, and considers the interactions between the components and their interfaces and parameters to be investigated. The simulation models that describe a product should be connected to each other in order to address the interactions that exist among various subsystems of the product. The problem of modeling mechatronic products becomes more challenging as the number of components and the functional layers increases.

Parameters

Most disciplines are confronted with a design in which major parameters are already fixed. Parameters of various physical components appear in various

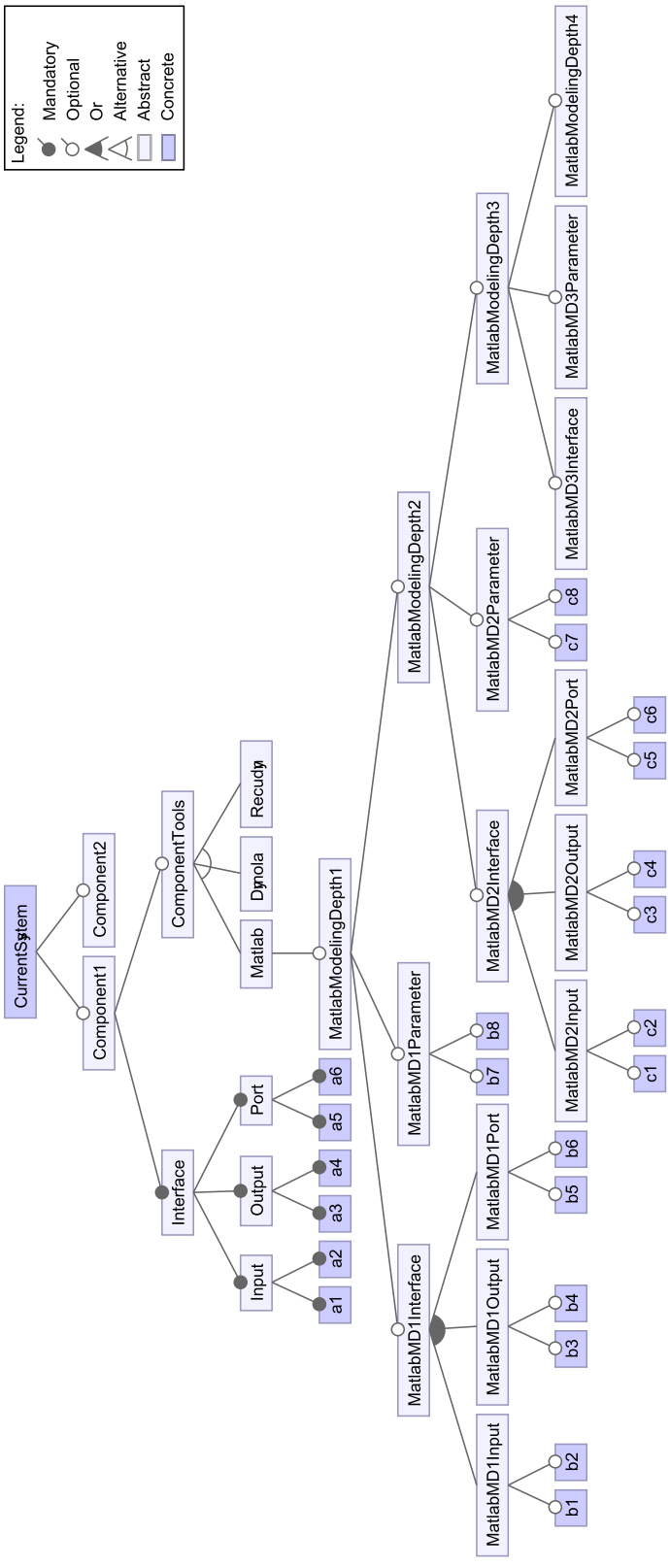
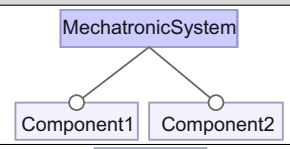
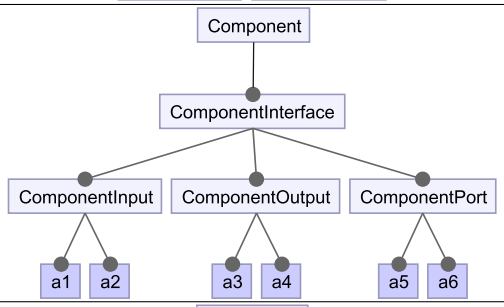
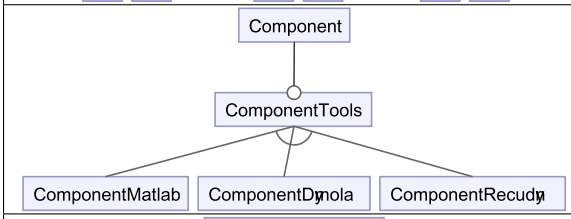
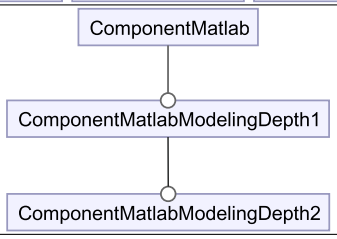
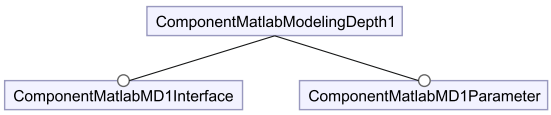
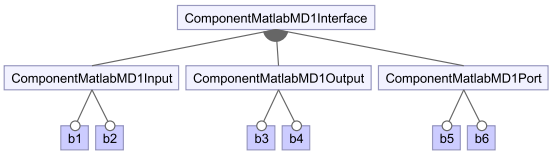
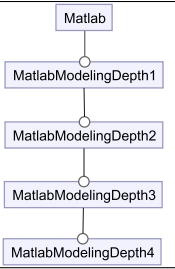


Figure 4-3: Template of feature model for components of a mechatronic system

Table 4-1: The basic structural of feature model for mechatronic systems and their explanation

Feature Construction	Explanation
	Solitary feature with cardinality [0..1], i.e., optional feature. A mechatronic system composed of some system elements (component).
	Solitary feature with cardinality [1..1], i.e., mandatory feature. A component model has some interfaces in system model. An interface divided into input, output and port. a1 and a2 are inputs, a3 and a4 are outputs, a5 and a6 are ports.
	Feature group with cardinality (1 – 1), i.e., xor-group. A component could be modeled in more than one modeling tools.
	Solitary feature with cardinality [0..1], i.e., optional feature. A component model in a specific tool could be modeled in more than one modeling depth.
	Solitary feature with cardinality [0..1], i.e., optional feature. A component model in specific modeling tool and specific modeling depth has interfaces and parameters. Under them several interfaces and parameters are categorized.
	Feature group with cardinality (1 – k), where k is the group size, i.e., or-group. An interface divided into input, output and port. b1 and b2 are inputs, b3 and b4 are outputs, b5 and b6 are ports.
	Solitary feature with cardinality [0..1], i.e., optional feature. A component model with the specific modeling tool and specific modeling depth has another modeling depth with more detail.

combinations and at various locations. To investigate the effects of parameter changes, they are represented in feature model. Every model of each component with a modeling depth has parameters.

Interfaces

Every component has interfaces such as inputs, outputs and ports. Each interface must include at least one input, one output or one port, which is defined as both input and output, because every component has a connection to another component, as depicted in the Figure 4-3. As an example a pneumatic cylinder has pneumatic pressure as an input and piston displacement as an output.

General interfaces

The general interfaces and properties of a component are considered here as a common features, which are valid and useful in all modeling tools, in order to avoid the repetition of this interface or property in every modeling tool. The advantage of having general interface is if the feature once is changed, it is valid in all existing model of the component and there is no need to change them separately in every model.

4.2.2 Variety of Modeling Tools

In this work, only the subclass of models that are codeable as computer programs, so-called mathematical models are considered. A model is always related to the tuple system and experiment. A smaller portion of a system is cut out, and thereby generate a new model which is valid for a subset of the experiments for which the original model was valid [Cel91].

Utilized tools in this work are based on languages that are compromised in terms of model expressiveness, simulation capability, and domain applicability and standardization. The basis of their paradigm and decorum such as differential algebraic equations, state machines, object-orientation, etc., are renowned and accepted by practitioners [Sch13].

Knowing the individual components that make up the product is necessary for simulating the behavior of a product. A product may be comprised of mechanical, electrical, or other components [Nik07]. If the mechanical components are allowed to move relative to one another, the product is called multibody. Various components or bodies can be interconnected through kinematic joints, springs, dampers, simple contact, or other elements [Nik07]. A simulation tool can be used for analyzing the behavior (e.g. the motion) of a multibody system. The simulation of mechanical components and their kinematic properties can be carried out via

Multibody Simulation (MBS), a software tool for CAE. A unique combination of Multibody Dynamics, Finite Element Analysis and Controls is offered by a modern CAE software suite called RecurDyn [Rec].

To generate the description of the block diagram, Simulink together with Matlab is used for specifying system by connecting boxes on the screen rather than writing a series of commands [Bol08]. But Modelica is a free, object-oriented language used for modeling large, complex, and heterogeneous physical systems. It is useful for multi-domain modeling, like mechatronic models [Mod12]. The Dymola environment uses the open Modelica modeling language, enabling users to freely create their own model libraries or modify the readymade model libraries to better match userś unique modeling and simulation needs [Dym].

To model the variety of tools, it can be assumed that a component is modeled in a powerful tool like Dymola or Matlab/Simulink or in Computer Aided Engineering tool like RecurDyn or Catia. Tools themselves are abstract features, because it is not mapped to implementation artifacts. However, the inputs, outputs and the parameters will be implemented and they are concrete features. If a feature change in a dynamic model results in the corresponding changes in other features, all of the changes are considered by updating the feature and consequently updating the relevant dynamic model in model configurator. A mechatronics system can have one or more components. Each component can also have one or more models in different simulation tools, each one with their modeling depth.

4.2.3 Variety of Modeling Depths

Checking the effort against the benefit of the model is necessary since the use of dynamic models sometimes implies a considerable effect. Create the models as detailed as needed but as abstract as possible is the aim of modeling. In the thesis of Matthias Lochbichler a methodology is developed for selecting the modeling depth of dynamic behavior models to support the developer to overcome this issue. Choosing the best modeling depth with respect to the requirements of the system is what this methodology offers. The following four predefined levels of modeling depth are proposed by this methodology, respectively called idealized function, basic feasibility, verification of the system-specific behavior, and optimization of components [LSB$^+$12]. Matthias Lochbichler et. al. [LSB$^+$12] defined the modeling levels as below:

Level I: Idealized Function
The function from a logical point of view is represented by models of level I, which includes important states containing time-discrete state machines. At this level there is no physically modeling. Only logical links are used, for example, a pneumatic cylinder is modeled as a switch. The cylinder model is extended by a

value and by zero value is retracted. The dynamic behavior of the system is not considered at this level of modeling depth and the models only consist of limits. Moreover, it is possible to examine the states of the system such as cycle times of a cylinder at discrete times.

Level II: Basic Feasibility
Time behavior completes the models of the basic feasibility and can be created on a physical context. At this level, modeling intermediate positions is simple. The system behavior is only illustrated at discrete times, for instance, the end positions of a cylinder. A highly idealized form is specified for the models and insignificant details such as friction or losses are disregarded. Soultion patterns typically includes simulation models of this level. It implies that there are no details about the real data of the components which are used for the developed system in solution patterns. In case of the pneumatic cylinder, it means that the piston moves without friction and there are no leakage currents. The cylinder is free to move in every position between given boundaries. The first analysis of the basic feasibility and observance of constraints can be carried out by such a model.

Level III: Verification of the System-Specific Behavior
Physical effects form the models of this level and side-effects are treated simply in a map or in a linear context. Solution elements replace the solution patterns of level II. With regard to the pneumatic cylinder, the friction between the piston and the cylinder wall is modeled. Moreover, at this level of modeling depth the leakage currents can be modeled. If necessary, the model parameters are specifically changed accordingly. For analyzing and verifying the system behavior or for controller design the detailed models are utilized.

Level IV: Optimization of Components
The optimization of components also uses physical-effect-based models. The largest modeling depth belong to these models and side-effects are modeled as comprehensively as possible. To optimize sensors and actuators, the models of this level are used. As a matter of fact, reducing costs and resources are the main objectives of optimization.

Feature model for modeling depth:
According to above definition of modeling levels, now they can be categorized for a component in a feature model. In the first three modeling depths, there are three kinds of subfeatures. One of them is the next model depth, and the other is property. The OR-binary shows that at least one of the subfeature must be selected. This means that the next depth either has more inputs, outputs or parameters than the previous depth, because it is more detailed than previous one, otherwise excludes the properties of the previous modeling depth. Modeling depths are abstract and just inputs, outputs and parameters will be implemented as concrete features in the feature model. Each modeling depth has four kinds

of mandatory subfeatures: inputs, outputs, ports and parameters to model the behavior of a components model.

In the product line, modeling depth is organized hierarchical from top with simple model to down with optimization model, see Figure 4-4. It outlines following issues:

- Feature *ModelingDepth1*: Selecting of Modeling depth is optional. The root feature is a simulation tool.
- ModelingDepth x+1: The root feature is the modeling depth x.

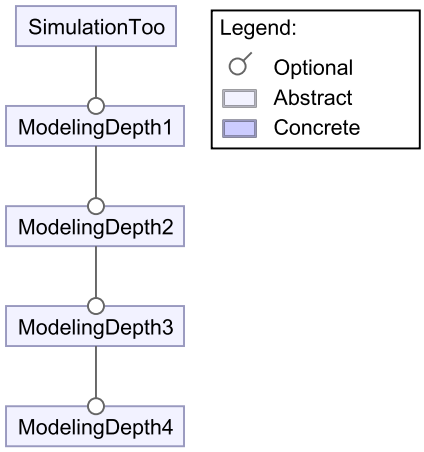


Figure 4-4: Feature model of modeling depths

In this section there are two used terms in this work, which must be separated from each other.

Modeling complexity: The complexity of a model is a structural feature. A high complexity means that the model consists of a large number of different components, which have many and also variable connections among each other. In contrast, models with low complexity have only a few elements, which are also hardly connected to one another.

Modeling depth: The modeling depth of a dynamic model describes how detailed or precise physical effects and their interactions are depicted. A high modeling depth corresponds with high detailing, so that the model can accurately represent the reality. Models with low modeling depth, on the other hand, contain many simplifications and assumptions.

Next section describes the use of constraints in feature model to define the dependency between elements of mechatronic systems.

4.2.4 Constraints of Feature Modeling

Constraints, called restrictions, can be attached to each component. Selecting a certain component may require or exclude the selection of another component. These relationships could be represented in restrictions. The modeling of the selected components is controlled via the feature model and the component family model constraints. In addition to the logical dependency of the feature elements, constraints between features can be used to show the dependencies which are not hidden in the logic connections of the diagram. There is a constraint to define the dependency of parameters on each other. It is proposed to keep a dependency between these two features. As an example, friction of the piston depends on the mass of the piston, i.e. if feature *friction* is selected, then feature *mass* must also be selected.

4.2.5 Interaction across Submodels in Different Domains

There is a numerous dependencies between submodels across the mechatronic design over interfaces. Their effects can be depicted in feature model. Feature model enable modeler to select the submodels, which are depending to other submodels. These interaction patterns help automating the process of putting submodels together to design the entire mechatronic system. Beside this automation of some aspects of feature models need the manual intervention to show the entire interactions. In feature model all possible interactions between involved model of subsystems are described.

4.3 Summary

In this chapter the management of variability in design of mechatronics system is investigated to provide the grammar for structuring a variability model. Each individual subsystem design is considered as variation with the consideration of technical side of the design. This subsystems have to be designed individually and have their own life-cycles. The strategy is, first, to focus on common design character of a component and then what is different. To facilitate system design customization, the artifacts used in different system components have to be sufficiently adaptable to fit into the different systems design in the product line. Several product of the same product line has for instance different designs. For example, a cylinder is designed in a way that allows a common approach to support the different piston designs with their different sizes.

For this purpose, functional or non-functional characteristics as commonality and variable characteristic as possible variability in system design are defined.

Thereby the characteristic of a product specific design as individual are supported. Here, features as basic concepts for variability representation are used. Feature, end-user-visible behavior of a software product, represent system components and their characteristics. A feature model shows the information of all possible system design in terms of features and relationships among them.

According to this definition, to represent variabilities in mechatronic systems interfaces of different design aspects as modeling tools and modeling depth are implemented as features. Modeling tools includes Matlab, Simulink and Simscape for specifying system by connecting boxes on the screen. Also, Modelica is used as object-oriented language for physical systems. Additionally, RecurDyn or Catia are considered as Computer Aided Engineering tools. Modeling depth describes detail of precise physical effects and their interactions.A grammatical feature model is developed to define their relationships.

The hierarchy of feature model for a component of a mechatronic system is explained explicitly in the subchapter 4.2. All these hierarchies are depicted in Figure 4-3. The Table 4-1 shows how the basic structure of mechatronic system is impacted by feature models according to variability of modeling tools and modeling depth(details). Also, a grammatical feature model is developed to define the relationships between submodels of a mechatronic product.

5 Implementation of a Multifunctional Framework

The previous chapter proposed a detailed description of a design method of mechatronic systems. The design model described the phases and the entire tasks in each phase to develop a modular and reconfigurable mechatronic system. The proposed procedure model showed that there is a need for a software tool to support the development process. This chapter introduces an implementation prototype of the Multifunctional Model Client (MMC).

This implementation follows a classical waterfall approach beginning with a development phase, and a closing testing cycle. The waterfall approach is the traditional approach used in smaller and larger projects (Figure 5-1). There are dependencies and corrective loops circulating back to earlier phases in order to adjust distinct phases [SH09].

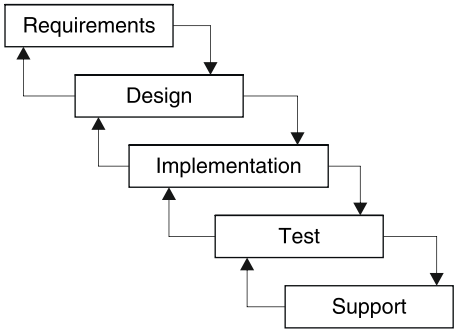


Figure 5-1: Traditional waterfall approach

5.1 Multifunctional Model Client

In order to design intelligent technical systems in multidisciplinary system development by the method presented in the previous chapter a MMC is implemented. It facilitates the designers work of handling and managing different models, solution and system knowledge. The MMC offers a guidance for development of mechatronic systems step by step according to the V-model [OAL+16], as the presented methodology in chapter 3. MMC has three main functions [OAL+16]:

- *configure dynamic behavior models (e.g. combine components into DBM of the system using feature model)*
- *ensure consistency (e.g. manage changes of parameters and interactions)*
- *access reusable solution knowledge (e.g. enable reuse of DBM/make DBM available for reuse)*

In the next section, functions, tasks and general requirements which will be supported by MMC are presented.

5.2 Requirements

Both functional and non-functional requirements need to be covered and implemented by the MMC. Definitions of the functional and the non-functional requirements of the configuration tool are offered in the following:

5.2.1 Functional requirements

The functional requirements describe the functions and tasks which should be supported by the MMC. The MMC should:

1. interact with the system knowledge, which organizes and stores components as system models.
2. interact with the solution knowledge, which organize and stores components in different simulation tools with different levels of detail.
3. interact with variability and feature models.
4. interact with automation systems.
5. connect the library of DBM to system model.
6. allow the developer to select a combination of elements, which are stored in the database, in order to generate a model of the entire mechatronic system or part of the system.
7. ensure that the selected combinations of components are consistent and compatible to each other.
8. allow co-simulation between subsystems in the case of consistency and compatibility of components.
9. show an error which contains the inconsistent or incompatible component or interface, if the selected combination of elements is inconsistent or incompatible.
10. have the ability to load, view and modify any previously generated models by parsing a previously generated configuration file.

5.2.2 Non-functional requirements

The non-functional requirements explain some general requirements regarding the software stability and usability. The most significant non-functional requirements are listed below:

1. To ensure the usability of the software, the MMC should include a self-explaining Graphical User Interface (GUI).
2. The MMC should assist the user in each window stepwise.
3. The MMC should be modular and extendable.
4. The MMC should be platform-independent, signifying the necessity of its compatibility in an executable package independent of the operating system.
5. The MMC should simplify some processes for the user such as database operations or check algorithms. A GUI is all the user has to deal with. Separating the concerns between the user interface, algorithms and database is the duty of the configuration tool.

To cover these requirements, next section defines the design concept of MMC through UML diagrams [Obj11] and its main purposes.

5.3 Design Concept

During this phase a detail design for the complete system of MMC as well as for each of the individual components is developed. The design is prepared to be translated directly into code in the next phase.

The present methodology of integrated model-based design is adapted by the concept of MMC (see Figure 5-2). In order to analyze the integrated multidisciplinary dynamic behavior of the system there is a need of simulation models that analyze the dynamic behavior of a product. To simplify the consistency with the help of feature model, a list of existing simulation models is supplied and from this list an existing simulation model could be selected. MMC accesses system knowledge formalized in a semantic system model to fulfill the requirements. Additionally, efficient integration of approved solution knowledge is in this way enabled e.g. DBM that are prepared for reuse (cf. [MMP+04],[PDS+01]) [OAL+16].

Three main purposes are followed by MMC, as shown in Figure 5-2 [OAL+16]:

- Semi-automatically configures dynamic behavior models by combining models of the components for the specific problem and retrieve information about new re-usable DBM.
- provides bi-directional access to the system knowledge/system model and therefore able to ensure consistency between dynamic behavior models.
- provides bi-directional access to reusable models (solution knowledge) that were made available via semantic web ontologies, which is focused in the dissertation of Oestersötebier [Oes17]

According to the feature model the designer is able to analyze the integrated system by means of the assembled simulation model and find appropriate combination of models or add information about new model for component respectively. Furthermore, the desired level of detail can be taken into account by MMC. Thus all component models are currently available at its disposal [OAL+16].

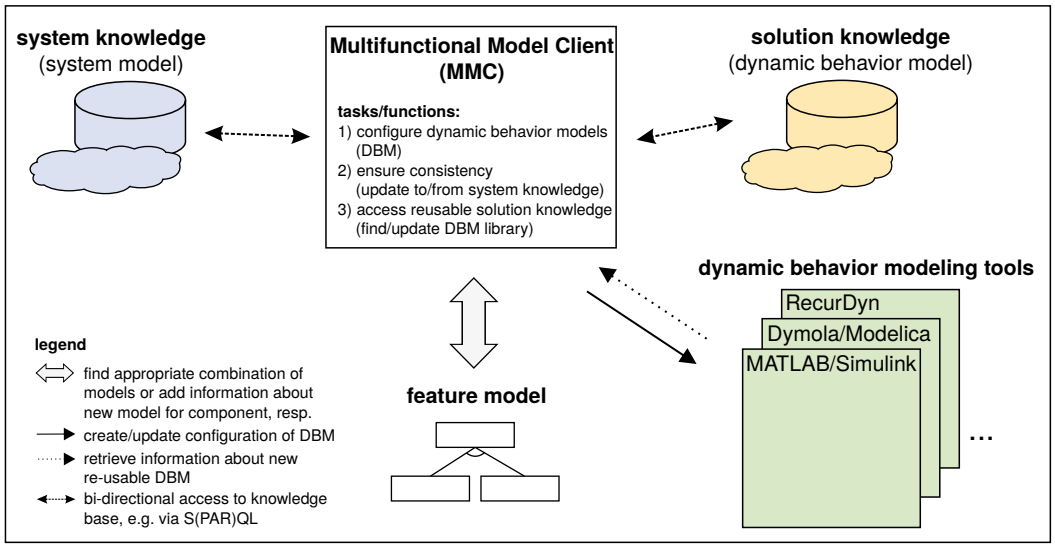


Figure 5-2: Design concept of Multifunctional Model Client [OAL+16]

In the following, the design concept is defined in details by the use of use case diagram, class diagram, state machine and activity diagram.

5.3.1 Activity diagram

To describe the dynamic aspect of the MMC Figure 5-3, 5-4 and 5-5 represent a model that shows the flow of elementary and executable actions in MMC in three phases of modeling design. These diagrams consist of activities that are made up of actions which apply to behavioral modeling technology to cover the design development based on the V model. This includes simple actions of developer and MMC in different phase of system design that create or update the objects. All this actions to create an executable simulation model and test it in an early stage are presented. In each Figure, there are three partitions determination of objective, synthesis and analysis, and two objects, MMC and user. In the following, activity diagram notations and then MMC activities by activity diagrams are explained.

Activity diagram notation

Activity: *organizes and specifies the participation of subordinate behaviors, such as Subactivities or Actions, to reflect the control and data flow of a process[Ent].*

Datastore: *used to define permanently stored data[Ent].*
Object: is a particular instance of a class at run time[Ent].
Partition: *is used to logically organize development phases in the Activity diagram[Ent].*

Discipline-spanning conceptual design (see Figure 5-3):

It begins with the initializing of system knowledge from solution knowledge. That means the library of both system and dynamic model must have the same elements. The result of this phase is principle solution, which will be integrated as solution pattern in the solution knowledge for later design. Following steps are done by MMC and user for discipline-spanning conceptual design phase, as shown in Figure 5-3:

1. MMC: initialize of system knowledge to create the system knowledge from solution knowledge
2. MMC: system knowledge as a permanently stored library in Enterprise Architect.
3. User: choose solution pattern made up of solution elements
4. User: create active structure:
5. MMC: crate feature model from active structure
6. User: choose desired components from configuration tree
7. MMC: create dynamic model from configuration tree
8. User: modify the dynamic model
9. MMC: update feature model from dynamic model
10. MMC: update system knowledge based on feature model
11. User: analyze the simulation model, if the result is not acceptable the dynamic model will be modified again
12. MMC: if the end result of simulation is accepted the stored system knowledge will be updated and stored.
13. Datastore: a principle solution

Discipline-specific design (see Figure 5-4):

It begins with the definition of a requirement list from previous phase. From it a part of concrete solution is resulted. It depicts the principle solution of last phase with more details, which will be saved as solution element in solution knowledge. Following steps are done by MMC and user for discipline-specific design phase, as shown in Figure 5-4.

1. User: complete requirements to get a requirements list

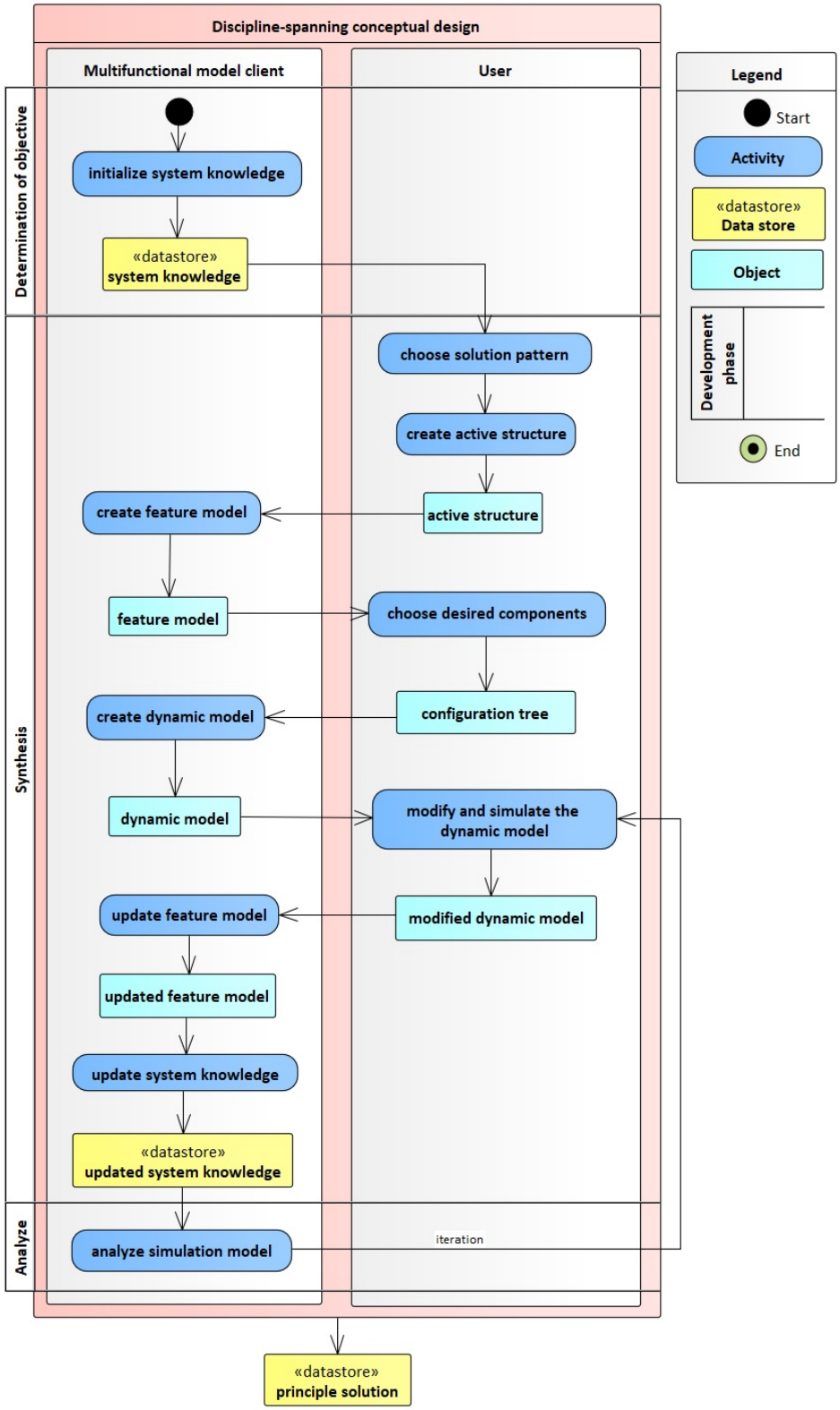


Figure 5-3: Activity diagram of discipline-spanning conceptual design of Multifunctional Model Client

2. User: choose solution element for updating the active structure
3. User: update the existing active structure from the discipline-spanning conceptual design phase
4. User: update system knowledge based on modified active structure
5. MMC: update feature model based on the new modified active structure and updated system knowledge
6. User: choose desired components from configuration tree based on feature model
7. MMC: create dynamic model based on user's selection of components
8. User: modified dynamic model with more details
9. MMC: update feature model based on the modified dynamic model
10. User: analyze the simulation model to validate the dynamic model
11. User: update the solution knowledge based on validated model
12. Datastore: solution knowledge
13. MMC: update system knowledge based on the updated solution knowledge
14. Datastore: system knowledge
15. Datastore: concrete part solution

System integration (see Figure 5-5):

It begins with development of integration strategy to choose an automation system for SiL and continuing with the HiL. It results a concrete solution, which encompasses the entire system and just like other solutions will be integrated in solution knowledge. Following steps are done by MMC and user for system integration phase, as shown in Figure 5-5.

1. User: develop integration strategy
2. User: choose automation system
3. MMC: update feature model based on the chosen automation system
4. User: choose desired components
5. MMC: configure dynamic model based on chosen components
6. User: generate an automation module based on the dynamic model
7. User: integrate of automation module in an automation system
8. User: simulate and validate entire dynamic model
9. User: update solution knowledge based on the dynamic model
10. MMC: update system knowledge based on the solution knowledge
11. Result: datastore of concrete solution

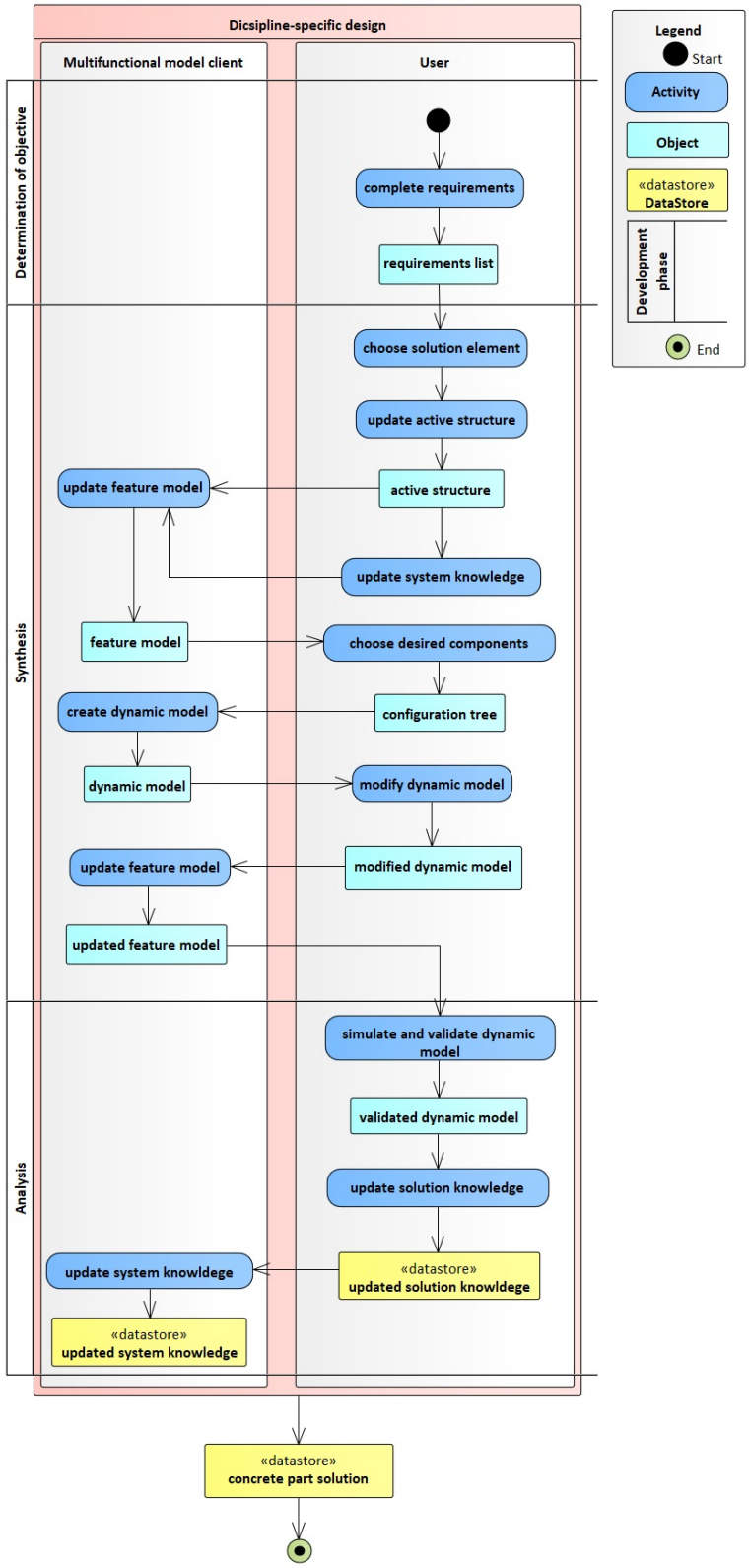


Figure 5-4: Activity diagram of discipline-specific design of Multifunctional Model Client

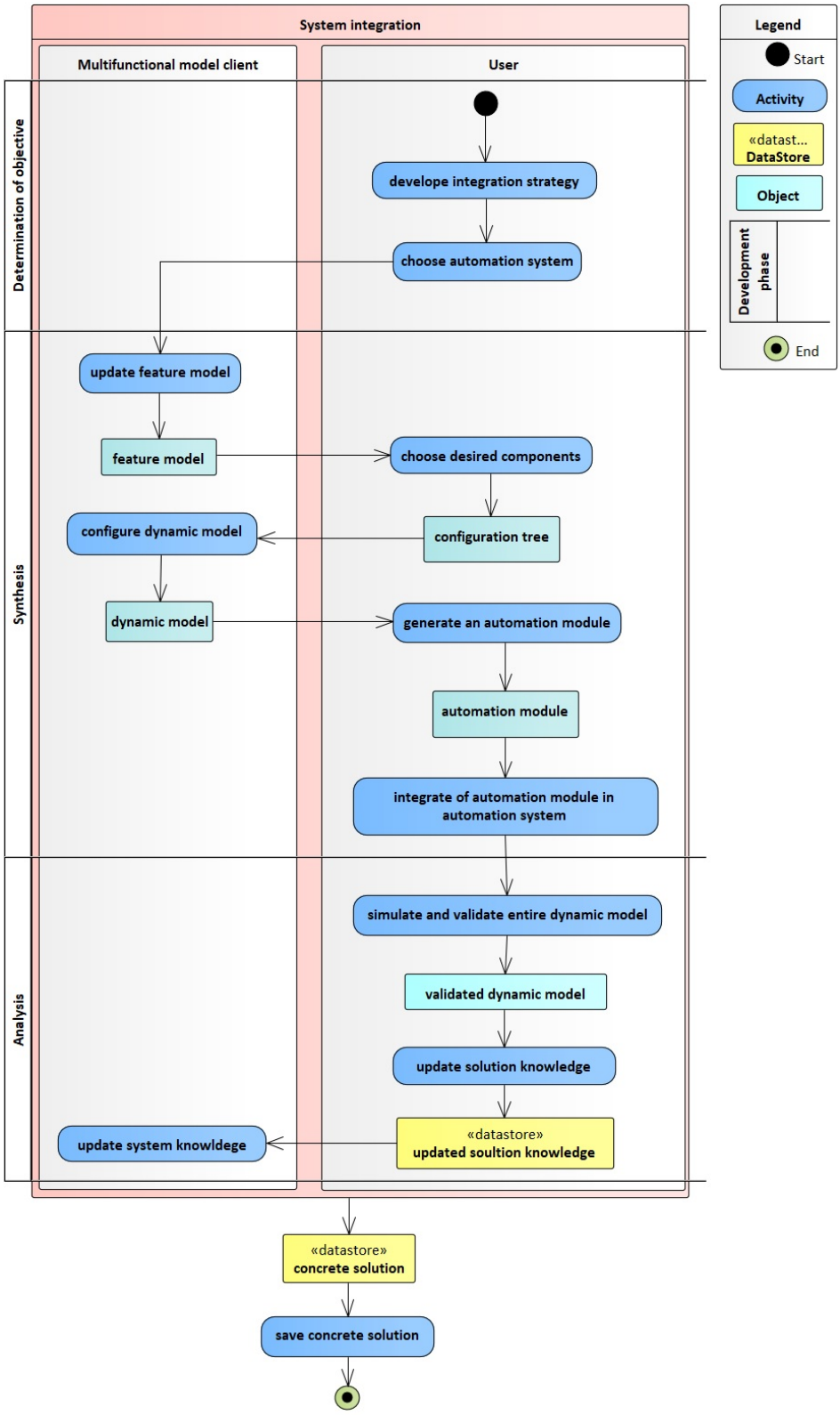


Figure 5-5: Activity diagram of system integration of Multifunctional Model Client

5.3.2 Use Case Diagram

The Use Case model is a catalog of system functionality described using UML Use Cases. The system boundary shows the logical interface between users and the system being described. Use case diagram includes some scenarios that describes the behavior and interaction of the mechatronic system with the MMC and a developer. In Figure 5-6 shows the developer as actor interaction and the MMC as system interaction. The nine use cases indicate the scope of the system, as shown in Figure 5-6. They are listed below:

- Task of user: create system model, complete dynamic modeling and update dynamic model, are done by users.

- Task of MMC: initialize knowledge base, create dynamic model and create feature model are done

- Interaction between user and MMC: update feature model, update system model and update knowledge base

Definition of used terms in the Figure 5-6:
User: developer
Multifunctional Model Client: prototype of a software tool
Initialize knowledge base: align system knowledge library with simulation model library and inverse by MMC
Create system model: create an active structure as a system model in Enterprise Architect based on selected solution pattern or solution knowledge by developer
Create dynamic model: create a dynamic model in Simulink or Dymola based on selected components from configuration tree of feature modeling by MMC
Complete dynamic model: complete dynamic model by developer if something is missing in the model
Update dynamic model: update dynamic model based on modified system model by developer
Create feature model: feature model is created based on active structure as system model by MMC
Update feature model: update feature model based on modified dynamic model by MMC
Update system model: update system model based on modified feature model by MMC
Update knowledge base: update knowledge base library based on the modified a model in the library by MMC

Figure 5-6: Use case diagram of Multifunctional Model Client

5.3.3 Class Diagram

In class diagram the structure and behavior of objects with the same characteristics and semantics are described by a class. Attributes describe the structure, while operations describe the behavior [Wei07]. In class diagram, the data, or more generally, the static structure of a system are defined[PBL05]. In various software development phases can be used class diagrams. It helps in implementation perspectives. Here, they are used for implementations in development of MMC in Java language and application. Therefore in this subchapter, a system means the MMC as a piece of software. To clarify how this prototype is developed, the classes and their behavior in the MMC java code are explained in the following class diagrams. This system is divided into multiple class diagrams to represent graphically each specific part of the system and make it more understandable.

As shown in Figure 5-7, MMC allows creating of several `Feature projects`, which system model library and dynamic model libraries are derived to build a system model and dynamic models respectively. In the structure of MMC a `Feature project` is the superclass of `Feature model`, while it has one `Configuration` class.

The explanation of each class of class diagram in Figure 5-7 in details are following.

`ModelXml`: the transformation result of source file. The system model and dynamic behavior models files will be referred to the *ModelXML*. To list all metadata of a model, a uniform XML schema is defined as *ModelXML*(see attachment A.1). In next part the structure of ModelXML in detail is described.

`System model library`: a database that contains system elements with relevant characteristics, properties and interfaces.

`System model`: is an active structure gathers the system elements, their attributes as well as the relation of the system elements.

`Interface of system model`: a collection of attribute definitions of system model that define cohesive set of behaviors. It is implemented in `System model library`, `ModelXml`, `Feature Project` and `Dynamic model`.

`Dynamic behavior model library`: database of reusable dynamic behavior models with specific modeling detail and individual components model, integrated in the system dynamic model.

`Dynamic behavior model`: model of entire system dynamic or part of it

`Interface of dynamic model`: a collection of attribute definitions of dynamic model that define cohesive set of behaviors. It is implemented in `ModelXml`, `Feature Project` and `Dynamic model`.

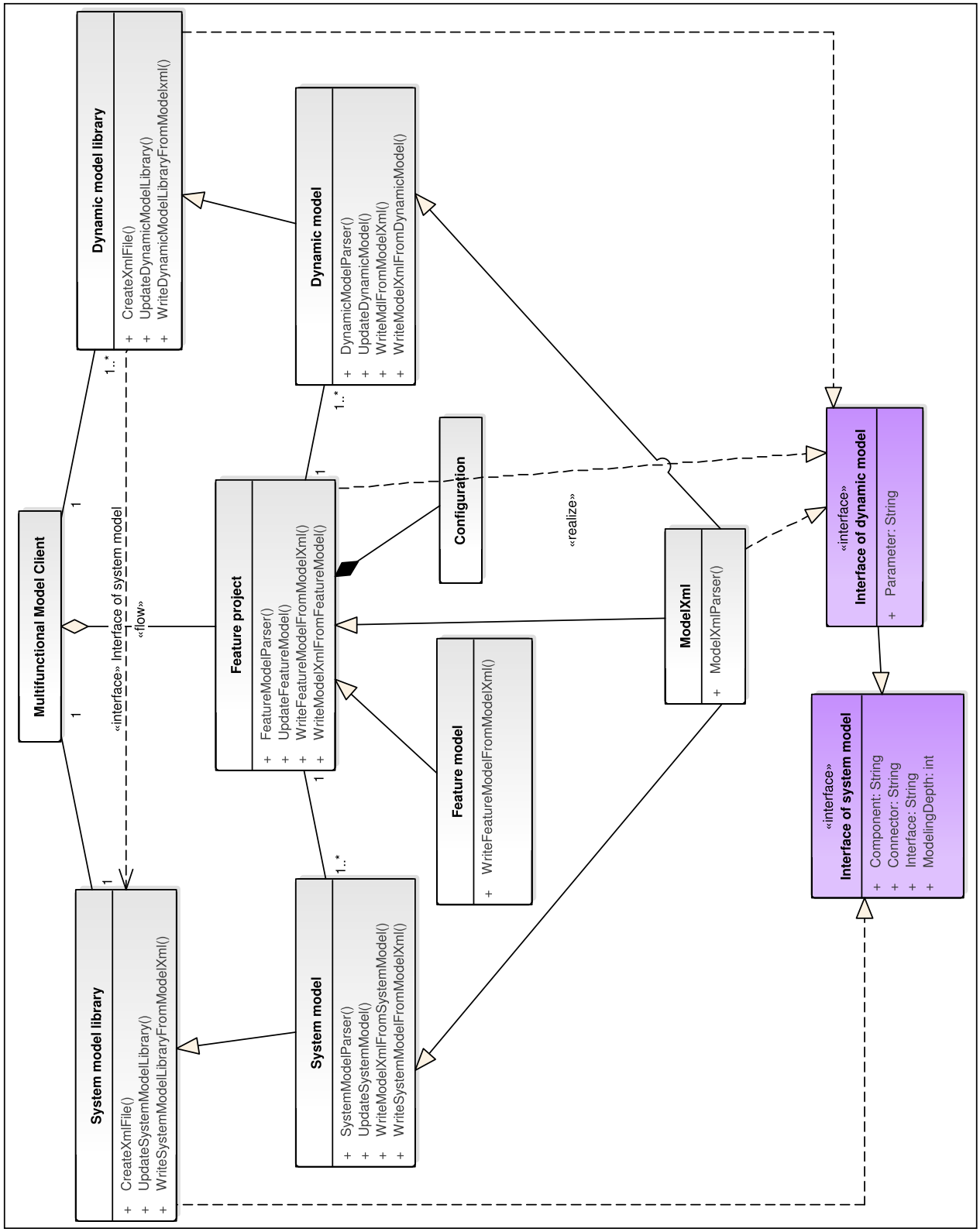


Figure 5-7: Class diagram of Multifunctional Model Client

`Feature Project`: gets the feature model from domain analysis as input and offers configuration choices [TKB+14].

`Feature model`: captures the variabilities and commonalities of mechatronic system models, models are mapped to the features

`Configuration`: customizes mechatronic system via feature model

Following points are in the Figure 5-7 to be considered:

- The Configuration and Feature model classes created in a Feature Project.
- Dynamic model can be made by the elements of dynamic model library (solution knowledge).
- System model can be made by the elements of system model library (system knowledge).
- System model library elements are reproduced from dynamic model libraries.
- Feature project sets and get the selection of system and dynamic elements.
- Feature project, Dynamic model and System model are transformed in ModelXML format, then its class inherits all properties of Feature Project, Dynamic model and System model classes.

To comprehend the structure of feature model-class diagram in Figure 5-9, there is to show in Figure 5-8 a primary construct of a component that makes up a mechatronic system in a class diagram. It is a conceptual diagram to describe a component in the mechatronic system. Following points describes specially the relationship of `component` class to others, as shown in Figure 5-8:

- A component can have many models which can be modeled in different modeling tools and in four different modeling depth.
- A component has interfaces and parameters.
- A component model could be a system or a solution knowledge element.

The class diagram of feature model is shown in Figure 5-9 to depict static classes and their relationships in the implementation of MMC software. Here the structure of feature model for a mechatronic system is shown. This type of class diagrams is designed from implementation perspective in java language and its application. Following points explain the Figure 5-9:

- `RootFeature` (right side of Figure 5-9) is the main feature and all other features are its subfeatures.
- `Component`, `InterfaceOfComponent`, `ParameterOfComponent`, `ModelingToolOfComponent`, `ModelingDepthOfComponent` are of type Feature.

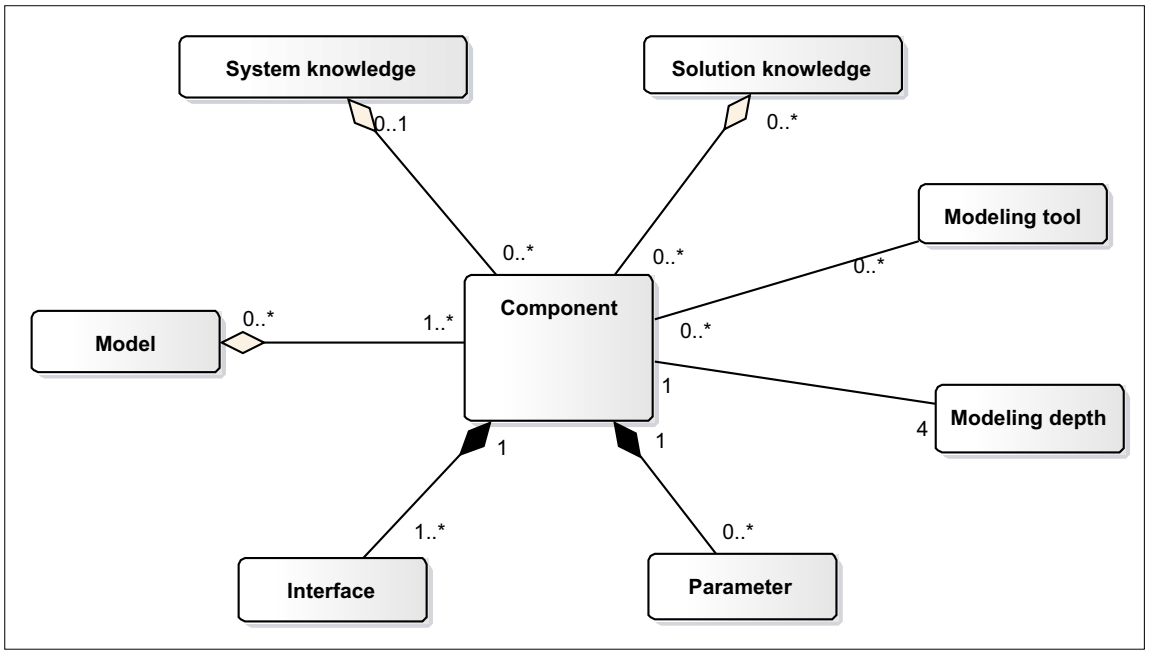


Figure 5-8: Class diagram of a component

- `MechatronicSystem` is composed of Feature.
- Though `InputOfComponent`, `OutputOfComponent` and `PortOfComponent` inherited all properties of `InterfaceOfComponent`.
- `ModelingDepth` is made up of `InterfacOfComponent` and `ParameterOfComponent`. It means a `ModelingDepth`-feature has subfeature of `InterfacOfComponent` and `ParameterOfComponent`.
- `ModelingDepth` composed to `ModelingTool`. It means a specific modeling depth belongs to a `ModelingTool`.
- `InterfaceOfComponent` and `ParameterOfComponent` have same properties and methods.
- `Interface Characteristic` is a collection of attribute definitions of characteristic that define cohesive set of behaviors. It is implemented in `ParameterOfComponent` and `InterfaceOfComponent`.

ModelXML
ModelXML is a XML file that defines a set of rules for encoding all models information of a system, i.e. it is a transformation result of system model and DBM files. Here are all metadata of a model listed. A uniform XML schema is defined as ModelXML(see attachment A.1).

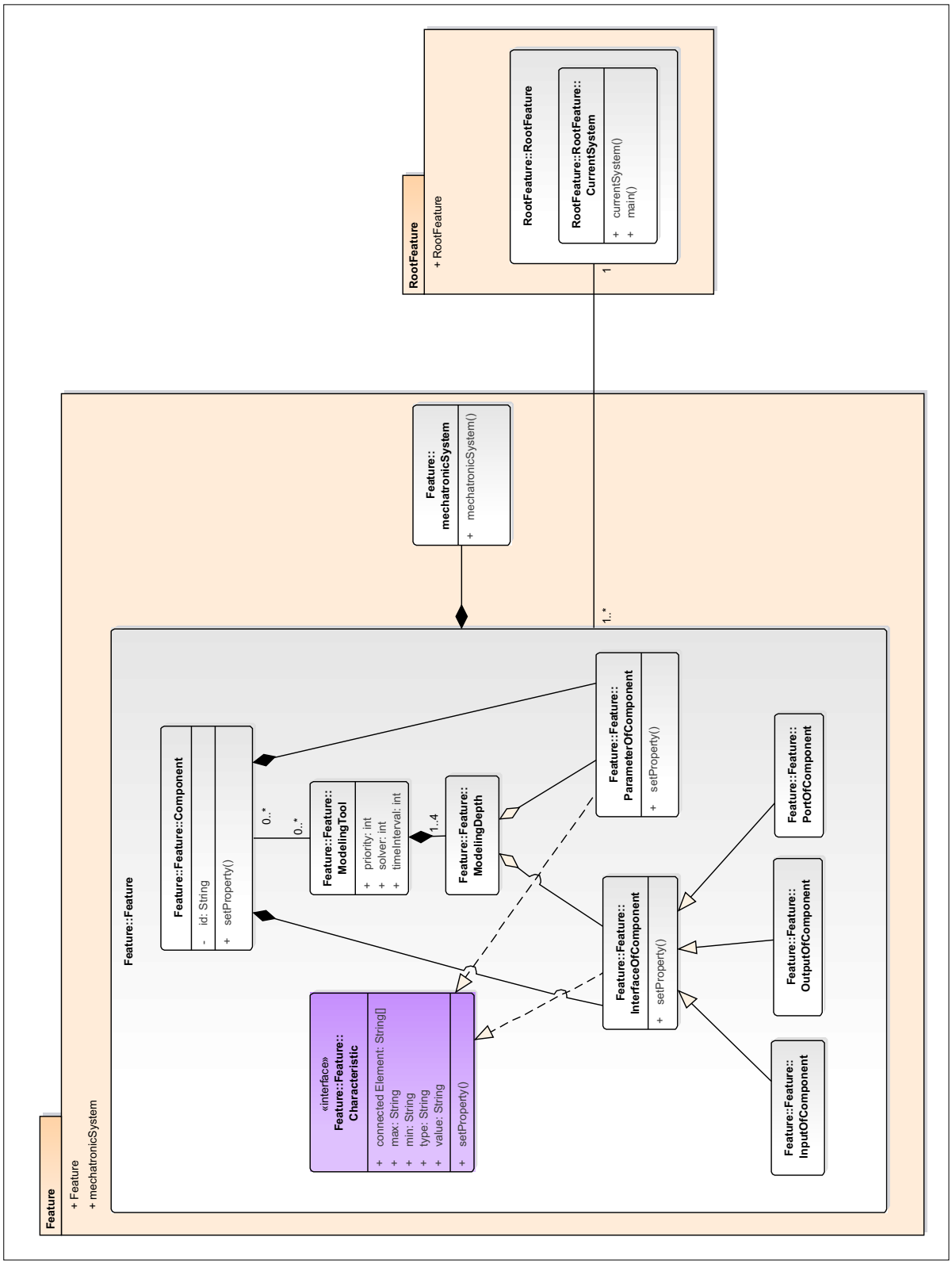


Figure 5-9: Class diagram of feature model

A ModelXML represents the entire data of a system, as represented tool, system and simulation model characteristics, all used interfaces and parameter properties. To translate data from one XML grammar to another a ModelXML is applied as a uniform XML schema (see attachment A.1). The XML is a standard for interchanging data in our implementation to simplify the transformation. As shown in Figure 5-10, the meta model of ModelXML has following constructs:

- `Modelxml` is made up of packages.
- `Type` defines the type of ModelXML, system model, system knowledge, solution element and solution knowledge.
- `Model` is made up of other Models, Components and their Connections.
- Component is made up of SystemElementModel, Parameter and Interface.
- `Package` is made up of other Packages.
- `Tool` is made up of Models. It defines the software, with which the package is modeled.
- `Connection` defines connections between Interfaces of different components.
- `Component` is made up of Interface, Parameter and SystemElementModel.
- `SystemElementModel` is a part of component. It belongs to SystemKnowledge.
- `Modifier` describes parameters of a system element.

5.3.4 State machine

A part of states and state transitions between feature model, system model and dynamic model in MMC is depicted in the Figure 5-11. The ModelXML (bottom of the Figure 5-11) is the central state, all other data is transformed to or from it to simplify the exchange of information. Each model has his own XML format or another format, which must be transformed to ModelXML to standardize the collection of information about the models in different domains.

Following transitions are executed in MMC:

- From `system model` is a `system model xml` created.
- From `system model xml` is a `modelXml` created.
- From `modelXml` is a `feature model xml` created.
- From `feature model xml` is a `featue model` and `feature model configuration` created.

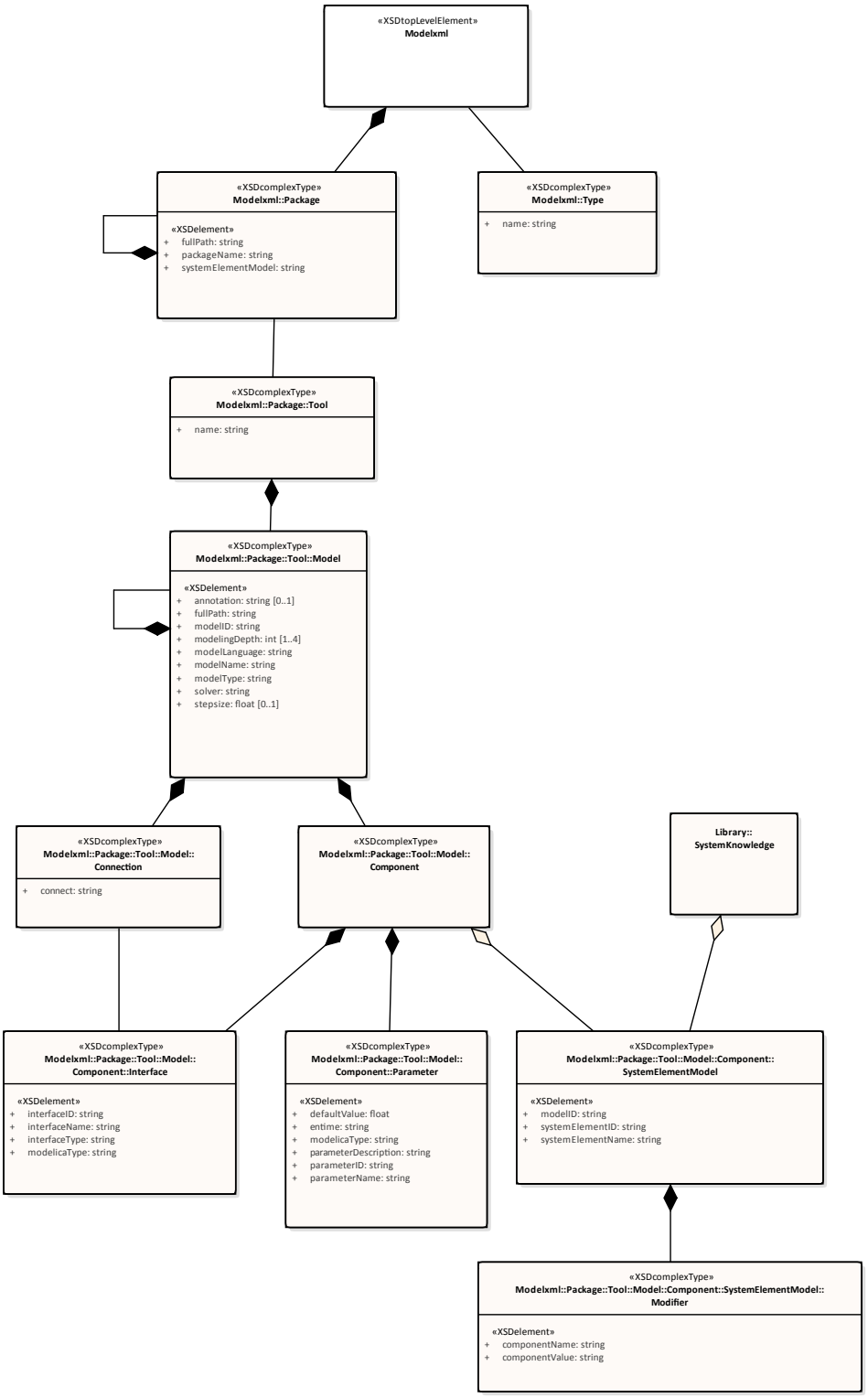


Figure 5-10: XML schema of ModelXML in form of class diagram

- From `featue model` and `feature model configuration` is a `modelXml` created.
- From `feature model xml` is a `modelXml` created.
- From `modelXml` is a `dynamic model` created.
- From `dynamic model` is updated.
- From `dynamic model` is a `dynamic model xml` created.
- From `dynamic model xml` is a `modelXml` created.
- From `modelXml` is the `feature model xml` updated.
- From `modelXml` is a `system model xml` created.
- From `system model xml` is a `system model` created.

Here, there is a special data transition between libraries of system and solution knowledge. The transition of data between libraries or knowledge is depicted in Figure 5-12 to synchronize the library of system and solution knowledge in Matlab/Simulink. Here, from a dynamic model library (solution knowledge) a system model library (system knowledge) is created, hereby the element of the dynamic model library in system model is used. As before, the ModelXML is central state of transformation. As mentioned above, the transformation of ModelXML in Matlab/Simulink is executed directly to a model (mdl) file.

Following transitions are executed between libraries of system model and Matlab/Smulink:

- From `system model library` is a `system model library xml file` created.
- From `system model library xml file` is a `modelXml` created.
- From `modelXml` is a `dynamic model library` created.
- From `dynamic model library` is a `dynamic model library xml file` created.
- From `dynamic model library xml file` is a `modelXml` created.
- From `modelXml` is a `system xml model library` created.
- From `system model library xml file` is a `system model library` created.

5.4 Implementation of Prototype

In this section, a prototype of MMC is implemented. Thereby the GUI, its tasks and functions, covered by the tool, are described. The software is implemented in

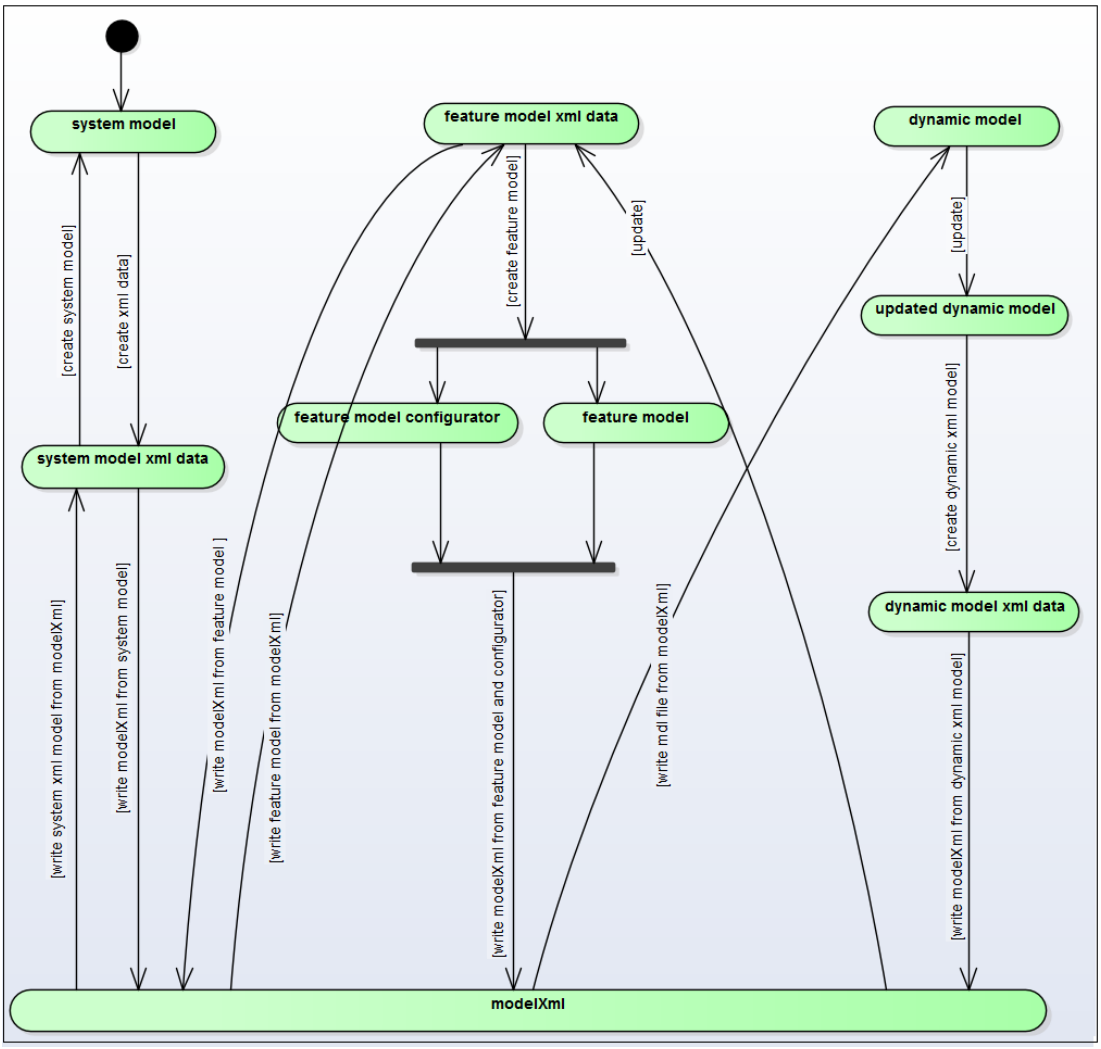


Figure 5-11: State machine of Multifunctional Model Client

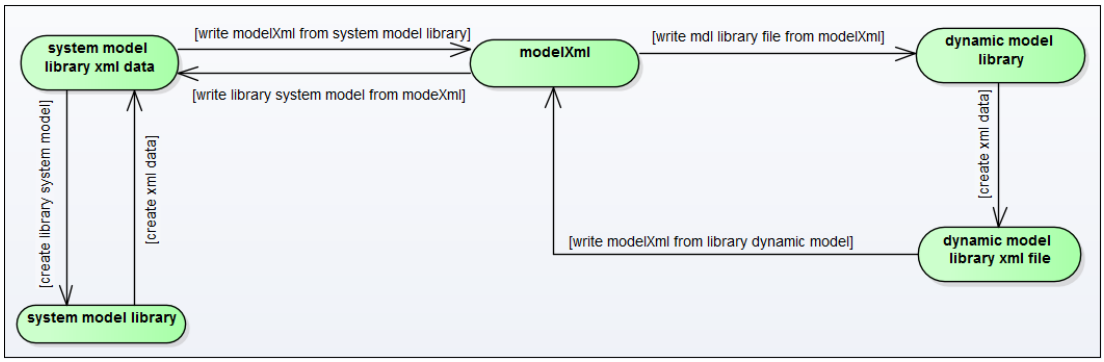


Figure 5-12: State machine of system model library and dynamic model library

an open source development environment, eclipse [Ecl]. The system and solution knowledge is implemented in the form of ontology to make them explicitly available, based on the work of Oestersötebier [Oes17]. It makes implicit knowledge explicitly usable. Further, the functions and algorithms are implemented in Java, whereas GUI is implemented in a bi-directional Java GUI designer, WindowBuilder [Win]. In addition to system model designed in the Enterprise Architect and Papyrus, the mechanical design is performed and refined in Simulink, Modelica and Recurdyn. Also SiL is performed in TwinCat from Beckhoff [Twi].

In the following sections the implementation of database and GUI is explained to show the dealing approach with the developed MMC tool.

5.4.1 Database of system and solution knowledge

In the thesis of Felix Oestersötebier [Oes17] an approach is employed to communicate with the solution knowledge. For this propose, a concept, as shown in Figure 5-13, is worked out and partially implemented at critical points [Oes17].

The generation or parameterization of SPARQL Protocol And RDF Query Language (SPARQL) queries is happened between Ontology server and model information. The required OWL/Ressource Description Framework (RDF) Triple for updating the knowledge model and the CCAE model based on the information are obtained using SPARQL, because some CAE-tools use already Structured Query Language (SQL) interfaces to database, e.g. CATIA, Solid Edge, MATLAB) As shown in Figure 5-13 the relation between OWL and XML is used to ease the sharing information between different sources, because this format is established in many CAE tools. However, only some of the information are transferred to a solution knowledge, further the whole model is stored in the model repository [Oes17].

According to the Figure 5-13, following steps are indicated [Oes17]:

1. An Extensible Stylesheet Language Transformation (XSLT)[Cla99] is used to extract relevant model information. The source file and a predefined XSL file with the transformation instructions are given to an XSL processor. The result is an XML file so called ModelXML.
2. The information transferred to the Assertional Box (A-Box) of ontology. For this purpose, the terminology in the Engineering Model ontology is used to generate an OWL Triple. For that, an XSLT can be used for the XML based ontology language.The so-called JXML2OWL-Mapper6 supports [TRC06] provides a graphical interface that maps the XML tags to the Terminological Box (T-Box) elements to generate the transform file. Generatd A-Box parts can be added to the knowledge base by updating of SAPRQL.

3. By request information from the solution knowledge model the ModelXML file is updated.
4. For updating the model a generic object-oriented parser for the ModelXML format and a specific Modelica interface in Java is created. Modelica interface uses data from a given ModelXML to customize the Modelica source code in the appropriate places.

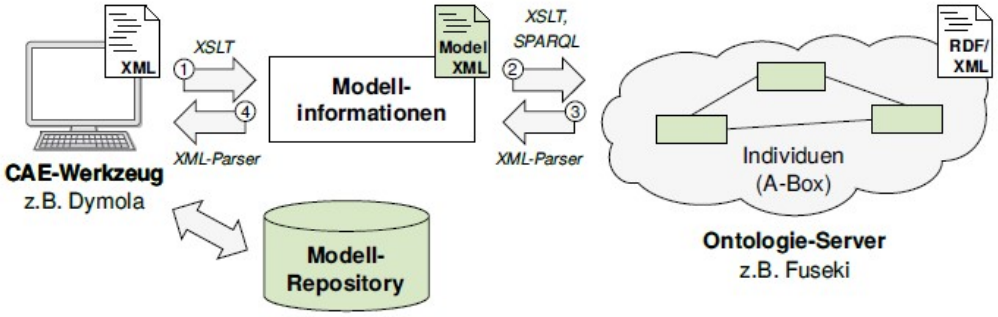


Figure 5-13: Bidirectional linking of model and ontology in 4 steps [Oes17]

5.4.2 Graphical User Interface

A GUI is mainly used for dealing with the developed MMC tool. The main operations of the MMC and their correlation to the various phases of the development procedure model are shown on the start screen as presented in Figure 5-14. Here, in addition to automation of some steps, is a check list provided to guide the user step by step through the V-model.

The three screen operations of the MMC, conceptual design, specific design and integration are described according to the V model in the following subchapters. Additionally, data exchange is a key requirement of MMC to execute the model transformation during the design process, which it will be discussed in the next subchapter.

Transformation Approach

In the following, a family of model-level transformations is characterizes by each metamodel specialization to convert one model to another. Families of horizontal transformations are executed in terms of relationships between metamodels that specify source and target UML models. A horizontal transformation transforms a source model into a target model at the same level of abstraction as the source model. Model evolution is supported by horizontal transformation, i.e. adding

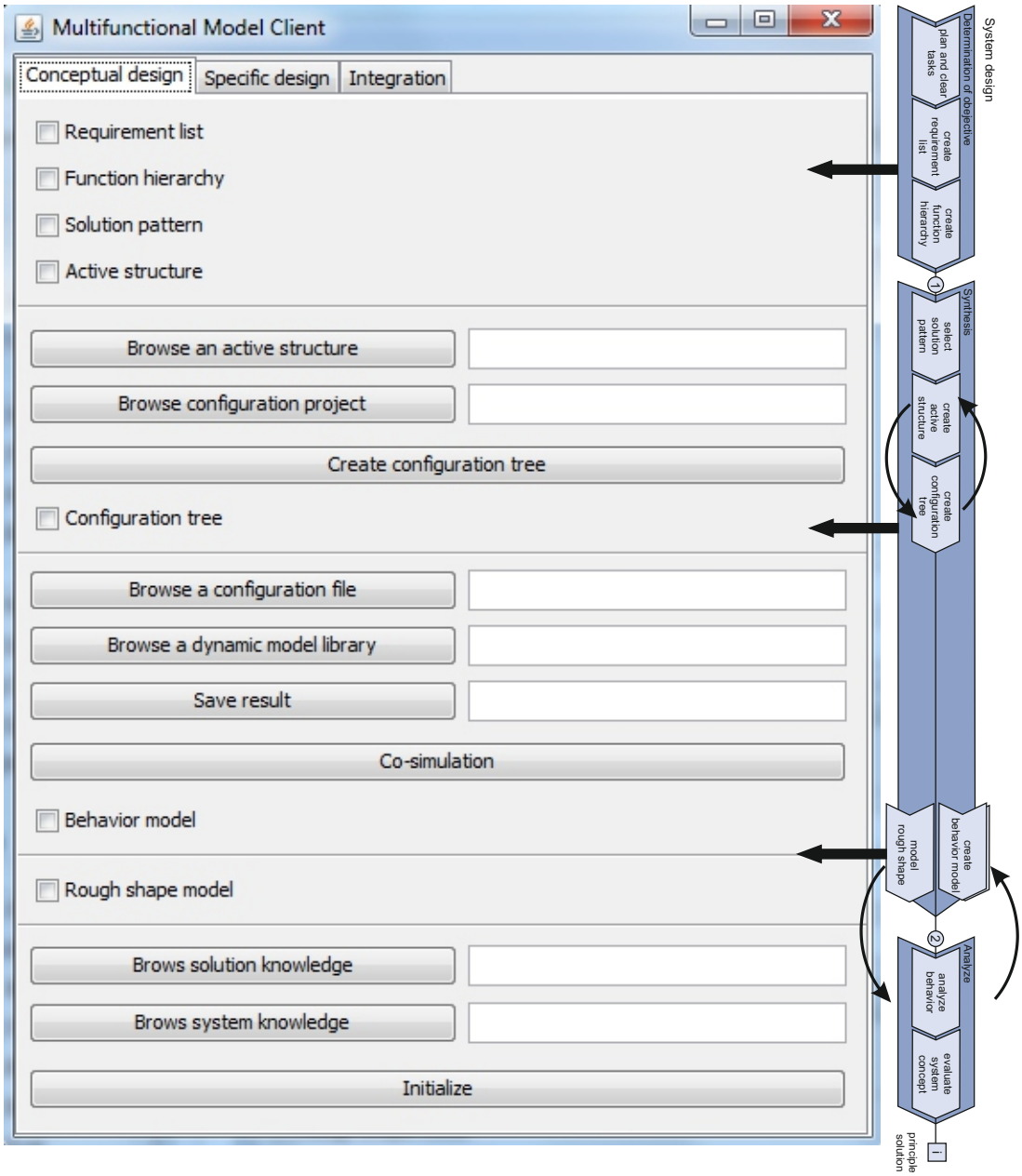


Figure 5-14: GUI of the Multifunctional Model Client: conceptual design

new features to a design and restructuring a design to enhance existing features [RJFLC03].

Here, the transformation takes place according to Meta-Model approach [FGT14]. Based on the Figure 5-15 start point is the source model, which is conform to its specific metamodel. A schema source metamodel is performed a mapping to the schema of target model, which presents the metamodel of target model. Both the source model and a set of mapping rules of metamodels obtained a specific target model.

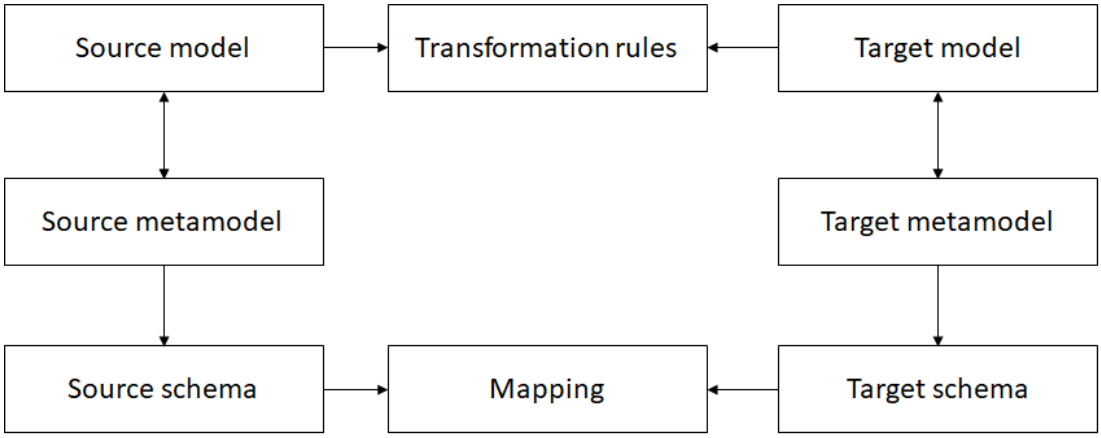


Figure 5-15: Metamodel transformation architecture

A mapping model consists of characterizations of Source and Target metamodel, and constraints on relationships between source and target elements. The Source and Target metamodel characterizes source and target UML metamodels. Here is a semi-automatic model transformation architecture is used, as in Figure 5-15. Following steps takes place when using mentioned approach for model transformation.

1. The source and target schemas are abstracted into source and target metamodels
2. Between the source and target metamodels the mapping are found and specified.
3. From the specified mapping an executed mapping is generated.
4. Source metamodel is abstracted into source schema
5. According to executed mapping rules from the source schema a target schema is generated
6. A target model corresponding to the source model is generated through executable mapping rules to transform source model to target model [WMR+11].

The ModelXML is conducted as source and target metamodel to create an interface between all models. In Figure 5-16 is shown all models are mapped with ModelXML, actually they all can be mapped with each other through ModelXML. If a new metamodel is observed to be mapped, it needs to be mapped just with metamodel of ModelXML, i.e. it is mapped to all other metamodels.

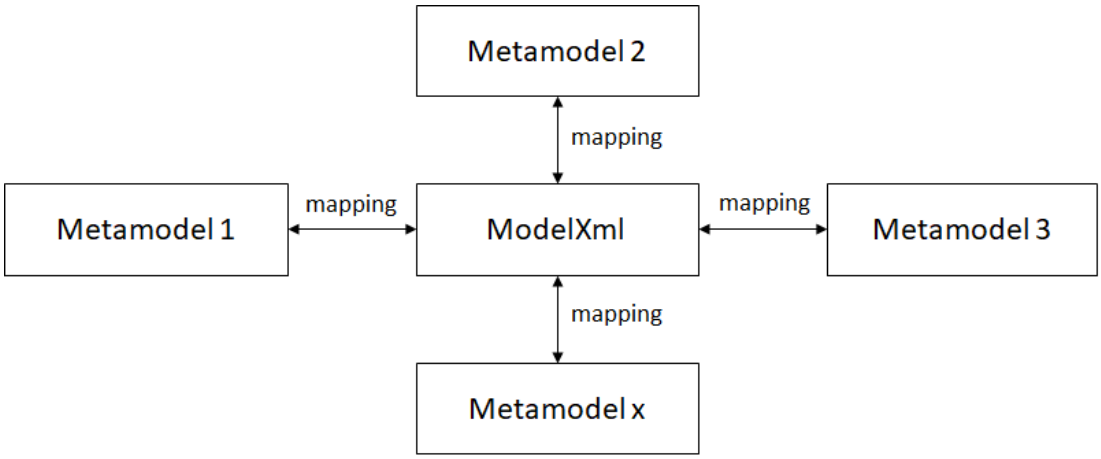


Figure 5-16: Mapping of ModelXML with metamodels

Conceptual design

In the following, all parts of the rubric of Conceptual design are listed and described. In the first section, as shown in Figure 5-14, there is a check list according to presented method in the chapter 3, so that the developer can be aware of executing all step of discipline spanning conceptual design sequentially.

Create configuration tree
Sequentially following functionalities are presented in second section of Conceptual design-rubric, as shown in Figure 5-14:

- Selection between knowledge base and partial models and active structure: load a file from the system or solution knowledge to create a feature model
- Browse an active structure: set a file from the library or load a new created active structure model
- Browse a configuration project: set a FeatureIDE project, which is created in eclipse for creating a feature model
- Create a configuration tree: the editor of FeatureIDE gets the feature model from knowledge base or active structure diagram as input and offers configuration choices.

To create a configuration tree, at first the creation of a feature model from active structure is required. For this target a transformation is performed, as shown in Figure 5-17. Here, the metamodel of the active structure is mapped to ModelXML metamodel and then it is mapped to a feature metamodel. According to the source meta model, an active structure describes one *System* and several *System Elements* similar to a component, which is part of *System*. System elements can be connected to each other by three different kinds of flow relations, namely: energy flow, material flow, and information flow [BBB+12]. To connect the system elements (components) to each other, interfaces of type *Flow Specification* are defined. The basic elements of feature model are depicted in the target meta model of feature model in Figure 5-17. The two central elements of the meta model of feature model are the *Feature* and *Subfeature* classes. A *Feature Dependency* is the association class of an association between the *Feature* and the *Subfeature* classes. Each subfeature is associated with other subfeature. The feature dependency is defined as an abstract class [PBL05]. The feature dependency relationship is specialized into the *Mandatory*, *Optional*, *Alternative* and *Or* classes.

Create dynamic model
A possible configuration tree base to the feature model is generated here. It depends on a system or part system, from which aspect, the user has the possibility to select between target modeling depth and tool from the configuration tree for creating a new dynamic model.

According to using the active structure to create a dynamic model, tasks of MMC are executed as follows:

- If the element in the library of selected simulation tool exists, it will be added in the dynamic model
- If the element in the library of selected simulation tool does not exist, a block with the known interfaces from the active structure will be added in the dynamic model
- The connection between elements will be created
- If matching the interfaces cause a semantic error, the process returns for a correction of interfaces. Since they could not be connected.
- Task of developer: complete and/or modify dynamic model

Sequentially following functionalities are presented in third part of Conceptual design, as shown in Figure 5-14:

- Browse a configuration file: set the file address of a configuration file from a FeatureIDE project
- Browse a dynamic model library: set the file address of solution knowledge

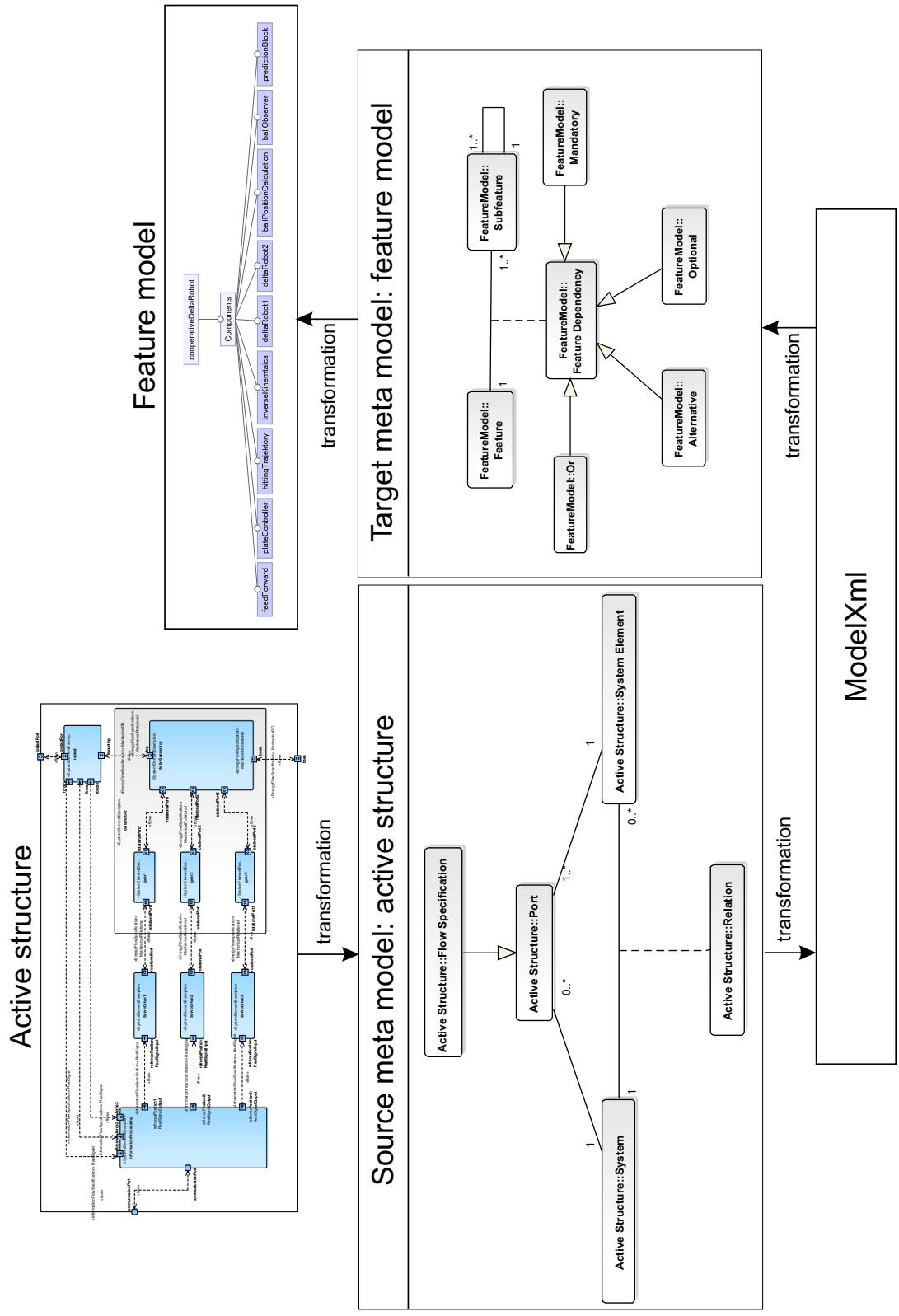


Figure 5-17: Transformation of active structure to feature model

- Save result: get the file address, where the new created dynamic model will be saved.
- Co-simulation: according to the user selection in configuration tree, a new dynamic model will be created. It couples two or more simulation softwares in the integrator level to co-simulate the submodels together, if they are modeled in different simulation software. Here, according to the Figure 5-18 the FMU is used to bring physical modeling tool into the Matlab/Simulink environment by the FMI toolbox from Modelon [Moda]. After creating the connection between Matlab and a Modelica simulators to describe the state of a FMU and the information flowing into/out of a FMU different kinds of variables are defined. Finally the co-simulation is executed stepwise as shown in Figure 5-18.

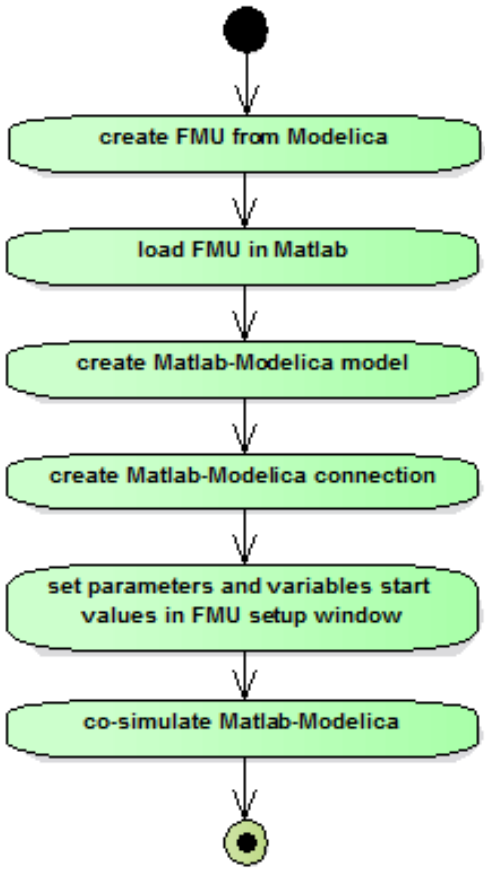


Figure 5-18: State diagram of co-simulation with FMI

To transform a feature model to dynamic model, such as Matlab/Simulink, Dymola, or Recurdyn, according to the metamodel approach following steps are done:

1. Transform feature model to ModelXML representation according to configuration tree
2. Transform ModelXML to dynamic model
3. Perform the co-simulation

As an example, in the Figure 5-19, the transformation of feature metamodel to ModelXML and to Simulink metamodel is shown.

The structure of the Simulink metamodel is depicted in right side of the figure. It shows a *System* that may contain a hierarchy of *Subsystem*s with *Block*s as leafs is called a Simulink model. *Block*s are the atomic processing units, connected to each other by connecting their *Outport*s and *Inport*s via *Line*s. Moreover, *Block*s have attributes in the form of *PropertyName* pairs that are either atomic or consist of properties in turn. It is directed from one *Block* to another one identifying first the fight *Inport* or *Outport*, following then a list of *Line*s to the opposite *Outport* or *Inport*, and eventually crossing the link from this element to its own *Block*. Just as importantly, patterns must be identified while writing analysis rules,where a block has a certain combination of property values or a certain number of outgoing or incoming connections [ALS+08].

Initialization
In the last part of Conceptual design-rubric is an initialization functionality to see in Figure 5-14. To initialize system knowledge according to solution knowledge a transformation is defined within eclipse represented in Figure 5-20. Following steps are executed for the transformation of dynamic model library to system model library according to the metamodel approach:

1. Transform solution knowledge (dynamic model library) to ModelXML representation
2. Transform ModelXML to system knowledge (systen library)

According to the source meta model in the Figure 5-20 following classes are defined:

1. *SolutionKnowledge* is first defined as a class.
2. A solution can be profiled through a *ModelingTool*, this means that a component can be modeled in different modeling tools.
3. A library of modeling tools includes *Model*, *Component* and *Interface*.
4. Whereas interfaces are modeled separately, *Parameter* belongs to components.
5. The *Package* and *Type* defines name, path and type of the modeling tool, respectively.

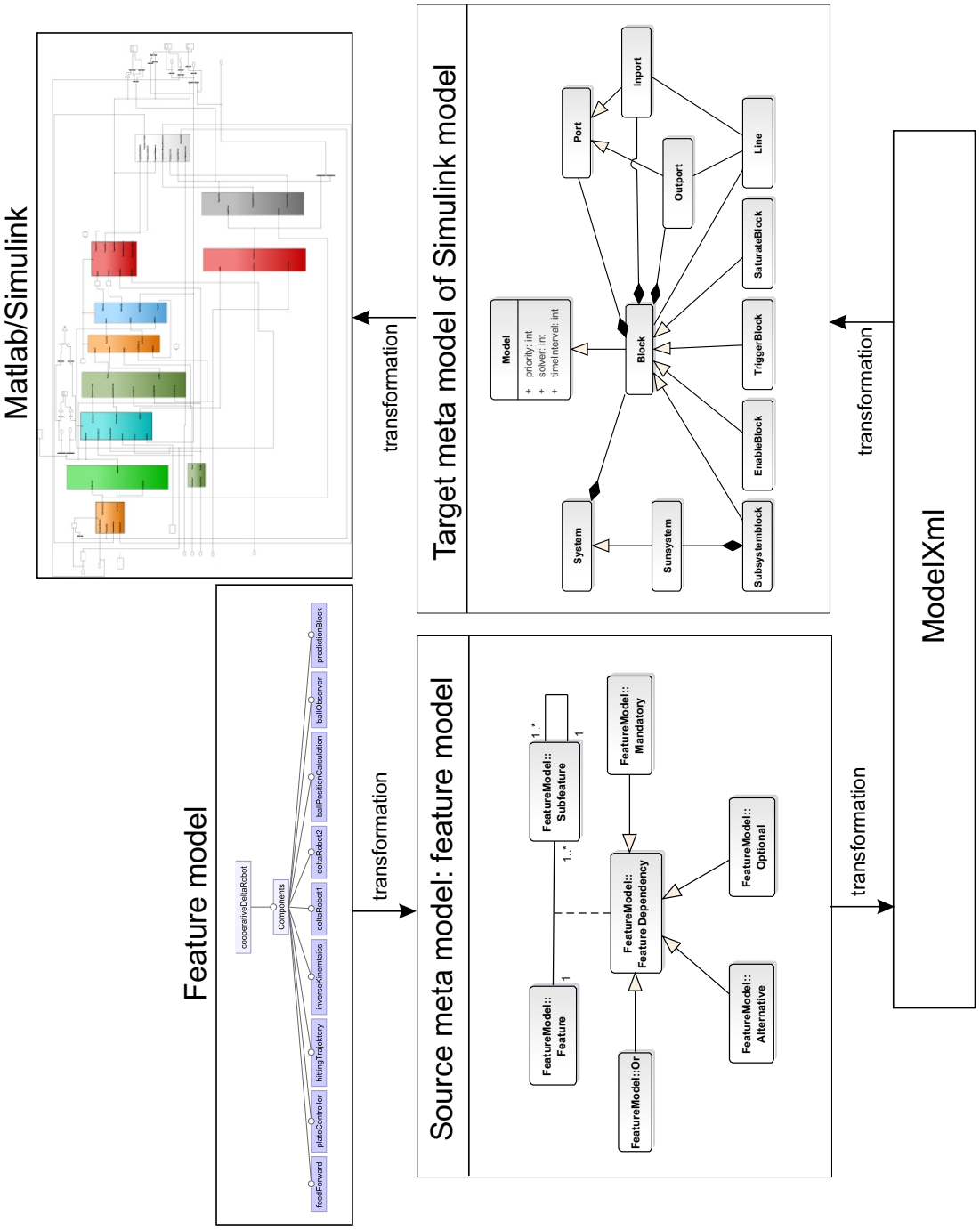


Figure 5-19: Transformation of feature model to Matlab/Simulink model

The right side of Figure 5-20 shows the target meta model. The *SystemKnowledge* as the library of system model includes *Model*, *Component* and *Interface*, which will be available for system modeling.

Specific design

Following all part of Specific design-rubric are listed and described. The intention of *Update*-functions, shown in Figure 5-21, is to keep consistency during the design process of mechatronic system, i.e. consistency between system model, feature model and dynamic models from one side and from other side consistency between system knowledge and solution knowledge is supported.

Update from dynamic model:
This part of Specific design-rubric is considered to update configuration project and active structure according to dynamic model. The functionality of buttons are:

- Browse a dynamic model: set the file address of a dynamic model
- Browse a dynamic model library: set the file address of solution knowledge
- Browse a configuration project: set an existing configuration project, which will be updated according to the selected dynamic model
- Browse an active structure: set an exiting active structure, which will be updated according to the selected dynamic model

Update from system model:
The last part of Specific design-rubric is considered to update configuration project and dynamic model according to system model. The functionality of buttons are:

- Browse an active structure: set the address of system knowledge, which must be updated
- Browse a configuration project: set the address of a FeatureIDE project, which must be updated
- Browse a dynamic model: set the file address of a dynamic model
- Browse a dynamic model Library: set the file address of solution knowledge

Activity diagram in Figure 5-22 shows the flow behaviors of MMC in the case of changing system model or dynamic model. To keep consistency between them, changed elements of active structure are mapped to dynamic model and reverse.

Changes in active structure causes changes in the feature model according to the system knowledge. Sequentially the created dynamic model follows changes from feature model through ModelXML. As shown in the Figure 5-22, in the case of updating active structure according to system knowledge the ModelXML is

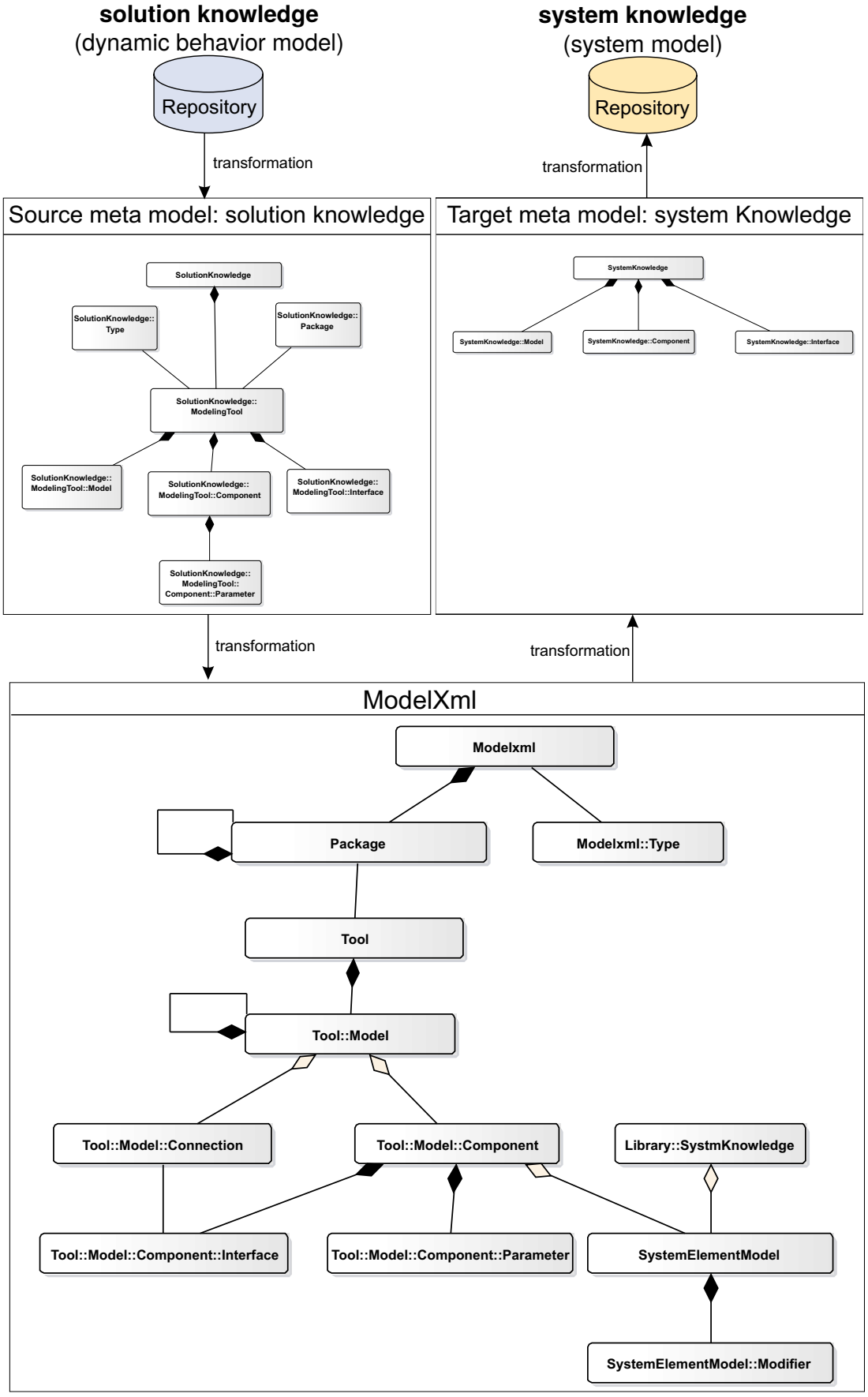


Figure 5-20: Transformation of solution knowledge to system knowledge

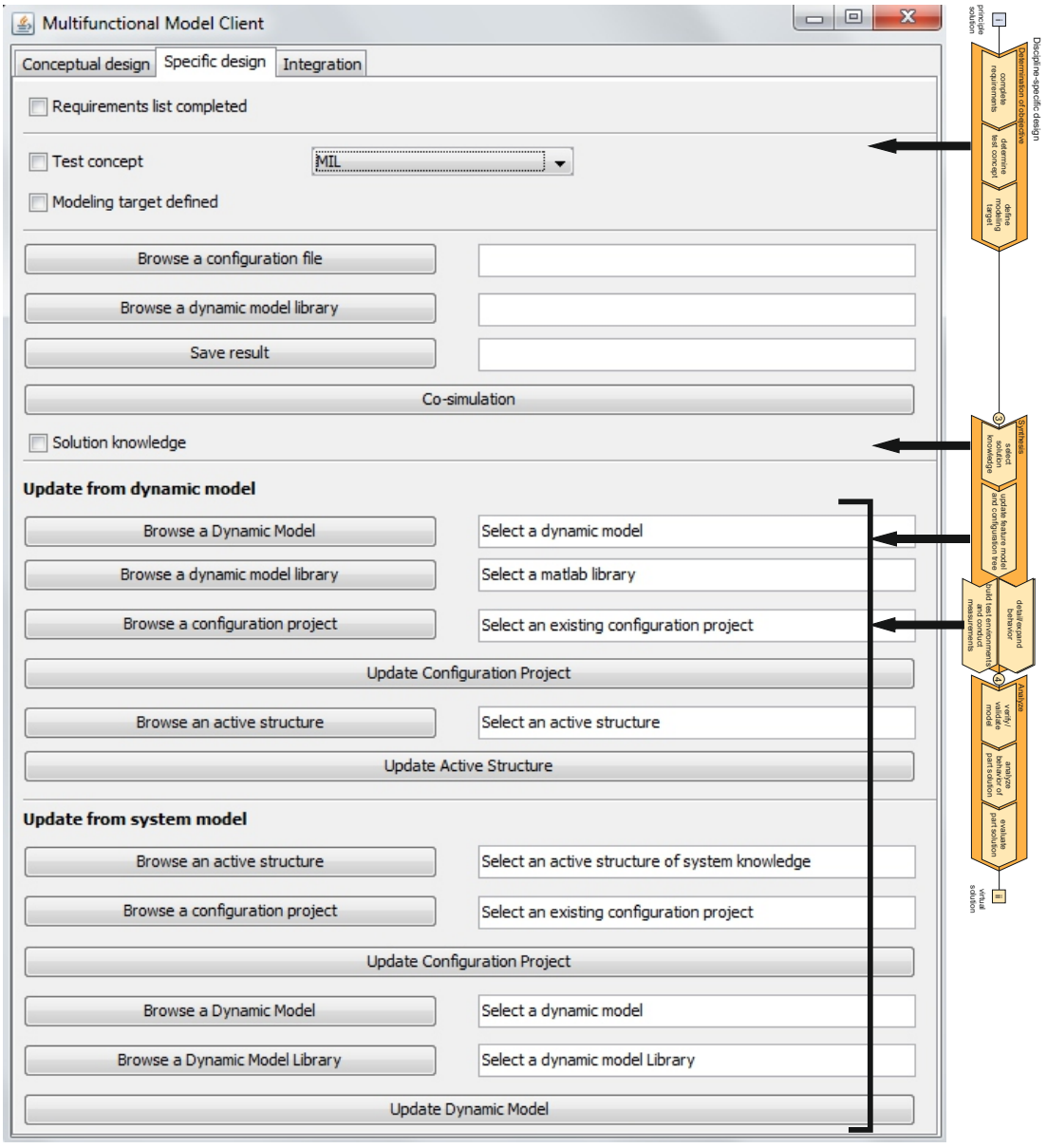


Figure 5-21: GUI of Multifunctional Model Client: specific design

created. Then based on the new ModelXML the current feature model is updated. Finally, based on the ModelXML created from updated feature model, dynamic model will be updated.

In another case, feature model and active structure will be updated parallel according to solution knowledge and changes in dynamic model. They could executed parallel because active structure as a part of partial models is independent of feature model. As shown in the Figure 5-22, in the case of updating dynamic model according to current solution knowledge the modeldXml is created. Then based on the new ModelXML feature model and active structure will be updated parallel.

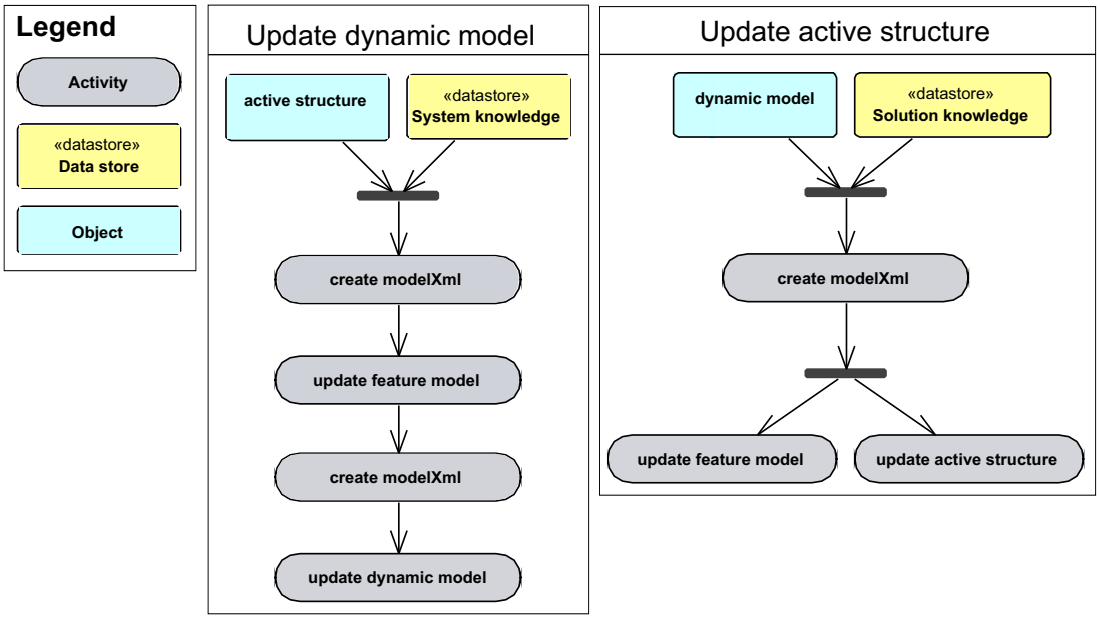


Figure 5-22: Flow behavior of update window of Multifunctional Model Client

The class diagram of update processing is shown in Figure 5-23. There are three class categories: parser, update and writing. While the *UpdateSystemModel* and *WriteModelXmlFromSystemModel* classes require *SystemModelParser* and *ModelXmlParser* classes for their full implementation, the *UpdateFeatureModel* and *WriteModelXmlFromFeatureModel* classes need *FeatureModelParser* and *ModelXmlParser* classes. Respectively, the *UpdateMatlabModel* and *WriteModelXmlFromMatlab* use *MatlabMdlParser* and *ModelXmlParser*. Since all models must be transformed to modelXML, all classes which write or update a model use *ModelXmlParser* as a central class.

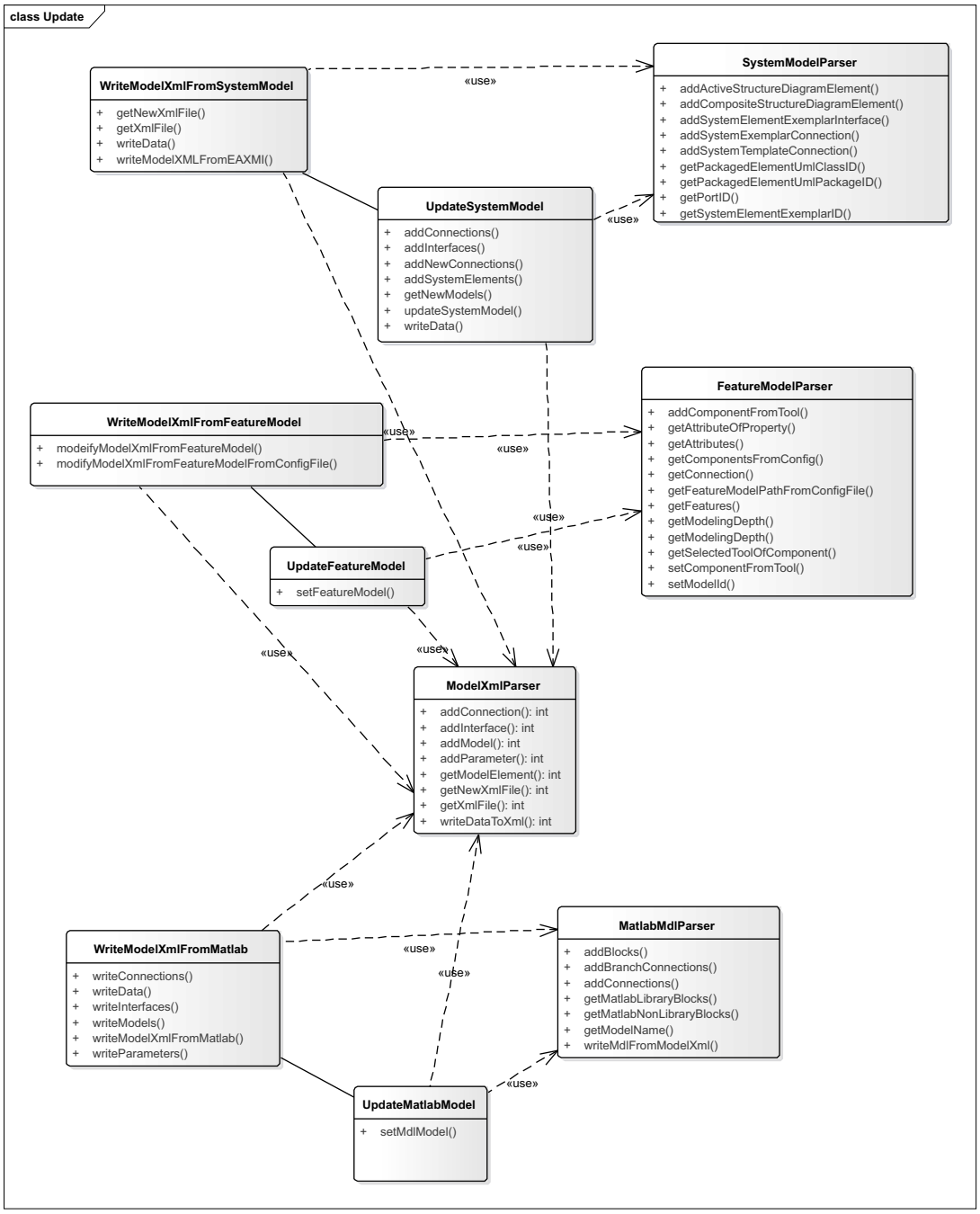


Figure 5-23: Class diagram of update process

Automation integration

In the Figure 5-24, the rubric of automation integration in MMC is depicted. It helps the execution of SiL and HiL to make software and hardware work together. The purpose of the shown window is the design of controllers at simulation level. Here, the simulated dynamic model in Simulink can subsequently be evaluated in an automation environment. Following steps, SiL and HiL, should be pursued subsequently to enable the user to generate real time capable modules of Simulink models. This prototype can be executed on the Twincat3 (The Windows Control and Automation Technology) from Beckhoff with the launch of PC-based control technology[Twi]:

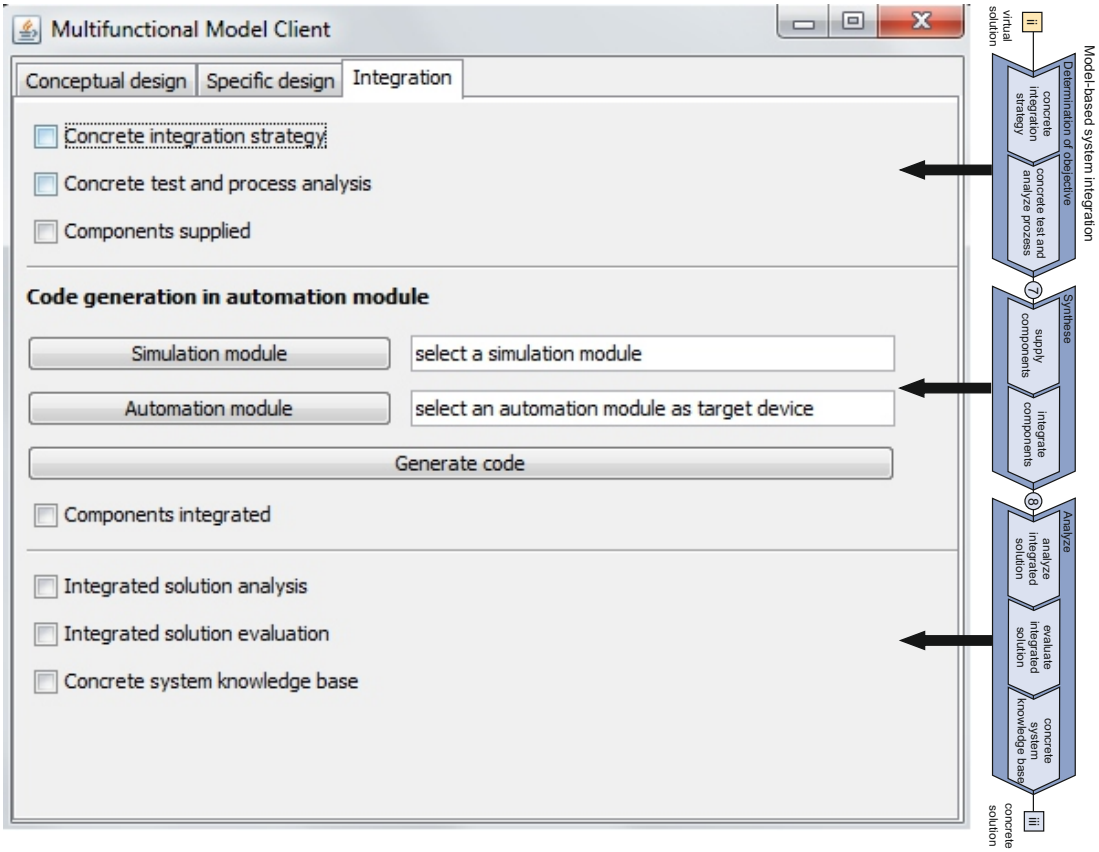


Figure 5-24: GUI of the Multifunctional Model Client: Integration

Software-in-the-Loop:
The SiL is done in following steps by TwinCat as the prototype:

1. Select a Simulink model
2. Select an automation system (Twincat3 or Automation Studio)

3. Select the system target file (TwinCAT.tlc for Twincat)
4. Select the solver type in Simulink : fixed step (to ensure real-time capability of the Simulink model)
5. Build process for generating executable files (generating C++ code and the project file for Visual Studio)
6. Generate a TwinCAT module from Simulink
7. Create Twincat3 project
8. Integrate an instances of the generated module in a TwinCAT3 project
9. Add a module from "Custom Modules-> Simulink generated modules"
10. Create multiple instances of the module created via the context menu of the parent node of the C++ project
11. Find Simulink sample times for the module under the Context-tab
12. Cyclic call of module instances under *System Configuration -> Task Management*
13. Match *Cycle Time* to *Fixed-step size* in Simulink (*Depend on: Task Properties* in Twincat3)

Code generation in automation module
In the second section of Integration-rubric, as shown in Figure **??**, following buttons are presented to realize the SIL:

- Browse a simulation module: select a Simulink model
- Browse an automation module: select an automation system (Twincat3 or Automation Studio)
- Generate code: generate a TwinCAT module from Simulink

Hardware-in-the-Loop:
In the last part of Integration-rubric a check list is performed for HiL tests. To test environment for assurance of properties the integration of HiL is done by TwinCat as the prototype. *The TwinCAT Component Object Model (TcCOM) defines the characteristics and the behavior of the modules. The model derived from the Component Object Model (COM) from Microsoft Windows describes the way in which various independently developed and compiled software components can co-operate with one another. One or several TcCOM modules are consolidated in a driver [Twi].*

Adding existing TcCOM modules to TwinCAT configuration and linking variables of a TcCOM module instance to PLC/IO or other TcCOM modules can be done by

using regular Automation Interface mechanisms and Description file tcm defines interfaces [Twi].

In the Figure 5-25 feature model is transformed to ModelXML and then will be mapped to a part of automation project. The meta model of Programmable Logic Controller (PLC) contains SimulinkObject, Task, Interface, Input and Output. SimulinkObject is a real-time-capable automation module. The class-Task defines the real time properties like cycle time and priority of tasks. For each of the SimulinkObject a task has to be specified. The SimulinkObject is listed with all interfaces defined in Simulink model, which are individually linked to the corresponding PLC variables.

5.5 Summary

In this chapter an implementation prototype of the Multifunctional Model Client according to a classical waterfall approach is introduced. Requirements, design and implementation phase of waterfall approach are tackled in this chapter, test and support phase will be addressed in next chapter. The implementation covered a list of functional and non-functional requirements to configure dynamic behavior models, ensure consistency and access reusable solution knowledge. Also, the design concept is outlined in subchapter 5.3 according to the Figure 5-2. Afterwards the detail of design concept and its functionality is defined by the use of activity diagram, case diagram, class diagram and state machine. Then, the GUI, tasks and functions, covered by the tool, are described. Here, is outlined that the software is implemented in an open source development environment, eclipse [Ecl] and in a bi-directional Java GUI designer, WindowBuilder [Win]. So, the three screen operations of the MMC, conceptual design (Figure 5-14), specific design (Figure 5-21) and integration (Figure **??**) are described according to the V model.

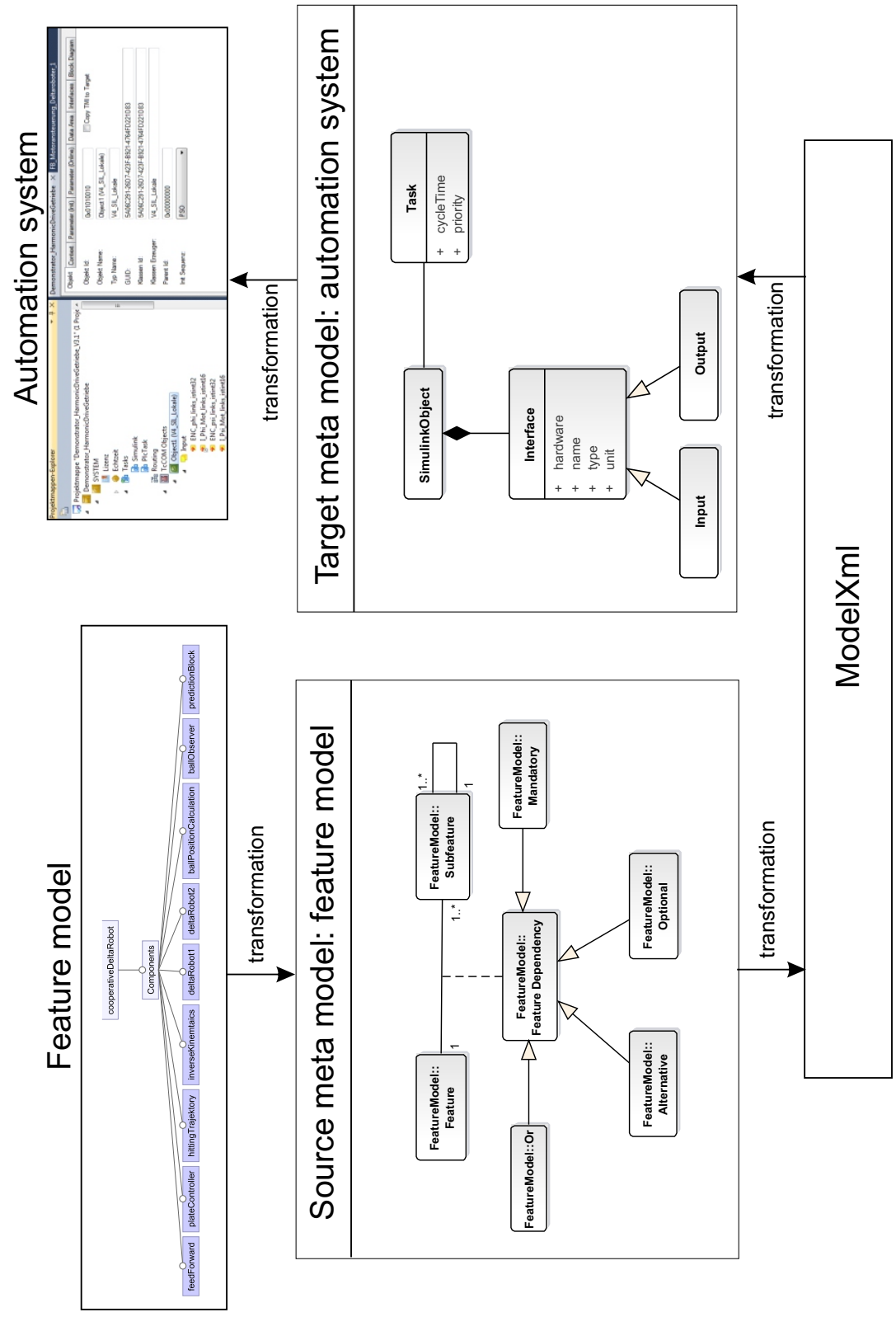


Figure 5-25: Transformation of feature model to automation system

6 Case Study for Validation - Delta Parallel Robots

The design method for developing a modular and reconfigurable mechatronic system is validated with the help of a cooperating delta parallel robots system. In this chapter, the steps and development structure of the delta robots according to the developed methodology is described using MMC. The steps are highly iterative during the process. The flow chart of process provides an important orientation for the developer for synthesis and analysis of each phase.

6.1 Structure of Delta Robots

To illustrate the proposed approach for an integrated model-based design process, an application example of two cooperating delta robots that juggle a ball by passing it to each other is used. The delta robot is a parallel robot, meaning multiple kinematic chains are connected from base to end effector. The use of parallelograms in structure is the essential concept behind the delta parallel robot design. The moving platform is always kept parallel to the fix base by the parallelogram structure, providing only translational motion for moving platform in x-, y-, and z-direction [SJT+15].

A control loop-algorithm for the cooperative ball juggling delta robots without visual guidance is presented. As a substitute for a visual feedback system e.g. cameras, an observer is designed using reflection laws to calculate the states of the ball (position and velocity). Additionally, the next striking velocity, striking position and striking time are predicted. This control loop is independent of any sensor feedback except impact times and it performs well even in the presence of unmodeled parameters. Different simulations and experiments performed on real hardware validate the proposed control loop. Applicability of the designed observer, prediction block and overall loop are shown by experimental results [SJT+15].

The system comprises two identical, autonomous delta robots, which are equipped with a movable racket on their tool-center-points (see Figure 6-1). The rackets consist of tilt kinematics, three piezo force sensors and a racket plate. Since no optical sensors are used to track the ball trajectory, the striking robot has to detect the point of impact using the three sensor signals, predict and communicate the ball flight trajectory by means of a dynamic model. On the one hand, this denotes an ambitious task for control engineers. Due to the limited information feedback, model-based observers and frequent communication of the decentralized control units (e.g. for the gaming strategy or for the prognosis of the next strike)

are needed. On the other hand, measuring the ball impact during the strike requires close collaboration of mechanical, electrical and control engineers. In order to meet the requirements of fast and reliable ball-detection, integration and integrated simulation of the discipline-specific results is essential. In summary, the cooperating delta robots constitute a representative of a new class of intelligent dynamic or cyber-physical systems that lead to new requirements on a systematic design process considering different disciplines. In the following sections, important aspects of the described methodology to design intelligent dynamic systems like the cooperating delta robots will be explained exemplary [OAL+16].

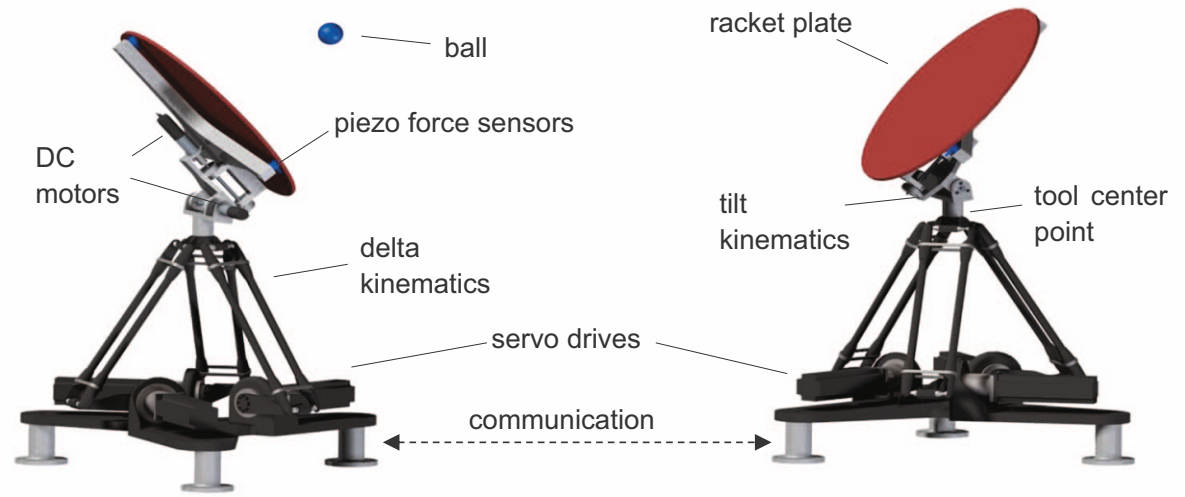


Figure 6-1: Cooperative Delta Robots

6.2 Discipline Spanning Conceptual Design

The development begins with the phase of the interdisciplinary system design. In the following subsections, the system design of delta parallel robots divided into the subordinate steps of the determination of objective, synthesis and analysis.

6.2.1 Determination of Objective

The first step in the system design phase is to specify the tasks and boundary conditions clearly. For this purpose, environmental model, application scenarios, requirements and functions are first prepared in the course of the planning and analysis of the task.

Environmental model:
An environment model is used to describe the interrelationships of delta robots

with surrounding systems. This structure model is shown in Figure 6-2. It includes the operative relationships between the robot 1 and its environment modeled by the surrounding elements as user, underground, environment, game object, energy source as well as robot 2. Relationships are presented using energy, material and information flows, which can be both disruptive and functionally beneficial [GTS14b].

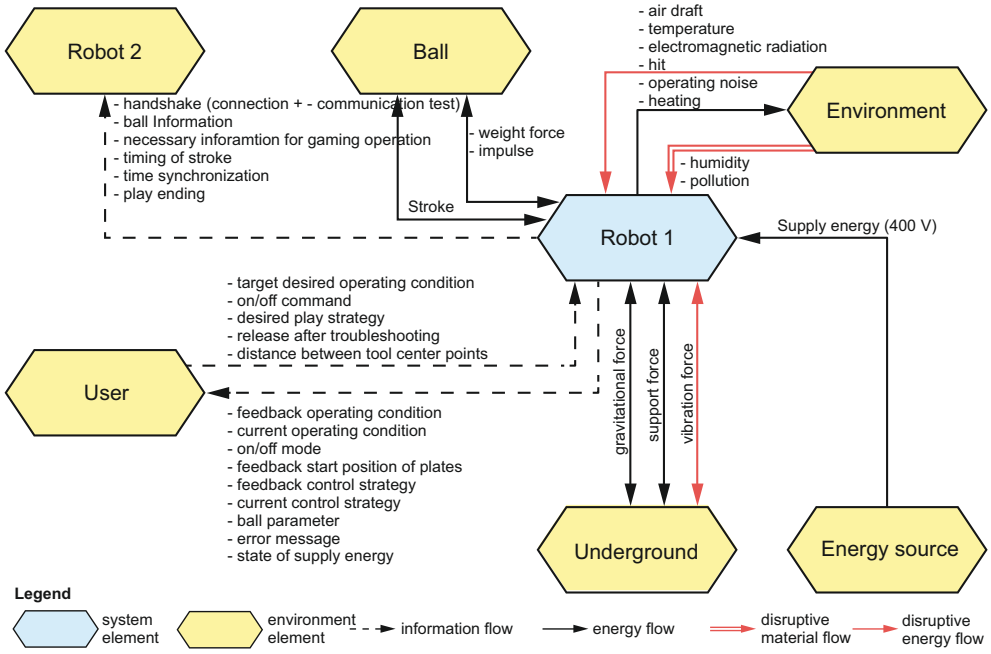


Figure 6-2: Environment Diagram of Cooperative Delta Robots

Application scenarios:
For the cooperating delta robots, eight different application scenarios are formulated, briefly: Initialization, start of play, successful pass request, incorrect pass request, successful play, faulty play, end of play and defect. For example, Table 6-3 shows the application scenario of successful play in a desired sequence [GTS14b].

Requirements:
Based on the first two partial models, additional requirements can be systematically derived. A distinction is made between demand requirements and wish requirements. This classification is indicated in the requirements list by the labels D (demand) and W (wish), as shown in Table 6-1.

In the delta robots, they are grouped into a list and broken down into categories such as geometry, controlled movement behavior, energy, security etc., see Table 6-1. In the course of the development process, the list is further and further detailed and supplemented by [GTS14b].

AW: 5	Application scenario: successful play	19. Sep. 2017

Description of situation
The initialization of the robots was successful and both are in the Play state.

Description of the principal behavior
A robot waits for the setpoint parameters from the other robot and receives them. The robot calculates the parameters for its own stroke. The ball is hit by these values to the other robot, the hitting robot registering the impact force during the stroke and equalizing the value with the target value. The target actual adjustment has the effect that new target impact parameters are defined taking into account the game strategy. These target parameters, in turn, are sent to the other robot as described at the outset. The sequence described is repeated.

Sketch

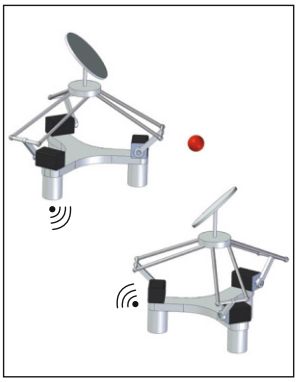


Figure 6-3: Application Scenario of Cooperative Delta Robots

State: 19.09.2017		Requirements list				Page 1	Sheet1	
Nr.	Demand Wish	Requirements of delta robots	Numeric value			Unit	Change	Creator
			Min.	Exact	Max.			
1		**Geometry**						
1.1	W	small size						
1.3	W	simple structure						
...								
4.		**Energy**						
4.1	W	Use of electrical energy						
...								
7.		**System**						
7.2	W	light weight						
7.3	D	no floor mounting necessary						
7.4	W	high stiffness of robots						
...								
8.		**Kinematics**						
8.1	D	rackets movement in 3 directions						
...								

Table 6-1: Requirements of Cooperative Delta Robots

Functions:
The main function of the delta robot is to hit an object. The hitting function can be broken down into control of the movement path, generation of impact force, calculation of the impact path and determination of the impact force. For these functions, solutions are searched for. If no suitable solutions are found, they are further subdivided. This is done until a solution can be found for each function. This is shown in Figure 6-4.

6.2.2 Synthesis

Solution elements associated with the solution patterns or further specific and/or more concrete solution patterns can now be selected in the synthesis step. For this

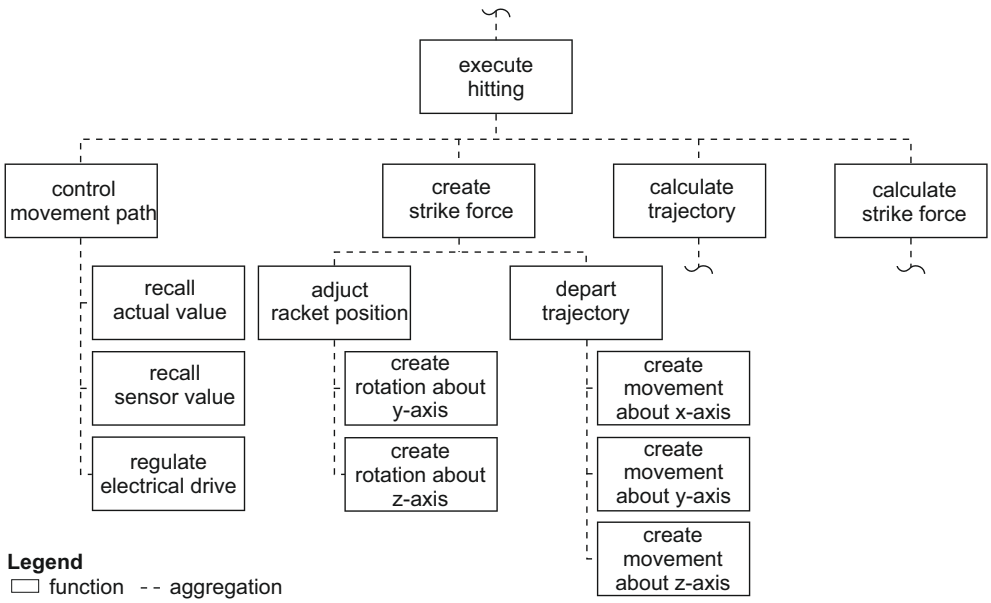


Figure 6-4: Function Hierarchy of Cooperative Delta Robots

purpose, active structure, feature model, shape, behavior and activity diagram of delta robots are illustrated.

Active structure:
The connections of the system elements within the active structure can be explained using the example of the drive unit. The drive unit consists of three servo drives, three levers and three legs, which are connected to the racket. The servo drives are controlled by the information processing. This relationship becomes clear by the information flow setpoint manipulated variable. By means of the drive unit, the subfunctions movement in x-direction, y-direction and z-direction are generated.

Figure 6-5 shows the active structure of the cooperating delta robot system. Each delta robot consists of a delta kinematics, which is moved by three rotary servo drives. On the TCP of the delta kinematics, a racket with a tilting kinematics is mounted in order to make the racket plate able to turn in two axes. This is the task of two DC drives. Under the racket plate, there are three piezo-force sensors, with the aid of which the sensing of the impact position takes place (ball sensing). On the basis of this information, a model-based observer should continuously estimate the ball position. The knowledge of the position is a prerequisite for the control and regulation of the striking [Oes17]. According to the specification of the active structure, it is necessary to complete, analyze and evaluate the partial models of the solution variants by means of behavioral and shape models. The coherent partial models form a cross-disciplinary system model, which can also be represented by SysML diagrams using the CONSENS profile [IKD+13].

For the development of two cooperating delta robots the structure of active principles is shown in Figure 6-5 to utilize partial models. Whereas both robots are built exactly the same, only one of them is shown. The structure of active principles discloses, *e.g. the connection between the system elements racket and deltaKinematics to be a Mechanical 3D-connection, which constitutes an energy flow (EnergyFlowSpecification). That implies the exchange of translational, as well as rotational movement between the two system elements [OAL+16].*

Actuators convert energy into motion in order to manage or move mechanisms, while electrical actuators provide mechanical torque for powering systems. The result for both is high torque at lower shaft speeds or RPM.

Information processing can be designed by using the plant model. A controller and the ball observer is designed for the ball projectile in the conception phase to determine the designed functional principle. Interdisciplinary character of intelligent technical systems demands dynamic properties. Because they are influenced by several disciplines basic functionality of system dynamics are designed in early stage of development in the concept phase [Oes17]. By use of feature model, the user knows which combination of submodels exists to select.

To assess the feasibility of the cooperating delta robots the simulation in particular is applied. A suitable ratio of the plate mass and the playing object is obtained based on the the necessity of high acceleration of ball and detection of impact position by means of force sensors. to reduce the modeling effort the system information processing can be characterized by coupling between event-driven and continuous behavior through modeling the submodels [Oes17].

Feature model:
The feature model of delta robots is exported from the active structure as explained in previous chapter. The feature model is demonstrated in Figure 6-6. The feature model consists of two parts: The first part depicts the models and interfaces of the delta robot and second part represents its components, such as racket, delta kinematics etc.

The properties of components and their interfaces are defined in the features of the feature models. For example, component *drive1* has following property:
String modelId = ”SEMDrive10102016132430”;
Interface *drive1_referencePosition* has following properties:
String unit = ”degree”;
String type = ”RealSignalInput”;
String sourceInterface = ”informationProcessing_refrencePosition1”;

These properties are derived from the active structure. Here, there is still no selection possibility for the model/models of components. Only after connection to the knowledge base by IDs tracking is determined, if s a model of the component exists

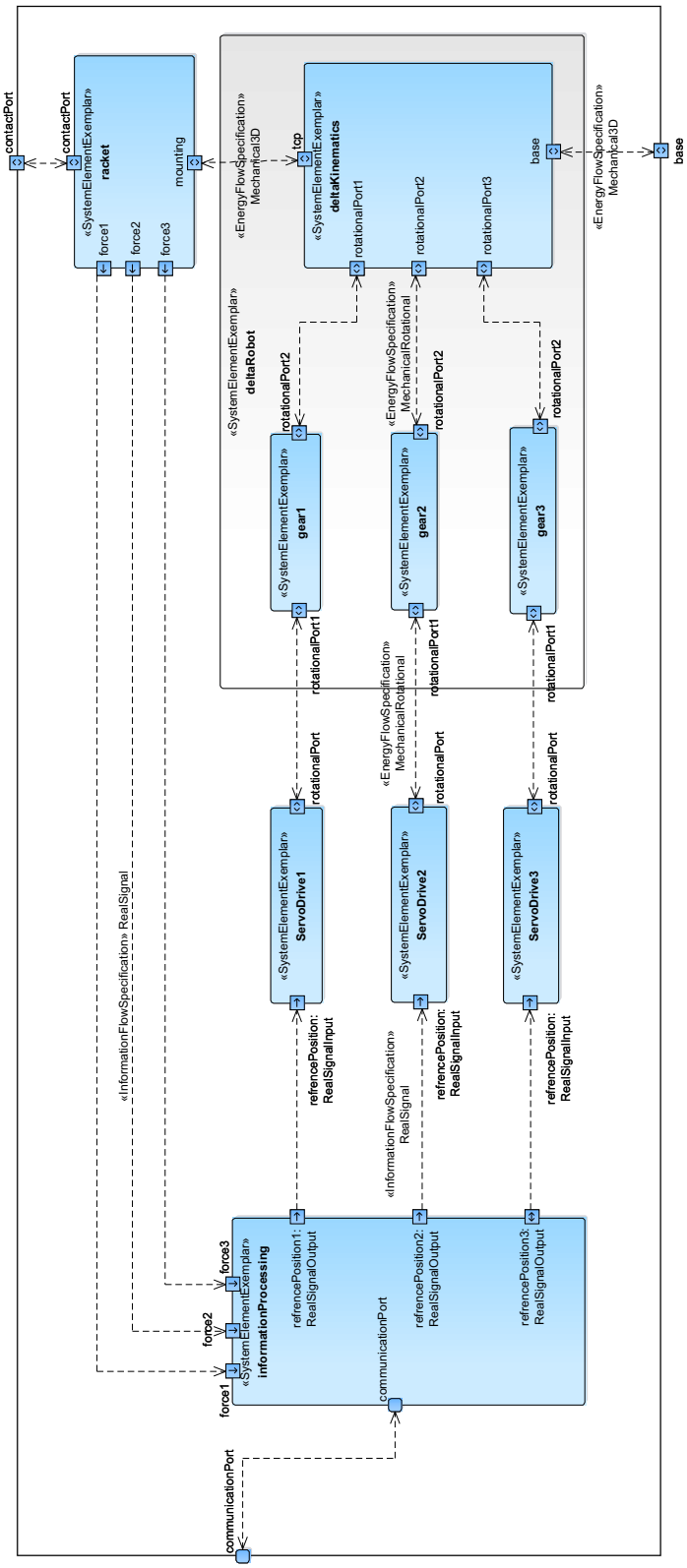


Figure 6-5: Active Structure of the Robot of Cooperative Delta Robots

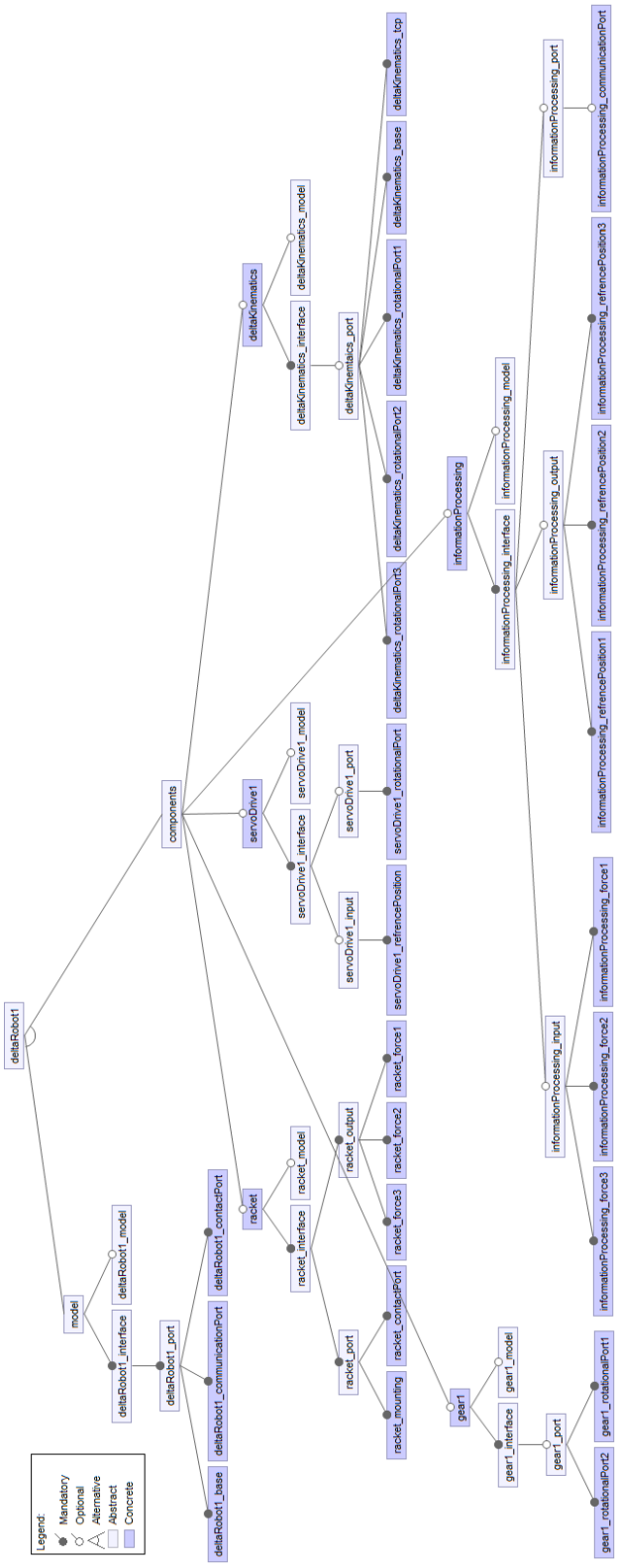


Figure 6-6: Feature model of one of the robots

or not. The system model consists of racket, delta kinematics, gear, drive and information processing. Among them, racket, delta kinematics, drive and gear are recorded with an id. It means these components are derived from the system knowledge and solution knowledge by MMC. Based on this relation, the feature model is extended by existing DBM of components, as shown in Figure 6-7. It is expanded by `deltaRobot_tools`, included `deltaRobot_matlab`, `deltaRobot_dymola` and `deltaRobot_recurdyn`. From the expanded part following issues are specified:

- There is a model in Dymola with modeling depth II.
- Dymola model are included some interfaces (ports and outputs).
- There is a model in Recurdyn with modeling depth III.
- Recurdyn model are included some interfaces (inputs and outputs).

Based on the integrated simulation, the user is then able to verify the configuration of the delta robot.

Shape: In accordance with the system elements and flow relations described in the operating structure, the shape model is created. The modeled servo drives are connected to a bar construction. On the connection point is the racket plate. A floor mounting is not provided according to the predefined requirements. The shape is designed in such a way that a high stiffness can be achieved with a low weight. Figure 6-8 shows the CAD model of the shape of the delta robot in an early phase [GTS14b].

Behavior: The abstract operating states of the delta robot are shown. When the robot is switched on by the user, the robot first carries out the initialization, which checks whether the system is functioning. If the initialization is terminated successfully, the robot changes to the standby state. If, however, an error is detected, the robot changes to the defect state. The robot can change from any state to this state. In standby, the robot remains where it is until another event is triggered. If, for example, the robot detects that a playing object is inserted, the robot changes to the play master state. The slave is then assigned to the second robot by the master. The second robot thus switches from standby to the slave state. The two states are similar. However, the master, for example, forwards any strategy changes to the slave or communicates the end of the game. After the game has finished, both delta robots return to the standby state. Figure 6-9 shows an abstracted version of the partial model behavioral states of the delta robot [GTS14b].

Activity diagram: Figure 6-10 shows the activities of the standby state [GTS14b]. If a play object is thrown in, the upper activity line (ball is thrown in) is to be traversed: first of all, the play object must be recognized. Subsequently, the play strategy is requested in parallel and the other robot communicates with the

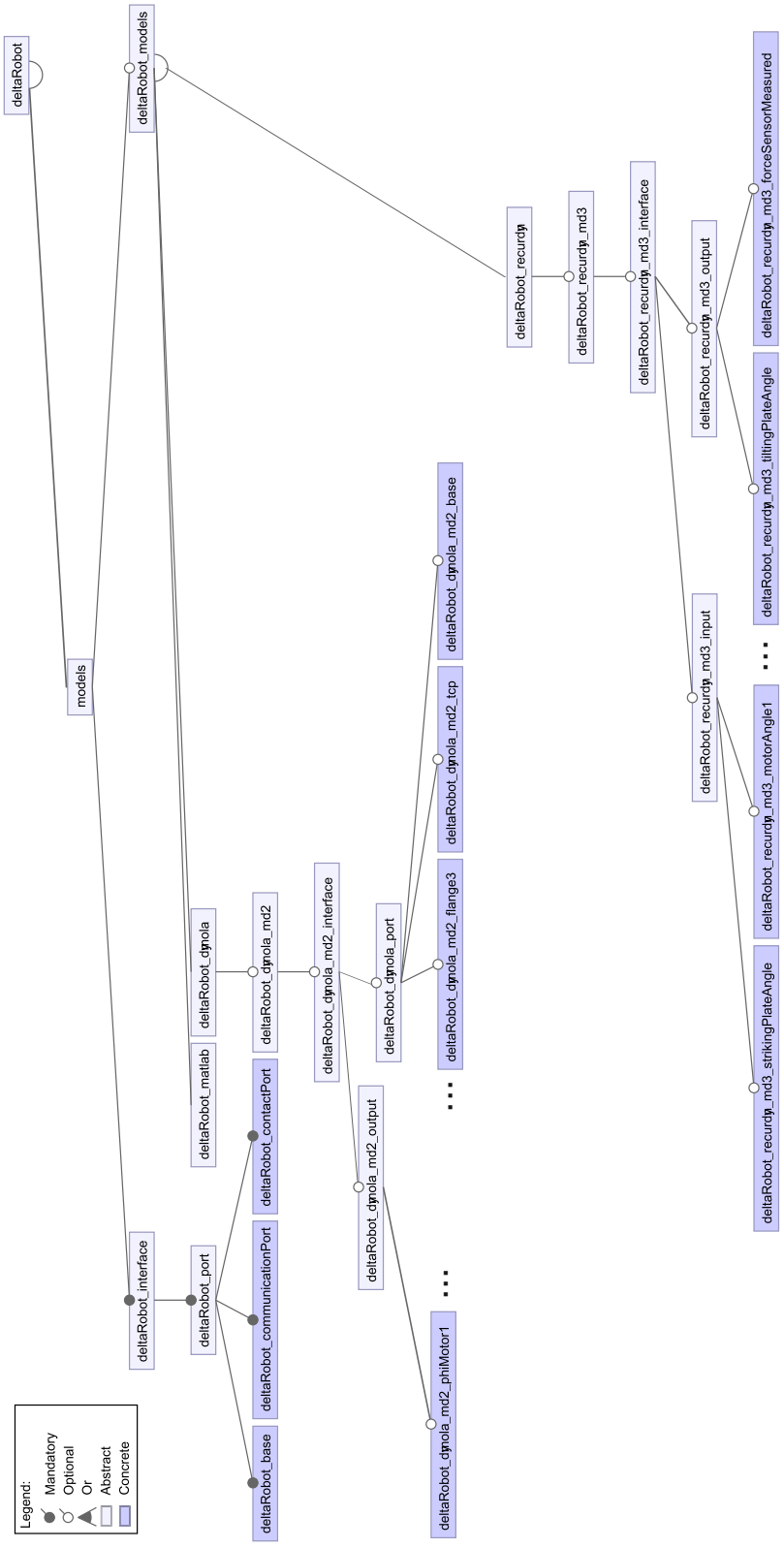


Figure 6-7: Feature model of the cooperative delta robot

Figure 6-8: Shape of delta robot

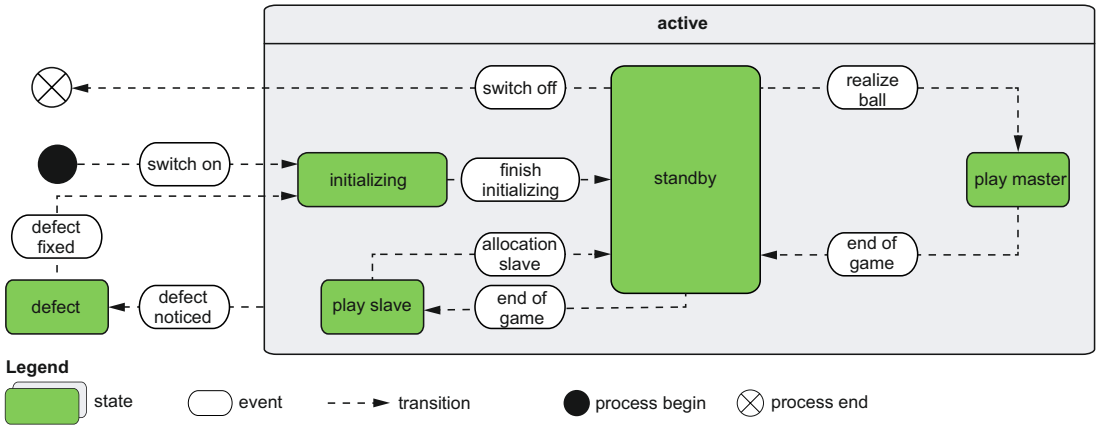


Figure 6-9: Abstract state diagram of delta robot

slave state. After receiving the game strategy, it is transmitted to the other robot. If the slave is assigned the slave role, the lower activity is executed (Information from other robot).

6.2.3 Analyze

In the idealized dynamics model suitable parameter can be determined and tested in many iteration. For example, here, the desired trajectory of the ball and the robots distance are provided during the conceptual phase of system design. In addition, a feasibility analysis is done for the detection of the impact position by means of force sensors and the necessary masses and stiffness ratios are determined.

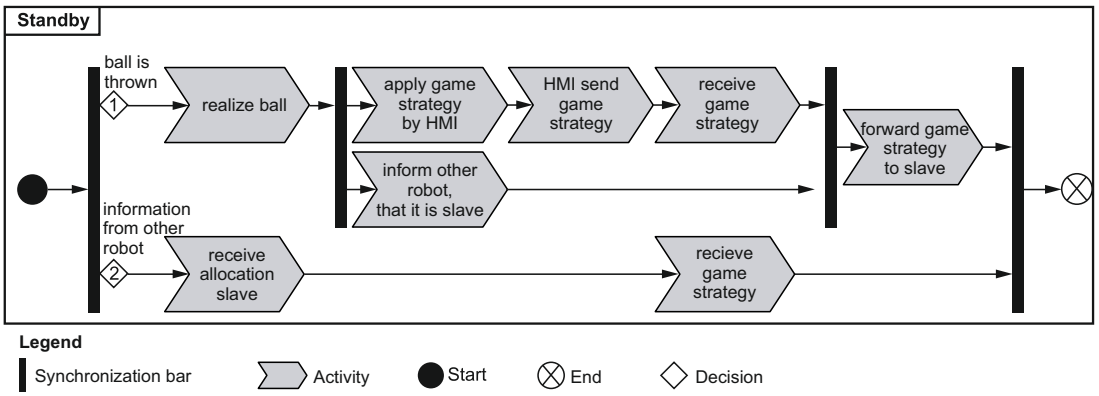


Figure 6-10: Activity diagram of Standby-state

In this phase, the feature model is used in order to give the user the opportunity to select a possible combination of models from a choice tree to build the principle level by means of idealized dynamics models (with a low level of detail), as shown in Figure 6-11 left) [OAL+16]. In the configuration tree, a selection between models of delta robot and the components group, has to be made.

Here, to create different scenarios of simulation model an access to instance of solution knowledge is created to operate on an object of it. Then assembling the desired DBM is executed as scenarios below:

Scenario one: simulation of delta robot in Dymola
In the configuration tree (as shown in Figure 6-11 right) Dymola-option is selected. Due to this selection other option of components are deactivated. According to this selection, a dynamic model of delta robot in Dymola is created (see Figure 6-12).

Scenario two: analyze the detail behavior of delta robot
The servo drive from Matlab and the delta kinematics from Recurdyn in configuration tree are selected. The result is shown in Figure 6-13. The following tasks are done by the MMC:

1. Servo drive is instantiated from the library of Simulink.
2. Delta robot with the selected interfaces is instantiated from the library of Recurdyn.
3. A S-Function of delta robot is created in Matlab/Simulink.
4. The servo drive and delta robot are connected to each other in Matlab/Simulink.
5. A memory-block is created to integrate a step delay for actual angle of the servo drive. Actually, for every motor a memory-block is considered, i.e. it is created automatically with the motor-block to set the initial condition.

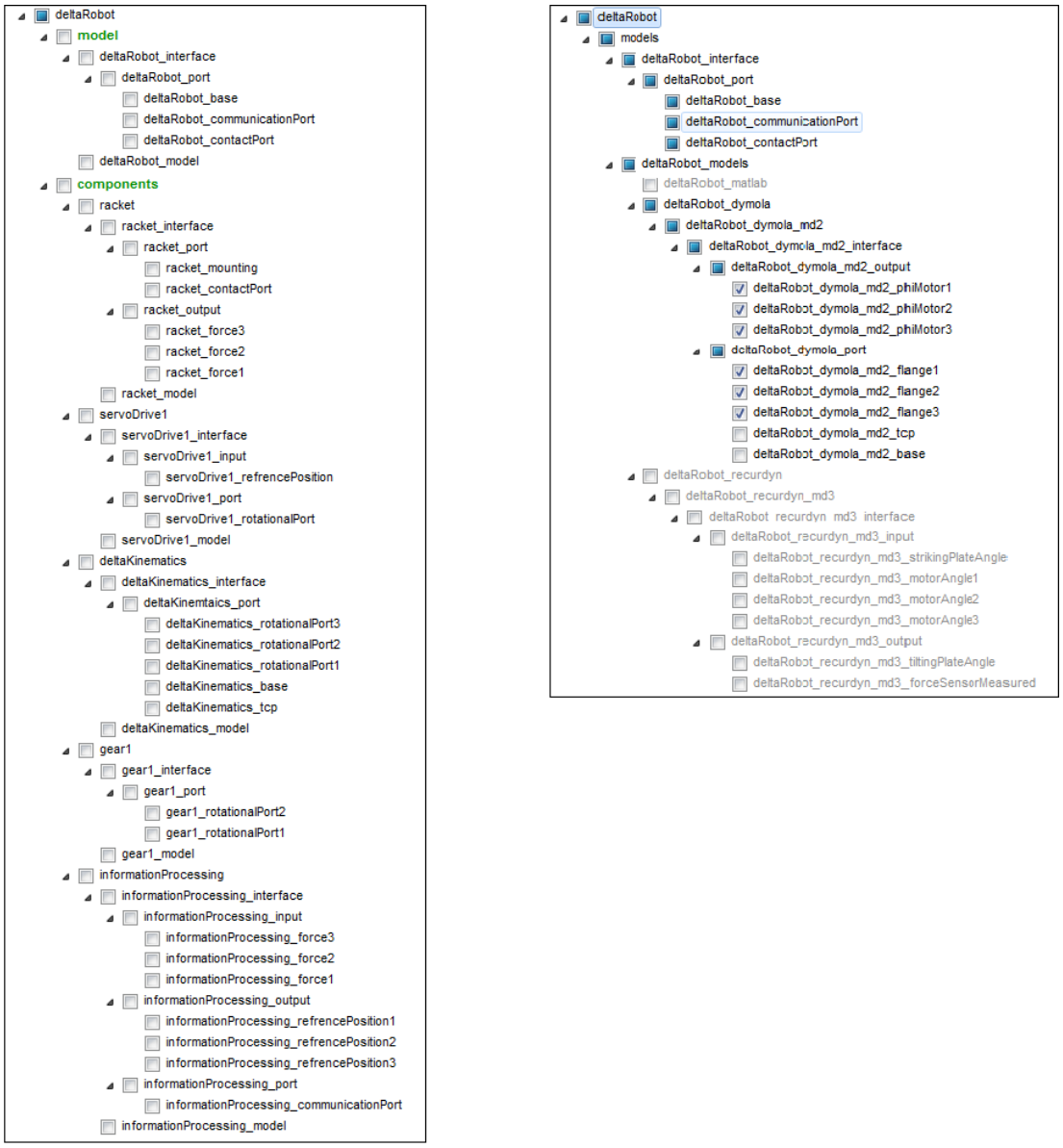


Figure 6-11: Left: Configuration tree of the delta Robot of Cooperative Delta Robots Right: One configuration of delta robot

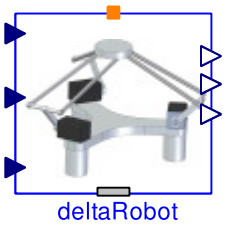


Figure 6-12: Dymola model of delta Robot

6. Signal connection between memory block, actual angle of delta robot and actual angle of servo drive is created.

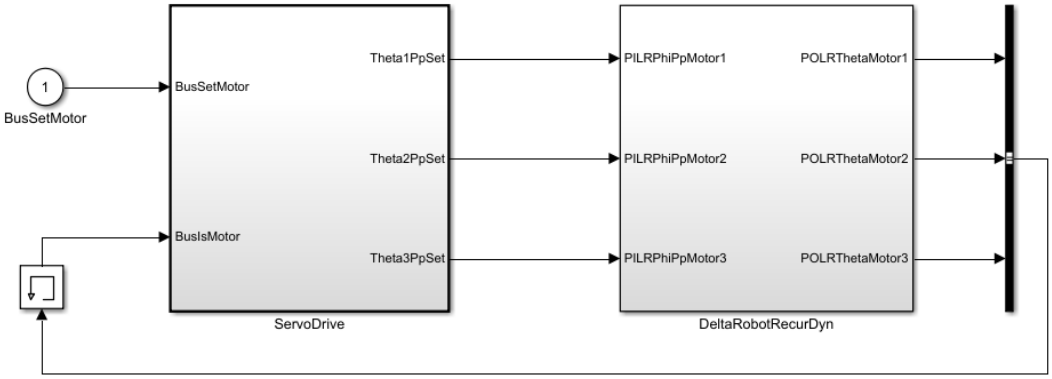


Figure 6-13: Simulink model of servo drive in Simulink and delta robot in Recurdyn

A servo drive monitors the feedback signal from the servomechanism and continually adjusts for deviation from expected behavior. In this scenario, as shown in Figure 6-13, there is no controller in the servo drive model. The task of this block is precisely control of velocity and acceleration. Here, the following changes must be subsequently followed to expand the servo drive model by a PI controller:

1. update solution knowledge (custom library of Matlab/Simulink) by a PI controller
2. update the system knowledge by a PI controller
3. update active structure. According to the updates, PI controller is used instead of drive.
4. update feature model. A motor controller-option is added to the feature model. The Figure 6-14 shows the configuration tree respectively.

Scenario three: characterize the model with more detail
In Figure 6-15 all components, servo drive, delta kinematics and racket in Dymola

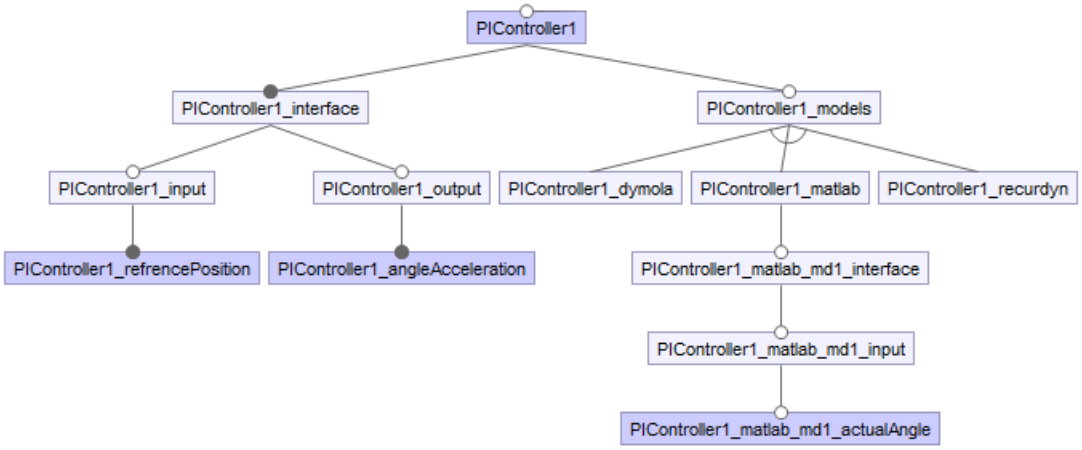


Figure 6-14: Feature model of motor controller

are selected. To pursue this target, a gear with one drive and one driven shaft will be connected to the servomotor and delta kinematics by user to increase torque and performance while reducing speed, as shown in Figure 6-16. According to the new model by updating the feature model in MMC, it will be expanded by the gear.

The dependency between the submodels of its subcomponents is recorded as a constraint that depicts the connections of system elements in the active structure. The desired integrated model is built up by mapping the necessary interfaces based on the valid configuration selected. A configuration is valid if the combination of features is permitted by the feature model (i.e. it fulfills the semantics of groups and all cross-tree constraints). Similarly, different modeling depths of these components can be selected.

In iterative analysis the model parameters are roughly interpreted until the required functionality is achieved. Now there are concrete requirements for the servo drives (torque, speed, acceleration) and the masses of the mechanical bodies. After evaluation of system's functionality the system is validated as principle solution, where responsibilities and sequences are determined [Oes17].

Scenario four: structuring interfaces based on feature modeling
In this scenario is explained the algorithm of DBM generation from SysML. Furthermore, is outlined how submodels interfaces are matching together according to feature model to connect DBM submodels.

MMC matches in particular interface mismatching between SysML and DBM. Also all port (Bi-directional flow specification) properties are synchronized by MMC. It generates a DBM from SysML according to following algorithm:

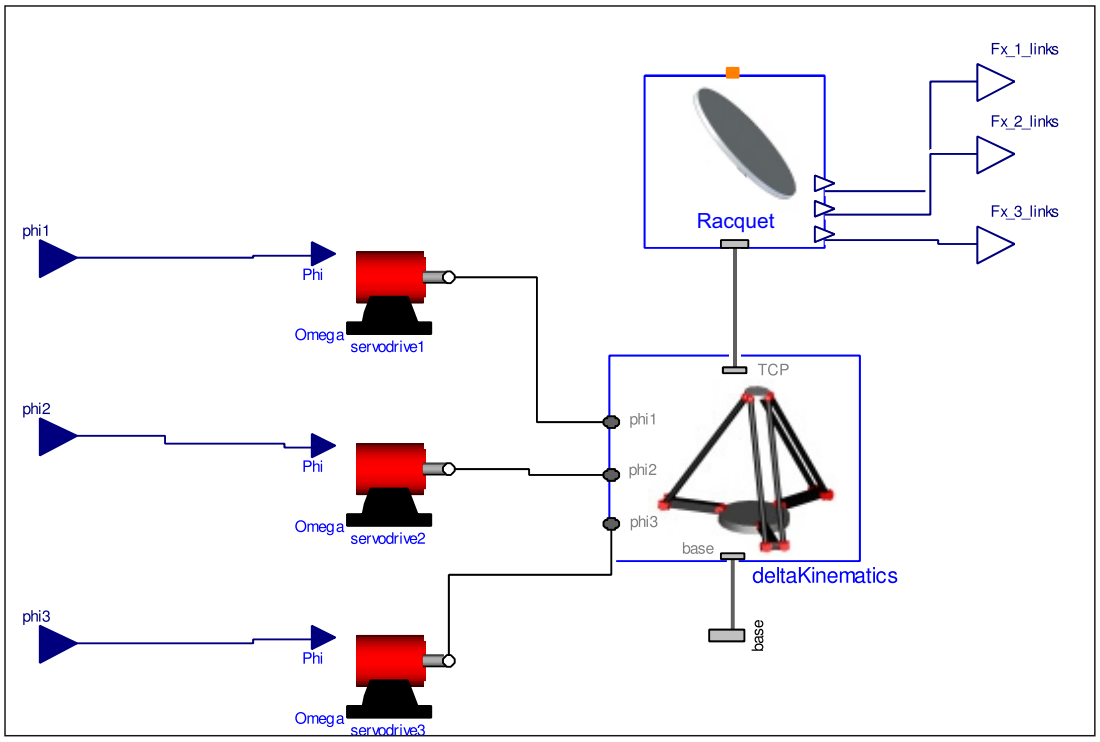


Figure 6-15: Simulink model of servo drive and delta kinematics

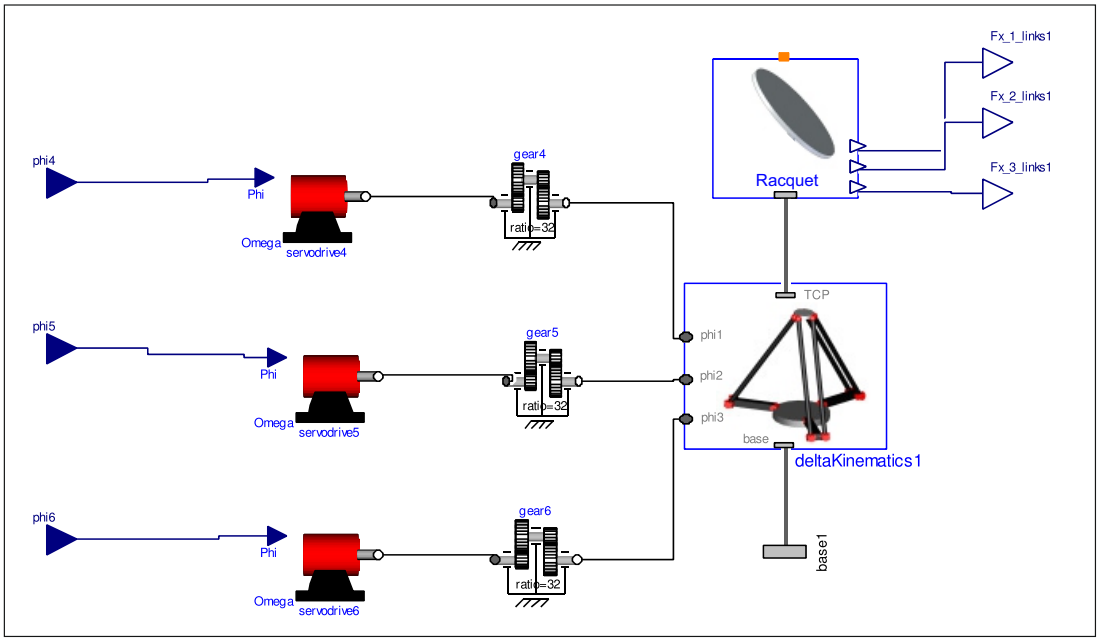


Figure 6-16: Simulink model of servo drive, gear and delta kinematics

1. Analyze the system model to identify components and their interfaces
2. Generate a configuration tree of interfaces
3. Generate a DBM in different tools based on the user's selection
4. Modify DBM in selected tools
5. Generate the FMU model description file
6. Compile and import FMU model

Each used interface in the active structure of delta robots is an instance of a flow specification from the diagram shown in Figure 3-8. In Figure 6-17, the features of an excerpt of the involved system elements are given from the feature model.

Figure 6-17 shows the top-level feature diagram, where each subfeature represents a system element with major points of interface variation. Their description is presented bellow.

This feature model is used by MMC in order to give the user the opportunity to select a possible combination of models from a configuration tree, as shown in Figure 6-18. According to the integrated simulation, the user is then able to verify the configuration of the racket and delta kinematics. The desired integrated model is built up by mapping the necessary interfaces based on the selected valid configuration. A configuration is valid, if the combination of features is allowed by the feature model (i.e. it fulfills the semantics of groups and all cross-tree constraints) [TKB+14].

According to the selection in Figure 6-18, one section of the physical model can be created in a signal flow tool and the part of the constructive geometry in another specialized simulation tool. In order to join them, an appropriate connection is required, because of structural differences in the tools and the models. Hence, all relevant properties such as inputs, outputs, ports and parameters of a solution element are illustrated with type and unit in the feature model. By processing information in the feature model, the adapters and unit converters required for the interfaces are determined in order to connect the submodels.

In the delta robot example, the configuration of DBM is limited to Matlab/Simulink and Dymola. To become explicit about interface configuration, different simulation tools are purposefully selected. The delta kinematics are modeled in Dymola and the racket is modeled in Matlab/Simulink. The inputs, outputs, and ports of the block are set according to the top level of ports in the configuration tree. Their data type is set to the corresponding flow property of flow specification, derived from the system model. The name of inputs and outputs is set to the name of flow properties. The first step for ports illustration in the feature model is detection of their variables, as shown in Table 6-2. Bases on the presented algorithm, in

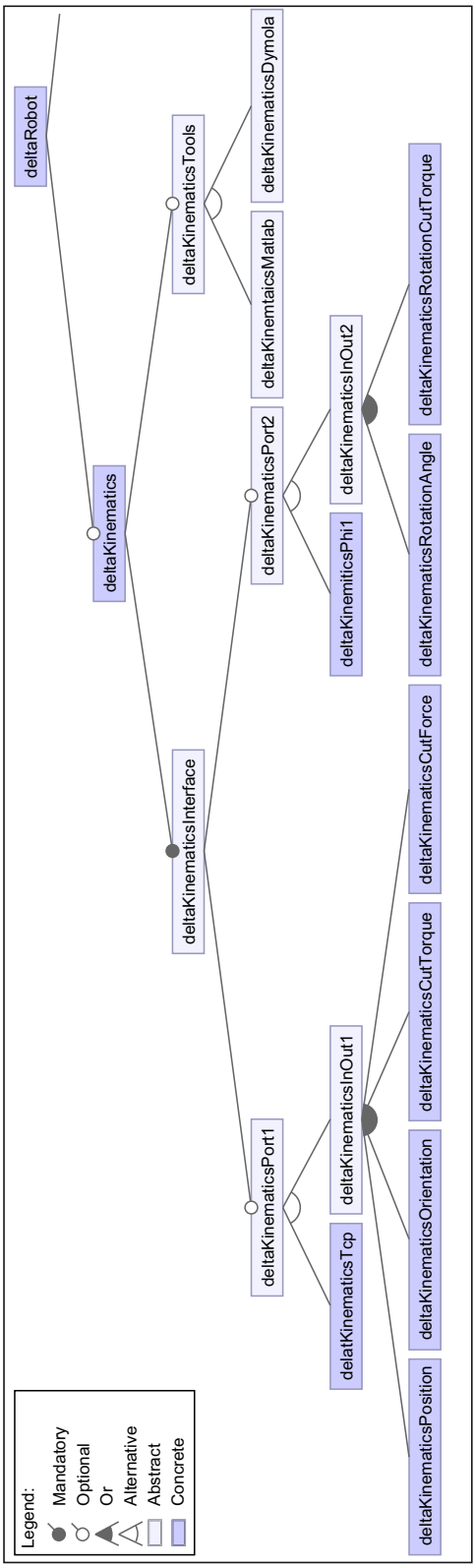


Figure 6-17: Excerpt of feature model based on active structure

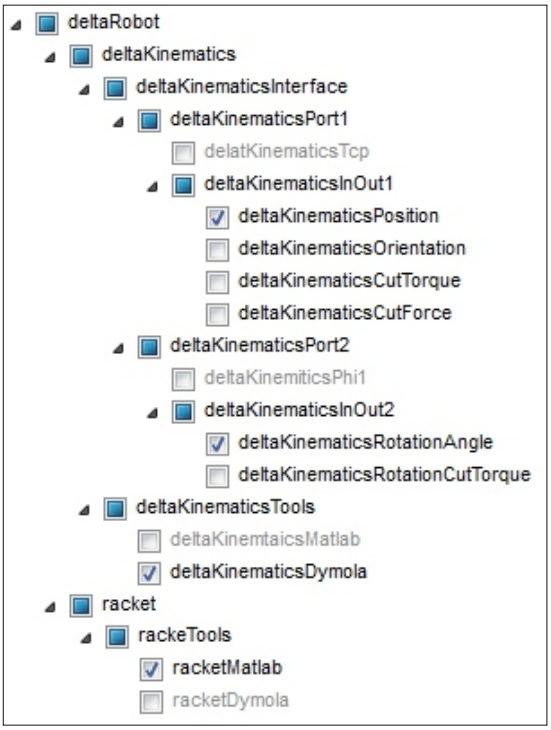


Figure 6-18: Excerpt of one of the valid configuration

the next step, the creation of a FMU model from the delta kinematics (Dymola model) to enable the importing of submodels to Matlab/Simulink is required. Since bi-directional flow ports are not supported in the FMI standard, variables called inputs and outputs are needed to be created from the standard ports. Accordingly, all variables of a port are listed by MMC and can be selected as input or output of the model (Figure 6-18). Thereby a product-line is defined where each element contains interfaces and tools, where interfaces from the type port is an alternative feature. In other words, only one of *deltaKinematicsTCP* (port), and *deltaKinematicsInOut* (in/output) can be presented in a model. The TCP-port consists of the following subfeatures variables: position, orientation, cut torque, and cut force. In Figure 6-19 right, the delta kinematics model created by MMC in Dymola with the correspondent inputs and outputs is shown. These new interfaces need to be connected to ports in Dymola by appropriate sensors and adapters, like position sensors, as shown in Figure 6-19 right.

For this purpose the MMC looks for appropriate sensors and adapters in the standard library i.e. the solution knowledge, in order for a given interface to be able communicate with other elements. If an appropriate adapter is found in the library, it is added to the model. The MMC also performs a check on the units

Table 6-2: Setup for TCP-port of delta kinematics, which is connected to mounting-port of racket

TCP	Unit	Type	Connected element
Position	meter	real output	racket-mounting
Orientation	-	boolean output	racket-mounting
Cut force	newton	real output	racket-mounting
Cut torque	newton.meter	real output	racket-mounting

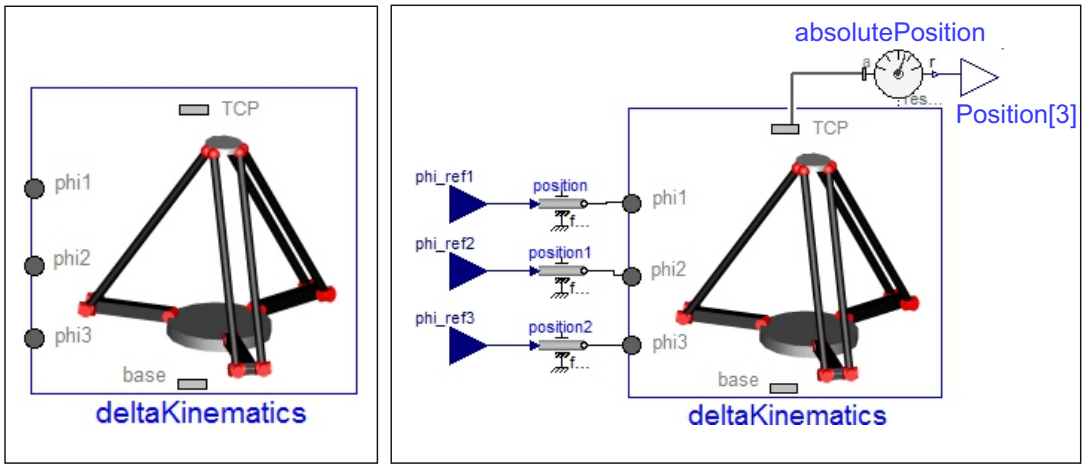


Figure 6-19: Left: delta kinematics model in Dymola
Right: delta kinematics model in Dymola created by MMC

between interfaces. Further in the case of a unit inconsistency, matching SI units and possibly recalculation could be executed [AOT15].

6.3 Discipline Specific Design

In this section, the parallel elaboration of the system takes place in the individual disciplines after the milestone principle solution. However, the disciplinary system model will be kept up-to-date. In the following subsections, the different disciplines are divided into the subordinate steps of the determination of objective, synthesis and analysis.

6.3.1 Determination of Objective

In the early analysis phase is possible to determine the contact process and contact partner behavior that influence strongly the system behavior. Therefore, in this step to select a suitable ball type different ball instance are tested. The target is to find the ball type, which testing is worked out as quickly as possible. Here, to complete requirements and determine test concept the modeling objectives are prepared for behavioral models in detail. Based on the developed test concept, level of detail and complexity of model is defined. For completion of requirements and determination of test concepts is necessary to know how is the identification and validation later [Oes17].

For that, according to the contact process and different type of balls the requirements list is extended by new requirements, as shown in Figure 6-20.

State: 19.09.2017		Requirements list				Page 1	Sheet1	
Nr.	Demand Wish	Requirements of delta robots	Numeric value			Unit	Change	Creator
			Min.	Exact	Max.			
1		**Geometry**						
1.1	W	small size						
1.2	D	dimension			1,5x1,5x1,5	m		
1.3	W	simple structure						
…								
4.		**Energy**						
4.1	W	Use of electrical energy						
4.2	D	Temperature range	-20		60	°C		
4.3	W	Max energy supplier			230	V		
…								
7.		**System**						
7.1	D	weight			60	kg		
7.2	W	light weight						
7.3	D	no floor mounting necessary						
7.4	W	high stiffness of robots						
7.5	D	no change in the position or orientation of the frame design of the system during operation						
7.6	W	minimize friction forces						
7.7	D	light ground vibrations must not interfere with the system						
8.		**Kinematics**						
8.1	D	rackets movement in 3 directions						
8.2	D	sensitive force control						
…								

Figure 6-20: Requirements of cooperative delta robots

6.3.2 Synthesis

From the point of view of continuous and event-discrete signals the developed structure of the cooperating delta robots is shown in Figure 6-21. A required flight path and plate actuator are calculated at first for the servo drive controller based on the required flying height and the predicted impact speed. Here, tow controllers are needed For the tilting, rotation and hitting speed of the plate. Thereby desired height of the ball and hitting the middle point of the plate are important to reach [Oes17].

This is based on value comparison the actual/set velocities and observed velocities of the preceding hitting. to adjust the tilting and rotation of the racket the predicted impact position is used. In information processing of the second robot the precalculation and the estimated velocities are calculated on the previous hitting. The measured motor angles as well as the sensor signals of the plate force sensors are used by ball observer. In the ball position calculation block the necessary parameters are calculated for the ball observer. To estimate the ball speed of the ball observer block uses pulse balances after impact [Oes17].

In the control loop the reference height of the ball is input with respect to the global coordinate system. As the distance between the robots is already known (d = 1,5m) and the incoming velocity components are also known, the velocity and angle of the striking plate can be calculated using the projectile motion equations [SJT+15].

After providing the Simulink model of delta robots a feature model is derived according to the Simulink model. An excerpt of the entire feature model is shown in Figure 6-22, as some components, their interfaces and connection between them. *RobotPlant*, *DeltaRobot*, *BallPoaitionCalculation* and *BallObserver* are depicted in details in the Figure 6-23 and Figure 6-24. The ports are divided in inputs and outputs to provide the possibility to select them separately. In this way, ports are more clarified to understand the dynamic model and to connect the blocks to each other if it is necessary.

According to the feature model an active structure can be created, as shown simplified in the Figure 6-21. Here is to see the delta robots with all system elements and their interfaces as a big system with the control loop of the ball juggling. All these system elements can be stored in system knowledge to be reused for other system analysis and other system designs. Also flow specification can be expanded according to the new interfaces, like position and prognosis of ball. In this regard and according to the Figure 6-22 the use of MMC is elucidated in details by the following scenarios.

Scenario one: a dynamic model of the entire system in Matlab/Simulink
As shown in Figure 6-23, a connection is needed between *RobotPlant* and *BallPositionCalculation* to make the simulation possible. In the *Robot Plant* block TCP of Delta robot is calculated by forward kinematics of parallel robot using the angle information of three base joints. In the *BallPositionCalculation* block, position of the ball in local and global coordinates is calculated using the information of base motors angles and the impact forces. In the *Connector* block between hardware and model can be switched and *RobotPlant* and *BallPositionCalculation* are connected.

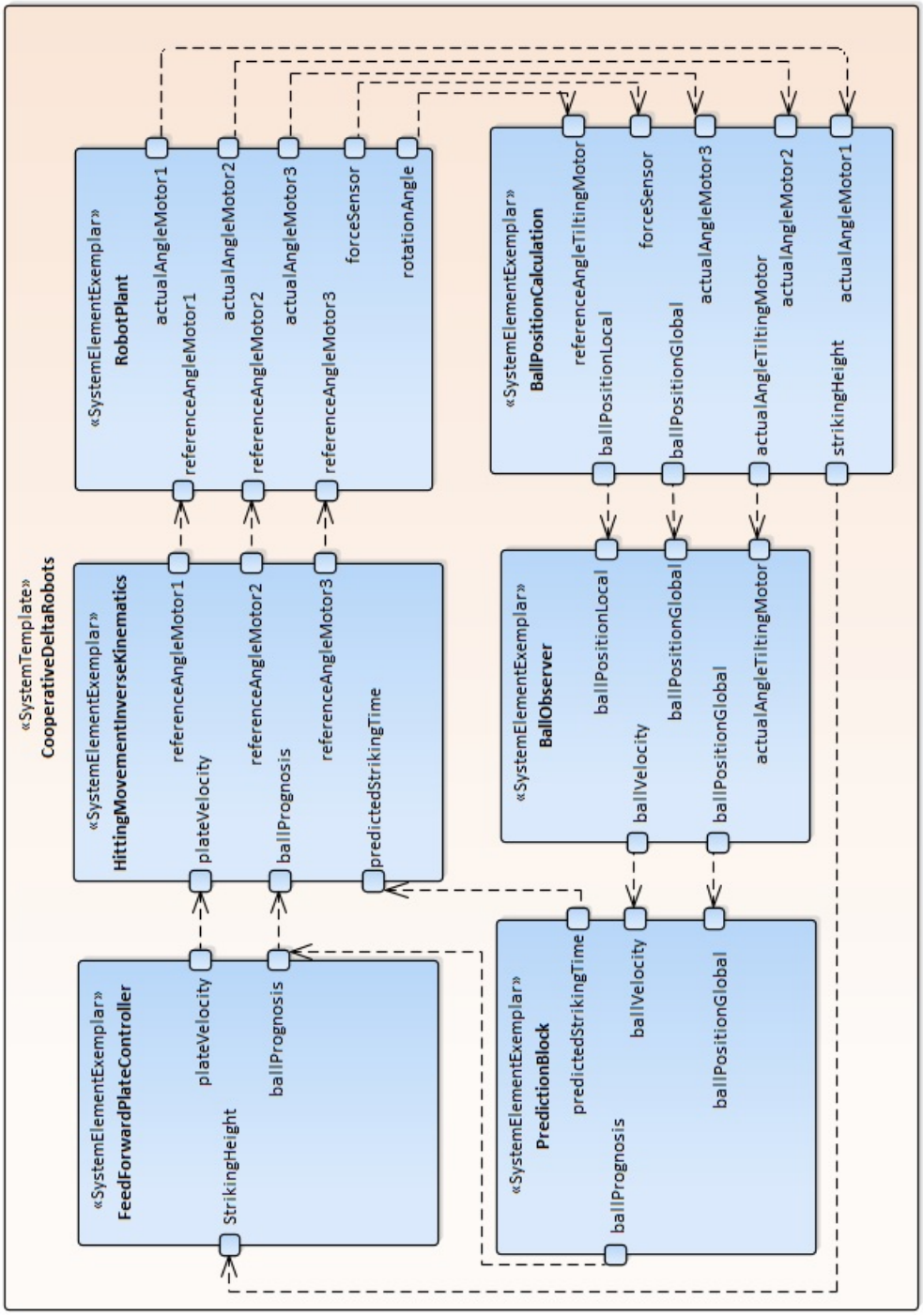


Figure 6-21: Active structure of cooperative delta robots controller

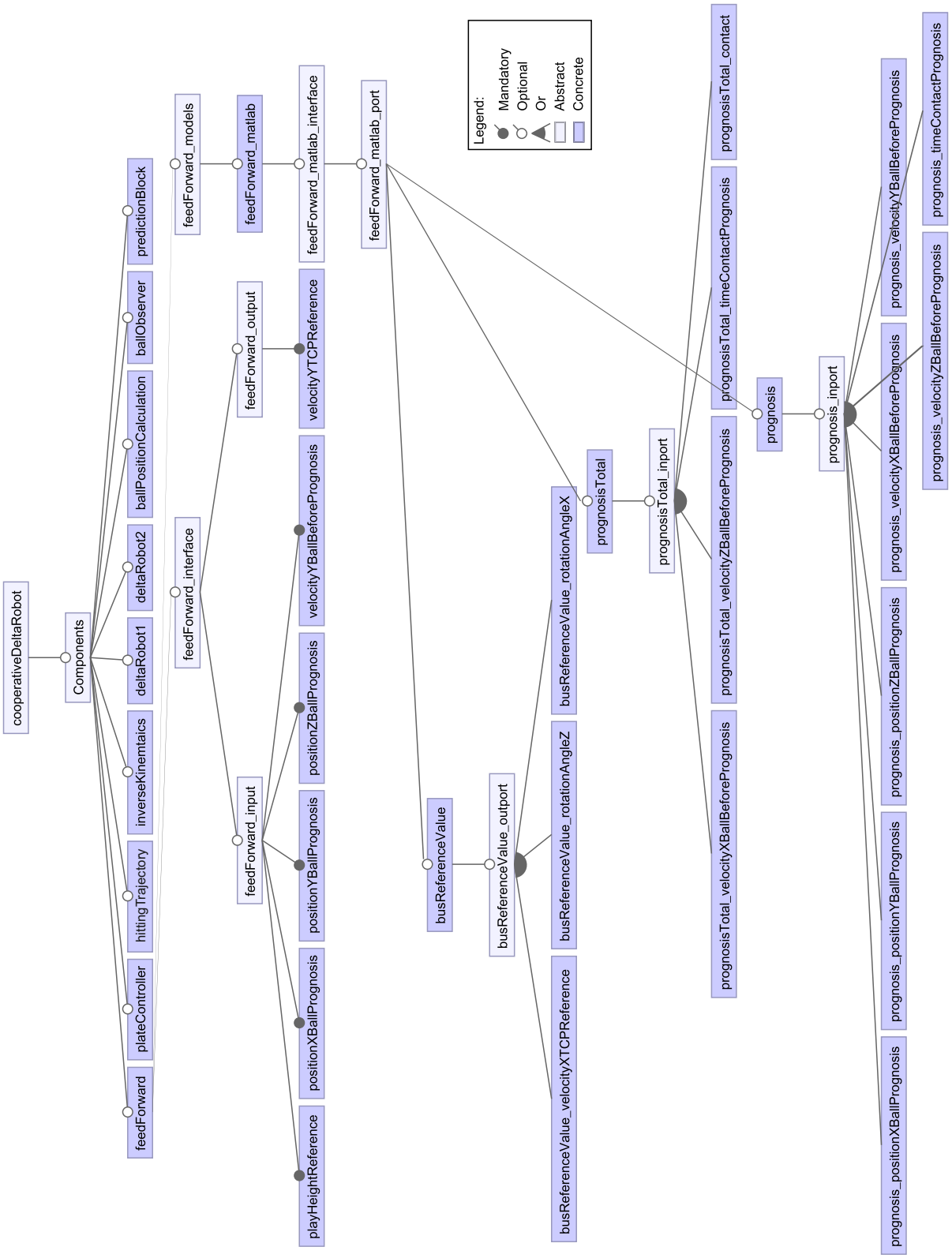


Figure 6-22: Expanded feature model of cooperative delta robots

Before connect them together a feature model is created by MMC to know how to connect the interfaces. As constraints the order of connections are given by user: *BallPositionCalculation => RobotPlant*

To connect the output signal of one block (*Robot Plant*) as bus signal to an input bus signal of another block (*BallPositionCalculation*) a bus connector is needed, because a bus signal represent a set of signals. To access them individually, the feature model by creation separates the signals of a bus, i.e. there is a feature for each signal. Based to that, MMC create a *Bus Selector* between the two blocks with all set of signals.

Also a *Switch* is created and connected to the *Switch Real Playing* input by MMC. This block is used to switch between hardware and model.

In the next step, required filters should be chosen by user if needed. As shown in Figure 6-23, a Butterworth filter is required to have a frequency response as flat as possible in the passband for angle of rotation about the x-axis(psi) and z-axis(phi). A final connection must be done by the user. *Hardware* bus and *BallPositionCalculation* bus are connected to the Switch. In the Table 6-3 the signal set of *Sensor Hardware Sensor Model* bus in the *Connector* block is listed.

Table 6-3: Bus Signals of RobotPlant and BallPositionCalculation

Signal Name	**Description**
F1 - F3	force sensor
phi	angle of rotation about the z-axis
psi	angle of rotation about the x-axis
teta 1-3	angle information of three base joints
phi motor	tilting angle of the motor

Scenario two: delta robot in RecurDyn, rest of system in Matlab/Simulink
The system units used in the RecurDyn is not SI system contrary to Simulink, therefore between these two blocks, a converter is used to convert the RecurDyn system unit to SI basic unit and inverse.

As shown in Figure 6-24, *BallObserver* block and *BallPositionCalculation* block are in Simulink and *DeltaRobot* in RecurDyn simulated. To connect these three blocks to each other a converter is needed to convert in one side meter to millimeter and in other side millimeter to meter.

The *BallObserver* calculates the instantaneous outgoing velocity of the ball after striking the plate and also the striking height and the position of the ball on the plate. In the *DeltaRobot* block TCP of Delta robot is calculated by forward kinematics of parallel robot in RecurDyn. In the *BallPositionCalculation* block,

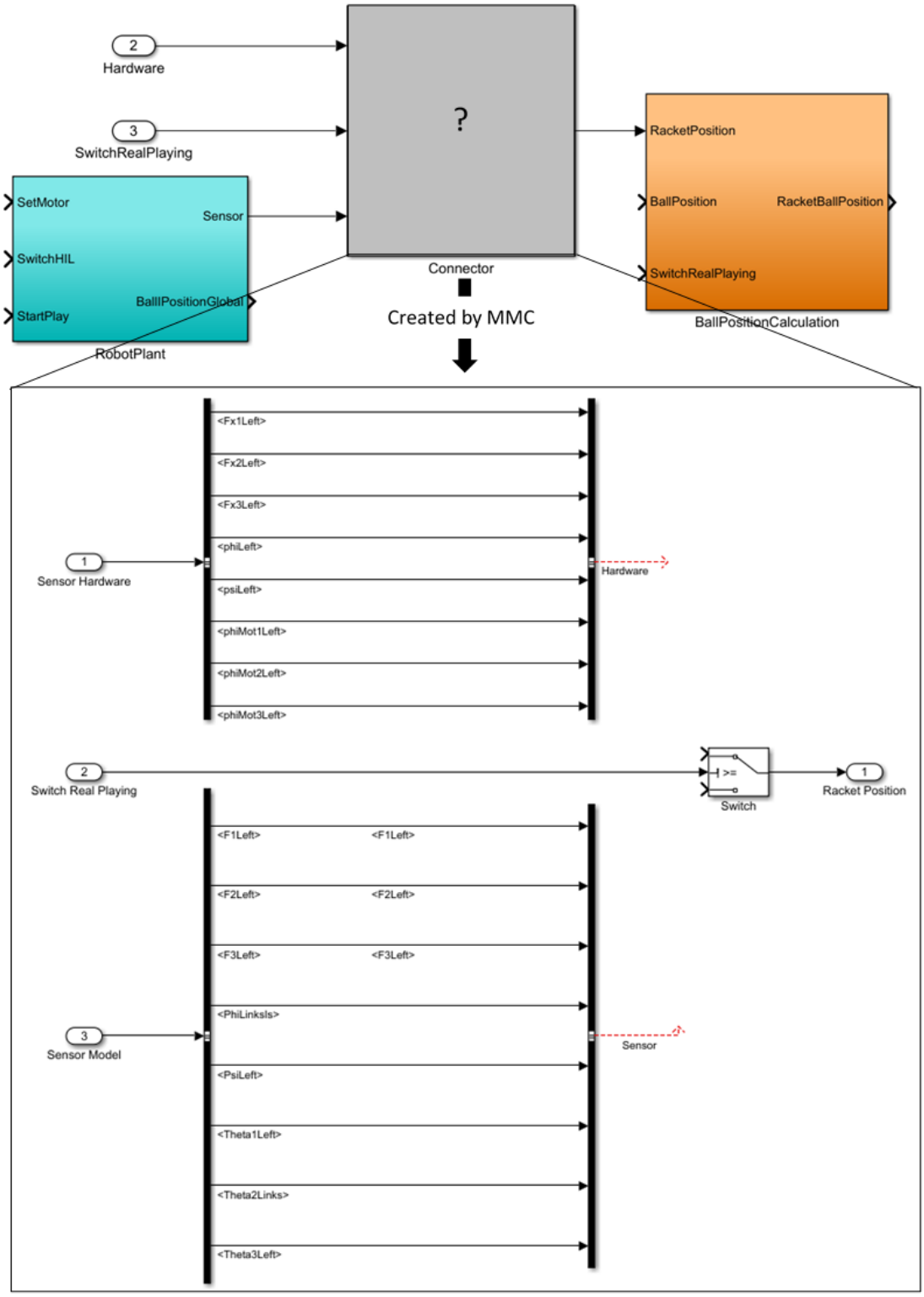


Figure 6-23: Connector between robot plant and ball position calculation for racket position calculation in Simulink

position of the ball in local and global coordinates is calculated using the information of base motors angles and the impact forces. In the Table 6-4 the used converters in the converter block are listed.

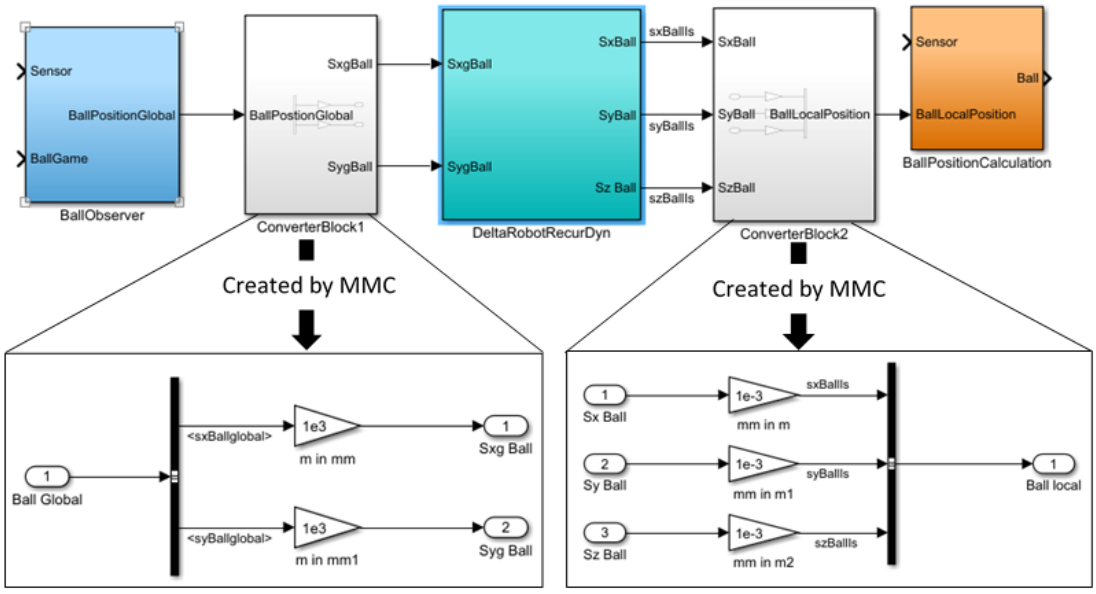


Figure 6-24: Converters between ball observer in Simulink, delta robot in Recurdyn and ball position calculation in simulink for local ball position calculation in Simulink

6.3.3 Analyze

To verify and validate the model, the specified test environments are set up and the measurements are carried out parallel to the information processing in the plant model. For example, stiffness and impacts of different play objects are calculated in the ball observer block to analyze behavior of part solution.

6.4 System Integration

System integration begins with respect to all disciplines, after the holistic model-based analysis has been successfully completed. Here again, the previously created simulation models can provide support in many places. In the following subsections, the integration divided into the subordinate steps of the SiL and HiL in the Matlab/Simulink and TwinCAT of Beckhoff [Twi].

Table 6-4: Connection Block between BallObserver, DeltaRobot and BallPosition-Calculation

Connection Block	Name	Description
Between Ball Observer and Delta Robot	Sxg Ball	x-component of balls position in global coordinates
	Syg Ball	y-component of balls position in global coordinates
Between Delta Robot and Sensor	Sxl Ball	x-component of balls position in local coordinates
	Syl Ball	y-component of balls position in local coordinates
	Szl Ball	z-component of balls position in local coordinates

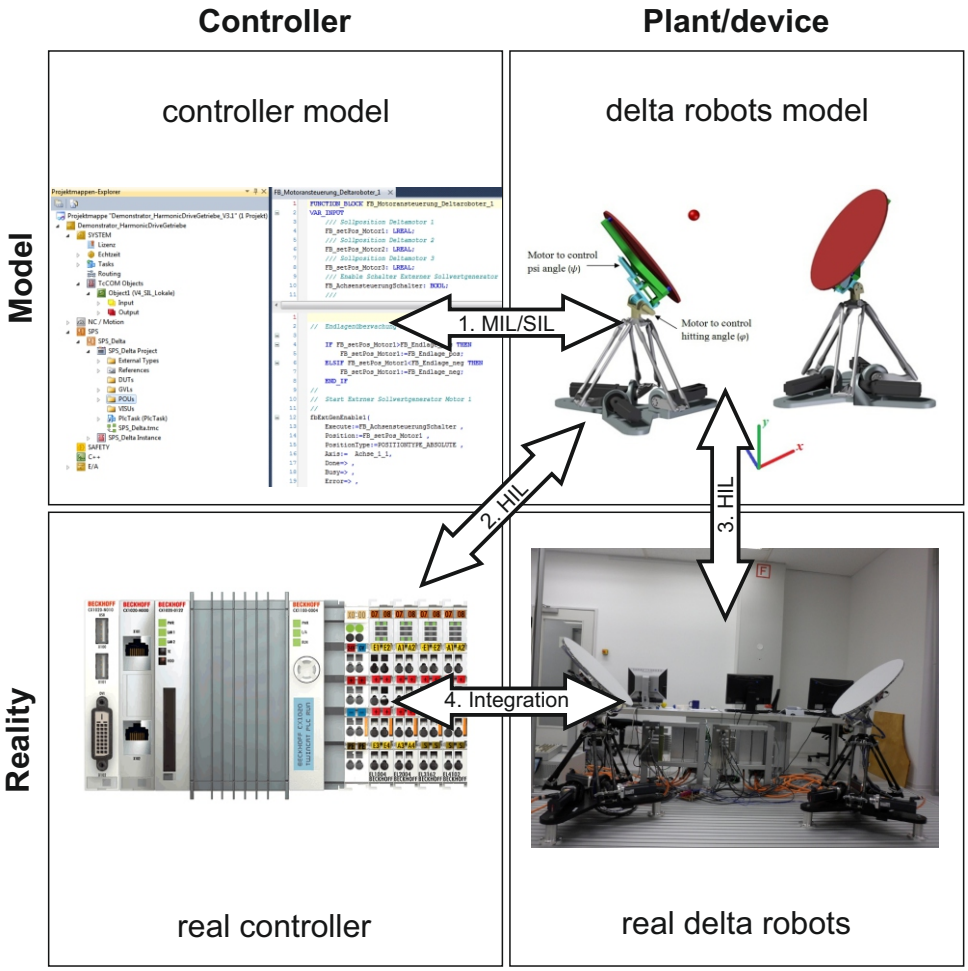
6.4.1 Software-in-the-Loop

In the related SiL simulation, the control is not executed on the target hardware, but it is also simulated. Thus, the interface between the controller and the virtual machine is not defined by any external system (see Figure 6-25) [PC02].

The virtual commissioning could be performed subsequent to the concretization of the entire solution. Virtual commissioning is a commissioning of the real control system and the original software in combination with a virtual plant. An early integration testing and a reduction of the commissioning time were provided by the virtual commissioning. Since the behavioral models could be used for the design of new components and for the commissioning simultaneously, their benefits increased [STB+13].

The controller model of cooperating delta robots is subsequently compiled by a mouse click into a TwinCAT module [1], *which works as a controller for the real controlled system in real-time [Twi].* Due to the fact that inputs, outputs and ports are defined via standard Simulink blocks, the interfaces must be equal to the interfaces of the module instance in TwinCAT. The Simulink model of cooperating delta robots is translated to a TwinCAT module in order to test a Programmable Logic Controller program in real-time, before the real system is connected. After instantiating the TwinCAT module generated from this model, a real-time simulation can be executed [Twi].

[1] *TwinCAT modules consist of a range of formally defined attributes and interfaces. They enable general application of the modules with each other and externally. The predefined interfaces enable cyclic calling of the internal module logic, for example. Each module implements a state machine that controls the initialisation, parameterisation and linking of the respective module [Twi].*



Legend:
1. MIL/SIL: Function verification
2. HIL : Controller test
3. HIL: Test
4: Integration

Figure 6-25: Transition from model to reality

In cooperating delta robots both the system as a model and the control program of the real control is already available. The entire system can be tested as part of a SiL with the system model. To make SiL simulation possible, the interfaces between system model and the control program must be prepared by MMC, as shown in Figure 6-26.

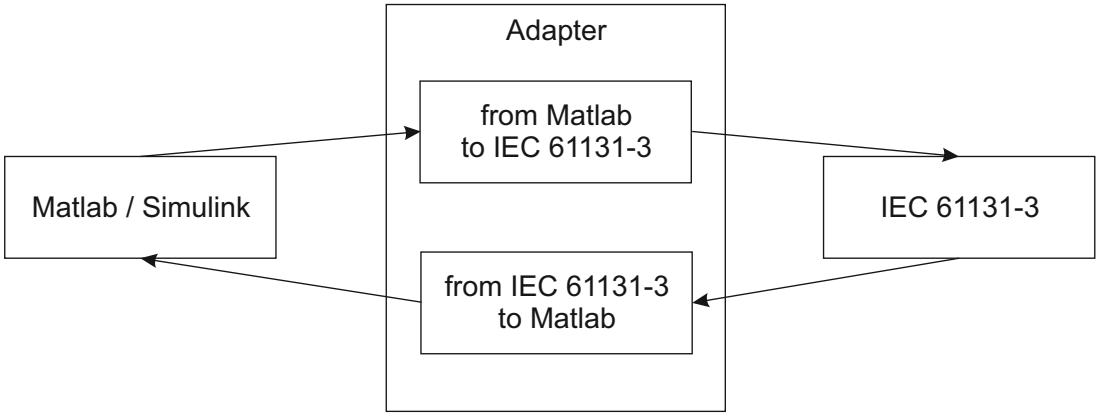


Figure 6-26: Interface conversion between Matlab and IEC 61131-3

Figure 6-27 and 6-28 show adapter between Matlab and control program consisting of following conversions:

1. conversion of data type of Matlab into data type of IEC61131-3 [2] and reverse by using standard data type conversion of the Matlab library, as shown in Figure 6-26. As shown in Table 6-5 the unit system of IEC61131-3 in compare to international unit system is listed, i.e. the different data types cover different value ranges. By reference to this list the type *conversion* is determined, which one should be used, e.g. INT32 to DINT in the case of angle (done by MMC).
2. conversion of units, e.g. angle degree (phi) to int16, as show in the Figure 6-27.
3. limit inputs through saturation, i.e. for conversion of degree to int16 a limit is needed from $(2^{16-1}-1)$ to $-(2^{16-1}-1)$ to ensure the limit size of the int16 variable (done by MMC).
4. gear used in encoder (done by user).
5. bus selector or creator, that its output using a bus object (done by MMC).

[2] *IEC 61131-3 not only describes the PLC programming languages themselves, but also offers comprehensive concepts and guidelines for creating PLC projects [JT10].*

In the Figure 6-27 following columns are shown the conversion of a Matlab model interface to an IEC interface. It is done by MMC to make them compatible for each other, e.g. motor and motor position:

- Input from Matlab: source is BusSet (reference rotation angle about z-axis) from Matlab.
- Bus selector: composed of many bus elements that feed subsystems (done by MMC).
- Gear: sets serve to transfer rotational motion and torque at a known ratio from one drive line axis to another [Mat], as PhiSetReset (reset rotation angle about z-axis) (done by user).
- Unit conversion: converts PhiPSet (rotation angle about z-axis) from degree to integer16. In the next case, converts PhiSetReset from degree to integer32 (reset rotation angle about z-axis) (done by MMC).
- Limitation: produces an output signal that is the value of the input signal bounded to the upper and lower saturation values. PhiPSet (set rotation angle about z-axis) limited from -32767 to 32767. ThetaSet (position of motor) must be limited by the user (done by MMC).
- Type conversion: converts values to Integer and Boolean which are needed for IEC61131, as all motor values (done by MMC).
- Output to IEC: preparing outputs for IEC target.

In the Figure 6-28 following columns are shown the conversion of an IEC interface to a Matlab model. It is done by MMC to make them compatible for each other, e.g. motor and force sensor. Motor is shown with encoder and actual value.

- Input from IEC: the source are values of motor encoder, motor angel and force sensor in IEC.
- Type conversion: converts an integer input signal to the double data type, like all motor and force sensor values (done by MMC).
- Unit conversion: converts from encoder value(integer) to Grad/s. In case of force, integer16 is converted to newton (done by MMC).
- Gear: sets serve to transfer rotational motion and torque at a known ratio from one drive line axis to another [Mat], as Phi (rotation angle about z-axis) (done by user).
- Bus creator block: combines a set of signals into a bus (done by MMC).
- Output to Matlab: preparing outputs for Matlab model.

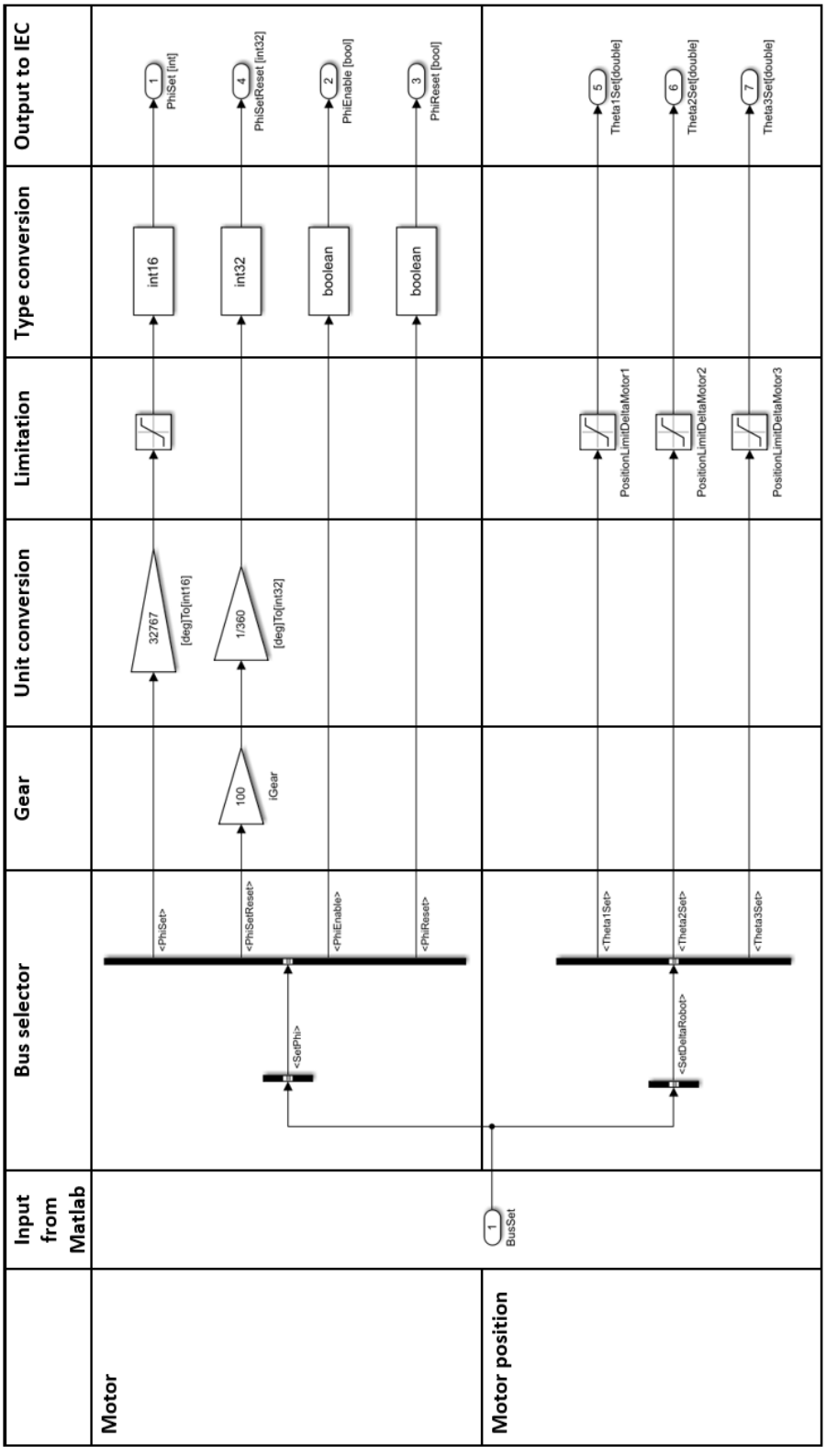


Figure 6-27: Structure of an adapter between Matlab model and IEC 61131-3

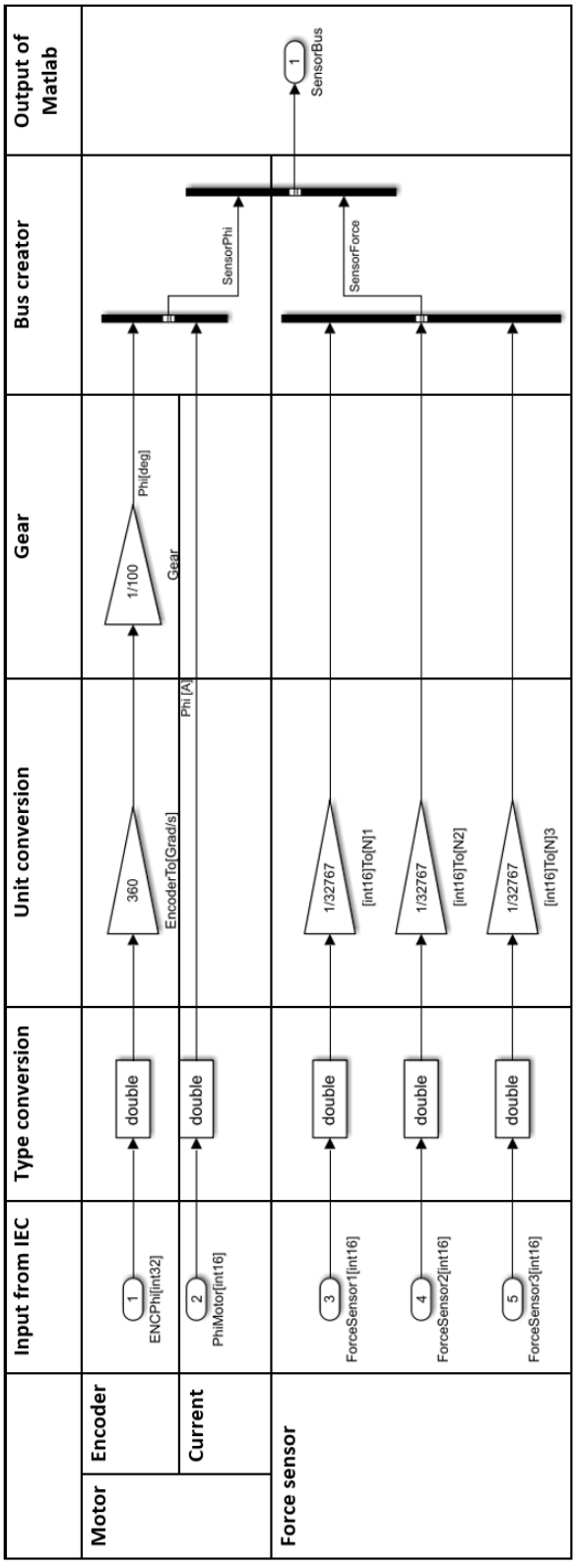


Figure 6-28: Structure of an adapter between IEC 61131-3 and a Matlab model

System Manager	IEC61131-3
BIT	BOOL
INT16	INT
INT32	DINT
...	...
UINT8	SINT
...	...
DOUBLE	LREAL
FLOAT	REAL

Table 6-5: List of type conversion in general system and IEC 61131-3 [Twi]

In the Table 6-6 some required inputs must be provided by the developer from the data sheet of each component. Other information can be exported from feature model. Bellow some of them are listed:

- gear and lines per revolution for resetting the encoder
- position limit of each motor
- measurement range and gear(100) of force sensor

The MMC adds this new information to the feature models. When from the updated feature model, a DBM based on the configuration tree is modeled, the reproduced adapters between Matlab and IEC 61131-3 are completed, as shown in Figure 6-27 and 6-28. The Simulink Coder for Matlab/Simulink environment contains a code generator, that works with the TwinCAT Target for Matlab/Simulink. Additionally, to ensure real-time capability of the Simulink model, a fixed-step solver must be configured in the solver settings [Twi].

To run a dynamic model designed in Matlab/Simulink as real time capable program in Twincat the following steps must be followed:

- Configuration of the Simulink model: the Twincat as system target file is selected in *Configuration Parameter* tab of Simulink.
- Generating a Twincat module from Simulink: the Simulink model of the delta robots is compiled to generate a module of it.
- Integration of the module in TwinCAT: instances of the generated module are integrated in a TwinCAT3 project [Twi].
- Cyclic call by a real-time task: this depends on the configuration parameters of the Simulink. If there are no matching tasks in Twincat, tasks can be created manually. *Fixed Step* Size specified in Simulink as cycle time for the allocated task is expected by the module. For all internal calculations,

From Matlab to IEC 61131

Interface	Input from Matlab	Gear	Unit conversion	Limitation	Adapter	Output to IEC
motor position	degree/second	-	$2^{x-1} - 1/360$	$[(2^{x-1}-1) \quad -(2^{x-1}-1)]$	integer x adapter	integer x
encoder	reset degree	100	4096 / 360	-	integer x adapter	integer x
	enable	-	-	-	boolean adapter	boolean
	reset	-	-	-	boolean adapter	boolean
	disable	-	-	-	boolean adapter	boolean
	degree	-	-	position limit	double adapter	double

From IEC 61131 to Matlab

Interface	Input from IEC	Adapter	Unit conversion	Gear	Filter	Output to Matlab
force sensor	integer x	double	$100/2^{x-1} - 1$	-	butterworth filter	double
motor sensor	integer x	double	360 / 4096	1 / 100	-	degree / second

Table 6-6: Table of conversion structure of interfaces from Matlab to IEC 61131-3 and inverse

a module uses the sample time set in Simulink. This setting is primarily intended for use in simulations within the TwinCAT environment [Twi]. For this reason the sample time can not be smaller than 0.5 milliseconds, because minimum cycle time in Twincat is 0.5 milliseconds.

The configuration is now completed and the model can be activated on the Twincat system, as shown in Figura 6-29. After activating and running the configuration the *Block Diagram* shows some online values in Twincat.

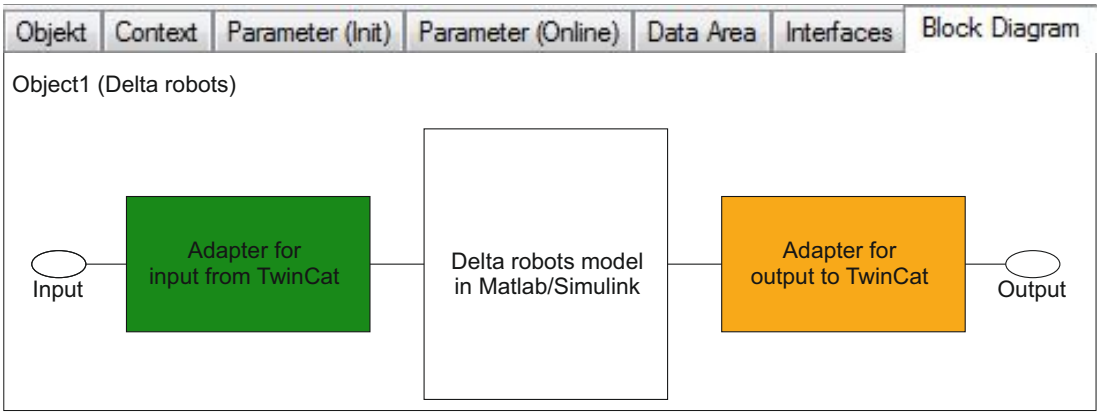


Figure 6-29: Block diagram of cooperated delta robots in Twincat

6.4.2 Hardware-in-the-Loop

All inputs and outputs of the real control system must be connected to the system model in order to test the functionality of the real controller program. This method is known as Hardware-in-the-Loop [STB+13].

With a high degree of maturity of the development, the results of a simulation for assurance of properties should be directly transferable to reality. In order to ensure the transferability of the results, the controlled behavior is integrated into the simulation.Here, the real controller is used and the real machine is replaced by a virtual machine (Figure 6-25). The virtual machine then has to behave as much as the real machine to the control system [PRR05]. This allows easy testing of the controllers.

The interface of the virtual machine is therefore defined by the real target system of the controller. Data exchange via this interface must be guaranteed within the cycle time of the controller, as in the real system. A time-deterministic behavior must therefore be ensured by the controller as well as by the virtual machine. The time-deterministic behavior is ensured by the use of the real target system of the control and the realization of the virtual machine on a real-time system. The

described requirements are met by a HiL. If this interface is reproduced exactly, there is no difference with regard to the interchangeability to the HiL [Kuf11].

A HiL simulation of the information processing on the target hardware - an industrial computer - is to be carried out. The control signals are then tapped and fed to the real servo controllers so that only the movement of the delta robots can be tested independently of the ball detection and control. The ball is virtual in this HiL simulation. Subsequently, the sensor values of the motor angles are converted into the information processing [Oes17].

At the end, the real system of cooperating delta robots is put into operation successively, whereas individual system components are still integrated as simulation models. The dynamic model of the entire system is still implemented in Simulink and a Twincat module (`Object1` as shown in Figure 6-30) is generated from it. *The simulation module can evaluate sensor data and calculate the systems response [Twi].* For that the available I/O hardware devices must be scanned to connect them to the simulation model. Then, every variable in the tree view of I/O devices are listed with their information and settings. Here, each variable could be linked to the variable link of the selection diagram, which shows all potentially linkable target input/output variables. In Figure 6-30 the motor position of both robots and ball position as a part of hardware output devices are listed, which are ready to be linked to input variables of simulation as a Simulink model with compatible type (double) and size(64 bit). This compatibility is provided by MMC to prepare the connection between interfaces of simulation model and Twincat module.

6.5 Evaluation of Variability Management Approach

The essential requirements of the design framework procedure model for developing MMC have been formed. It defines the required phases in a hierarchy to do the V model steps for design of a mechatronics system. The developers can use the design framework described in this work in order to create and design a mechatronic system.

The analysis of the existing methods and approaches towards a traditional V model has shown that this method affects positively the previously defined approach. The evaluation is described as follows:

Procedure model:

- Systematic procedure:
  Target: The development phases should be systematically described based on the detailed V model [GTS14b]. The design variants should be structured to be offered to the user for creating the desired model.

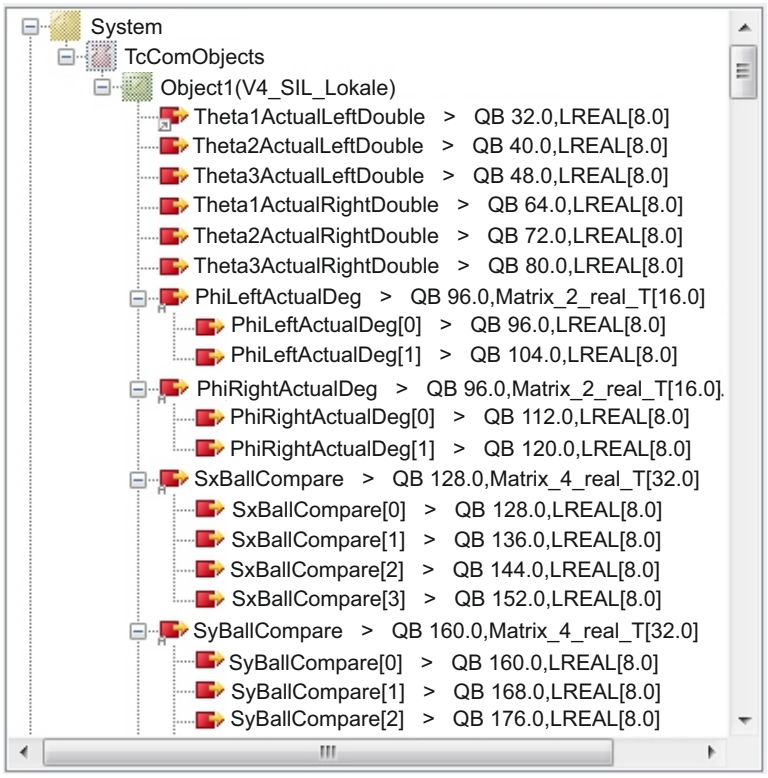


Figure 6-30: All potentially linkable target variables depending on the selected variable type

Solution: The used variability management methods in V model offers the user to create the desired model.

- Complexity reduction:
  Target: Several technologies are used in a complex mechatronic system, therefore, the complexity of design procedure should be reduced.
  Solution: The complexity of design procedure is reduced through:
  - GUI to follow the developer during the design procedure
  - Feature modeling to manage the design variability
  - Highlighting the gaps of modeling
  - Automatism of some steps
- Multi domain deal:
  Target: A mechatronic system consists of mechanical components, electronics components, control components and software components. Here, the procedure should be considered as an interdisciplinary design.
  Solution: The procedure is considered as an interdisciplinary design. Here,

the different modeling and simulation tools are considered to cover interdisciplinary design of a mechatronic system.

- Ensure consistency:
  Problem: In current approaches there is not any automation to ensure consistency along the design method. It must be considered by the user manually. Normally it does not reflected to other steps because of dependency lack between steps.
  Solution: In the prototype software an automation is considered to ensure consistency along the design method, e.g. updating the system model and simulation model automatically through feature model and adaption of system model changes to simulation model and reverse.

Design model:

- Modular and extendable design model:
  Problem: In current approaches by adding/removing a new component to/from a system design the entire system is affected so that the design must be restructured.
  Solution: The design method helps to modularize the subsystems of a mechatronic system, to avoid the restricting of the design model by adding or removing a component. Here, there is a possibility to store the modules of design system in a repository and reuse them partially in a later design system.

Configuration software tool

- Design variant:
  Problem: There is not an approach to configure different design variants which allows the user to model, simulate and test a system from different aspects.
  Solution: Here, an approach is provided to configure different design variants of a mechatronic system. According to this approach the system can be analyzed from different aspects.
- Identical steps:
  Target: The discussed design method consists of sequential design phase which must be followed to reduce the common errors and ensure the consistency. There is not any tool which identified these steps sequentially.
  Solution: The designed GUI helps the developer to identify the steps sequentially according to the V model. The sequential design phase can be followed to reduce the common errors and ensure the consistency.
- Automation possibility:
  Problem: There is no automatism to implement a mechatronic system according to the V model. For each phase there is just an isolated application

which can be provide by user manually.
Solution: The developed method some steps are implemented automatically according to V model to invest less time for design development of a system. Here, there is a possibility to deal with the design system centric in one tool.

6.5.1 Quality and Quantity of Feature Models

Feature modeling method used to describe variability in the design of mechatronics system. A usable feature modeling tool environment enables effectively to communicate model information. Here, the quality and quantity of feature diagrams for case study is analyzed to evaluate the effort of presented approach usability. Jaksic et al. [JFC+14] shows:

- Larger textual model: increased cognitive efforts of users
- Larger visual model: linear increase of feature model quality
  - They do not have to deal with the increased cognitive load of textual model
  - They dedicate more of their cognitive processing power on tasks that would yield higher model quality

Jaksic et al. [JFC+14] in the Table 6-7 breaks down the criteria which are used to assess the quality of a feature model.

According to the 6-7 assigned weight to the feature models of this case study is 0.8, since here feature model includes a comprehensive features set but might fail to accurately represent certain group dependencies, e.g. used AND-group when XOR-group would be more appropriate.

According to Jaksic et al. [JFC+14] the quantity as a measure of FM completeness is calculated in the case study as follows.

In current approach the main hierarchy of feature model with the feature number in case study in the conceptual system design phase is listed below:

- Number of components = 9
- Number of tools = 3
- Number of modeling depth = 4
- Number of interfaces and parameters > 2
- Total number of features > (9 x 3 x 4 x 2) = 216

For the 9 components considered in the case study, each component can either be excluded from a configuration or included with one selected model variant. Since up to 12 model variants are available for each component, resulting from 3

Table 6-7: Feature model quality [JFC+14]

Quality	Assigned Weight	Description
Poor	0,2	FM cannot be properly parsed by the tool (i.e., model contains an inconsistent and/or an invalid element(s), or simply it is not syntactically well-formed).
Satisfactory	0,4	FM is properly loaded by the tool but lacks majority of features and/or groups.
Good	0,6	FM is mostly complete (i.e., includes various Audi model lines) and has neither inconsistencies nor invalid elements.
Very good	0,8	FM includes a comprehensive features set but might fail to accurately represent certain group dependencies (e.g., used AND-group when XOR-group would be more appropriate).
Excellent	1,0	FM includes a comprehensive features set, and provides a solid foundation for further breaking down the model as an SPL artifact. Different feature groups, dependencies and constraints were used in terms of both quantity and quality. (i.e., this FM, if offered with an online con figurator, has enough details to allow a customer to produce a model of a custom Audi car tailored for her needs).

tools and 4 modeling depths, each component has 13 possible states. 1 state for excluding the component and 12 states for selecting exactly one of the available model variants. Therefore, the number of possible non-empty configurations is calculated as:

$(1 + 12)^9 - 1 = 13^9 - 1 = 10{,}604{,}499{,}372$

The subtraction of 1 excludes the empty configuration in which no component is selected.

The feature model quantity is evaluated based on the 4 criteria listed in Table 6-8.

According to Jaksic et al. [JFC+14], the completeness of a feature model is assessed using four equally weighted criteria: number of features, number of constraints, number of valid configurations, and feature model depth. Each

Table 6-8: Feature model completeness evaluation based on [JFC+14]

Criterion		Case Study		
Category	**Weight**	**Score**	**Value**	**Description**
Feature	0.25	1	>216	50 or more
Constraint	0.25	1	>5	5 or more
Valid config	0.25	0,25	10,604,499,372	Above 25k
FM depth	0.25	1	>4	Depth of 4 or more

criterion contributes 25% to the overall FMQuantity value. Based on the criterion weights and the scores obtained in Table 6-8, the FM quantity is calculated as:

$$FMQuantity = (0.25 \times 1) + (0.25 \times 1) + (0.25 \times 0.25) + (0.25 \times 1) = 0.8125$$

The feature model quality is assigned a value of 0.8 according to Table 6-7. Therefore, the user effectiveness is calculated as:

$$UserEffectiveness = FMQuantity \times FMQuality = 0.8125 \times 0.8 = 65\%$$

According to Jaksic et al. [JFC+14], the user effectiveness is used as an indicator for evaluating how effectively a feature model supports the modeling task. The metric combines both the completeness of the feature model (FMQuantity) and the assessed quality of the model (FMQuality).While FMQuantity reflects the degree to which relevant features, constraints, valid configurations, and hierarchy depth are represented, FMQuality evaluates the correctness and consistency of the resulting feature model. Therefore, a user effectiveness of 65% indicates that the proposed feature model provides a comprehensive and consistent representation of the system variability, although further improvements regarding model completeness and dependency representation are still possible.

6.5.2 Effort Estimation for Each Phase of V Model

Safavi and Shaikh estimate the effort of each software development life cycle phase at the very abstract level. According to effort estimation model for each Phase of Software Development Life Cycle [SS10] the percentage of work duration associated with its decomposes the ratio of effort put in each phase. As depicted in the table 6-9, 40% of work effort is put in detailed discipline-specific design code and unit test phase. The rest effort is put in different areas of the project development life cycle. This estimation approach is adapted from the effort estimation of Safavi and Shaikh[SS10].

Effort advantage of V model with variability management approach as shown in the Table 6-10:

Table 6-9: Work duration of V model [SS10]

Activity	**Standard Work Effort%**
System Design:	
Determination of objective	6%
Synthesis and analyze	10%
Discipline-specific design:	
Determination ofobjective	14%
Synthesis and analyze	40%
System Integration:	
Determination of objective	20%
Synthesis and analyze	10%
Total Effort	100%

Conceptual system design phase:
Create configuration tree: create behavior model needs less effort (x1)
Update feature model and configuration tree: iteration needs less effort (x2)

Discipline-specific design:
Create configuration tree: detail/expand behavior model is with less effort (y1)
Update feature model and configuration tree: iteration needs less effort (y2)

Model-based system integration:
SiL: Concrete test and analyze process needs less effort (z1)
HiL: Integrate components needs less effort (z2)

The affect of using MMC on the effort of standard task set is listed in the Table 6-10.

The total effort of current approach according to the Figure 6-31 is:
x1 + x2 + y1 + y2 + z1 + z2

Finally, effort coefficient has to calculated with the effectiveness of feature model(74%). Then the final effort is:
0,74 * (x1 + x2 + y1 + y2 + z1 + z2)

6.6 Summary

In this chapter, an application example of two cooperating delta robots that juggle a ball by passing it to each other (see Figure 6-1) is used to illustrate the proposed approach for an integrated model-based design process. The cooperating

Table 6-10: Work duration of V model with variability management approach

Activity	**Standard Work Effort%**
System Design:	
Determination of objective	6%
Synthesis and analyze	10% - (x1 + x2)
Discipline-specific design:	
Determination of objective	14%
Synthesis and analyze	40% - (y1 + y2)
System Integration:	
Determination of objective	20% - (z1 + z2)
Synthesis and analyze	10%
Total Effort	100% - (x1 + x2 + y1 + y2 + z1 + z2)

delta robots lead to new requirements on a systematic design process considering different disciplines.

Environmental model, application scenarios, requirements and functions are prepared in the course of the interdisciplinary system design. In synthesis phase, active structure, feature model, shape and behavior and activity diagram of delta robots are illustrated to select a specific and/or more concrete solution patterns. By using feature models, the idealized dynamics model of the system, suitable parameter configurations are determined and tested. These specify the component requirements and idealized dynamics models with a low level of detail and assured on the principle level before switching to the detailed elaboration. Here, is explained the algorithm of DBM generation from SysML. Furthermore, is outlined how submodels interfaces are matching together according to feature model to connect DBM submodels.

At this point, the search for solution elements in the course of the discipline-specific elaboration is provided. In this step, among concretizaion of requirements and detailed activity structure, in the first instance different scenarios are studied and measured. According to this basis, the modeling objectives for behavioral models are prepared or detailed. They contain target sizes for the model, degree of detail and complexity, which are dependent on the test concept developed.

In the course of System integration, SiL and HiL for the delta robots are executed (see Figure 6-25). In the related SiL simulation, the control is not just executed on the target TwinCAT PLC, but is also simulated in the Matlab/Simulink. In this case study both the system as a model and the control program of the real control is already available. The entire system is tested as part of a SiL with the system model. In the HiL, the results of a simulation for assurance of properties is

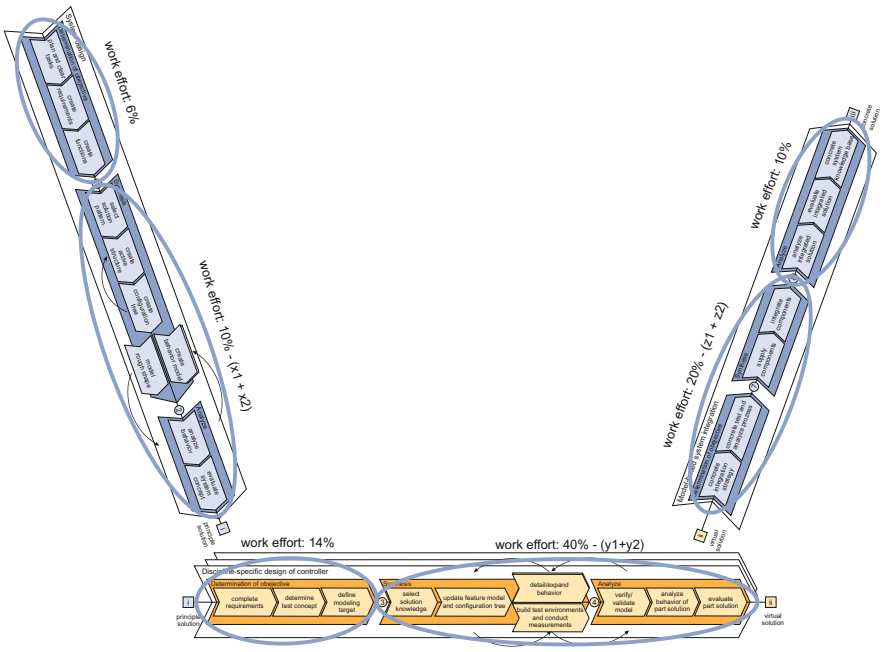


Figure 6-31: Effort estimation of extended V model

directly transfered to reality. In order to ensure the transferability of the results, the controlled behavior is integrated into the simulation.

At the end, developed design method is evaluated by analysis of the methods and approaches towards the traditional V model. Here, the effectiveness of feature model is calculated to estimate total effort of design method in the case study.

7 Conclusion and Outlook

The findings gained in the context of this thesis are summarized in this chapter and then an outlook on possible, further scientific contribution topics will be given.

Mechatronics product is outcome of a global concurrent engineering or integrated design approaches. The task distribution between different disciplines and the creation of synergies are parts of the system design. The advance modeling and simulation are important as well as many varieties of design that caused by the number of realized prototypes. The handling of individual disciplines are mostly focus of numerous methodologies for design of mechatronic system. One of them is presented by VDI Guideline 2206 [Ver04] based on an adapted software development method, the V model.

Here, an approach for the integrated model-based design of intelligent dynamic systems is presented. A concept for the establishing a cross-domain solution is the basic target of this system design. To reduce the design complexity because of involving different modeling tools across the system, consistency between multi domains and dependencies between subsystems are considered. In this work, a guideline with more flexible procedural model to follow a fixed schedule form for design processing is proposed. For this purpose, the original V-model of the VDI guideline is further broken down into subfunctions. Also, some additional subfunctions are added to the developed system design process of Oestersötebier [Oes17]. Considering that, a method of variability management is adapted to use commonalities and differences of the various implementation fragments of the resulting product. To handle this, Feature-Oriented Software Development is applied to provide configuration options and to facilitate the generation of model-based system design based on a selection of features.

The nature of the mechatronic product and FOSD are briefly defined in Chapter 2. Here, the relevant literature and methods to this work is highlighted to identify current available supports for thos challenges.

In the next chapter, the result of using variability management in a mechatronic system is presented. Also the concept of a Multifunctional Model Client is introduced, which combines outcomes of the different disciplines to analyze mechatronic systems and to explain the design of a mechatronic system. The focus of chapter 3 is on creating, combining and configuring of dynamic behavior models of the components in an appropriate level of detail. So an adequate method to manage variabilities is developed to make a modular and reconfigurable mechatronic subsystems. Through the adaption of feature modeling approach, consistency

between different disciplines is assured. The facilities of the feature model semantic is expanded as well. This concept supports sharing of common information between development team members, because bi-directional information exchange to system and solution knowledge is established.Therefore, both information about the system and information on suitable solution elements is modeled. In summary, this concept helps developers to follow an integrated model-based design process of mechatronic systems step by step. In order, chapter 3 classifies this thesis in terms of the known design method types in VDI Guideline ([VDI04]) and design methods of intelligent mechatronics system [GTS14b].

Also a fundamental definition of variability management and feature modeling is largely used to develop the methodology in this study. To understand different uses of them in the research methodology, their contributions are highlighted in Chapter 4. The Table 4-1 shows how the basic structure of mechatronics system is impacted by feature models according to variability of modeling tools and modeling depth(details). Also, a grammatical feature model is developed to define the relationships between submodels of a mechatronic product.

Then, in the Chapter 5, a framework based on the java programming language is implemented based on this concept, thereby allowing the developer to handle the whole procedure of designing the mechatronics system. To achieve seamless integration of the presented approach in everyday work, focuses placed on coupling of widely used tools like Enterprise Architect, MATLAB/Simulink or Dymola. Therefore, the knowledge bases, including solution elements and reusable models, as well as adapters and unit converters for matching interfaces, are developed. However, as the representation of system knowledge and solution knowledge comply, every successful development possibly supplies new solution elements. The evolution is also included in the framework.

Additionally, to cover the interaction of different modeling tools within the product development process, the co-simulation ability is adapted in the model configurator. To achieve this target, the configurator makes the subsystems compatible to each other to prepare them to couple. This approach involves handling the present subsystems model with various depths in different tools by using feature models. This work shows the possibility to connect them by considering their interactions. A configurator was created by realizing the variety of modeling depth and modeling tool as a feature, which gives the ability to select desired features and create a valid configuration of the model. To execute all these tasks the methods of model transformation are used to support integration and communication between the models. For model communication where models are exported from one tool to another, co-simulation and model exchange are applied.

To show more task details of MMC, this configurator creates the inputs, outputs and parameters that needs to be defined, such that the present subsystems can be

connected to one another. If a desired modeling depth is missing in a desired tool, it prepares the necessary inputs, outputs and parameters of the absent subsystem. While considering them, the user can create the absent part of the model. The model configurator bases on the defined dependencies and desired simulation model and prepares the required type of adapter and unit converters. The model configurator helps the developer to follow the generic procedure for designing step by step. It supports the sharing of common information between development team members. On the basis of feature model, the subsystems can be updated or changed while considering the interactions and dependencies.

To evaluate the developed method based on defined criteria and implementation of MMC, a delta parallel robot is presented in Chapter 6. The steps of developing the robots are conducted semi-automatically by MMC. The feature models are applied to demonstrate every existing model of subsystems. So the user can synthesize and analyze each step iteratively. The model is integrated in the automation software TwinCAT and executed in SiL tests, to provide an early integration testing and to reduce the commissioning time. For this purpose automation modules are created base to the simulation model and among others made its interfaces compatible with automation system. In HiL the real control system is connected to the system model to test the functionality of the real controller.

The results are listed as follows:

- Refine model-based system design process for a mechatronic product
- Guide developer to design a complex product through different involved domains
- Develop a tool which helps developer to configure the submodels with considering dependencies and relations
- Vertically: across the system design
- Horizontally: through different domains regarding to modeling and simulation tools
- Show cooperative work and information exchange among the engineers
- Exchange of design models and data
- Manage the variability of subsystems
- Make library from commonalities to make subsystems reusable
- Couple the subsystems from different disciplines

In this thesis, according to the concept of variability management the variety of design in mechatronics system are considered to build up variety model of one product. In the future work, variety of components can be investigated to prepare variety of end-product.

Here, the model-based system design of mechatronics system is discussed. According to this concept, a software is implemented to help the user to execute the design step by step. Here, some few simulation tools and system tools, like Matlab/Simulink, Dymola, Recurdyn, Papyrus and Enterprise Architect are supported to do some steps semi-automatically. In the future work the implementation of software can be continued. So that more modeling and simulation tools are supported.

In this thesis, the tasks and steps are performed semi-automatically. In this point, there is a room to automation. Since the case studies are from the research area (sorting machine and parallel delta robots), with the further industrial examples the concept could be prepared for the industrial use. Also, beside the existing model libraries as system knowledge, solution knowledge and adapters, in the future work can summarize and standardize more elements from other projects for the libraries.

8 Bibliography

[Aba] *Abaqus*. https://www.3ds.com/de/produkte-und-services/simulia/produkte/abaqus/, . – Online; accessed on 06-April-2020

[ABB+01] Atkinson, C.; Bayer, J.; Bunse, C.; Kamsties, E.; Laitenberger, O.; Laqua, R.; Muthig, D.; Paech, B.; Wüst, J.; Zettel, J.: Component-Based Software Engineering. The KobrA Approach. (2001)

[AC04] Antkiewicz, M.; Czarnecki, K.: FeaturePlugin: feature modeling plug-in for Eclipse. In: *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange* ACM, 2004, S. 67–72

[Ada04] Adamsson, N.: Model-based development of mechatronic systems?Reducing the gaps between competencies? In: *5th International Symposium on Tools and Methods of Competitive Engineering Lausanne, SWITZERLAND, APR 13-17, 2004* Bd. 1 MILLPRESS SCIENCE PUBLISHERS, 2004, S. 405–413

[AFr] *A Framework for Software Product Line Practice, Version 5.0*. http://www.sei.cmu.edu/productlines/frame_report/pl_is_not.htm, . – Online; accessed on 06-April-2020

[AFT+10] Alvarez Cabrera, A.; Foeken, M.; Tekin, O.; Woestenenk, K.; Erden, M.; Schutter, B. D.; Tooren, M.; Babuska, R.; van, F. H.; Tomiyama, T.: Towards automation of control software: A review of challenges in mechatronic design. In: *Mechatronics* 20 (2010), Nr. 8, 876–886. http://doc.utwente.nl/80227/

[AK09] Apel, S.; Kästner, C.: An Overview of Feature-Oriented Software Development. In: *Journal of Object Technology* 8 (2009), S. 49–84

[ALR+05] Apel, S.; Leich, T.; Rosenmüller, M.; Saake, G.: FeatureC++: on the symbiosis of feature-oriented and aspect-oriented programming. In: *GPCE* Bd. 3676 Springer, 2005, S. 125–140

[ALS+08] In: Amelunxen, C.; Legros, E.; Schürr, A.; Stürmer, I.: *Checking and Enforcement of Modeling Guidelines with Graph Transformations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–89020–1, 313–328

[AOT15] Abrishamchian, F.; Oestersötebier, F.; Trächtler, A.: Feature Model Approach For Managing Variability of Dynamic

Behavior Models in Mechatronic Systems. In: *Proceedings of the ASME 2015 International Mechanical Engineering Congress & Exposition*. Houston, Texas: ASME, November 2015

[Arn10] Arnold, M.: Stability of sequential modular time integration methods for coupled multibody system models. In: *Journal of computational and nonlinear dynamics* 5 (2010), Nr. 3, S. 031003

[ASW+11] Apel, S.; Speidel, H.; Wendler, P.; Rhein, A. von; Beyer, D.: Detection of Feature Interactions Using Feature-aware Verification. In: *Proc. of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2011, S. 372–375

[Bat05] Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: *Proc. of the 9th International Conference on Software Product Lines*, Springer-Verlag, 2005, S. 7–20

[Bat06] Batory, D.: A tutorial on feature oriented programming and the ahead tool suite. In: *Generative and Transformational Techniques in Software Engineering* (2006), S. 3–35

[BBB+12] Becker, S.; Brenner, C.; Brink, C.; Dziwok, S.; Heinzemann, C.; Löffler, R.; Pohlmann, U.; Schäfer, W.; Suck, J.; Sudmann, O.: The MechatronicUML Design Method – Process, Syntax, and Semantics / Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn. 2012 (tr-ri-12-326). – Forschungsbericht. – Vers. 0.3

[BC96] Brownsword, L.; Clements, P.: A Case Study in Successful Product Line Development. / CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST. 1996. – Forschungsbericht

[BCC+99] Bergey, J.; Campbell, G.; Clements, P.; Cohen, S.; Jones, L.: Second DoD Product Line Practice Workshop Report / CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST. 1999. – Forschungsbericht

[BCD02] Bernardo, M.; Ciancarini, P.; Donatiello, L.: Architecting families of software systems with process algebras. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11 (2002), Nr. 4, S. 386–426

[BEKL+14] Biehl, M.; El-Khoury, J.; Loiret, F.; Törngren, M.: On the modeling and generation of service-oriented tool chains. In: *Software & Systems Modeling* 13 (2014), Nr. 2, S. 461–480

[BFK+99] Bayer, J.; Flege, O.; Knauber, P.; Laqua, R.; Muthig, D.; Schmid, K.; Widen, T.; DeBaud, J.-M.: PuLSE: A methodology to develop software product lines. In: *Proceedings of the 1999 symposium on Software reusability* ACM, 1999, S. 122–131

[Big] *BigLever Software Gears Product Line Engineering Solution.* `http://www.biglever.com/solution/product.html`, . – Online; ccessed on 06-April-2020

[BJS09] Botterweck, G.; Janota, M.; Schneeweiss, D.: A design of a configurable feature model configurator. (2009)

[BOA+11] Blochwitz, T.; Otter, M.; Arnold, M.; Bausch, C.; Elmqvist, H.; Junghanns, A.; Mauss, J.; Monteiro, M.; Neidhold, T.; Neumerkel, D. u. a.: The functional mockup interface for tool independent exchange of simulation models. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany* Linköping University Electronic Press, 2011, S. 105–114

[Bod07] Bode, H.: *MATLAB-SIMULINK: Analyse und Simulation dynamischer Systeme.* Springer DE, 2007

[Bol07] Boldt, R.: Combining the Power of MathWorks Simulink and Telelogic UML/SysML-based Rhapsody to Redefine MDD. In: *Telelogic White Paper* (2007)

[Bol08] Bolton, W.: *Mechatronics: A Multidisciplinary Approach.* Pearson Prentice Hall, 2008 (Mechatronics: a multidisciplinary approach v. 10). `https://books.google.de/books?id=gPTcxA3f_SIC`. – ISBN 9780132407632

[BON+07] Brisolara, L.; Oliveira, M.; Nascimento, F. A.; Carro, L.; Wagner, F. R.: Using UML as a front-end for an efficient Simulink-based multithread code generation targeting MPSoCs. In: *DAC 2007 Workshop, UML-SoC*, 2007

[Bon08] Bonnema, G. M.: *Funkey architecting: an integrated approach to system architecting using functions, key drivers and system budgets.* University of Twente, 2008

[Bor10] Borches, P. D.: A3 Architecture overviews. In: *Views on Evolvability of Embedded Systems.* Springer, 2010, S. 121–136

[Bos00] Bosch, J.: *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach.* New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. – ISBN 0–201–67494–7

[BPS12] Brink, C.; Peters, M.; Sachweh, S.: Configuration of Mechatronic Multi Product Lines. In: *Proc. of the 3rd International Workshop on Variability & Composition*, ACM, 2012. – ISBN 978–1–4503–1101–4, S. 7–12

[BPSP04] Beuche, D.; Papajewski, H.; Schröder-Preikschat, W.: Variability management with feature models. In: *Science of Computer Programming* 53 (2004), Nr. 3, 333 - 352. http://dx.doi.org/http://dx.doi.org/10.1016/j.scico.2003.04.005. – DOI http://dx.doi.org/10.1016/j.scico.2003.04.005. – ISSN 0167–6423

[BSL+10] Berger, T.; She, S.; Lotufo, R.; Wąsowski, A.; Czarnecki, K.: Variability modeling in the real: a perspective from the operating systems domain. In: *Proceedings of the IEEE/ACM international conference on Automated software engineering* ACM, 2010, S. 73–82

[BSSP02] Beuche, D.; Spinczyk, O.; Schröder-Preikschat, W.: Fine-grain Application Specific Customization for Embedded Systems. In: *Proceedings of the IFIP WCC 2002 Stream 7 on Distributed and Parallel Embedded Systems (DIPES 2002)*, 2002, S. 25–30

[BST+07] Benavides, D.; Segura, S.; Trinidad, P.; Cortés, A. R.: FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In: *VaMoS* 2007 (2007), S. 01

[Buc10] Buchmann, T.: *Modelle und Werkzeuge für modellgetriebene Softwareproduktlinien am Beispiel von Softwarekonfigurationsverwaltungssystemen*, Dissertation, 2010

[Buu90] Buur, J.: *A theoretical approach to mechatronics design*, Technical University of Denmark (DTU), Dissertation, 1990

[BWM+06] Bhattacharya, P.; Welakwe, N. S.; Makanaboyina, R.; Chimalakonda, A.: Integration of CATIA with Modelica. In: *proceedings of Modelica conference*, 2006

[CAT] *CATIA? 3DEXPERIENCE® 3D Software - Dassault Systèmes®*. https://www.3ds.com/products-services/catia, . – Online; accessed on 06-April-2020

[CE00] Czarnecki, K.; Eisenecker, U. W.: *Generative Programming: Methods, Tools, and Applications*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000

[CEKJ+00] Cherki, S.; El Kaim, W.; Josset, P.; Paris, F.: Domain analysis and product-line scoping: a Thomson-CSF product-line

case study. In: *Software Product Lines: Economics, Architectures, and Implications* (2000)

[Cel91] Cellier, F. E.: *Continuous System Modeling*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1991. – ISBN 0387975020

[CI01] Clauss, M.; Intershop, J.: Modeling variability with UML. In: *GCSE 2001 Young Researchers Workshop* Citeseer, 2001

[CJB00] Coriat, M.; Jourdan, J.; Boisbourdin, F.: The SPLIT method. In: *Software Product Lines*. Springer, 2000, S. 147–166

[CK05] Czarnecki, K.; Kim, C. H. P.: Cardinality-based feature modeling and constraints: A progress report. In: *International Workshop on Software Factories*, 2005, S. 16–20

[Cla99] Clark, J.: *XSL Transformations (XSLT) Version 1.0, W3C Recommendation*. `http://www.w3.org/TR/1999/REC-xslt-19991116`. Version: November 1999

[CLP11] Cao, Y.; Liu, Y.; Paredis, C. J.: System-level model integration of design and simulation for mechatronic systems based on SysML. In: *Mechatronics* 21 (2011), Nr. 6, S. 1063–1075

[CN02] Clements, P.; Northrop, L.: *Software product lines*. Addison-Wesley„ 2002

[CS07] Chen, K.; Schaefer, D.: MCAD-ECAD Integration: Overview and Future Research Perspectives. In: *ASME International Mechanical Engineering Congress and Exposition, ASME*, 2007

[Dav95] Davis, M. J.: Adaptable, reusable code. In: *ACM SIGSOFT Software Engineering Notes* Bd. 20 ACM, 1995, S. 38–46

[dll19] *Was ist eine DLL?* `https://support.microsoft.com/de-de/help/815065/what-is-a-dll`, 2019. – Online; accessed on 06-April-2020

[DSB+04] Dean, M.; Schreiber, G.; Bechhofer, S.; Harmelen, F. van; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; Stein, L. A.: OWL web ontology language reference. In: *W3C Recommendation February* 10 (2004)

[DSB+11] Dhungana, D.; Seichter, D.; Botterweck, G.; Rabiser, R.; Grunbacher, P.; Benavides, D.; Galindo, J. A.: Configuration of multi product lines by bridging heterogeneous variability modeling approaches. In: *Software Product Line Conference (SPLC), 2011 15th International* IEEE, 2011, S. 120–129

[Dym] *Dassault systemes*. http://www.3ds.com/products-services/catia/products/dymola/, . – Online; accessed 06-April-2020

[EB12] Eppinger, S. D.; Browning, T. R.: *Design structure matrix methods and applications*. MIT press, 2012

[EBP+03] Engelson, V.; Bunus, P.; Popescu, L.; Fritzson, P.: Mechanical CAD with multibody dynamic analysis based on Modelica simulation. In: *Proceedings of the 44th Scandinavian Conference on Simulation and Modeling*, 2003, S. 18–19

[Ecl] *Eclipse - The Eclipse Foundation open source community website.* https://eclipse.org/, . – Online; accessed on 06-April-2020

[eco] *ecos-2.0-cdl-guide-a4.pdf*. http://www.gaisler.com/doc/ecos-2.0-cdl-guide-a4.pdf, . – Online; accessed on 06-April-2020

[EGK08] Eck, C.; Garcke, H.; Knabner, P.: *Mathematische Modellierung*. Bd. 2. Springer, 2008

[EKB+08] Erden, M. S.; Komoto, H.; Beek, T. J.; D'Amelio, V.; Echavarria, E.; Tomiyama, T.: A review of function modeling: Approaches and applications. In: *Ai Edam* 22 (2008), Nr. 2, S. 147–169

[Ent] *Enterprise Architect*. http://www.sparxsystems.com/products/ea/, . – Online; accessed 06-April-2020

[EUP] *Projekt Acosar: Weltweiter Standard für die Echtzeit-Co-Simulation*. https://www.springerprofessional.de/automobil---motoren/projekt-acosar-weltweiter-standard-fuer-die-echtzeit-co-simulati/7069626, . – Online; accessed on 06-April-2020

[FD13] Fosse, E.; Delp, C. L.: Systems engineering interfaces: A model based approach. In: *Aerospace Conference, 2013 IEEE*, 2013. – ISSN 1095–323X, S. 1–8

[FE98] Fritzson, P.; Engelson, V.: Modelica?A unified object-oriented language for system modeling and simulation. In: *European Conference on Object-Oriented Programming* Springer, 1998, S. 67–90

[FGT14] Falcone, A.; Garro, A.; Tundis, A.: System Dependability Analysis Through Platform-Independent Simulation Models, 2014

[FHP+10] Follmer, M.; Hehenberger, P.; Punz, S.; Zeman, K.: Using SysML in the product development process of mechatronic systems.

In: *DS 60: Proceedings of DESIGN 2010, the 11th International Design Conference, Dubrovnik, Croatia*, 2010

[FMI14] Modelica Association: *Functional Mock-up Interface for Model Exchange and Co-Simulation*. `https://www.fmi-standard.org/`. Version: 2014

[FMS14] Friedenthal, S.; Moore, A.; Steiner, R.: *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014

[FOR+09] Frey, E.; Ostrosi, E.; Roucoules, L.; Gomes, S.: Multi-domain product modelling: from requirements to cad and simulation tools. In: *INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'09* INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'09, 2009

[GDP+10] Gausemeier, J.; Dorociak, R.; Pook, S.; Nyssen, A.; Terfloth, A.: Computer-aided cross-domain modeling of mechatronic systems. In: *DS 60: Proceedings of DESIGN 2010, the 11th International Design Conference, Dubrovnik, Croatia*, 2010

[Ger90] Gero, J. S.: Design prototypes: a knowledge representation schema for design. In: *AI magazine* 11 (1990), Nr. 4, S. 26

[GFD+08] Gausemeier, J.; Frank, U.; Donoth, J.; Kahl, S.: Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme des Maschinenbaus. In: *Konstruktion* (2008)

[GFM01] Gausemeier, J.; Flath, M.; Mohringer, S.: Conceptual design of mechatronic systems supported by semi-formal specification. In: *Advanced Intelligent Mechatronics, 2001. Proceedings. 2001 IEEE/ASME International Conference on* Bd. 2 IEEE, 2001, S. 888–892

[Gie08] Gielingh, W.: An assessment of the current state of product data technologies. In: *Computer-Aided Design* 40 (2008), Nr. 7, S. 750–759

[GK04] Gero, J. S.; Kannengiesser, U.: The situated function–behaviour–structure framework. In: *Design studies* 25 (2004), Nr. 4, S. 373–391

[GLL12] Gausemeier, J. (Hrsg.); Lanza, G. (Hrsg.); Lindemann, U. (Hrsg.): *Produkte und Produktionssysteme integrativ konzipieren – Modellbildung und Analyse in der frühen Phase der Produktentstehung*. München: Carl Hanser Verlag, 2012

[GPW09] Gausemeier, J.; Plass, C.; Wenzelmann, C.: *Zukunftsorientierte Unternehmensgestaltung – Strategien, Geschäftsprozesse und IT-Systeme für die Produktion von morgen.* München: Carl Hanser Verlag, 2009

[Gri97] Griss, M. L.: Software reuse architecture, process, and organization for business success. In: *Computer systems and software engineering, 1997., proceedings of the eighth israeli conference on* IEEE, 1997, S. 86–89

[GSG+09] Gausemeier, J.; Schäfer, W.; Greenyer, J.; Kahl, S.; Pook, S.; Rieke, J.: Management of Cross-Domain Model Consistency During the Development of Advanced Mechatronic Systems. In: Bergendahl, M. N. (Hrsg.); Grimheden, M. (Hrsg.); Leifer, L. (Hrsg.): *Proceedings of the 17th International Conference on Engineering Design* Bd. 6, Design Methods and Tools, Design Society, August 2009 ((ICED'09)), S. 1–12

[GTS14a] Gausemeier, J. (Hrsg.); Trächtler, A. (Hrsg.); Schäfer, W. (Hrsg.): *Semantische Technologien im Entwurf mechatronischer Systeme: Effektiver Austausch von Lösungswissen in Branchenwertschöpfungsketten.* München: Hanser, 2014. – ISBN 978–3–446–43630–5

[GTS14b] Gausemeier, J. (Hrsg.); Trächtler, A. (Hrsg.); Schäfer, W. (Hrsg.): *Semantische Technologien im Entwurf mechatronischer Systeme: Effektiver Austausch von Lösungswissen in Branchenwertschöpfungsketten.* München: Carl Hanser Verlag, 2014. – Autoren: Anacker, H.; Bauer, F.; Borcherding, H.; Dziwok, S.; Frank, U.; Gausemeier, J.; Herden, R.; Hoppe, G.; Just, V.; Kiele-Dunsche, M.; Kruse, D.; Oestersötebier, F.; Papenfort, J.; Pohlmann, U.; Reddehase, H.; Rieke, J.; Schäfer, W.; Schierbaum, T.; Seifert, L.; Stichweh, H.; Teichrieb, H.; Trächtler, A.; Wagner, R.; Wessels, S.

[HC+88] Hauser, J. R.; Clausing, D. u. a.: The house of quality. (1988)

[HMP04] Hooman, J.; Mulyar, N.; Posta, L.: Coupling Simulink and UML models. In: *Proc. Symposium FORMS/FORMATS*, 2004, S. 304–311

[HRG11] Han, S.; Rüdenauer, A.; Geimer, M.: Die GUSMA-Plattform: In sechs Schritten zum virtuellen Produkt. In: *Mobile Maschinen* 2 (2011), S. 32–34

[IKD+13] Iwanek, P.; Kaiser, L.; Dumitrescu, R.; Nyssen, A.: Fachdisziplinübergreifende Systemmodellierung mechatronischer

Systeme mit SysML und CONSENS. In: Maurer, M. (Hrsg.); Schulze, S.-O. (Hrsg.): *Tag des Systems Engineerings*, Carl Hanser Verlag, November 2013, S. 337–346

[Ing13] Ingenierure, V. D.: Simulation von Logistik-, Materialfluss- und Produktionssystemen-Begriffe. 01. In: *Aufl.(Dezember 2013)* (2013)

[iQU] *ISID and iTiD announce the latest edition of "iQUAVIS" conception design support system for manufacturing industry - CNET Japan.* https://japan.cnet.com/release/30051537/, . – Online; accessed on 06-April-2020

[Ise08] Isermann, R.: *Mechatronische Systeme – Grundlagen.* 2., vollst. neu bearb. Aufl. Springer, 2008

[Jan10] Janschek, K.: *Systementwurf mechatronischer Systeme: Methoden – Modelle – Konzepte.* Berlin: Springer, 2010. http://dx.doi.org/10.1007/978-3-540-78877-5. http://dx.doi.org/10.1007/978-3-540-78877-5. – ISBN 978–3–540–78876–8

[JFC+14] Jakšić, A.; France, R. B.; Collet, P.; Ghosh, S.: Evaluating the Usability of a Visual Feature Modeling Notation. In: Combemale, B. (Hrsg.); Pearce, D. J. (Hrsg.); Barais, O. (Hrsg.); Vinju, J. J. (Hrsg.): *Software Language Engineering.* Cham: Springer International Publishing, 2014. – ISBN 978–3–319–11245–9, S. 122–140

[JKP+12] Johnson, T.; Kerzhner, A.; Paredis, C. J.; Burkhart, R.: Integrating models and simulations of continuous dynamics into SysML. In: *Journal of Computing and Information Science in Engineering* 12 (2012), Nr. 1, S. 011002

[JT10] John, K. H.; Tiegelkamp, M.: *IEC 61131-3: Programming Industrial Automation Systems Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids.* 2nd. Springer Publishing Company, Incorporated, 2010. – ISBN 3642120148, 9783642120145

[Kai14] Kaiser, L.: *Rahmenwerk zur Modellierung einer plausiblen Systemstruktur mechatronischer Systeme*, Fakultät für Maschinenbau, Universität Paderborn, Dissertation, 2014

[KCH+90] Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S.: Feature-oriented domain analysis (FODA) feasibility study / Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst. 1990. – Forschungsbericht

[ker] *www.kernel.org*. `https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt`, . – Online; accessed on 06-April-2020

[KP10] Kerzhner, A. A.; Paredis, C. J.: Model-based system verification: a formal framework for relating analyses, requirements, and tests. In: *International Conference on Model Driven Engineering Languages and Systems* Springer, 2010, S. 279–292

[Kuf11] Kufner, A.: *Automatisierte Erstellung von Maschinenmodellen für die Hardware-in-the-Loop-Simulation von Montagemaschinen*, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart, Dissertation, 2011

[LAL+09] Liebig, J.; Apel, S.; Lengauer, C.; Leich, T.: RobbyDBMS: a case study on hardware/software product line engineering. In: *Proceedings of the First International Workshop on Feature-Oriented Software Development* ACM, 2009, S. 63–68

[LK99] Law, A. M.; Kelton, D. M.: *Simulation Modeling and Analysis*. 3rd. McGraw-Hill Higher Education, 1999. – ISBN 0070592926

[Loc] Lochbichler, M.: *Systematische Wahl einer Modellierungstiefe im Entwurfsprozess mechatronischer Systeme*, Universität Paderborn, Dissertation. – noch nicht veröffentlicht

[LOT14] Lochbichler, M.; Oestersötebier, F.; Trächtler, A.: Dynamic Behavior Models and their Modeling Depth in the Design Process of Mechatronic Systems. In: *Proceedings of the ASME 2014 International Mechanical Engineering Congress & Exposition IMECE 2014*. Montreal, Québec, Kanada: ASME, November 2014, S. V011T14A051

[LSB+12] Lochbichler, M.; Schmüdderrich, T.; Brökelmann, J.; Trächtler, A.: Methodology For Selecting The Modeling Depth Of Object-Oriented Behavioral Models. In: *International Conference on Modeling and Simulation*. Zürich: World Academy of Science, Engineering and Technology, Juli 2012, S. 327–331

[LSR07] Linden, F. J. v. d.; Schmid, K.; Rommes, E.: *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. – ISBN 3540714367

[Mat] *Mathworks*. `http://www.mathworks.com/products/simulink/`, . – Online; accessed 06-April-2020

[MBC09] Mendonca, M.; Branco, M.; Cowan, D.: SPLOT: software product lines online tools. In: *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* ACM, 2009, S. 761–762

[MLS+00] McComas, D.; Leake, S.; Stark, M.; Morisio, M.; Travassos, G.; White, M.: Addressing variability in a guidance, navigation, and control flight software product line. In: *Product Line Architecture Workshop at Software Product Line Conference (SPLC1, Proceedings)*, 2000

[MMC+10] Maxwell, B.; McNeilly, S.; Capey, N.; Ugavina, N.; Kumar, V.; Xie, S.; Mancarella, S.: Extending UML With Enterprise Architect. In: *Aerospace Conference, 2013 IEEE*, Sparx Systems, 2010

[MMP+04] Mocko, G.; Malak, R. J.; Paredis, C. J.; Peak, R.: A Knowledge Repository for Behavioral Models in Engineering Design. In: *ASME Design Engineering Technical Conf., Computers and Information in Engineering Conf.* American Society of Mechanical Engineers (ASME), 2004, S. 943–952

[Moda] *FMI Toolbox for MATLAB/Simulink*. `http://www.modelon.com/products/fmi-toolbox-for-matlab/`, . – Online; accessed 06-April-2020

[Modb] *modeFrontier | ESTECO*. `http://www.esteco.com/modefrontier`, . – Online; accessed on 06-April-2020

[Modc] *ModelCenter Integrate | Model Based Engineering Software | Phoenix Integration*. `https://www.phoenix-int.com/product/modelcenter-integrate/`, . – Online; accessed on 06-April-2020

[MS10] Munir, Q.; Shahid, M.: *Software Product Line: Survey of Tools*. 2010

[MTS00] Morisio, M.; Travassos, G. H.; Stark, M. E.: Extending UML to support domain analysis. In: *Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on* IEEE, 2000, S. 321–324

[mus] *6_PS_Simulationsprogramme_Vergleich_Muschau.pdf*. `https://docplayer.org/38670211-Simulationsprogramme-im-vergleich-andreas-musch html`, . – Online; accessed on 06-April-2020

[Nik07] Nikravesh, P. E.: *Planar Multibody Dynamics: Formulation, Programming and Applications*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 2007. – ISBN 1420045725, 9781420045727

[NSH+08] Nagel, R. L.; Stone, R. B.; Hutcheson, R. S.; McAdams, D. A.; Donndelinger, J. A.: Function design framework (FDF): integrated process and function modeling for complex systems. In: *ASME 2008 international design engineering technical conferences and computers and information in engineering conference, IDETC/CIE*, 2008, S. 273–286

[OAL+16] Oestersötebier, F.; Abrishamchian, F.; Lankeit, C.; Just, V.; Trächtler, A.: Approach for an Integrated Model-based Design of Intelligent Dynamic Systems Using Solution and System Knowledge. In: *Procedia Technology* 26 (2016), 436 - 446. `http://dx.doi.org/http://dx.doi.org/10.1016/j.protcy.2016.08.056`. – DOI http://dx.doi.org/10.1016/j.protcy.2016.08.056. – ISSN 2212–0173. – 3rd International Conference on System-Integrated Intelligence: New Challenges for Product and Production Engineering

[Obj11] Object Management Group (OMG): *Unified Modeling Language (UML) 2.4.1 Superstructure Specification (formal/2011-08-06)*, 2011

[Oes17] Oestersötebier, F.: *Modellbasierter Entwurf intelligenter mechatronischer Systeme mithilfe semantischer Technologien*, University Paderborn, Dissertation, 2017

[PBF+13] Pahl, G. (Hrsg.); Beitz, W. (Hrsg.); Feldhusen, J. (Hrsg.); Grote, K.-H. (Hrsg.): *Pahl/Beitz Konstruktionslehre: Methoden und Anwendung erfolgreicher Produktentwicklung*. 8. Berlin: Springer Verlag, 2013

[PBL05] Pohl, K.; Böckle, G.; Linden, F. Van d.: *Software product line engineering : foundations, principles, and techniques*. Berlin, New York: Springer, 2005 `https://www.springer.com/de/book/9783540243724`. – ISBN 3–540–24372–0

[PC02] Pritschow, G.; Croon, N.: *Wege zur virtuellen Werkzeugmaschine - Verschiedene Betrachtungsweisen und Modellierungsansätze.wt Werkstattstechnik online 92 (2002) 5, S. 194...199. Wege zur virtuellen Werkzeugmaschine - Verschiedene Betrachtungsweisen und Modellierungsansätze.wt Werkstattstechnik online 92 (2002) 5, S. 194...199.* 2002

[PCF14] In: Pereira, J. A.; Constantino, K.; Figueiredo, E.: *A Systematic Literature Review of Software Product Line Management Tools*. Cham: Springer International Publishing, 2014. – ISBN 978–3–319–14130–5, 73–89

[PDS+01] Paredis, C.; Diaz-Calderon, A.; Sinha, R.; Khosla, P.: Composable Models for Simulation-Based Design. In: *Engineering with Computers* 17 (2001), S. 112–128

[Pra01] Pratt, M. J.: Introduction to ISO 10303?the STEP standard for product data exchange. In: *Journal of Computing and Information Science in Engineering* 1 (2001), Nr. 1, S. 102–103

[PRR05] Pritschow, G.; Röck, S.; Rüdele, H.: *Echtzeitfähige Simulation von Werkzeugmaschinen.wt Werkstattstechnik online, Jahrgang 95, Heft 5, S. 302...308. Mai 2005. Echtzeitfähige Simulation von Werkzeugmaschinen.wt Werkstattstechnik online, Jahrgang 95, Heft 5, S. 302...308. Mai 2005.* 2005

[pur] *pure-systems - The leading provider of software for product line and variant management tools | pure::variants.* `http://www.pure-systems.com/products/pure-variants-9.html`, . – Online; accessed on 06-April-2020

[Qam13] Qamar, A.: *Model and Dependency Management in Mechatronic Design*, KTH-Royal Institute of Technology, Dissertation, 2013

[Rec] *Recurdyn solution 4 engineers.* `http://www.functionbay.org/`, . – Online; accessed 06-April-2020

[RJFLC03] R Judson, S.; France, R.; L Carver, D.: Specifying model transformations at the metamodel level. (2003), 01

[RS10] Rosenmüller, M.; Siegmund, N.: Automating the Configuration of Multi Software Product Lines. In: *VaMoS* 10 (2010), S. 123–30

[RT12] Rahmani, K.; Thomson, V.: Ontology Based Interface Design and Control Methodology for Collaborative Product Development. In: *Comput. Aided Des.* 44 (2012), Mai, Nr. 5, 432–444. `http://dx.doi.org/10.1016/j.cad.2011.12.002`. – DOI 10.1016/j.cad.2011.12.002. – ISSN 0010–4485

[RW06] Reiser, M.-O.; Weber, M.: Managing highly complex product families with multi-level feature trees. In: *Requirements Engineering, 14th IEEE International Conference* IEEE, 2006, S. 149–158

[SAM12] Sayyad, A.; Ammar, H.; Menzies, T.: Software Feature Model recommendations using data mining. In: *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, 2012, S. 47–51

[Sca19] *SCADE Suite - Control Software Design | Esterel Technologies*. http://www.esterel-technologies.com/products/scade-suite/, 2019. – Online; accessed on 06-April-2020

[Sch13] Schamai, W.: *Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool*. Department of Computer and Information Science, Linköping University, Doctoral thesis No 1547, 2013. http://dx.doi.org/10.3384/diss.diva-98107. – DOI 10.3384/diss.diva–98107

[SH09] Stober, T.; Hansmann, U.: *Agile Software Development: Best Practices for Large Software Development Projects*. 1st. Springer Publishing Company, Incorporated, 2009. – ISBN 3540708308, 9783540708308 Titel anhand dieser ISBN in Citavi–Projekt übernehmen

[Sha98] Sharp, D. C.: Reducing avionics software cost through component based product line development. In: *Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE* Bd. 2 IEEE, 1998, S. G32–1

[SHF+10] Schamai, W.; Helle, P.; Fritzson, P.; Paredis, C. J.: Virtual verification of system designs against system requirements. In: *International Conference on Model Driven Engineering Languages and Systems* Springer, 2010, S. 75–89

[SJT+15] Shareef, Z.; Just, V.; Teichrieb, H.; Trächtler, A.: Design and Control of Cooperative Ball Juggling Delta Robots without Visual Guidance. In: *Robotica* (2015), Juni, S. 1–17

[SKS+10] Shah, A. A.; Kerzhner, A. A.; Schaefer, D.; Paredis, C. J.: Multi-view modeling to support embedded systems engineering in SysML. In: *Graph transformations and model-driven engineering*. Springer, 2010, S. 580–601

[SLB+13] Schmüdderrich, T.; Lochbichler, M.; Brökelmann, J.; Trächtler, A.: Methodik zur anforderungsgerechten Wahl der Modellierungstiefe von Verhaltensmodellen für die virtuelle Inbetriebnahme. In: *Tagungsband Mechatronik 2013*, 2013, S. 43–48

[SLN+12] Simko, G.; Levendovszky, T.; Neema, S.; Jackson, E.; Bapty, T.; Porter, J.; Sztipanovits, J.: Foundation for model integration: Semantic backplane. In: *ASME IDETC/CIE* (2012)

[SR09] Sage, A. P.; Rouse, W. B.: *Handbook of systems engineering and management*. John Wiley & Sons, 2009

[SS10] In: Safavi, S.; Shaikh, M.: *Effort estimation model for each phase of software development life cycle*. 2010, S. 270–277

[SSP09] Shah, A. A.; Schaefer, D.; Paredis, C.: Enabling multi-view modeling with sysml profiles and model transformations. In: *The 6th International Conference on Product Lifecycle Management* University of Bath, 2009, S. 527–538

[SST+07] Sjöstedt, C.-J.; Shi, J.; Törngren, M.; Servat, D.; Chen, D.; Ahlsten, V.; Lönn, H.: Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and transformations. (2007)

[STB+13] Schmüdderrich, T.; Trächtler, A.; Brökelmann, J.; Gausemeier, J.: Procedural Model for the Virtual Commissioning on the Basis of Model-Based Design. In: Abramovici, M. (Hrsg.); Stark, R. (Hrsg.): *Smart Product Engineering*. Springer Berlin Heidelberg, 2013 (Lecture Notes in Production Engineering), S. 23–32

[Str04] Streitferdt, D.: *Family-oriented requirements engineering*, Dissertation, 2004

[Suh01] Suh, N. P.: Axiomatic design: Advances and applications (the oxford series on advanced manufacturing). (2001)

[SV89] Salminen, V.; Verho, A.: Multi-disciplinary design problems in mechatronics and some suggestions to its methodical solution in conceptual design phase. In: *International Conference on Engineering Design (ICED89)* Bd. 1, 1989, S. 533–554

[Tas95] Task, I.: SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM. (1995)

[TDU+07] Tomiyama, T.; D?Amelio, V.; Urbanic, J.; ElMaraghy, W.: Complexity of Multi-Disciplinary Design. In: *CIRP Annals - Manufacturing Technology* 56 (2007), Nr. 1, 185 - 188. `http://dx.doi.org/http://dx.doi.org/10.1016/j.cirp.2007.05.044`. – DOI http://dx.doi.org/10.1016/j.cirp.2007.05.044. – ISSN 0007–8506

[TEkB+12] Törngren, M.; El-khoury, J.; Brodtkorb, D.; Dahle, H. P.: Systematic and cost-efficient tool integration for embedded systems: the iFEST approach. In: *ARTEMIS Technology Conference 2012*, 2012

[The14] The Eclipse Foundation: *EMF Feature Model, Eclipsepedia*. Online. `https://wiki.eclipse.org/EMF_Feature_Model`. Version: 2014. – https://wiki.eclipse.org/EMF_Feature_Model Online; accessed 06-April-2020

[TKB+14] Thüm, T.; Kästner, C.; Benduhn, F.; Meinicke, J.; Saake, G.; Leich, T.: FeatureIDE: An extensible framework for feature-oriented software development. In: *Science of Computer Programming* 79 (2014), S. 70–85

[TKE+11] Thum, T.; Kastner, C.; Erdweg, S.; Siegmund, N.: Abstract features in feature modeling. In: *Software Product Line Conference (SPLC), 2011 15th International* IEEE, 2011, S. 191–200

[Trä14] Trächtler, A.: *Mechatronics (lecture manuscript WS 2013/2014)*. 2014

[Tra13] Trampe, H.: *Modeling and Simulation - Overview*, University Hamburg, Informatic Scientific Computing, Project work, 2013

[TRC06] Toni Rodrigues, P. R.; Cardoso, J.: Mapping XML to existing OWL ontologies. In: *International Conference WWW/Internet Citeseer*, 2006, S. 72–77

[tSH+05] **Streitferdt**, D.; Sochos, P.; Heller, C.; Philippow, I.: Configuring embedded system families using feature models. In: *Proc. of Net. ObjectDays*, 2005, S. 339–350

[Twi] *Beckhoff*. `https://infosys.beckhoff.com/index_en.htm`, . – Online; accessed 06-April-2020

[UKS+05] Umeda, Y.; Kondoh, S.; Shimomura, Y.; Tomiyama, T.: Development of design methodology for upgradable products based on function–behavior–state modeling. In: *Ai Edam* 19 (2005), Nr. 3, S. 161–182

[UTT+90] Umeda, Y.; Takeda, H.; Tomiyama, T.; Yoshikawa, H.: Function, behaviour, and structure. In: *Applications of artificial intelligence in engineering V* 1 (1990), S. 177–193

[VD06] Vanderperren, Y.; Dehaene, W.: From UML/SysML to Matlab/Simulink: current state and future perspectives. In: *Proceedings of the conference on Design, automation and test in Europe: Proceedings* European Design and Automation Association, 2006, S. 93–93

[VDI04] Richtlinie VDI2206 Juni 2004. *Entwicklungsmethodik für mechatronische Systeme*

[Ver04] Verein Deutscher Ingenieure (VDI): VDI Guideline 2206: Design Methodology for Mechatronic Systems. In: *Beuth Verlag* (2004)

[VHG10] Völker, L.; Han, S.; Geimer, M.: Unternehmensübergreifende Simulation mobiler Arbeitsmaschinen: Ein Vorschlag für eine standardisierte Vorgehensweise. In: *1st Commercial Vehicle Technology Symposium, Shaker, Aachen*, 2010, S. 407–416

[VIN] *VI Network*. https://www.vi.net/vi-infrastructure/vi-network/,

[W+99] Weiss, D. M. u. a.: Software product-line engineering: a family-based software development process. (1999)

[WACT+10] Woestenenk, K.; Alvarez Cabrera, A. A.; Tragter, H.; Tomiyama, T.; Bonnema, G. M.: Multi domain design: integration and reuse ASME, 2010

[WDS+09] White, J.; Dougherty, B.; Schmidt, D. C.; Benavides, D.: Automated reasoning for multi-step feature model configuration problems. In: *Proceedings of the 13th International Software Product Line Conference* Carnegie Mellon University, 2009, S. 11–20

[Wei07] Weilkiens, T.: *Systems engineering with SysML/UML: modeling, analysis, design*. Burlington, MA, USA: Morgan Kaufmann Publishers, 2007

[Wha] *What Is an S-Function? - MATLAB & Simulink*. `https://www.mathworks.com/help/simulink/sfg/what-is-an-s-function.html`, . – Online; accessed on 06-April-2020

[Win] *WindowBuilder*. `https://eclipse.org/windowbuilder/`, . – Online; accessed on 06-April-2020

[WMR+11] Wang, S.; Morin, B.; Roman, D.; Berre, A.-J.: A Semi-automatic Transformation Approach for Semantic Interoperability in MDE, 2011

[WS09] Wölkl, S.; Shea, K.: A computational product model for conceptual design using SysML. In: *ASME Paper No. DETC2009-87239* (2009)

[ZBLD+14] Zheng, C.; Bricogne, M.; Le Duigou, J.; Eynard, B.: Survey on Mechatronic Engineering: A Focus on Design Methods and Product Models. In: *Adv. Eng. Inform.* 28 (2014), August, Nr. 3, 241–257. `http://dx.doi.org/10.1016/j.aei.2014.05.003`. – DOI 10.1016/j.aei.2014.05.003. – ISSN 1474–0346

A Attachment

A.1 ModelXML Schema

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"\\ xmlns:xs="http://www.w3.org/2001/XMLSchema">\\
  <xs:element name="modelxml">\\
    <xs:complexType>\\
      <xs:sequence>\\
        <xs:element name="package">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="fullPath"/>
              <xs:element type="xs:string" name="packageName"/>
              <xs:element name="model">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="modelID"/>
                    <xs:element type="xs:string" name="fullPath"/>
                    <xs:element type="xs:string" name="modelType"/>
                    <xs:element type="xs:string" name="modelName"/>
                    <xs:element type="xs:string" name="solver"/>
                    <xs:element type="xs:float" name="stepsize"/>
                    <xs:element type="xs:float" name="modelingDepth"/>
                    <xs:element name="component" maxOccurs="unbounded" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="SystemElementModel">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element type="xs:string" name="SystemElementName"/>
                                <xs:element type="xs:string" name="SystemElementID"/>
                                <xs:element type="xs:string" name="modelID"/>
                                <xs:element name="modifier" maxOccurs="unbounded" minOccurs="0">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element type="xs:string" name="componentName"/>
                                      <xs:element type="xs:string" name="componentValue"/>
                                    </xs:sequence>
```

```
35                                    </xs:complexType>
36                                  </xs:element>
37                                </xs:sequence>
38                              </xs:complexType>
39                            </xs:element>
40                          </xs:sequence>
41                        </xs:complexType>
42                      </xs:element>
43                      <xs:element type="xs:string" name="
    modelLanguage"/>
44                    </xs:sequence>
45                  </xs:complexType>
46                </xs:element>
47                <xs:element type="xs:string" name="SystemElementModel"
     maxOccurs="unbounded" minOccurs="0"/>
48                <xs:element name="package" maxOccurs="unbounded"
    minOccurs="0">
49                  <xs:complexType>
50                    <xs:sequence>
51                      <xs:element type="xs:string" name="fullPath"/>
52                      <xs:element type="xs:string" name="packageName"
    />
53                      <xs:element name="model">
54                        <xs:complexType>
55                          <xs:sequence>
56                            <xs:element type="xs:string" name="
    modelID"/>
57                            <xs:element type="xs:string" name="
    fullPath"/>
58                            <xs:element type="xs:string" name="
    modelType"/>
59                            <xs:element type="xs:string" name="
    modelName"/>
60                            <xs:element type="xs:string" name="
    solver"/>
61                            <xs:element type="xs:float" name="
    stepsize" minOccurs="0"/>
62                            <xs:element type="xs:float" name="
    modelingDepth" minOccurs="0"/>
63                            <xs:element name="component" maxOccurs="
    unbounded" minOccurs="0">
64                              <xs:complexType>
65                                <xs:sequence>
66                                  <xs:element name="parameter"
    minOccurs="0">
67                                    <xs:complexType>
68                                      <xs:sequence>
69                                        <xs:element type="xs:string"
     name="parameterName"/>
```

```
70                                        <xs:element type="xs:string"
      name="parameterID"/>
71                                        <xs:element type="xs:string"
      name="modelicaType"/>
72                                        <xs:element type="xs:string"
      name="parameterDescription"/>
73                                        <xs:element type="xs:float"
      name="defaultValue"/>
74                                        <xs:element type="xs:string"
      name="ENTIME"/>
75                                      </xs:sequence>
76                                    </xs:complexType>
77                                  </xs:element>
78                                  <xs:element name="interface"
     minOccurs="0">
79                                    <xs:complexType>
80                                      <xs:sequence>
81                                        <xs:element type="xs:string"
      name="interfaceName"/>
82                                        <xs:element type="xs:string"
      name="interfaceID"/>
83                                        <xs:element type="xs:string"
      name="modelicaType"/>
84                                        <xs:element type="xs:string"
      name="interfaceType"/>
85                                      </xs:sequence>
86                                    </xs:complexType>
87                                  </xs:element>
88                                </xs:sequence>
89                              </xs:complexType>
90                            </xs:element>
91                            <xs:element type="xs:string" name="
     annotation" minOccurs="0"/>
92                            <xs:element type="xs:string" name="
     modelLanguage"/>
93                          </xs:sequence>
94                        </xs:complexType>
95                      </xs:element>
96                      <xs:element type="xs:string" name="
     SystemElementModel" minOccurs="0"/>
97                    </xs:sequence>
98                  </xs:complexType>
99                </xs:element>
100             </xs:sequence>
101           </xs:complexType>
102         </xs:element>
103       </xs:sequence>
104     </xs:complexType>
105   </xs:element>
```

```
106 </xs:schema>
```

Anlage in der Dissertation:
Zitation von studentischen Arbeiten

Erklärung zur Zitation von Inhalten aus studentischen Arbeiten

In Ergänzung zu meinem Antrag auf Zulassung zur Promotion in der Fakultät für Maschinenbau der Universität Paderborn erkläre ich gemäß §11 der Promotionsordnung und unter Beachtung der Regelung zur Zitation studentischer Arbeiten:

Die von mir vorgelegte Dissertation habe ich selbstständig verfasst, **und ich habe keine anderen** als die dort angegebenen Quellen und Hilfsmittel benutzt. Es sind ~~**Inhalte**~~ / **keine Inhalte** studentischen Ursprungs (studentische Arbeiten) in dieser Dissertation enthalten.

Ich habe die verwendeten Arbeiten entsprechend der Regelung „Zitation aus studentischen Arbeiten in Dissertationen" zitiert.

Ort, Datum: Köln, 17.06.2026

Unterschrift