Architecture-Centric Design: Modeling and Applications to Control Architecture Generation

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 10 oktober 2011 om 10:00 uur
door

Andrés Alberto ALVAREZ CABRERA
Diplôme D'Ingénieur, Génie Mécanique Conception,
Institut National des Sciences Appliquées de Lyon

geboren te Bogotá, Colombia

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. T. Tomiyama

Samenstelling promotiecommissie:

Rector Magnificus	Voorzitter
Prof. dr. T. Tomiyama	Technische Universiteit Delft, promotor
Prof. dr.ir. M.J.L. van Tooren	Technische Universiteit Delft
Prof. dr.ir. F.J.A.M. van Houten	Universiteit Twente
Prof. dr.ir. J.M.P. Geraedts	Technische Universiteit Delft
Dr. C. Paredis	Georgia Institute of Technology, Verenigde Staten van Amerika
Dr. D.A. van Beek	Technische Universiteit Eindhoven
Dr.ir. J.L. Herder	Technische Universiteit Delft
Prof. dr.ir. P.P. Jonker	Technische Universiteit Delft, reservelid

This research has been carried with the support of the Innovation-Oriented Research Programme 'Integral Product Creation and Realization (IOP IPCR)' of the Netherlands Ministry of Economic Affairs, Agriculture and Innovation.

ISBN: 978-90-6562-281-5

To my growing family

Through our mere existence, all living
beings defy the fate of the universe,
predicted by thermodynamics to be chaos.
However, intelligent beings have further
choice, ranging from sustainably ordering
the surrounding universe (creation),
to accelerating on the road to chaos up to
the point it consumes us (destruction).

The choice is ours…

Andrés A. Alvarez Cabrera

Abstract

Design activities, including control design, are becoming increasingly difficult due to a corresponding increase in product and product development complexity. Model-based (or driven) engineering, development and design have become common concepts related to modern complex product development practices. However, it is argued here that currently such approaches only remain successful within a domain-specific context. This work has as main contributions the analysis of desirable characteristics and a proposal for a model which can effectively support model-based development in general (i.e., not only within specific domains), coined here as "architecture-centric". Another contribution of this work is an intensive review (though hardly complete) on existing tools and methods related to the model-based development of control architectures for complex mechatronic systems.

Synthesis, analysis, and verification of the proposals are based on the generic case of control (architecture) design, which represents most of the relevant characteristics and problems in current design practices for complex mechatronic products. Besides the main contributions above, the case studies for control architecture generation provide an overview of the control design process, as well as additional insight into the required characteristics of the model and possible methods to effectively implement it and use it in the context of industrial product development.

Samenvatting

Ontwerp activiteiten, inclusief regelsysteem ontwerp, worden steeds moeilijker door toename van zowel product als product ontwikkeling complexiteit tegelijkertijd. Model gebaseerd (of gedreven) technische ontwikkeling en ontwerp zijn gemeengoed geworden in hedendaagse, complexe product ontwikkelingen. Er kan echter worden gesteld dat dergelijke aanpakken slechts succesvol worden toegepast in domein specifieke context. De voornaamste bijdragen van dit werk zijn een analyse van de gewenste karakteristieken en een voorstel van een model dat effectief model gebaseerde ontwikkeling ondersteund in het algemeen (d.w.z. niet slechts in een domein specifieke context), hier "architectuur-centrisch" genoemd. Een andere bijdrage is een uitvoerig (echter nauwelijks volledig) literatuur onderzoek naar bestaande instrumenten en methodes op het gebied van model gebaseerde ontwikkeling van regelsysteem architecturen van complexe mechatronische systemen.

Synthese, analyse en verificatie van de voorstellen zijn gebaseerd op het ontwerpen van regelsystemen(architectuur) in het generieke geval, waarin de meeste van de relevante kenmerken en problemen van de huidige ontwerppraktijk voor complexe mechatronische producten vertegenwoordigd zijn. Naast bovengenoemde bijdragen geven de case studies, waarin regelsysteem architecturen zijn gegenereerd, een overzicht van het regelsysteem ontwerpproces. Daarnaast geven de case studies verbredende inzichten in de benodigde model eigenschappen en mogelijke methodes voor effectieve implementatie en toepassing in een industriële productontwikkeling context.

Preface

Here, I provide an overview of my activities during the last years, trying to give the reader another point of view and some insight into how this document has been conceived. The work for this thesis started with the goal of the project for which I was hired at the Technical University of Delft as part of the project "Automatic Control Software Generation for Mechatronic Systems": To generate input information for control code analysis models out of a high-level model. At that time, the goal and the means to achieve it were mainly defined through a diagram (*cf.* Figure 1) describing an overview of a design process supported by this high-level model (i.e., the function model in the figure). My job was to develop the block corresponding to the functional model and to provide enough input information for the control design processes through the use of artificial intelligence techniques (qualitative-reasoning).

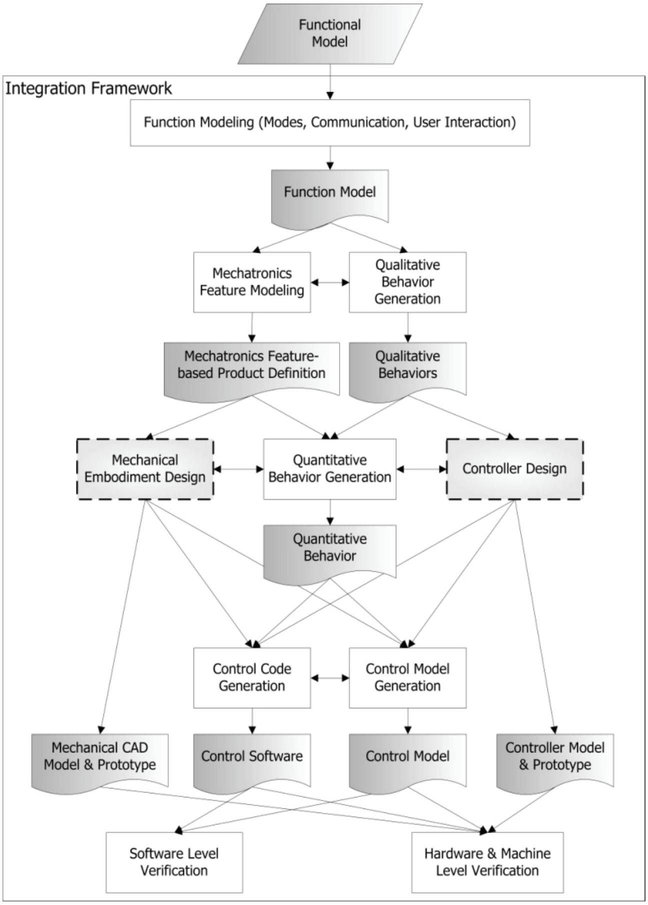

Functional Model
Integration Framework
Function Modeling (Modes, Communication, User Interaction)
Function Model
Mechatronics Feature Modeling
Qualitative Behavior Generation
Mechatronics Feature-based Product Definition
Qualitative Behaviors
Mechanical Embodiment Design
Quantitative Behavior Generation
Controller Design
Quantitative Behavior
Control Code Generation
Control Model Generation
Mechanical CAD Model & Prototype
Control Software
Control Model
Controller Model & Prototype
Software Level Verification
Hardware & Machine Level Verification

Figure 1. Architecture of integration framework. White blocks represent tools to be further developed. Dashed-line blocks correspond to existing commercial software tools. Iterations are not shown.

The first period of my PhD was spent gathering and reading related material which, to my surprise, covered a wide range of literature from fundamental design theory to control design methods, passing by artificial intelligence techniques to automatically transform information. After gaining some basic understanding about the models I could use as input and the models I needed to obtain as an output, the work began by trying to model a couple of different systems, discovering that maintaining usability and readability of the input model and placing enough information to achieve my goals of model generation was rooted on:

- selecting a proper group of generic modeling primitives
- providing basic modeling interface mechanisms which allowed to handle parts of the information while maintaining its connection to other data in the model

A first set of primitives was provided by the Function-Behavior-State (FBS) model methodology developed by Professor Tomiyama (my promoter) and his colleagues. The first tests proceeded by modeling with tools like Visio and some SysML modeling software. Visio allowed to quickly produce graphic models, but failed to facilitate managing parts of the information while keeping them connected to a single model. Thus, when I presented the Visio models to the project's industrial partners, the point of maintaining the information linked bellow was completely lost, and I could make very little practical progress. SysML allowed creating very rich models using a graphical input, but much of the modeling effort had to be spent creating formal class representations in the different diagrams to try obtaining a "complete" and coherent model. Thus, actually modeling any system was not easy with any of these tools, and required an immense amount of effort.

The first experiences modeling with the available tools provided more confidence in the modeling primitives, but made evident the need for a "tool" to better support the modeling. Through the Graphical Modeling Framework (GMF), Eclipse provided a fast means of describing our primitive modeling classes and implementing a basic tool to create instances of them and actually model systems. The resulting tool was far from perfect, but allowed to demonstrate the main points of building an integrated information model for the generation of other models. Then, some of the industrial partners could see more clearly the advantages of the proposed approach and the first case studies were successfully implemented, modeling part of a system and demonstrating how the information could be used to generate other models for analysis and synthesis. Also, an important change to the modeling primitives was made by recognizing the parameter as an information object that allows relating the other primitives and building explicit descriptions of the system. The work up to this point constitutes the first contribution of this thesis by recognizing the set of modeling primitives and providing a research tool (not robust enough for industrial application) which allows building models using such primitives.

Then the generation of control design models could be addressed more directly. The first step was to identify more clearly the information which is available from the first stages of development and which relates to control design: structural and topological relations of parts of the system, its functions and their defining parameters. Then, I had to select some candidate target models that could take as input such information after some transformation, and that would help verifying that it is indeed enough for starting to develop a controller. The transformations impose additional constraints to the model, especially to build descriptions of the behavior. This part forms the second contribution of this thesis, by providing some modeling rules that allow modeling unambiguously enough information to generate the target control design models.

Delft, June 2011 *Andres Alberto Alvarez Cabrera*

Architecture-Centric Design: Modeling and Applications to Control Architecture Generation

1 Introduction

Many modern complex systems nowadays can be categorized as mechatronic, i.e., involving the synergic integration of mechanical, electrical, and software subsystems. Software in such systems ranges from embedded control to user interface and database access. Control (software) plays a fundamental role in ensuring the correct behavior of the system. This is demonstrated by the control-originated bugs and other failures affecting products ranging form mobile phones to automobiles, airplanes, and rockets [86]. The origin of such failures is connected in many cases to the design of the controller itself [162] and not to accidents or misuse. Additionally, developing the controller is also a painstaking task, especially when considering the complex interactions taking place in modern mechatronic products and in their development processes.

Therefore there is a desire to support, and if possible automate, many tasks involved in the design process of a controller. Here, design is considered as a process containing a series of activities involving mainly analysis, synthesis and transformations of models. This desire, or rather need, to better support design processes has fueled the development of many tools and methodologies, from which many successful members belong to what is called model-based/driven design/-engineering/development approach. Though such terms represent slightly different approaches, in this work they appear grouped and are referred from now on simply as MBD.

The objective of this work is to contribute towards satisfying such needs. Looking at the available solutions it is possible to learn more about how to improve supporting design activities. A good starting survey is provided by the INCOSE [89]. MBD proposes approaching design problems by modeling the design problem and/or solution using an implementation-independent language that can be automatically transformed into a formal description of the implementation. This approach offers many advantages ([72], [73], [97]), which mainly include facilitating to ensure consistency and completeness during design. However, on a closer look to the available academic and practical implementations ([77], [78], [96]), MBD is mainly used by a few of the best development companies within specific domains (e.g., software development) or within a specific sector (e.g., automotive), using what has been called the domain-specific languages [77] (DSL). Modeling with DSL raises the level of abstraction while at the same time narrows the design space [77]. As it is currently implemented, the use of MBD delivers some its advantages within specific domains but does not concurrently reach the design activities because the work of stakeholders does not become truly integrated. Therefore, it is argued here that more generic languages that enable fully implementing MBD are still in their infancy. This argument is one of the core points which justify the proposals of this thesis.

A more generic implementation of MBD requires considering the need to maintain a connection among many design processes to achieve efficient collaboration. To this end, the challenge of supporting (and automating) control design tasks is analyzed from a novel viewpoint: the definition and use of a high-level model containing generic design information from which control information can be extracted and subsequently analyzed. When required, the results of such analysis should be fed back to the generic model. Here this is called an 'architecture-centric approach' to design of control. It must be highlighted that the goal is that such generic model remains useful for all design stakeholders, and not to make it a domain-specific language for control design. One contribution of this thesis lies in defining desirable characteristics of the language used to build such generic models, while another contribution is the proposal of a specific modeling language presenting such characteristics.

Regarding terminology in this work, 'design stakeholders' (or simply stakeholders) makes reference to the actors involved in a development process, such as designers, engineers, managers, and even their working tools. Considering the great span of literature related to system design and architecture, basic definitions for some stakeholders and their roles are provided at this point with the help of Figure 2. As shown in the figure, the stakeholders are responsible of creating and/or maintaining system descriptions (models) at different levels of detail and use them for managing their own efforts or the work of other stakeholders, and to deliver results from such models to other concerning stakeholders. In practice these titles are not fixed, and terms like system architect or system engineer are used in this thesis to refer to stakeholders working mainly with system descriptions with small number of details or pertaining to overviews, while terms like domain specialist or designer are used to talk about stakeholders dealing with detailed descriptions which normally are handled by members within a specific domain of expertise.

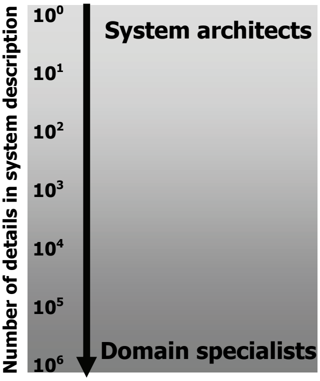


Figure 2. Relation of some stakeholders to detail in system descriptions

Besides the small analysis regarding the current state of MBD practices, the previous discussion touches three different aspects which this thesis addresses:

- There is a need to improve collaboration in current development practices.
- A generic model can help improving collaboration among stakeholders.
- There is much room for improving support of controller design activities.

The next section contains an overview of the chapters in this book, followed by other three sections which provide additional background on the three aspects mentioned above.

1.1 Structure of this thesis

This book compiles most of the work documented by the author during the research period to obtain his doctor degree, adding the line of thought which unites the whole produce coherently. The rest of the book is divided into two parts: Part I is composed of chapters 2 to 4 and explains the architecture-centric design approach and supporting material, while Part II applies the proposals on control design, spanning over chapters 5 through 7. Next the main topics of each chapter are introduced, including the references to the works on which they are based.

Chapter 2 ([3], [6]) documents the initial efforts to gather information on approaches describing "high-level" models and methods which could be used as input for controller design support and automation, and also provides insight on the challenges related with such approaches and their use in product development. The core of the proposals addressing such challenges is presented in Chapter 3 ([3], [7], [11]). Then, Chapter 4 ([4], [5], [7], [11]) presents case studies which led to the proposed approach, followed by different case studies where the proposed approach was used. Given the nature of the case studies, the discussions around them provide intuitive justifications and support to the proposals, rather than statistical or measurable usage data.

In Chapter 5 ([9], [10], [11]) the reader will find an overview of the controller design process and a proposal for controller design based on MBD supported by the architecture-centric approach presented in this thesis. Chapters 6 ([8], [10]) and 7 [9] provide more detail on the transformations and tasks involved in the design proposal, respectively for regulatory and supervisory control.

The Discussion chapter highlights the contributions of this thesis as well as the limitations of the proposals, and presents possible future research directions. The book ends with a Conclusions chapter.

1.2 Information and collaboration in product development

As stated by Bishop [21], in the context of mechatronic design, communication between all the stakeholders and transparency of the design decisions in the various domains are essential for success. In the same work, Bishop also voices the need for models of systems that allow preserving the dominant parameters while at the same time provide an interface to the (control) design and simulation tools that engineers use. Such needs have not been fully satisfied yet [107].

Why is it necessary to provide more support to the information exchange during design? Is this really a problem in modern industry? Existing benchmark reports on industrial practices ([28], [92]) partially address these questions. This section summarizes some ideas of a recent study on information exchange through email communications performed by Wasiak *et al* [188], providing an overview on the process of information exchange during engineering product development. Email data is the center of that work because it has been identified as one of the most widely used information sharing tools in companies. The study categorizes the type of topics in the exchanged information ("what"), the purpose of exchange ("why"), and the way in which the information is conveyed ("how"). Several conclusions giving an insight on the overall situation of design practices are presented, though not claimed to be generalizable:

- The purpose of 70-80% of the analyzed communications is to passively distribute information.
- Apart from the company-related communications (regarding methodologies, customers, etc), approximately 60-70% of the analyzed emails between engineers and engineering managers are product rather than project-related.
- Engineering design is an information intensive activity and therefore heavily dependent on the ability of engineers to access a good amount of accurate and up-to date information.
- There are many difficulties in searching and using data from emails (and other textual sources).

Another important aspect related to information exchange during design, is information reuse. Reusing information is related to finding ways to "package it" in a container that allows storing, retrieving, and transferring the design information efficiently.

With respect to collaboration, some goals compiled by Whitehead [190] are recalled next: Establish scope and required capabilities, converge design towards an architecture, manage dependencies and reduce them if possible, identify, document and fix errors, and create organizational memory. It is possible to see that information exchange relates to all these goals.

Considering that much design information is transferred/discussed through (semi)informal mechanisms, it can be conclude at this point that it is poorly documented and reuse is not formally supported.

1.3 A generic model and the model interpretation problem

The desire to have a generic model useful to all stakeholders is justified by the fact that the mechatronic nature of a system implies that people representing different disciplines are necessary to design such systems. Even at the early stages when the idea of a new system is just being conceived, specialists from each discipline interact with other specialists to provide new information and refine the current design. In current design practices, most of these interactions are carried out through informal communication channels and, when documented, exchange the information using either domain-specific models which are hardly understood for non-specialists (e.g., block diagram, 3D model), or generic representations which do not facilitate reusing the involved design information (text documents, emails, calculus sheets, etc.). A domain-specific model (DSM) is one that is commonly used by a group of specialists in a certain domain. A domain corresponds to a common work, scientific, or education field, such as automotive, chemistry, or mechanical engineering.

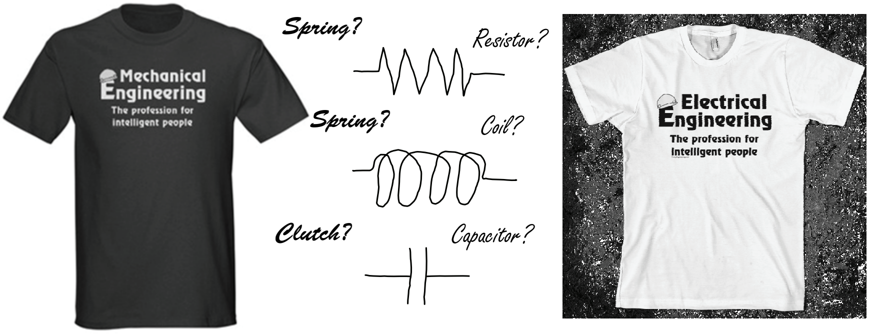


Figure 3. Different interpretation of modeling objects by two different domain specialists

To further explain why correct interpretation of domain-specific models by people outside each domain can be so difficult to achieve, the “model interpretation problem” is introduced: *Model interpretation is relative to the knowledge of the observer!* A clear and simple example is depicted in Figure 3, where three symbols used by mechanical engineers to represent mechanisms coincide with symbols used by electrical engineers to represent electric circuits. A less evident example can be taken from geometric (e.g., 3D) model interpretation: where a well trained mechanical engineer may see assembly directions, structural members, possible manufacturing methods, and functional surfaces, a non-specialist may not identify more than a couple of sub-systems and the rough volume of the objects if a proper reference is provided.

The previous discussion can lead to the question: is there a modeling language that can be interpreted independently from domain knowledge? This thesis contributes by identifying generic language primitives that can help overcoming that challenge, and exemplifies the use of such primitives in a prototype language: the Architecture Model (AM). The AM language provides a shared format for the exchange of design information. Recurring concepts in the proposal include the use of product “functionality” to support modeling and integration of dissimilar information and “parameters” to support information exchange. The AM has been used as intermediate or input model in the demonstrative use cases appearing in this thesis.

Following the MBD line of reasoning, the AM proposal can be considered as a DSL corresponding to the system architecture domain. However, it is argued in this thesis that system architecture is generic enough to be understood and modeled by all stakeholders through their individual contributions. As the reader will see in the description of the AM in Chapter 3, this point is justified by the definition of system architecture model embraced in this work: system architecture is modeled by the aggregation of views corresponding to the interests coming from different stakeholders.

1.4 Supporting controller design

Throughout the book the case of controller design is studied often, though contents and conclusions in Part I are generally applicable to design. In part, this is justified because the control design case provides representative examples of many of the characteristics and problems present in general for current design practices, as control design process entails intensive interaction among design disciplines and is inherently multidisciplinary [94]. The choice also seeks to address part of a current problem in the mechatronic industry: design is not carried out in a concurrent way to exploit the synergy among domain experts and many problems are detected late and forcefully solved in the control software domain at an advanced development stage. These practices compromise the quality of the resulting software and the product. Additionally, MBD methods stimulate designers to formalize models for their specific domains but do not help specifying a common factor among the models which facilitates use across disciplines, besides a common product or company ontology.

As many other design tasks, designing a controller is not straightforward and, in a general sense, lacks much formalization before quantitative “plant” models are made available. Also, looking at current industrial practices, some may wrongfully conclude that controller (software) development is an almost isolated design task corresponding to later stages of design. However, controller design is intimately related to other design activities, mainly because of the multiple interactions that control systems must have in order to estimate and influence the status (or state) of the system at any given moment. The controller is another system component for which many design tradeoffs related to other components have to be considered in order to obtain a well integrated and synergistic system.

It must be highlighted that, though “controller design” covers a very wide spectrum of methods, design stages, and applications, in this book the term is primarily intended to represent controller software architecture design unless specified otherwise. Thus, there is almost no material in this thesis which specifically addresses topics like development of other software (e.g., user interface), analog control, control tuning, control type selection, etc.

By applying the proposed architecture-centric approach, results from the second part of this thesis demonstrate how to empower the control engineer with a model that can:

- Facilitate him gathering the information from multiple sources to do his job.
- Allow exposing his concerns so he can influence designs to be performed by experts of other disciplines.

From a more technical point of view, Part II deals with analyzing the controller design process regarding two of its main tasks: regulation and supervision. Regulation refers to the task of maintaining parts of the system under specific reference conditions, also understood as modes or states. In turn, these conditions can be characterized by the values of representative variables or parameters, which are used to define the reference and to assess system performance. Supervision refers to the task of switching among the different modes to ensure that a process runs as designed.

PART I. ARCHITECTURE-CENTRIC DESIGN AND SUPPORTING MODELING LANGUAGE

2 Challenges in Complex Product Development Processes

This chapter defines and explores key aspects of complex product development. More specifically contains an analysis of the development of mechatronic systems, which involve an increased complexity due to the need to tightly and synergistically integrate their components, as well as the people who design them ([126], [172], [187]). Therefore, such integration needs to extend to the development processes. This leads to find a series of challenges in current product development practices and in how MBD is used. Analysis of the challenges sheds some light on how to support development processes through a full implementation of a MBD development approach. The challenges share common grounds regarding information integration and sharing. After revising current attempts to address the pinpointed challenges, it is mainly concluded that:

- Methods based on higher abstraction levels play an important role, but that their implementation is an issue.
- Multidisciplinary design optimization and verification of both hardware and software require suitable modeling paradigms and tool support.

With these findings in mind, the ground is set to propose the outline of an integrated design support framework for mechatronic systems later in Chapter 3.

The first section presents an overview of design practices for mechatronic systems. Then, the challenges related to many pervasive problems in current development practices are presented in Section 2.2, followed by a review of approaches that seek overcoming those challenges in Section 2.3. The chapter finalizes with some conclusions.

2.1 Mechatronic systems and their control

First, it is convenient to establish some common ground about mechatronic systems, mentioning some distinctive aspects and problems related to their design and control.

A definition for mechatronic systems that shares some of the most common aspects from the different definitions that can be found in literature ([13], [17], [54], [197], [202]) is: *A mechatronic system is one that contains a synergic integration of applied principles from mechanics, electronics, and information technology, considering the driving phenomena, requirements, and constraints*. Usually the electronic and software parts of the system are responsible of the information handling and the control of the system.

2.1.1 Particularities in the design of mechatronic systems

With the previous definition in mind it can be said that, ideally, when designing a mechatronic system the possible interactions among disciplines must be considered at all stages. This allows obtaining a design that integrates synergistically the involved domains and subsystems. In fact, the concept of mechatronics shares much with the concurrent engineering approach of product development ([30], [74], [151]).

Nowadays, there are still situations where the design of mechatronic systems is carried in an independent manner for each involved domain ([105], [198]), at best, only maintaining in common the requirements specified at the conceptual stage. Then the resulting parts are assembled forming a consistent unit. Some times it is wrongfully assumed that the design is integral or concurrent because design activities are performed parallelly in time. Though this is a valid design approach, it is worth mentioning some related problems:

- Separate subsystems are designed considering only the *intended* interfacing between them, thus neglecting other *unintended* forms of interaction. Unintended interactions are hard to detect when viewing a problem from the point of view of a single domain.
- As the full set of relevant forms in which subsystems interact may not be identified, subsystem changes considered independent might lead to malfunction of the system as a whole because their impact is not appropriately evaluated.
- True global performance of the system cannot be properly optimized because it is not considered when performing a separate design for each domain.
- Because the designers cannot keep track of the needs of other designers, it is common to search solutions within the primary domain in use when problems arise in the middle of the design of one of the subsystems. In that way, better solutions that may come from other domains are disregarded.
- Design of the controller does not receive enough attention in the conceptual stages, leaving the solution of problems that presumably can be handled by the controller for later design stages. This can increase the number of design iterations and the development time.

Complexity also plays a role in difficulting an integrated and coordinated development process because it makes it more difficult to understand. There are several reasons that explain complexity in this context ([13], [17], [54], [197]):

- The multidisciplinary nature of mechatronic systems requires careful consideration of interactions coming from integrating subsystems.
- The modern tendency to seek flexibility (as multiple function achievement and as adaptability) in mechatronic products. A flexible mechatronic product is more complex than one that performs less functions or that only works in a very restricted environment.
- The sheer “size” of the mechatronic product design. Even mechatronic products that do not contain many mechanical components possess some form of

interfacing (system and user interfacing) and control that requires detailed models and specifications for software and electronics.

2.1.2 Highlights about controller design in mechatronic systems

Control constitutes a significant part of the design of mechatronic systems ([13], [17]). A proper choice of the control architecture (*cf.* Chapter 5) in a system is not only critical to guarantee that the necessary system variables are controlled [197], but also effectively influences the development efficiency and the performance of the mechatronic system.

As mentioned before, quite often the controller design is not considered carefully in the conceptual stage of product development. An example of this is given in [198], where the authors comment on the consequences of disregarding controller design in the conceptual phase: "*The common approach for the machine tool design would (...) come up with a very stiff structure, and then move on ahead for a prototype machine tool manufacturing for servo tuning. This process can be very expensive when the servo controller does not match well with the prototype system. It gets even worse when developing high precision machine systems.*" This is a clear example of how the solution space can be restricted to one domain. In the example, the problem was considered to be solved with the mechanical design, and only tests on the prototype showed that the conceived controller could not perform properly in this design when the problem could not be managed by the controller alone. If considered from the beginning, in numerous cases the controller can compensate imperfections of subsystems from other domains, thus allowing to decrease manufacturing costs [13], including those of prototyping.

2.2 Challenges in mechatronic design

Both academic and industrial sources have reported on challenges related to the design and development of mechatronic systems, such as:

- Exchange of design models and data ([147], [187]).
- Cooperative work and communication among the design engineers ([28], [111], [131], [147], [187]).
- Multidisciplinary modeling ([31], [55], [118], [131], [191]).
- Simultaneous consideration of designs from different disciplines ([28], [31], [55], [92], [191]).
- Early testing and verification ([28], [31], [55], [131]).
- Persistence of a sequential design process ([111], [187], [191]).
- Lack of tools and methods supporting multidisciplinary design ([28], [92], [187], [191]).
- Support of the design of control software ([28], [118], [172]).

Examining these challenges, three core issues can be identified, which influence many of the problems in the development of mechatronic systems. These challenges relate to design integration, design verification, and generation of control software. In the next subsections, these will be discussed in more detail.

2.2.1 Design integration

Figure 4 depicts a representation of the current mechatronic design process, where spaces represent common gaps between the different design phases and the tools used in the design. Design teams are often composed separately according to their area of expertise and often work at different locations. The integration phase is postponed until the moment when physical prototypes are available. These points are elaborated in Section 2.3.

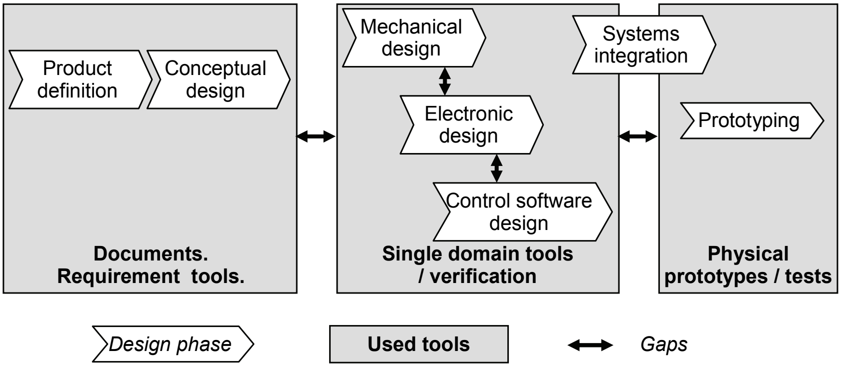


Figure 4. Common current design practice phases and tools

Integration has been directly identified as an important research direction and a key element in the design of mechatronic systems by industry [131] and by authors like Craig [55], Schöner [147], and Wikander [191]. Tomizuka [172] and Wang *et al.* [187] identify the importance of aspects closely related to integration, such as cooperative work of designers, data sharing, knowledge management, design project management, and simultaneous design in different domains (e.g., design of the control algorithm and of the system to be controlled). A recent report on industrial practices [28] shows that the leading mechatronic product manufacturers opt for integration oriented towards management of specialist designers and tools that support such an approach, rather than using tools that encompass all detailed design aspects. The desired tools, as identified by these manufacturers, should handle information at the system level and track requirements and design changes to efficiently support integration of design activities, thus approaching MBD. Apart from the need for tools, it is also necessary to consider the design methods these tools support.

Appropriate methods and tools to support design integration are required, both in the conceptual phase as well as in the detailed design phase, as has been identified by academia [187] and by the engineering community ([111], [131], [148]). The role of the human actors is also important, as communication of ideas and information between designers from different domains is necessary ([28], [55]). These three factors (i.e., methods, tools, and human role) will be discussed in the next subsections.

2.2.1.1 Design methods

Despite many research contributions aimed at providing a theoretical framework for the design process, this goal has not been achieved yet [22]. As depicted in Figure 4, design activities might be separated in the sense that parts of the design might depend on data provided by other parts (e.g., the design of a controller may require knowledge of certain physical characteristics of the system). Traditional methods in engineering design broadly exhibit either a sequential or a concurrent flow of activities.

As reported by Wang *et al.* [187], sequential design has proven to be unsuitable because of its lack of flexibility, which increases design cost and development time. This perception is supported by engineers in industry [111]. Rzevski [143] recommends stepping out of the conventional end-to-end (i.e., sequential) design process in favor of a concurrent approach to deal with design of mechatronic systems.

The core of traditional concurrent engineering approaches (see e.g., [151]) is to consider all phases of the life cycle of the product as early as possible in the design in order to deal with issues related to later life-cycle phases, such as production and disposal [108]. But even traditional concurrent approaches have proven to be limited when dealing with complex design situations, in the sense that strong interdependencies might have unpredicted effects on the overall performance [191]. As mentioned by Wikander *et al.* [191] and Rzevski [143], a typical approach for the design of mechatronic systems is to build the system by assembling single-domain subsystems and by paying special attention to the design of interfaces among them. Wikander *et al.* remark that such traditional methods can merely achieve a sound integration of the components (i.e., “something that works”), but not a synergetic integration. Therefore, research on mechatronics should also focus on the interactions of the different engineering disciplines [191] rather than only on the interactions between the subsystems that are being designed.

Dealing concurrently with the interactions of designers and of their designs is of paramount importance for the early detection of problems in product development.

2.2.1.2 Design tools

Recent reports on industrial practices confirm the use of different tools to manage design data, and state that the lack of tools that allow integration and shared use of such data is one of the main challenges in mechatronic product development ([28], [92]). As illustrated in Figure 4, a current tendency is that designers from different design domains rely on specialized tools. Similar to Wang *et al.* [187], a tool is considered as domain-specific if it supports the design in a single domain, e.g., SolidWorks supports mechanical design and OrCad supports electrical design. Furthermore, there are not many specialized tools that support the first stages of design and that also extend efficiently to the subsequent stages, although this limited reach is probably due to misuse of the existing tools. Examples of such tools are requirement management tools like Rational DOORS, and the tools which support approaches to capture requirements like Quality Function Deployment (QFD) [135] or Integration DEFinition for function modeling (IDEF0) [120].

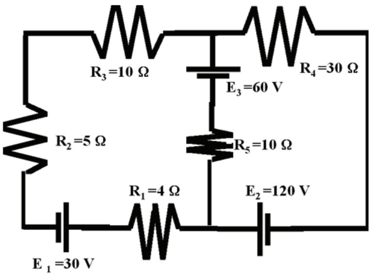
Mono-domain tools perform well within their own domains, but their specialization often makes it difficult to consider information from other domains. The tools used in the control design domain in general prove to be more flexible as they use mathematical models as modeling primitives, e.g., in the form of block diagrams or bond graphs [95]. Additional insights on tool integration can be found in the works of Cutkosky *et al*. [56] and Dolk *et al*. [63]. The varied nature of the different design tools interferes with a direct integration (i.e., direct mapping of the modeled objects) using a single tool or design environment. Examples that illustrate such variety are:

- In mechanical design, dimensions, shapes, and materials that correspond to the physical objects are the main interest. Thus, representing abstract concepts and grouping parts according to other criteria than physical proximity become problematic.
- In the design of controllers, the physical system, also referred as the plant, is often abstracted to a black box model. From such point of view it is difficult to find the explicit connection between the behavior and its physical causes.
- Electronics deals with the physical implementation of the control. The software packages for electronic design support predictions of behavior and execution time through logical and physical simulations.
- Electric engineering commonly designs “bridge” objects from electronic and mechanical domains, and tools related to it focus on the connectivity of components and the communication among them.
- Requirement management and capture tools focus on representing textual requirements information. The link to other design domains is mainly made through document referring, and it is the job of the user to (informally) connect such documents with the current design data.

Realizing that most of the design activity in industry is performed using separate modeling tools, one aim is to produce a framework for containing a complete model of the system and provide a mechanism of information exchange between modeling tools ([167], [170]). The authors of [62] comment that: “*Another area of integration that is becoming increasingly important (...) for all applications is software integration. The ability to link word processors, spreadsheets, databases, and graphics packages in a seamless fashion enhances the feasibility of modeling environments which can support paradigm integration.*” In this way, designers can continue using the available specialized modeling tools (which are highly efficient in their own areas), but at the same time will have the support of a framework to aid them to gather updated information on other systems, export updated data from their work, and view the existing design at different levels of detail [56]. It must also be considered that normally, detailed information is understood properly just by the experts on a specific field, and that simulating a model that contains such detailed information requires large amounts of computational power.

$$I1 + I2 = I3$$
$$-4\ I1 + (-30) - 5\ I1\ - 10I1\ + 60\ + 10I2 = 0$$

(a)



(b)

Figure 5. Mathematic (a) and graphic (b) models of an electric circuit (example taken from http://www.math.ucdavis.edu)

Challenges on tool integration are thus strongly related to the models the tools allow to build them (see Section 1.3). Models are crucial to support, communicate, and document the design activities. Model types and modeling tools used in design are mainly idiosyncratic (e.g., Figure 5), and depend on the desired aspects of the system to be represented (or simulated) and the level of detail to be specified. Other source of complexity that complicates modeling is the interaction of phenomena from different domains in the system. The existence of these interactions forces the designers to "couple" models made in tools that, frequently and in the first place, may not be designed to be compatible with each other. As a result, the coupling is performed, so to say, "manually" by the designers. The aforementioned problems associated with complex systems reduce the efficiency of classical models in terms of implementation feasibility and undermine one of the main goals of models: assist humans to understand the knowledge contained in the models. Other models and tools are necessary to overcome these limitations and obtain models that support effectively an integrated design approach.

2.2.1.3 Human factors

In part, the integration problem can be traced back to the early phases of design of a system in which its architecture is defined. In the conceptual design phase, the designers choose the solution principles, decomposition, interfaces, and design process planning that will guide the detailed design phases and the way in which designers will cooperate [143]. The selection of an architecture influences the choice of detailed solutions and the integration of those solutions in a rather straightforward manner; e.g., actuating an axis of a machine tool with a linear motor or with a precision ball screw completely changes the configuration of the machine at both the hardware and the software level, and therefore, different groups of specialists will need to interact in each case.

Human communication and cooperation are additional factors that affect design integration. One issue is to communicate the goals and requirements of the design and how they relate to the chosen solution, and to assign responsibilities for such requirements. In order to enable monitoring the requirements throughout the design process it must be possible to decompose the requirements and to make budgets of resources for them, down to the interfaces of the individual designers. Another issue is to inform the designers on how their part of the solution in the design affects other parts. Individual designers make choices that can inadvertently affect the system as a whole. The design should therefore be tested for consistency and validity throughout the design process.

Both issues strongly relate to the fact that there are currently few methods and tools that support systems engineering and architecting activities and that capture the information produced in these activities in order to facilitate the exchange of information between designers.

2.2.2 Lack of interdisciplinary verification

Verification is a necessary activity for quality assurance in wich it is evaluated wether the developed product complies with the desired specifications defined at the first stages of design (not to be confused with validation, which relates more to capturing the right requirements). The four classical verification methods are demonstration, test, inspection, and analysis [108]. Of these, the first three require physical prototypes to be developed, while the latter is based on a (usually mathematical) representation of the system, also known as a model. Developing appropriate models for analysis and a platform to verify various aspects of the system, including control software, represents a challenge. In practice, specific models are developed to perform tests at different stages of the design. Due to the use of domain-specific modeling tools, such models usually correspond to a specific point of view on the system, like either the electrical or mechanical aspects, or continuous dynamics and discrete, sequential behavior [92]. With the expected synergetic effects that characterize mechatronic systems, these separate views cannot capture the overall system behavior. Even more, the analysis of changing operation modes, defined in terms of state machines, requires reconfigurable multi-domain models, which are often not supported.

Schemes of co-simulation and model sharing incorporate data generated in other domain-specific analysis tools into control design models, for example, as implemented in the de-facto industry standard [137] Matlab/Simulink. However, often these dynamic models can be considered as an input/output box in the form of a transfer function, and the explicit relation with the original design input is lost. On the other hand, control and hardware co-simulations also require coordination among different specialists, and as discussed in Section 2.2.1.3, many challenges remain in that area.

For these reasons, verification and testing of control software still relies heavily on the use of hardware prototypes or breadboards, requiring considerable investment in terms of time and money [92]. In a way, complete system prototypes allow a concurrent, multidisciplinary verification that can reduce overall development time. On the other hand, besides their cost, the use of prototypes becomes less viable as their detailed design has to be relatively well specified for their construction. An approach typically used in the aerospace industry is the 'Iron Bird' concept, in which a combination of part of the final hardware and software is used to test and verify the behavior of on-board systems, such as the electrical and hydraulic actuation devices. In this way, system verification does not require building a fully operational system, but it still requires significant investment and the detailing of portions of the design.

2.2.3 Lack of automation in control software design

In practice, the control system development effort is around 20% to 40% of the total software development effort [83]. Modern Computer Aided Control System Design and Computer Aided Control Engineering tools (CACE is used to refer in general to both) such as Matlab/Simulink or dSPACE, and software development tools such as Rational Rose provide means to translate control algorithms, in the form of block diagrams and state transition diagrams, to machine-executable code. These code generators eradicate human coding errors, increase reliability and reusability, and reduce development time and effort. Nonetheless, a major part of the control system design is spent obtaining "working" formal models like block diagrams and the values for the parameters that configure each block. The aforementioned tools only help to transform those formal descriptions into control code.

Generating code from a model (e.g., a block diagram or a description in the Unified Modeling Language (UML) [124]) of the structure and logic of the software system is part of what is known as MBD for software. Only some of the top-level companies that design mechatronic systems take this approach and it is not a common practice [28]. In such cases, the primitives used for building such models usually represent objects clearly defined for certain specialists. To obtain a more transparent model that aids integration, it is desirable that the objects used in the model are familiar to the parties involved in the control design, which transcend the control engineers.

To arrive at a formal description that can be transformed into code, the designer must define a control structure (I/O groups) and strategy (controller type), and think about the implementation of functions for the measurement and filtering of system signals and for the application of the control outputs to the system (*cf.* Chapter 5). Once the structure and strategy are chosen, design rules and optimization routines can be applied to determine the controller parameters, provided that the requirements are given in a suitable form. Often, however, these requirements have to be derived by the experts first, as system requirements specifications are not defined in control terms from the beginning. There is still much to be gained by supporting and automating the control design tasks in the early stages of design. These topics are also treated with more dept in Chapter 5.

2.3 Review of available approaches

Both academia and industry have come up with methods and tools to deal with the challenges identified above. This section discusses a selection of these methods and tools, grouping them in the same way as in the previous section.

2.3.1 Design integration

2.3.1.1 Design methods

Various methods consider the modeling of functions, requirements, and other information that is usually defined at the conceptual stage of the design. Documenting such information helps the designer to maintain an overview of the system and to keep track of the evolution of the design. Multiple authors have proposed models that contain functional descriptions of systems, like Function-Behavior-State (FBS) [176], Functional Representation [41], Schemebuilder [29], and MACE [87], to guide and improve choices made in the first phases of product design. These models represent knowledge about the functions of the system, complemented with information about how the function is accomplished and which objects, both hardware and software, are involved. For example, some functional modeling approaches complement this information with qualitative (e.g., Qualitative Process Theory and Qualitative SIMulator [18]) or quantitative (e.g., differential equations, bond graphs) data. Example applications are mentioned by Erden *et al.* [65]. The FBS methodology has been implemented in the software framework KIEF [171] to integrate tools from various domains and to facilitate the transfer of information, as will be discussed in Section 2.3.1.2. Other approaches use functional flow and block diagrams, and they model functions as transformation stages of matter, energy, or information ([67], [120], [126], [154], [196]). The IDEF0 method [120] offers a formalization for functional flow diagrams and various IDEF languages [98] model details of the system that could be connected more directly than the functions to other domain-specific models, but they do not provide a clear connection between the different IDEF models. The functions and key drivers method (FunKey) [24] proposes allocating budgets of resources to the functions of a system. In this way, FunKey pursues its goal of documenting the architecting process and of providing a means to compare product architectures.

The implementation of these methods is a challenge. As in the case of other theories related with design, either the approaches are not implemented in a tool, or the developed tools are not part of common industrial practice [22]. Furthermore, functional descriptions are mainly used to aid the designer in the identification of related information, but not to classify or identify such information with the help of an automated system. This stems from the fact that these abstract representations have proven to be hard to formalize, and recent experiments [2] indicate that even people have trouble using them in practice. Another important factor is that there is not even a consensus for definitions and formalisms in the field of design research ([2], [22]). Additionally, requirements information is not included in most of these methods. An exception is FunKey, which mainly focuses on the system budgeting aspect. In particular, QFD specializes in capturing user requirements and connecting them to characteristics of the system that can be used to measure the fulfillment of those requirements.

Muller has proposed the Customer objectives, Application, Functional, Conceptual, Realization model (CAFCR) to decompose the product architecture into the five views its name indicates [117]. This allows for independently capturing the needs of the customer, the functions the product performs, and the design of the product from the conceptual and realization standpoints. Its main purpose is to provide mechanisms to keep track of stakeholder concerns, like safety, usability or performance, in order to maintain integrated goals throughout the whole design process. The work of Muller mentions which relevant information should be considered to obtain a proper description of the architecture of a product, and suggests methods to capture such information. However, these methods are not strongly linked to each other. The large variety and number of methods mentioned in CAFCR brings more flexibility, but leaves to the systems architect or designer the, sometimes difficult, task of choosing the most appropriate out of all the presented methods.

The V-model [153] sets a general flow for the development process of a product. It indicates that each stage of the product definition should be used to systematically test the implementation as subsystems are integrated to arrive to the final product. Different stages of development and testing are defined depending on the source, but in general, requirements analysis, architecting, detailed design, and the corresponding verification/validation stages are defined. The model provides a structured base for the development process, but it is very general, and does not provide details for its implementation; there are no tools to fully support it, and companies have to carefully develop a (normally DSL-based) framework of tools to model each definition phase and to put the test phases into practice. Though not explicitly specified in the V-model references, analysis methods (*cf.* Section 2.2) are crucial to support the definition stages and to obtain correct models that can be used for verification. At this point it is worth mentioning that many other models with similar scope exist and that discussing the advantages of each of them is not within the scope of this work. As an example it is possible to refer to the spiral model [23], which has similar goals as the V-model, but which considers several iterations using prototypes to verify the design at one stage and to produce a base for the next one.

The axiomatic design method, presented by Suh [155] states that functional independence of the system's constituents leads to an optimal design. To attain this, the method provides guidelines, namely, the axioms of independence and information, to compare and evaluate early design choices. Suh and other authors also report that the method has been applied successfully in multiple situations [155]. A crucial point from the axiomatic design method is the importance of linking high-level information (functional requirements) to implementation specific information (design parameters). On the other hand, modern mechatronic products implement an increasing number of functionalities while maintaining constraints on space and costs, and thus, a tight integration of the subsystems is desirable, which makes it harder to obtain functional independence. This relates to the tradeofs between considering modular or integrated architectures [175].

Capturing and integration of information is important to deal with the challenges discussed here. The Knowledge and Information Management project [113] has proposed principles that describe the characteristics of engineering information that should be captured and kept for reuse.

This section has shown how several methods deal with one or more aspects related to integration, but there are implementation issues and gaps exist between early design phases and the detailed design phases.

2.3.1.2 Tool integration

According to Citherlet [47], there are four different approaches to multidisciplinary tool integration: stand-alone, interoperable, coupled or linked, and integrated programs. The first one is the least desirable, as the tools are unrelated and communication is not possible. Interoperable programs provide means to exchange or share models. Towards these goals, additional frameworks have been developed to streamline or automate the model exchange. This second approach will be treated in more detail later on in this section. Coupled or linked tools can communicate at run-time. Due to the flexibility of their modeling primitives (*cf.* Section 2.2.1.2) some tools used in the control design domain have taken the second or third approach. Finally, integrated programs facilitate work in different domains within a single tool. Vendors, especially those of mechanics CAD tools, have used this approach integrating tools from other domains into their software suites. As an example, CATIA V4 also supports electronics, systems and control modeling, and incorporates embedded control code generation for the latter. Though the existing coupled and integrated programs provide a way of predicting the behavior of a system, they specialize in running models used in detail design and lack a direct connection with information from earlier phases of the design process (e.g., goals, functions). Efforts to incorporate such information are being made in newer versions.

Within the interoperable integration approaches, the pluggable metamodel mechanism implemented in KIEF [199] and the framework of the Virtual Reality Ship (VRS) systems project [185] can be mentioned. The VRS project reference indicates that several tools used in the European ship building industry, including a physical testing platform, have been integrated, but unfortunately no details of how this is done could be extracted from the available material.

The core of KIEF is a knowledge base in which objects from different modeling tools are mapped to each other using "physical phenomena" as connecting points ([171], [200]), in what is known as the process-centered approach [65]. This knowledge base also contains information about modeling tools to support their integration into the framework. A metamodel of the system is built according to the ontology underlying the knowledge base and KIEF manages the data transfer and consistency between the domain specific modelers. An ontology can be defined as a formal representation of a set of concepts within a domain and the relationships between those concepts, and as such can define a language for communication between domains.

The software suite CORE [183] offers integration through a model-based systems engineering approach. The tool allows making models to capture requirements, modeling function decomposition and flows, and mapping them to models of system components and their interfaces. It implements a concurrent design process called 'the onion model' [43] to validate the product definition stages subsequently within its models. For comparison, it is highlighted that such a tool can support a good portion of the 'left arm' of product specification of the V-model (see Section 2.3.1.1), but lacks a direct link to the models and tools used in the detailed design and the subsequent testing phases (verification). Nonetheless, the models provided by this tool can be related manually by the designer, outside the CORE tool, at the level of components.

Although the approach in the CORE and KIEF methods is different, both rely on a product model based on components to integrate multiple views on the system. The models from the CORE tool can also be manually integrated to other design information at the component level. This originates in the fact that most parameters and data are directly related to these components. The object-oriented properties ensure that components sharing parameters or data can be easily grouped into a new composite component. The component-oriented approach may be intuitive and fast at the moment of building models, but each modeling object can only be used in a specific situation. For example, a "gear pair" component used in a transmission must be defined in a completely different way than a gear pair used to grind material. A process-oriented approach can help to deal with these kinds of situations, by separating behavior and modeling primitives. The metamodel in KIEF uses such an approach, relating all concepts of the system through their attributes to phenomena and laws, giving more applicability to each modeling object (*cf.* Figure 6).

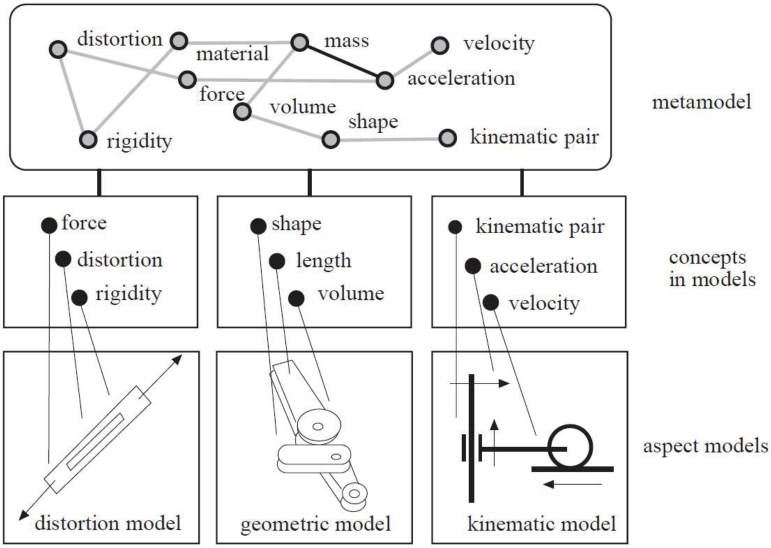


Figure 6. Metamodel mechanism in KIEF [28]

A component-oriented approach that also corresponds to interoperable integration is proposed by Peak *et al*. ([129], [130]). A framework based on the Systems Modeling Language (SysML) [123] is used to integrate information from different tools (e.g., CATIA, Ansys, Matlab/Simulink). Using a combination of SysML and the Composable Object (COB) [128] paradigm it is shown how to represent knowledge about a system and to link such knowledge to tools that can use it to build other models. COBs combine the structural and behavioral descriptions of a system. In this object-oriented approach the models can be built in such a way that they are both human- and machine-readable. COBs also form a basis for the integration of different views on a system, as shown by Peak [129].

In support of multidisciplinary design and optimization a framework called a Design and Engineering Engine (DEE) has been developed by La Rocca [102]. Relying on a knowledge-based engineering platform, a DEE is a domain-independent tool suitable for the design of a variety of systems from multiple domains. The core of a DEE is the ‘Multi-Model Generator’ (MMG), which is responsible for the instantiation of a product model built from a set of parametric, object-oriented modeling primitives. Furthermore, the MMG processes the product model to generate input for domain specific analysis tools, which are responsible for the evaluation of one or several aspects of the design. In this way, aspects such as aerodynamic performance and structural stiffness can be analyzed, all based on the same product model. Data sharing between the various tools is enabled by using an agent-based network [20].

Recent interviews with mechatronic product development companies [92] reveal a problem with the fact that different disciplines use separated design tools and data, which hampers communication among them. The same interviews show that better results can be achieved when using specialist engineers working in well-coordinated groups rather than mixed groups with cross-disciplinary managers. Based on this, it is possible to conclude that a promising approach is to provide different modeling environments tailored to each domain, while integration is handled at the “back side” of the tools as a communication support mechanism. The next section treats efforts to overcome the communication issue in more detail.

2.3.1.3 Human factors

As argued in Section 2.2.1.3, it is important to consider human factors involved in the design if one wants to achieve an integrated design approach. The communication between the stakeholders is of special interest. Tomizuka [172] mentions that effective communication with others is a necessary requirement for the engineering practice, even more when considering that nowadays engineers must work in teams in design mechatronic systems. Industry also recognizes the importance of the communication among engineers ([111], [131]).

Pahl *et al.* [126] identify communication and exchange of information between designers as one of the fundamental aspects of their systematic design approach that relates to division of work and collaboration. They mention methods like brainstorming and group evaluation to support the information exchange activities. As Pahl *et al.* comment, these methods are especially helpful for the search of solutions in the conceptual phase, and thus are focused towards that end in their work. Unfortunately, such methods seem less appropriate for being extended to later stages of design, because they have been conceived to deal with less detailed information than the one required for such design phases.

Although the importance of communication among engineers and information exchange has been widely recognized, to the best knowledge of the author, there are no tools supporting the design activity while extensively considering these aspects, e.g., integrating the individual work of the engineers using their own tools together with an overview of the system and its goals.

2.3.2 Lack of interdisciplinary verification

As discussed in Section 2.2, in practice the use of domain-specific modeling tools limits the design and the verification to a specific point of view on the system. Finite-element models are used to verify strength and stiffness of the mechanical design, CACE tools are used to develop and verify controllers, and data is transferred from one tool/domain to the other when required. Following an analysis method for verification plays an essential role in early multidisciplinary verification of the design; the onion model discussed in Section 2.3.1.2 is an example of this. Often, real multidisciplinary verification can only take place at late stages in the design process, when hardware prototypes are available. In relation to controller design, the use of hardware-in-the-loop and rapid control prototyping relies on these hardware prototypes. Though this is common practice, the reliance on prototypes makes this approach less suitable in a concurrent design environment. A goal of this review is to find alternatives to the use of physical prototypes, also to avoid the other disadvantages presented in Section 2.2.

The multi-domain dynamics models used in control design are often transfer functions, modeled with block diagrams in tools as Matlab/Simulink. Two other types of simulation models can be identified for this purpose: models of the first type are based on 'physical modeling' methods, which rely on differential equations and energy flows to describe the behavior of systems; models of the second type are based on geometric modeling, either in combination with finite-element meshes and solvers, or with multi-body dynamics solvers.

A drawback of the use of controller design tools to integrate multi-domain effects in system design is that the user often focuses on the design of the controller for the given model of the system. The ‘black-box’ nature of the plant models used supports that statement. In order to shift from controller design to system design, physical modeling languages like bond graphs [18], Modelica [165], and SimScape [164] provide the user with graphical modeling elements representing physical components from various domains, such as electrical motors, resistors, and mechanical gears. The obtained system of differential equations is subsequently solved by the supporting tool. These tools often also allow for the modeling of signals and discrete events ([53], [64]). Due to the port-based approach, simplified models which are used early on in the design process can be replaced with more detailed models as the design matures, though restrictions arise from the nature and number of ports.

The bond graph language from Karnopp *et al.* [95] has been promoted for the modeling of mechatronic systems by authors like Van Amerongen ([12], [13]). The bond graph tool 20-sim consists of a block modeler, a set of control analysis methods, and a basic 3D modeler which can be used to link the block diagram representation to a mechanical model. Ferretti *et al*. [69] state that mutual interaction between domains, modular and object-oriented modeling, and reuse of modeling components using libraries and customization are required for a modeling and simulation tool for mechatronic systems. Their conclusion is that the combination of the Modelica language and the tool Dymola satisfies most of these requirements. There are various similar modeling and simulation tools available, both commercial and academic. These tools include gPROMS [134], SABER [156], HyBrSim [115], and Smile [160].

A disadvantage of these multi-physics modeling tools is that the model is based on assumptions about the expected behavior, such that a significant (multi-domain) experience is required to know which assumptions are valid. For example, thermal effects can have a considerable influence on electronic components, but the designer needs to know the relative position of the heat source and the electronics to decide whether or not to take this into account. The use of first-principle based simulations, i.e., using finite-element analysis, is a way to (partially) circumvent this.

Simulation based on finite-element methods relies on 2D/3D CAD models. Various commercial CAD tools are available nowadays, and their use is a well-established industrial practice. Vendors of these tools often provide additional tool suites for finite-element analysis, covering domains such as mechanics and thermodynamics. Specialized multi-physics simulation tools, e.g., COMSOL, allow for simultaneous analysis of phenomena from different domains. To prevent consistency problems, often the geometry models developed in dedicated CAD tools are imported in the specialized tools, instead of being developed only for this purpose [92].

Results from these various analysis tools can subsequently be used in models that are used in the controller design, albeit via manual data transfer. The direct use of finite-elements tools in combination with controller design tools for verification purposes is computer-intensive and time-consuming, but might, however, in the long term be faster and cheaper than physical prototype-based testing.

To prevent the manual transfer of data, Voskuijl [184] has used a combination of a Simulink-based aircraft dynamics model and computational fluid dynamics analysis for the design and optimization of a blended-wing body aircraft. Albeit custom-developed, it shows that domain-specific analysis can be integrated in a multi-domain analysis and optimization tool. The DEE concept discussed in Section 2.3.1.2 applies a similar approach, in which multidisciplinary analysis, optimization, and verification are supported by an integration framework.

With respect to the verification of discrete, event-driven control algorithms, there are various methods available, depending on the formalism in which the algorithm is defined. These methods are used for checking the existence of dead-lock situations, unreachable states and transitions that do not occur, among others. For realistic model-based verification, the model of the system should reflect the changes in operation mode, e.g., by reconfiguring the active actuators. More details are discussed in Chapter 7. Also see MULTIFORM in the next section.

2.3.3 Lack of automation in control design

It must be stressed that in this work the automation of control software design covers more than just the generation of control code out of a detailed control software model, and extends to obtaining such model (*cf.* Section 2.2.3). There are various commercial code generators available, both for Matlab/Simulink-like environments and UML-based modeling tools. The Gene-Auto project has developed methods for automatic model transformations, focusing on a “correct by construction” approach [173], such that the code can be implemented on critical embedded systems in the aerospace and automotive industry. By verifying the code generator itself, it can be used without the need to verify the generated code. To integrate design formalisms for continuous and discrete-event control, an integrated design notation is used in both the PiCSi [92] and the Flexicon project [166]. Projects like MULTIFORM [118] have also started providing capabilities of interoperation with different formal descriptions and some supervisory control synthesis methods. UML [124] is used as a common language, into which both Simulink models and Sequential Flow Charts are transformed. From the combined control system, platform-independent Java code can be generated. Again, the use of proven, domain-specific tools and methods in combination with a translation to an integrated model is preferred above a new and integrated "do it all" language. In contrast to this, the application of DSM languages to raise the level of abstraction of control software design relies on specific modeling elements. It removes the need to map elements to domain-independent languages as UML before code generation can be applied and as such decreases development time [96]. For DSM to work, however, the language and code generation tools have to be developed by one or more domain experts.

In terms of automation of the control design much can be gained in the early phase when requirements are translated into control structure and logic. Message Sequence Charts and UML sequence diagrams can be used to specify required behavior, but these specifications are considered to have a weak expressiveness [81]. Instead, Live Sequence Charts have more expressive power. By formalizing communication between actors over a timeline, live sequence charts provide means to automatically derive control software logic and structure from them, e.g., in the form of UML. As discussed in Section 2.2.3, the generation of code from the latter description is possible, but not widely applied yet.

To get from requirements to control software, a method based on Requirements-Based Programming is proposed by Rash *et al.* [138]. This method should increase development productivity and the quality of the generated code by automatically performing verification of the software, which is supported by an approach that ensures that the application can be fully traced back to the initial requirements of the system. A more direct link between (functional) requirements and software has been achieved by the use of the Functional Block computer-aided design environment [68]. The prototype tool can be used to design and analyze reusable high-level control software components and to generate run-time code for distributed control systems. The applicability of such a direct approach, where functions and software code are directly linked, to continuous feedback control software is however not straightforward, because of the strong dependency on the system properties.

Another approach that starts from high level specifications is presented by Sakao *et al* [144]. The input specifications are modeled in FBS [176] using qualitative descriptions. Qualitative reasoning techniques are used to derive a sequence of activations from the actuators, and quantitative information can be added to the resulting sequence. The method is only implemented for a specific case, but a patent [177] shows aspects of the control sequence derivation that could be used in generic cases.

Partial automation of the control development process can be obtained by instantiating pieces of pre-developed control code from databases linked to specific system components, e.g., sensors or actuators. For example, this approach has been implemented on a large scale by a company specialized in handling and transport systems of goods [193]. In that company, around 80% of all the PLC controller code in a system can be generated from component descriptions and associated code elements. These code elements, stored in company-specific libraries, contain routines to execute most of the low-level tasks for each type of component; e.g., start up, shut down, and emergency handling sequences for an electric motor. Service functions and irregular situations have to be predicted by the engineers and programmed manually. Integrating generated code with manually written or existing library code reduces part of the advantages of automatic code generation in this case.

2.4 Conclusions

Development of mechatronic products brings new challenges for design because modern mechatronic systems tend to be complex by nature. The design of such systems requires the participation of experts from several domains that cooperate to solve problems from the point of view of their specialties. Appropriate modeling and design support tools are essential to deal with system complexity, and one alternative for support is to accomplish modeling tool integration.

The design of integrated mechatronic systems requires a paradigm shift towards concurrency and integration of design teams and work, paying special attention to the early phases of design. To obtain tighter integration, the design of mechatronic systems demands a holistic approach that considers interactions and interrelations among design domains. Tools to support such an approach are necessary and, at the moment, scarce. The use of domain-specific design methods and tools to develop an integrated, multidisciplinary system has inherent drawbacks, related to multi-domain modeling and the communication between designers and tools.

The identified challenges relate to the integration of tools, models, and human actors in the design process, the lack of multidisciplinary verification, and the lack of automation in control software development. The review shows that current methods and tools attacking these challenges focus on specific points and that developed implementations are rather scarce. Model and data sharing is a key issue to progress towards an overall solution. Furthermore, formalization of concepts like architecture, function descriptions, and requirements needs to be addressed to address their representation.

Regarding the efforts to overcome the identified challenges, industry tends to focus on tool-level integration, while academia focuses on underlying integration methods. Methods proposed by academia seem hard to implement due to the abstract system descriptions, but have a promising future. Also, proposals based on high level representations (e.g., descriptions using functions) have an intrinsic potential to manage model complexity and to support simple and efficient descriptions of system behavior.

3 Architecture-Centric Design

This chapter presents the proposals to address the challenges presented in Chapter 2: an architecture-centric approach for design and the Architecture Model (AM) language and tool implementation which support it. This proposal is also accompanied by a method which helps creating an AM with the required characteristics, where high-level information (i.e., functions) provides a backbone for navigation and overview of design information, while parametric information provides a means to integrate detailed design data. It will also be discussed how these proposals contribute to achieve true MBD implementations.

As seen in Chapter 2, many problems originate at the conceptual design phase, e.g., conceptual solutions from different disciplines are not shared/understood because of lack of flexible shared models, and tools and methods to support conceptual design and information exchange at that level are rather scarce and still have to overcome important shortcomings like the dissociation of information from different sources. Then, what can be done to step towards cooperation and concurrency in design? The hypothesis handled in this work is that representing design information lies at the bottom of such issues, and that using the system architecture is a key stepping stone for developing a common representation for the stakeholders. This hypothesis also responds to the needs rising from the mechatronic industry ([28], [169]), more specifically because of the predominance of (bulky and unstructured) textual information and of models which are not easy to understand and transfer outside their domain or specialty. Once a usable common representation is available, integration can be supported effectively and the design processes can be improved. In this context, integration in product development can be paralleled to achieving a knowledge intensive design environment [168], which aims to allow the flexible exchange and generation of knowledge. The language and tool implementations presented here and the case studies discussed in the next chapter are used to gather evidence that can help supporting these hypotheses.

As discussed in Chapter 2, the “V” development cycle [153] (depicted in Figure 7) and other similar methods are intended to guide the design process systematically, but not many tools support directly their usage. The proposal in this chapter aims to use the system architecture to support such guiding methods by providing:

- A base to document the decomposition phase.
- A formalization to capture design interfaces necessary for the integration phase and its analysis, including feedback of information and iterations.
- A mechanism to trace the effects of requirements on the designed implementations, and vice versa for verification.

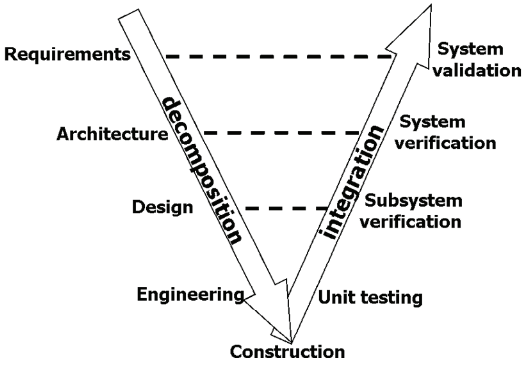


Figure 7. “V” development cycle. Verification follows the dashed lines.

Thus, in the case of supporting the “V” development, the model of the architecture can support a process of integration following the horizontal lines in Figure 7 propagating decisions from left to right, in addition to providing a base document to verifying correctness. An implicit premise behind the use of architectural descriptions here is that the use of “high-level” descriptions can improve multidisciplinary communication and integration. In an intuitive sense, this premise is supported by the fact that high-level design decisions need to be documented with high-level descriptions.

After this introduction, the chapter is structured as follows: Section 3.1 contains a discussion introducing the seminal ideas behind our proposal to support the product development processes. Then, Section 3.2 exposes the core concepts of the architecture-centric design approach. The AM language and modeling tool are presented in Section 3.3 as a possible implementation behind an architecture-centric design approach. The chapter ends with some conclusions in Section 3.4.

3.1 Overview and key aspects

A first concept is that the proposal contemplates an integration framework to support design (as defined in Chapter 1). It aims at supporting the communication among developers and model transformations between tools, similar to the architecture framework described by Browning in [34]. As depicted in Figure 8, a high-level model is meant as a backbone to navigate, give an overview, and classify detailed design information (i.e., managing correspondence), by capturing functions, requirements, and the architecture of the system. In this aspect, the proposal follows the line of reasoning of the methods presented in Section 2.3.1.1, additionally aiming towards integration. The basic hypothesis for the use of functions as integration elements is that from the functional point of view it is possible to describe a system at different levels of detail, focusing on the points of interest to the user while maintaining coherence of the model. The importance of modeling functions for machine and process design was already recognized in works of Rodenacker [142] and Pahl and Beitz [126]. In those references, design is seen as a process of transformation and mapping of information from abstract concepts (i.e., functions and requirements) to concrete descriptions of physical systems, that later will allow manufacturing a system. McDonough states that design is the first signal of human intention [112], and it is argued here that functionality can be used to express that intention. Thus, design cannot be done without the existence of these abstract concepts that specify what the system is expected to do. Careful documentation and modeling of the functional description is then as necessary as it is for any other information related to the design.

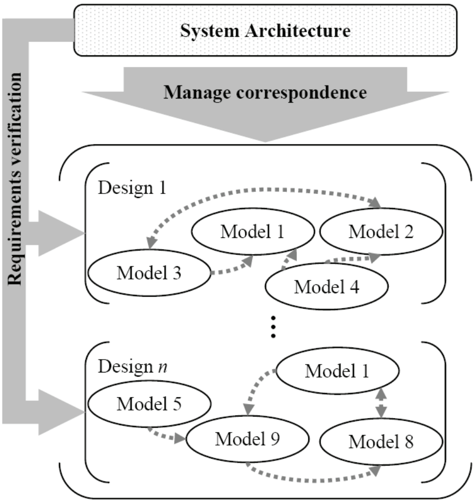


Figure 8. Diagram for proposed approach

Two main reasons explain why the framework is not meant as a single tool in which all design information can be performed. First, the design information of an entire system is too large and complex. On the one hand, creating a model that contains design information with the necessary detail would increase the model size, and create an access bottleneck [56]. On the other hand, providing the operations to model and handle the different kinds of detailed design data in a single tool constitutes another barrier. Second, existing tools are designed and optimized for specific domains, and the designers are proficient with these tools. In practice, each designer is responsible for creating and maintaining the models related to her or his discipline [34].

Besides high-level information, a model of the system requires detailed information at the different stages of design. The lower part of Figure 8 represents the set of stages of the design process. In each stage, the current state of the design is represented by a set of models ("Design *x*" groups in Figure 8, e.g., 3D CAD model, block diagrams). Each of these models can correspond to single or multiple domains and to different levels of detail. To form a consistent design at a certain stage, the models must correspond to each other by sharing certain parameters and their values. For example, at some point in time the control engineer requires the mass of a part in order to simulate a simple dynamic model attached to a controller model. At that point, the mass of the part can correspond to a precise value calculated from a geometric model or to an estimated value laying in a database corresponding to a parts list. An approach that shares several of these aspects can be found in the PACT project [56]. In this respect, an important difference of both proposals lies in the fact that in PACT there is no central shared model, and instead it creates the impression of a shared design model through interactions of agents and facilitators [56].

Information needs to be exchanged between different domain specific design tools in order to integrate the different design activities and to automate analysis, synthesis, and model transformation. An information manager should provide means to navigate, visualize, and ensure consistency of the system model and the associated modeling data. This also includes allowing to form different views which capture and trace the concerns and requirements of stakeholders ([88], [116]). System-level requirements must be decomposed or budgeted and tracked back to the various subsystems and the different domain specific design processes. The main ideas for extracting control related information were also outlined in the original proposal in [6], and the resulting proposal is presented in the second part of this thesis.

Apart from this overview, this section addresses more specifically the relation of the proposal to the topics of model integration (as part of design integration) and functional modeling.

3.1.1 Requisites for model integration

An integrated modeling paradigm that gives the designers a proper view of the system as a whole in several levels of abstraction, and that keeps track of the current state of design is fundamental to attain an integrated design that can cope with the problems brought by complexity [3]. To establish some common grounds for the integration of models, literature proposes some basic requirements which are shared by the current proposal:

- It is necessary to separate the modeler from the solver in order to deal with the definitional integration (i.e. of the models) and the procedural integration (i.e. integration of the solvers) processes separately [63].
- Definitional integration becomes possible as models can be represented in a common language. A conversion of external models to a common language is necessary [63].
- Procedural integration may be more suitable for situations where the models and their associated solvers are of diverse nature [63].
- It is necessary to detect correspondence of variables between models. This seeks to minimize necessary human intervention in the detailed levels of the model integration process. Typing schemes offer an alternative to aid in this process [63].
- Graphical user interfaces and views are crucial to provide model integration support [63].

- One shared database that contains *all* the data of the integrated models quickly becomes a bottleneck [56].
- Modularity, from the point of view of reusability, and the use of model libraries helps to speed up the modeling and verification processes [76].

3.1.2 Function modeling

To give a better idea about the high level model used in the proposed approach, next, a brief description of function modeling is given. Different authors have proposed models, like Function-Behavior-State (FBS) ([167], [170]), Functional Representation (FR) ([40], [41]), Schemebuilder [29], and MACE [87], which incorporate explicit knowledge about functions of systems and devices. For a complete review on functional modeling please refer to [65]. Though the proposals differ in several important aspects, some common points are maintained:

- It is agreed that for modeling purposes the description of a function incorporates the intention of the designer or the given use of a device. Therefore, the definition of a function is independent of the means used to attain it.
- The state of the system is described in terms of the values (qualitative or quantitative) of its state variables. A variable is linked to an object that makes part of a structural description of the system.
- A sequence of state transitions for the system constitutes the explanation of how the function is achieved. In the FBS references this is known as behavior.

These models are not exclusively meant to represent functional, "simplified," knowledge about the system. Thus, the definition of a function is completed with information about how the function is accomplished and which objects (including hardware, software, and knowledge objects) are involved in this process. Most of these modeling approaches define part of this knowledge based on developments such as Forbus' QPT, de Kleer and Brown's qualitative reasoning theories, Kuipers' QSIM (see [18] for an introduction to concepts of qualitative reasoning), and Bond Graph [95] representation theory. The complete structure can be used in applications such as those mentioned in [65].

Use of functional models can be advantageous for several reasons. First, they provide a way of representing the intention of the designers of the system, both for design and for use. Secondly, but not less important, functions can represent a system at several levels of detail, which allows to change the level of abstraction in which the model is seen while preserving, what could be called, the consistency of the model (i.e., the model can still represent the whole system while showing more detail where required). Additionally, functions can model indistinctively hardware, software, and systems from different domains. In a sense, functional models get very close to implicitly represent the architecture of a system.

3.2 An architecture-centric design approach

The architecture-centric approach presented here refers to the refined proposal derived from the core ideas from the previous section and the experiences implementing and using high-level models for supporting design activities (see Section 4.1). It is no surprise that the term "architecture-centric" has been used already to describe similar approaches. The concept of using product architecture as a base for development has been used in the software domain for several years now (*cf.* [26], [103]) to gain advantages such as improving clarity in decompositions of complex systems and allow reuse of design knowledge. In spite of their validity, it is evident that the focus on software on such references entails proposals with a predominant role of software architecture and a heavy use of concepts and tools specific to the software domain.

This section starts analyzing the role of product architecture according to literature under the general context of product development, and then highlighting goals which align to the key aspects presented in Section 3.1. Pursuing these goals and supporting reaching them is what defines the proposed architecture-centric design approach. Then, Section 3.2.2 provides a clear definition of a model of the architecture and the characteristics it should have in order to effectively support an architecture-centric approach to design, which addresses complexity in design. Discussions regarding the nature of the architecture (e.g., whether it should be modular [66] or integrated) do not fall within the scope of this proposal because, as a matter of fact, the proposed approach can be implemented regardless of the specific "type" of architecture of the product and the chosen development process (e.g., the "V" or spiral development processes discussed in Chapter 2).

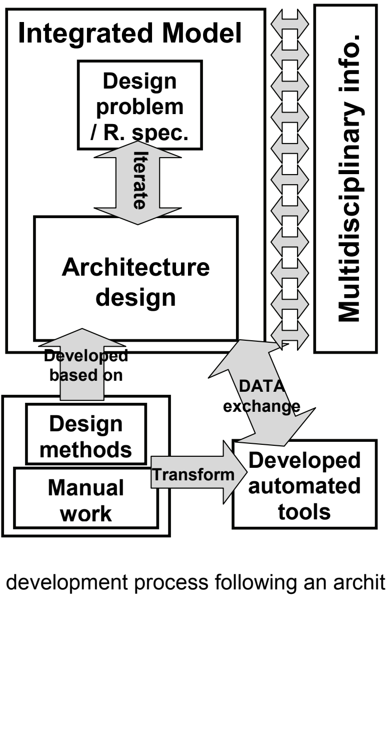


Figure 9. Elements in a product development process following an architecture-centric design approach

Figure 9 contains a schema which generalizes the elements in the workflow of a development processes following an architecture-centric approach. The arrows depict processes of exchange and transformations among blocks depicting sources of information and work. An integrated model representing the architecture covers also the specification of the problem. Such model is developed from multidisciplinary information, including the one related to design methods from different domains. Part of the design methods can be embedded in automated, domain-specific, tools which can exchange shared data with the integrated model.

3.2.1 The role of product architecture

Product development is the subject of many scientific publications as the review material of Krishnan *et al* [101] and Brown *et al* [33] demonstrate. Part of this research stream mentions the role of product architecture in product development explicitly (Ulrich [174], Ulrich *et al* [175]) or implicitly (*cf.* Pahl *et al* [126]) and its impact on a variety of aspects (e.g., Sosa *et al* [152]) such as project management, product innovation, manufacturability, evolvability, etc. However, most of the work centers on describing how the architecture should be (or its "types"), rather than addressing how to practically capture such architecture knowledge, model it, and directly use it to support the product development process. The work of Buede [36] gets close to such a practical view but (is the opinion of the author that), like many systems engineering authors, centers on the role of the system engineer or architect, and does not propose a direct involvement of other stakeholders (e.g., designers, engineers, managers, and their working tools) in the development and use of architecture-level knowledge and models.

The choice of product architecture affects how the product can be updated, its variety and performance, and has implications for manufacturing and product development management (*cf.* Ulrich and Eppinger [175]). Focusing on development management, dividing the system in different modules and parts not only affects how the development teams will be formed to design those parts, but also relates to the interactions between such teams and the planning activities. Ulrich [174] elaborates on this point specifying management differences for modular and integrated architectures. In the case of mechatronic systems, where several domains should be integrated to attain increased or new functionality [126], it results even more useful if the product architecture clearly marks the purpose of any development activity in order to help coordinating the work of stakeholders. In practice, interfaces are not always well known and most products are not modular; a certain level of integration is necessary and supporting integration remains important. Therefore, performance goals and constant communication among stakeholders must be supported to manage the "widespread propagation of changes" [174].

When does product complexity justify the need to support development through explicit guidance from the architecture? As pointed out by Ridley [141], no single person knows how to design from scratch a computer mouse, and no one can even make a simple pencil by himself because making these products requires the knowledge of many people. Architecture may have little use for development of simple products. However, it can be necessary to achieve efficiency when the knowledge of many people has to be used together to design a product, as it is the case in mechatronic product development. In this context the following goals of a product architecture model within an architecture-centric approach to design can be stated, i.e., in which product architecture provides the guidelines for all development activities:

- Provide overview: “Big picture” and “mental model”
- Support integration: Link information necessary for design
- Provide traceability: Allow finding the where/who/when/why for design decisions.

A more comprehensive list of goal uses for architectural models can be found in [88]. The goals above are stated in addition to other goals that can be found in literature referring to the architecture, and mainly concern how the architectural representation contributes arriving to an agreement among the stakeholders, i.e., a consensus. In many cases stakeholders may think that consensus has been achieved, but there is a high risk that no real consensus exists when unambiguous, readable, and well indexed documentation is not available. It is important to clarify that, as stated in previous sections, integration implies linking information from different sources and the tools for analysis, and not joining or merging the sources of information or analysis tools into a single design model or tool.

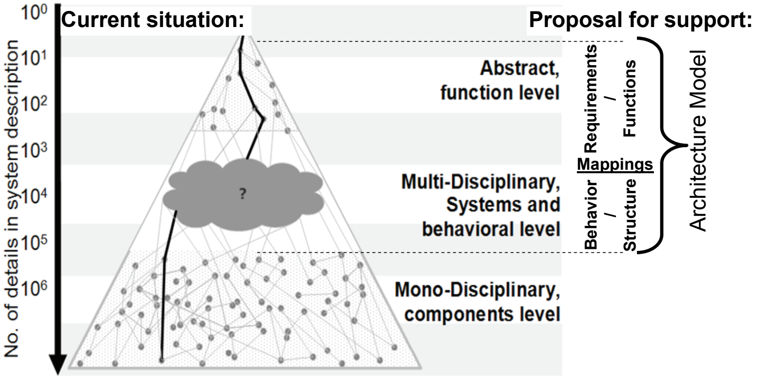


Figure 10. Representation of product complexity (left, adapted from [117] and [44]) and areas which need support for an architecture-centric approach to design (right)

To support these goals of an architecture-centric approach to design, the model which represents the architecture must possess certain characteristics. These are discussed in dept in Section 3.2.2.1; however, first there is place for defining in more detail what is meant by a “model of the architecture” and how such model is used in current practice. The left side of Figure 10 depicts product information at different levels as a pyramid, with an internal line following related concepts. The cloud in the middle of the pyramid represents a lack of traceability at certain levels. An architecture-centric approach to design requires addressing those missing connections. In this section it is proposed how to tackle this by using a model of the architecture containing the information depicted at the right side of the Figure 10.

3.2.2 Modeling and capturing product architecture

Before addressing how to model and capture the product architecture, it is necessary to define what is meant by architecture and its representation. According to [88] (see Figure 11), the system architecture embeds the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

This work adheres to the generic definition of product architecture representation from the ISO/IEC 42010 standard [88], corresponding to an aggregation of models organized by several views, as shown in the encircled area of Figure 11. The views correspond to concerns coming from the product development stakeholders (engineers, designers, architects, users, etc.). Consequently, these views must refer to the objects in the domain-specific models that the stakeholders use for development and design. However, here it is considered that these views must clearly relate to each other (possibly by a central model) and be represented in a language clear to all stakeholders. These requirements intend to deal with the model interpretation problem described in Chapter 1. Considering this definition, it is possible to discuss how models of the architecture are usually developed and utilized.

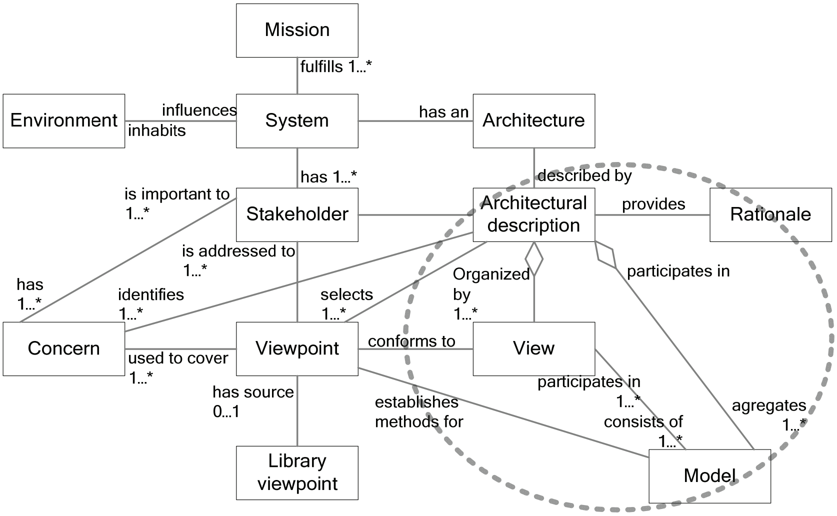


Figure 11. Conceptual model of an architectural description, adapted from [88]

3.2.2.1 Architecture models in current practice

The product architecture can be captured in many ways, which gather information from sources as requirements specifications, models describing the product interfaces and structure, relations to functional architectures, etc. These documents and models are available in many cases, but they can be rarely put together to get a comprehensive overview of the architecture.

It can be observed that in practice the model objects specific to most domains are hard to use as a support to communicate with other domains because of their variety, and in spite of their widespread use. For example, block diagrams are used extensively by developers in the control domain. Many engineers are acquainted with such diagrams, as they are introduced to most engineers during their education, but people from other disciplines and backgrounds can have a hard time following (or even misinterpret) an explanation supported by a block diagram. Authors like Jobling [94] have questioned the appropriateness of block diagrams to express other concepts than the signal flows and transformations these diagrams were created to represent, especially during conceptual design. Reflecting this feature, sometimes "conceptual block diagrams" are used in earlier phases of design with a broader user composition. A similar example for the case of bond graphs can be seen in the "word bond graphs" [95]. Yet another example can be found in the SysML which, despite its formal specification [123], still seems to require additional explanations giving a clearer purpose to its modeling constructs in works like the ones of Friedenthal *et al* [72] and Welkiens [189]. All these cases of models, which currently support most MBD approaches, provide evidence that MBD still is supported mainly by DSM as stated in Chapter 1.

In practice, text documents (e.g., reports, memos, emails) are used to support communication among stakeholders from different domains during product development ([25], [188]). These documents hold a significant amount of knowledge conveying different types of information concerning: the context of the discussion, the involved objects and their relations, the performed functions and purpose, the performance characteristics, and the correspondence between objects and functionality. The following example of a textual description illustrates these points: *The paper path transports the paper through the different processes required for printing a page using pinches. First, the paper is taken from the input trays and placed at the entrance of the paper path. Then, the paper is transported through different units that perform the steps of the process to print one side (of the paper). Lastly, the paper can be taken to the exit of the printing engine, flipped or not, or it can be flipped and taken to print the second side.*

Even assuming this information is complete and unambiguous, the text description mixes the "different types" of information (also see [188]), making it more difficult to understand than when information is properly identified or separated. In other words, useful descriptions include information from different sources and different characteristics. Just gathering the common information in its original format (i.e., domain-specific models) for a discussion is not enough because, without a clear association among the domain-specific concepts, it would require that each stakeholder understands the specific language of every other involved stakeholder.

Besides the text-document-based approach above, other approaches have been used to support modeling and using the product architecture. Some of those approaches have been tested in industrial settings but in spite of this remain marginally known and used in practice. A thorough revision and analysis of the available methods falls out of the scope of this chapter (*cf.* Chapter 2). Nonetheless, a brief comment on the current use of such approaches can be added by referring to a series of tests performed by Borches in a recent work [25], which presents characteristics of five different models used to represent parts of the architecture of a complex mechatronic product developed in a company. The study includes informal approaches without defined formal representations as well as approaches that use formal models such as SysML or design-structure matrices (matrix representation of different system relations). Though interpretation of those results can vary and its source seems to have a strong preference for models that do not use a specific software implementation or language, some conclusions regarding current use of such models in industry are drawn here, complementing the conclusions in Chapter 2:

- Most stakeholders find functional and physical views useful
- Simple languages promote communication but in exchange for model size and content detail
- Information visualization creates conflicts among model size, capturing the overview, and providing enough detail
- Complex languages/tools increase unacceptably the work load of stakeholders.
- Increased work overhead reduces model update and feedback
- Stating the source of information helps increasing trust in it

Additionally, it is concluded here that the use of models of the architecture in practice is low because stakeholders do not find an immediate practical use for them which justifies investing additional work for their creation. As discussed in Chapter 2, the existing (and rather scarce) tool support does not improve model intelligibility and accessibility enough to decrease the net required design effort. The panorama for current product development practices has been depicted in Figure 12. The left side contains the pyramid introduced in Figure 10. The right side depicts a generalization of current development processes, following the same conventions as in Figure 9, in which information and work from different domains have a weak connection with the architecture of the system. The correspondence of information in both ends of the figure is marked by dashed regions and arrows, showing overlaps and missing correspondence between available sources on the right and traceability on the left.

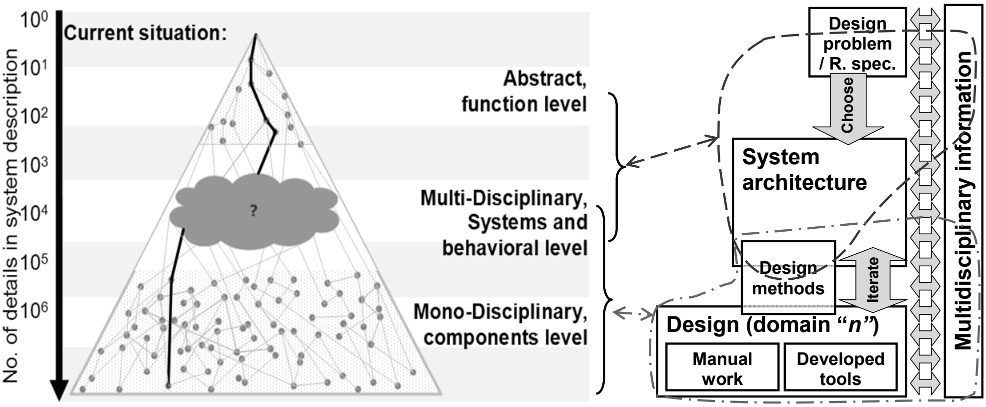

10⁰
10¹
10²
10³
10⁴
10⁵
10⁶
No. of details in system description
Current situation:
?
Abstract, function level
Multi-Disciplinary, Systems and behavioral level
Mono-Disciplinary, components level
Design problem / R. spec.
Choose
System architecture
Design methods
Iterate
Design (domain "n")
Manual work
Developed tools
Multidisciplinary information

Figure 12. Representation of product complexity (left, adapted from [44] and [117]) and relation to elements of current development practices (right)

3.2.3 Desirable characteristics of a model of the architecture

This section presents and discusses desirable characteristics of product architecture models to support architecture-centric design. The purpose is not describing in detail a specific language, nor comparing particular languages or methods with respect to the desirable characteristics described here. The characteristics are derived from the reviews and experiences documented in this thesis, from which it is possible to identify the need for a language which:

- Supports communication independent of stakeholder's background, which requires 'simple' modeling objects. Also, graphical representations are of great value for this purpose ([36], [63]).
- Enables relating any domain-specific data to a common model, so the data can be shared along with the part of the context (i.e., relations within and out of the domain) contributed by each stakeholder. This means that the model should allow automatically recalling the context of every object so there is no need to copy it every time a description is made.
- Allows representing the different types of information introduced in the previous section 3.2.2.1.
- Permits promptly identifying what each kind of information is, and what it relates to.
- Facilitates identifying completeness of the information.

It is preferred that the model has a computer-readable format, because computers allow fast, precise, and safe data transfer, as well as reducing the effort to replicate and modify information. Moreover, many domain-specific tools process digital data, and the data can be automatically formatted (i.e., transformed) if the source and destination languages are know.

For the proposal, the “views” concept introduced in Figure 11 is abstracted as identifying groups of objects. Since the objects must be structured somehow to form a single model, relations among the objects become necessary. Though in principle all groups are equally important, certain generic groups that can be used to support almost every discussion between stakeholders and that work as a backbone for navigation and indexing of the model have been identified. Based on the desired characteristics and the different types of information, it is possible to recognize the need for four different generic groups of objects, which work as views of the architectural description in Figure 11. The first three groups represent information that is used by almost every stakeholder (i.e., not domain-specific), while the fourth group provides a generic set with which it is possible to capture other particular (domain-specific) groups that the stakeholders may require.

- A first group representing all the desired or intended specifications of the product. This includes the functionality and requirements that the product must fulfill. There must also be a mechanism to express required behavior. This may include a hierarchical function decomposition that reflects design choices, as well as ordering of functions, because performing functionality in different order can lead to different products.
- A second group representing the actual (software and hardware) components of the system and the relations/interfaces among them. Objects in this group fulfill and implement the desired characteristics represented in the first group.
- A third group representing formalizations, such as mathematic relations, of the phenomena that direct the behavior of the system as represented in the second group.
- A fourth group of mappings linking elements from the three groups above to the domain-specific information structures/models that the stakeholders use within their particular domains. Mappings to the first group explain the purpose of the linked models. Mappings to the second and third groups indicate which objects and behaviors are (partially) represented in the specific model. This group should allow representing the global structure of the linked domain-specific models, which in turn allows referring to specific parts of them and facilitates understanding.

Additionally, in the proposal it is recognized that the product parameters/attributes/-variables (called here ‘parameters’) used to represent characteristics of the product in almost any specific model, are related to objects in all the previous groups and thus it is useful to represent them as objects in their own right. Parameters provide an additional mechanism to (indirectly) interrelate objects from the four groups described above, and to maintain consistency. All the objects and relations from the proposed language constitute a network that can be filtered and visualized, presenting to the user the relevant information for the task at hand. As said in [36], authors such as Levis [104] have identified concepts for the multiple-architecture approach which are very similar to the first three groups. However, the fourth group (which can be related to the viewpoints in Figure 11) are rarely explicitely addressed.

There are many possible implementations for a language with the characteristics presented here using (meta)languages that allow creating formal classes and instantiating models complying with such formalizations (e.g., UML). The next section captures previous work of the authors [11] and provides a detailed explanation of the implemented language and modeling method which have been used to test the architecture-centric approach to design.

3.3 The Architecture Model (AM)

Though there is no globally agreed definition of system architecture, it can be said that it should convey what the main parts of the product are and what they do individually and as a group. As stated previously, this proposal considers the definition presented in [88] (Figure 11). Similar perspectives considering multiple views are shared by authors like Muller [117] and Browning [34]. The views in the architecture represent relevant information to specific stakeholders. To obtain a model of the product such views have to be organized and linked to each other. An obvious choice is to place common software/hardware objects in the views and use them as the integrating part. Nonetheless, using only the real objects to join the views can be restrictive because (as Muller highlights in [117]) the architecture of a product is fuzzy and contradictory, and therefore difficult to capture consistently in a model. Using other abstract representations to overcome such problems is proposed in Section 3.2.3 and next subsections.

The proposal consists of a high level representation, closer to conventional systems engineering approaches that promote the use of system architecting. In spite of their high-level nature, the abstract elements allow representing situations at any level of detail. Functions are used in the model to amalgamate descriptions in different domains and levels of detail.

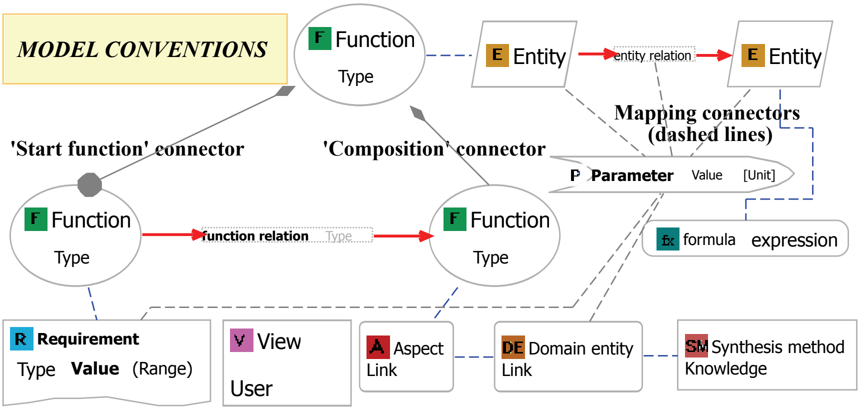


Figure 13. Conventions for the objects in the AM (all possible links not shown)

As indicated earlier in this chapter, apart from the characteristics stated in Section 3.2.3, the language is inspired by the work of Umeda *et al* on the FBS model [176]. Figure 13 gives an overview of how the information of the architecture can be modeled using the AM language, by instantiating several objects and placing relations among them. The objects are used to clearly identify different types of information, while allowing making descriptions using several object types (many other languages, like SysML, separate object types in different diagrams).

General definitions for the main components are given next (ordered alphabetically). The precise meaning and use of these objects will be explained in the following pages. All the allowed mappings among elements have not been completely specified, but some links and topologies will be suggested to represent several pieces of information. It is also important to mention that the objects have several attributes which allow embedding in them more information (*cf.* Section 3.3.6) but that the use of many of these attributes has not been fully specified and may vary between objects.

- Aspect - A: Represents a specific concern, which can link/group other related objects (e.g., F, De, SM)
- Design task (referred in some publications as synthesis method - SM): A representation of a design activity or process.
- Design task relation - DTR: Represents input/output information exchanged between design tasks.
- Domain entity - De: Representation providing a link to an object in an external model.
- Entity - E: A system object (software or hardware) which performs functions.
- Entity relation - Er: A relation among entities, entailing transfer of energy, information, or mass.
- Formulae - fo: Mathematic representation (equation, inequality, etc.) of a relation among parameters.
- Function - F: Description of an objective or function of the system from a subjective point of view.
- Function relation - Fr: Transition among functions used to describe behavior (i.e., sequences of functions).
- Parameter - P: Representing any variable or parameter which can be used to quantify a property. These objects can be mapped (directly or indirectly) to other objects allowing to build concrete descriptions.
- Requirement - R: Description of an objective or constraint related to values of parametric information.
- View - V: Used to arrange information in a pictorial representation to provide an explanation.

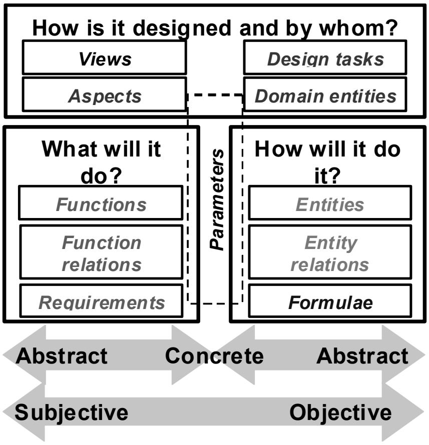


Figure 14. Relation of modeling elements to the architecture information spectrum

The AM language is an enabler to the uses presented in Section 3.2.1. In general, it allows to its user modeling the answers to the questions:

- What will the system do? Or what will the stakeholders want the system to do?
- How will the system do what it is required? Or how is the design implemented?
- How is the system designed and by whom? Or which methods, models and tools are used by the stakeholders?

These questions can be answered at an abstract (stated without reference to a specific instance) or concrete (relating to an actual, specific thing) level as well as with subjective (particular to a given person) or objective (presented factually) information. As discussed in Section 3.2.2.1, capturing information through the entire spectrum of these characteristics is necessary to provide a more complete understanding of a system. Figure 14 represents how the modeling elements relate to the questions above (one box for each question) and to the different information characteristics (arrows at the bottom). The lower left box elements are used to build a specification of the system corresponding to subjective needs. The lower right box groups the elements used to describe directly how the system is formed and how it behaves. The elements in the upper box represent particular points of view and a direct relation to other external, domain-specific, models (i.e., different than the AM, such as the documents presented earlier in the introduction).

A point of view similar to the one in Figure 14 is shown in Figure 15. However, for practical issues and corresponding to the four object groups presented in Section 3.2.3, this latter perspective makes a clear distinction among information describing the system and information describing the phenomena ruling its behavior. Under such point of view, four layers compose the AM. A graph of functions and related requirements constitutes one layer (Section 3.3.1). The function layer is mapped to a second layer containing information about the structure of the product (Section

3.3.2). In turn, the structure layer maps to a third layer containing information about the continuous behavior of the system (Section 3.3.3). The mapping between the behavioral and structural layers links the behaviors described in the domain-specific models. The mapping between the structural and functional layers allows tracing requirements data and providing a context to the implementation. The term ‘parameter’ is used indistinctively for all variables, attributes, and values that represent characteristics of the product in the different domain-specific models. Other indirect mappings (not shown in Figure 15) can also be found through parameters. Parameters of the set $P=\{p_1, p_2, \ldots, p_{Np}\}$ are identified by their names through the entire model. In all cases, ‘mappings’ refer to edges that form undirected bipartite graphs. A prototype modeling tool that allows partitioning the model for readability and that ensures its consistency has also been developed (Section 3.3.6). A fourth layer of the model (Section 3.3.4) links its parameters to external models and model creation methods used for specific purposes. This part of the model is coined here as the external communications layer. A ‘complete AM’ contains elements in these four layers, and ideally the mappings among them can be represented as onto functions, i.e., there are not “loose” elements which lack direct mappings (or to relax this condition, indirect ones) to other layers to which they can map. In that way, every function will be implemented by some entities, and every entity will have some behavioral representation. The ‘view’ nodes help organizing the information according to the specific interests of a stakeholder (*cf.* Figure 16 to Figure 19).

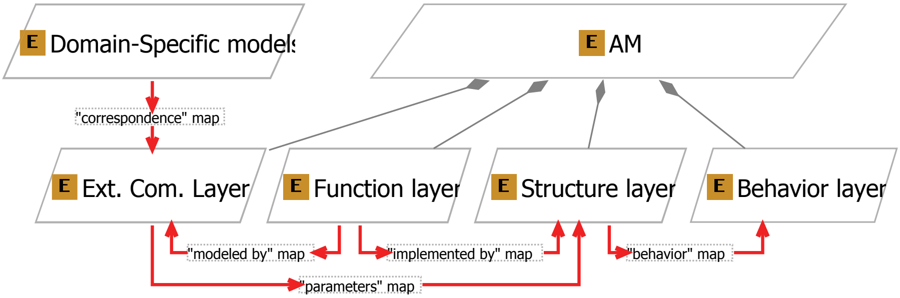


Figure 15. Structure of the AM depicting main objects and their mapping relations.

A commercial, configurable mobile robot platform known as Boe-Bot [127] is used in this section to show examples of the proposed models. In this particular case, the robot should move in an area and detect the position of obstacles in it. The data captured by the robot is transformed into a map of the area using an external PC to which the robot communicates as the exploration progresses.

3.3.1 Function Layer

Functions are used to describe what the product is expected to do. Much research has been done regarding functional representation and its purpose [65]. Some (e.g., [176]) regard functions as subjective descriptions that link the personal human intentions to the objective behaviors of the product. Other consider functions as transformers of energy, material and information (e.g., [85], [126]), describing almost directly what objects do. This work adheres to the first definition, with the purpose of providing a common model for all stakeholders. This common model reflects the design rationale and links desired characteristics of the product (i.e., requirements) to the domain specific models made by designers. This point of view about functions helps separating the definition of view from the restrictive ordering imposed by physical objects (i.e., software and hardware). Separating the functional and structural layers allows to correctly define a design, differentiating its intended use from its physical structure descriptions. For example, the differential drive of the Boe-Bot achieves translation and rotation functionalities in an integrated manner with a common structure, but decoupling at the same time translation and rotation functionalities during operation (i.e., the robot can rotate without translating). An implementation with additional direction wheels dedicates such components to the rotation functionality.

Before defining functions more precisely it is in place to introduce ‘requirement’ nodes. Requirement nodes belong to the set $R = \{r_1, r_2, \ldots, r_{Nr}\}$, and represent necessary or desirable characteristics of the product and its behavior (hence their name), agreeing with the definition presented by Buede [36]. To maintain a clear connection to the implementation and to allow verification, a requirement node should always refer to a parameter. This can be done directly through a mapping to a parameter node $Rp = \{Rp_{ij}\} \subseteq (R \times P)$, or indirectly, composing a requirement from other existing requirements using decomposition links $Rd = \{rd_{ij}\} \subseteq (R \times R)$, eventually mapping to parameters. A requirement defines characteristics on a group of parameters by constraining their values.

In Figure 16 the requirements are defined by decomposition or mapping to parameters, also identifying their type and, when mapped directly to a parameter, their value. The requirement of “obstacle is found” indicates a situation in which the distance to an obstacle is less (type ’<’) than a quarter of a meter. The requirement of “move straight forward” indicates that both, “remain in angle” AND “robot moving forward”, are considered together. In turn, “robot moving forward” specifies a situation where the value of “Boe-Bot speed” must be the same (type ’=’) as the value of “robot maximum speed”.

The functional layer of the architecture has graphs where the node set $F = \{f_1, f_2, \ldots, f_{Nf}\}$ corresponds to the ‘functions’ of the product (see Figure 17). Functions are described freely by the designer, but as a guideline, function nodes are named as actions in the form of “to do something”. The authors also explore possible uses of a formal classification for functions, which facilitates computational processing of functional information, by adding a ‘type’ to each function. Mapping a function and a requirement node $Rf = \{rf_{ij}\} \subseteq (F \times R)$ indicates that the function should try to obtain, maintain, or comply with the requirement. ‘Composition’ links $Fd = \{fd_{ij}\} \subseteq (F \times F)$ model how a function is decomposed into subfunctions, as in a function breakdown structure [108], adding more detail to its definition. The decomposition relations $Fd(Gfd)$ and the functions $F(Gfd)$ form a directed acyclic graph Gfd. Some guidelines for function decomposition can be found in [36] and [181].

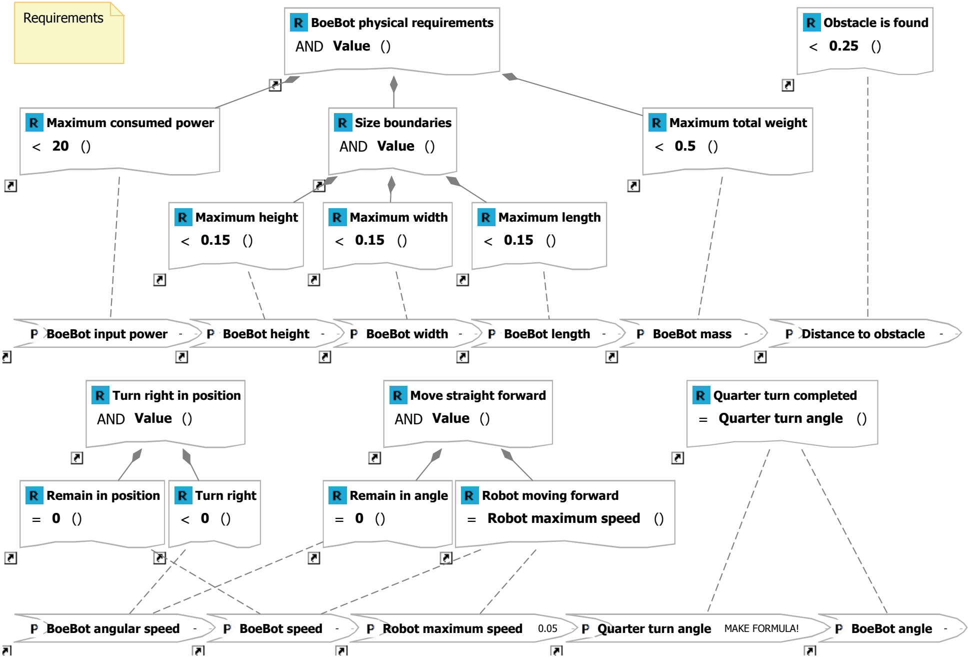


Figure 16. A requirements definition view in the robot AM

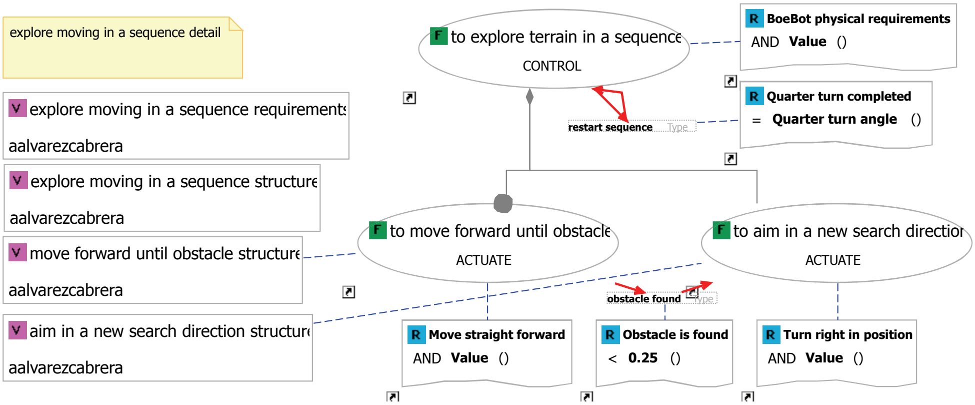

explore moving in a sequence detail
V explore moving in a sequence requirements
aalvarezcabrera
V explore moving in a sequence structure
aalvarezcabrera
V move forward until obstacle structure
aalvarezcabrera
V aim in a new search direction structure
aalvarezcabrera
F to explore terrain in a sequence
CONTROL
R BoeBot physical requirements
AND Value ()
R Quarter turn completed
= Quarter turn angle ()
restart sequence
Type
F to move forward until obstacle
ACTUATE
F to aim in a new search direction
ACTUATE
obstacle found
Type
R Move straight forward
AND Value ()
R Obstacle is found
< 0.25 ()
R Turn right in position
AND Value ()

Figure 17. Navigation functionality view in the robot AM

'Function relations' of the set $Fr = \{fr_1, fr_2, \dots, fr_{Nfr}\}$, and their links to functions $Frf = \{frf_{ij}\} \subseteq (F \times Fr)$ document how a process should be carried out. The necessary condition to perform a function is modeled by means of a mapping to a requirement node $Frr = \{frr_{ij}\} \subseteq (Fr \times R)$, indicating activating conditions for the function in the arrow end. Function relations $Fr(Gfr)$ and functions $F(Gfr)$ form a bipartite digraph Gfr, and add information of function activation criteria based on requirements data. Function relations are used effectively to introduce a partial (temporal) ordering among functions when describing a process that requires several steps under different conditions; i.e. at different instants.

Figure 17 depicts how a function is decomposed into two subfunctions, how each of these aims to achieve its requirements, and how the functions should activate under certain requirements. To "explore moving in a sequence" the robot will first "move forward until obstacle," as indicated by "moving straight forward". When the condition "obstacle is found" is reached, it will "aim in a new search direction" as specified in "turning right in position". The sequence is repeated upon the circumstances of a "quarter turn is completed", reactivating the main exploration functionality. Additionally, the exploration functionality and all its subfunctions must adhere to the constraints imposed by "Boe-Bot physical requirements", e.g., using less than the maximum allowed power (*cf.* Figure 16). The example shows the three roles that a requirement node can play, namely, constraining the values of parameters when performing a function, providing target values for the execution of a function, and indicating a set of parameter values under which a function will become active.

3.3.2 Structure Layer

Capturing the structure of the product is crucial to show how components relate to each other. The structure layer corresponds to the intuitive view of the product as it is seen through our eyes: a set of objects exchanging material, energy and/or data. The structural layer contains a set of 'entity' nodes $E = \{e_1, e_2, \dots, e_{Ne}\}$, representing hardware or software components. Also, entities can be composed of other entities using 'composition links' $Ed = \{ed_{ij}\} \subseteq (E \times E)$. Similar to the case of the functional layer, decompositions form an acyclic directed graph Ged. The relevant parameters of an entity node are mapped directly to it as $Ep = \{ep_{ij}\} \subseteq (E \times P)$.

Entities are related to each other through 'entity relations' $Er = \{er_1, er_2, \dots, er_{Ne}\}$, and their links $Ere = \{ere_{ij}\} \subseteq (E \times Er)$, which show the exchange of matter, energy and information among them, as indicated by the parameters that can be mapped to the relations $Erp = \{erp_{ij}\} \subseteq (Er \times P)$. The topology modeled in the structure layer does not consider time. Thus, through the bipartite digraph Ger, entities $E(Ger)$ and their relations $Er(Ger)$ form together a model of the possible topologies of the product on different modes. In the AM, entities are mapped by $Fe = \{fe_{ij}\} \subseteq (F \times E)$ to nodes in the function layer, showing how the functions are implemented.

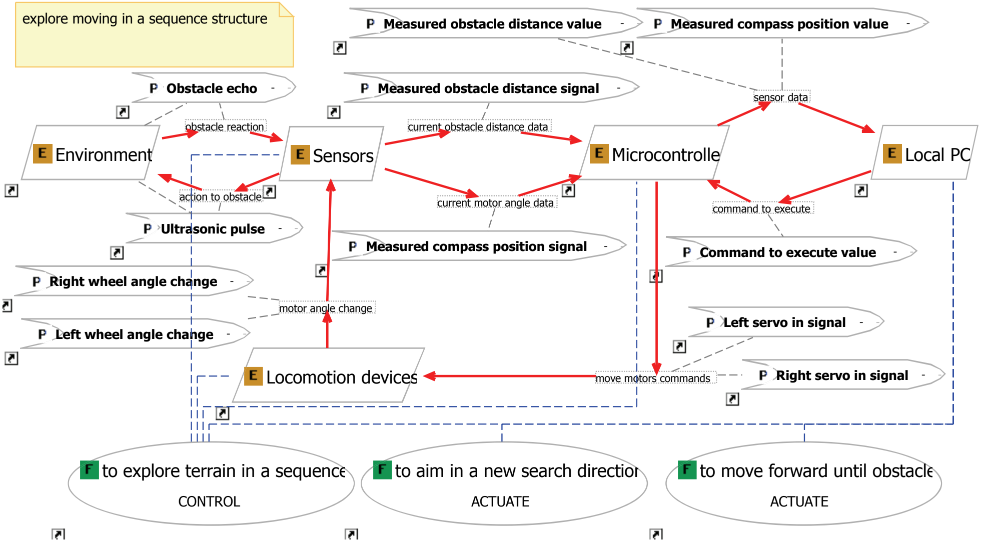

explore moving in a sequence structure
P Measured obstacle distance value
P Measured compass position value
P Obstacle echo
P Measured obstacle distance signal
sensor data
obstacle reaction
current obstacle distance data
E Environment
E Sensors
E Microcontrolle
E Local PC
action to obstacle
current motor angle data
command to execute
P Ultrasonic pulse
P Measured compass position signal
P Command to execute value
P Right wheel angle change
motor angle change
P Left wheel angle change
P Left servo in signal
E Locomotion devices
move motors commands
P Right servo in signal
F to explore terrain in a sequence
CONTROL
F to aim in a new search directior
ACTUATE
F to move forward until obstacle
ACTUATE

Figure 18. View presenting the structural description related to the navigation functionality in Figure 17

The example in Figure 18 shows some of the components of the robot and other external components, together with the relations among them which result interesting at the level described by the functional description in Figure 17. Sensors provide data to detect the obstacles and robot facing to a software module in a separate PC, which sends back the motion command to be executed back to the robot.

3.3.3 Behavior Layer

To obtain an understanding of the behavior of the product it is necessary to obtain information about its structure and combine it with information about the involved phenomena. Phenomena can be physical or man-made, e.g., Newton's second law or input-output relations in a software program. The main goal of the behavior layer is to contain a description of the behavior or phenomena that rule how the product works and the relations between its parameters. Such information is present in domain-specific models (e.g., state-space model of a system) because they can describe part of the behavior in detail. The intention is not to store all these details of the behavior in the AM, but to provide a "lite" version which can be shared and analyzed easily. Like that, it is possible to see the relations between the domain-specific models at a high level.

The behavioral layer is built from sets of relations (e.g., equations, causality) represented as 'formula' nodes fo_{No} from the set $Fo = \{fo_1, fo_2, \ldots, fo_{No}\}$, mapped directly to the individual entities $Ef = \{efo_{ij}\} \subseteq (E \times Fo)$, entity relations $Erf = \{erfo_{ij}\} \subseteq (Er \times Fo)$, and/or to the functions $Ff = \{ffo_{ij}\} \subseteq (F \times Fo)$. Each formula specifies relations among parameters, forming what is called here concrete behavior. A set of formulae can thus be interpreted as a consistent set of parameter relations defining a concrete behavior, for example, a system of differential equations. Such concrete behavior (set of formulae) can be mapped to entities or to functions, describing respectively the concrete behavior related to an object or to a specific task.

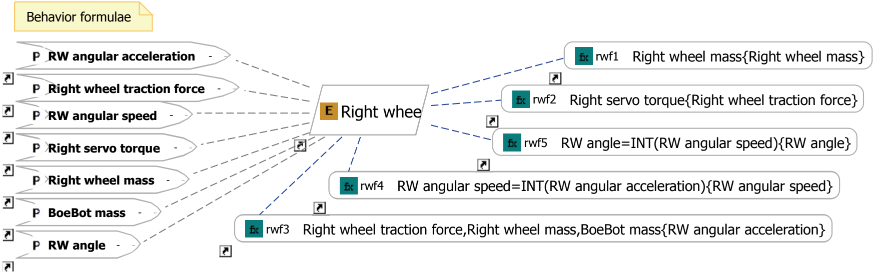

Behavior formulae
P RW angular acceleration
P Right wheel traction force
P RW angular speed
P Right servo torque
P Right wheel mass
P BoeBot mass
P RW angle
E Right whee
rwf1 Right wheel mass{Right wheel mass}
rwf2 Right servo torque{Right wheel traction force}
rwf5 RW angle=INT(RW angular speed){RW angle}
rwf4 RW angular speed=INT(RW angular acceleration){RW angular speed}
rwf3 Right wheel traction force,Right wheel mass,BoeBot mass{RW angular acceleration}

Figure 19. View presenting formulae describing dynamics of a wheel

The language for writing such relations is not specified in the proposal. However, depending on the use given to such information, additional specific information and/or format may be required. Concrete examples of this will be presented in Part II. Figure 19 depicts some formula definitions related with the global dynamics of the robot.

3.3.4 External communications layer

The main task of the external communication layer is to link the model parameters to external, domain-specific models. The AM parameters are directly linked to parameters in external models. However, the AM also has objects to model views that give an idea of the purpose and nature of the linked models.

Commonly, models are built during design to test whether the design will accomplish a function or to help defining details in the design, i.e. verification or synthesis. In both cases, the model refers to values that define the desired performance of the product while carrying a function, represented here with requirements. An ‘aspect’ node in $A = \{a_1, a_2, \ldots, a_{Na}\}$ acts as an envelope for a set of external models (model fragments) which are used to evaluate a common concern linked to a functionality, which may extend over several disciplines and stakeholders. An aspect node maps by $Fa = \{fa_{ij}\} \subseteq (F \times A)$ to a function node, linking it to its purpose and to the involved requirements. The external models themselves are represented with the ‘domain entity’ node set $D = \{d_1, d_2, \ldots, d_{Nd}\}$, which can be mapped $Dp = \{dp_{ij}\} \subseteq (D \times P)$ to parameters in the AM, forming a direct connection to provide updated data to the external models. The domain entities can be mapped directly to aspect nodes $Da = \{da_{ij}\} \subseteq (A \times D)$. Domain entities may be composed $Dd = \{dd_{ij}\} \subseteq (D \times D)$ to reflect the structure of external model fragments. Figure 20 depicts an example diagram in the external communications layer.

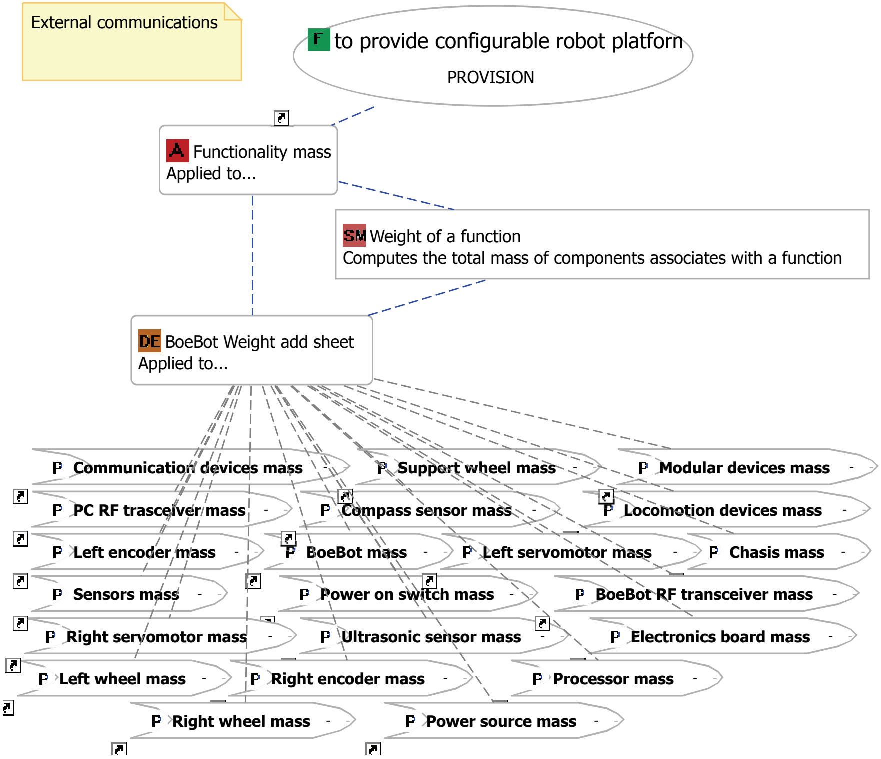


Figure 20. View representing the link between parameters of the robot and an external Matlab model

The objects above are used to represent the link to external models and their purpose. Additionally, it results useful to represent how these external models are obtained or computed. The ‘synthesis method’ node set $S = \{s_1, s_2, \ldots, s_{Ns}\}$ represents operations or data processing done with a set of domain entities and their mapped parameters. Thus, the synthesis method node can be mapped to an aspect node $As = \{as_{ij}\} \subseteq (A \times S)$ or directly to the involved domain entities $Ds = \{ds_{ij}\} \subseteq (D \times S)$. The synthesis method object counts with methods that allow exporting/importing data from/to the AM and identifying or executing the tools and process that handle such data. Practically speaking, the external communications layer supports the scenarios in which the AM has a direct link with an external modeling tool, by providing means to identify or label the information and document its intended use. There are several possibilities to perform what is coined here as labeling, e.g., direct identification of nodes through attributes, mapping of nodes to external label data. One mechanism considered here to be intuitive and close to actual design practices, while remaining human and machine readable, is based on the aspect nodes a_i and their mappings to the function and domain entity nodes (*resp.* fa_{ij} and da_{ij}). Detail concerning how the information is transferred among external models through the AM falls out of the scope of this chapter. See[193]-[195] for more details or Part II for specific examples.

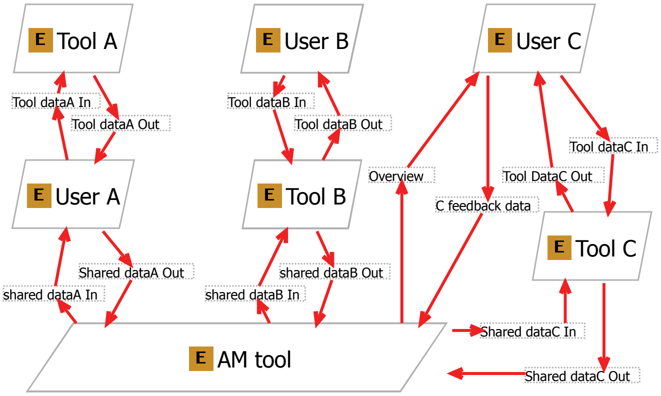


Figure 21. Structure of AM usage scenarios

The structure for several usage scenarios supported by the AM is presented in Figure 21. The tool implementing the AM (see "AM tool" in the figure) can be directly utilized by the users to edit the AM, which can be linked to models from specific domains ("Tool n" in Figure 21). Shared data (e.g., requirements) and an overview of the design process status are provided by the AM to the linked tools/users, so they can execute simulations/verification with updated data. This approach supports itself in the premise that designers use their "native" models and tools, and therefore, all the design computations (simulations, optimizations, transformations, etc.) are carried in the domain specific tools. Domain specific models are interlinked through the AM mainly by the parameters, and entities and functions provide common objects for discussion. The use of functional abstractions provides a base for information reuse. Details about reuse of information are not treated in this thesis, as they are mainly a contribution of Woestenenk *et al* [195]. However, some related aspects are discussed here at following sections covering modeling and implementation of the AM and in Part II and Section 3.3.7 when the objects in this layer are used.

3.3.5 Model construction process

Building the model methodically is vital to streamline the modeling process and its associated development process. Though layers of the AM do not need to be made in any particular order (the data structure is "flat", *cf.* Section 3.3.6), the proposed method to build an AM uses the functional layer as a backbone. The purpose of is to facilitate navigation by any user, by organizing the model based on the functional information. Figure 22 depicts the arrangement of the layers. Each functional diagram provides links (through 'view' nodes) to a requirements definition diagram, structural layer diagrams, and to other function layer diagrams. The requirements definition diagram models the mappings between requirements and parameters, keeping the function diagram leaner. Mappings among functions and entities are placed in the structural layer diagram; see the "Structure & Relations & (Function)" blocks in Figure 22. When mappings to functions clutter the structure diagram, the user can make a separate diagram to map functions and entities, in which entity relations and parameters are omitted. In turn, the structure diagram provides a link to a behavior diagram showing the mappings between entities and formulae, and a link to an external communications diagram.

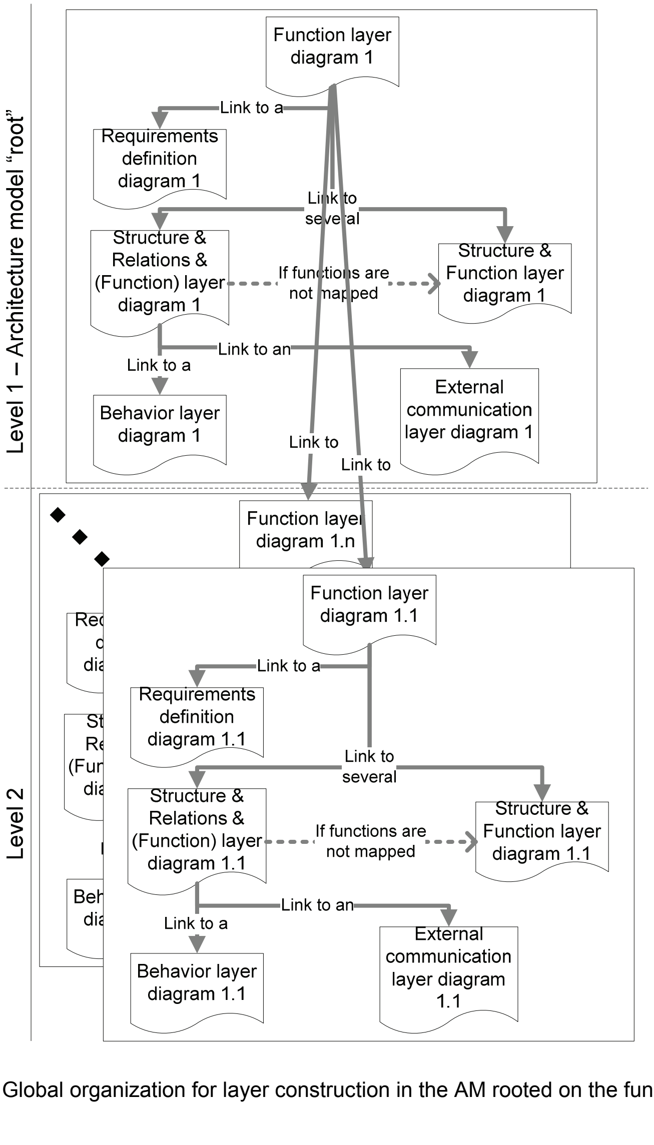

Level 1 – Architecture model "root"
Function layer diagram 1
Link to a
Requirements definition diagram 1
Link to several
Structure & Relations & (Function) layer diagram 1
If functions are not mapped
Structure & Function layer diagram 1
Link to an
Link to a
Behavior layer diagram 1
External communication layer diagram 1
Link to
Link to
Level 2
Function layer diagram 1.n
Function layer diagram 1.1
Link to a
Requirements definition diagram 1.1
Link to several
Structure & Relations & (Function) layer diagram 1.1
If functions are not mapped
Structure & Function layer diagram 1.1
Link to an
Link to a
Behavior layer diagram 1.1
External communication layer diagram 1.1

Figure 22. Global organization for layer construction in the AM rooted on the function layer

3.3.5.1 Modeling process flow

In addition to the AM structure suggested in Figure 22, two sample modeling process flows are depicted in Figure 23 and Figure 24. However, it must be highlighted that the objective is to present examples of how to build an AM in an orderly fashion and no statement suggests that any single modeling workflow is convenient for every situation.

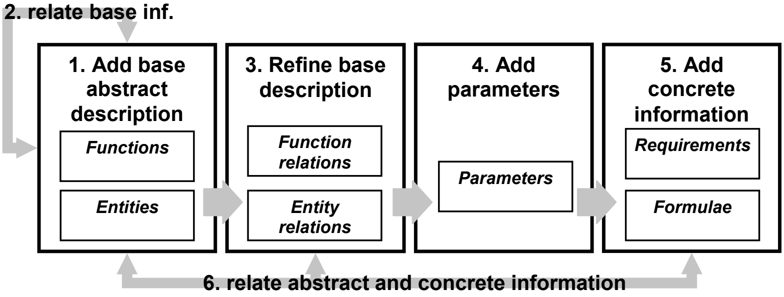

2. relate base inf.
1. Add base abstract description
Functions
Entities
3. Refine base description
Function relations
Entity relations
4. Add parameters
Parameters
5. Add concrete information
Requirements
Formulae
6. relate abstract and concrete information

Figure 23. Model objects (italicized) and proposed modeling steps (iterations, not shown) to make an AM by increasing concreteness

One way to build the model, which mainly corresponds to progressively increasing the concreteness of the information, is presented in Figure 23. This process results to be intuitive in many cases when the idea of a problem and a system solution are being conceived. Additionally, in the first step, it is possible to select the function or the structure layer as a basis for the model. Using the functional layer as a backbone may be better when what wants to be achieved is known beforehand and structure is designed around it, while the structure may also provide an appropriate backbone for some descriptive models of existing solutions.

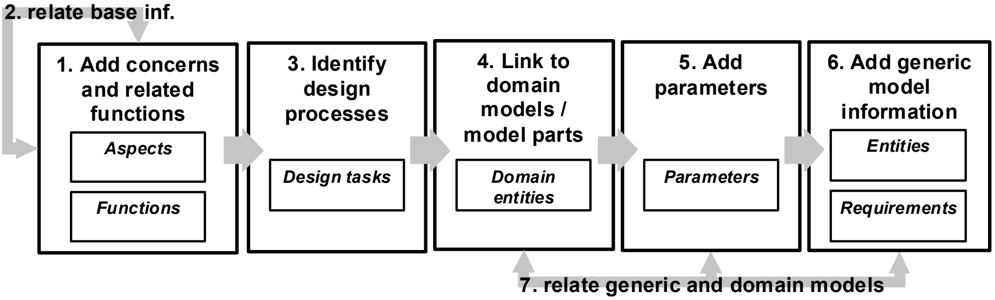
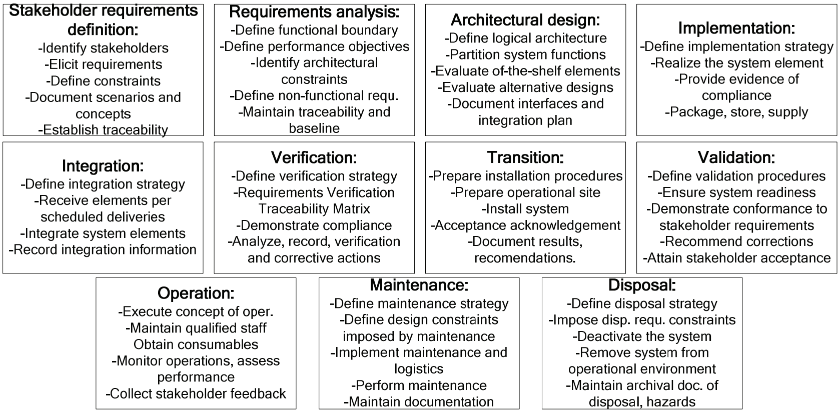

2. relate base inf.
1. Add concerns and related functions
Aspects
Functions
3. Identify design processes
Design tasks
4. Link to domain models / model parts
Domain entities
5. Add parameters
Parameters
6. Add generic model information
Entities
Requirements
7. relate generic and domain models

Figure 24. Model objects (italicized) and proposed modeling steps (iterations, not shown) to model an aspect in the AM

Yet another modeling process flow can be taken when there is a particular concern related to the system and an existing external model supports evaluations related to the concern or synthesis based on the available information. From the perspective of certain stakeholders, a model of the whole system might not be of interest. Modeling the aspects is a way of contributing to the whole model of the architecture while at the same time maintaining focus over a specific concern and its related models. This approach to model is depicted in Figure 24, and can start by adding an aspect hierarchy that roughly describes the main parts of the concern. On the one hand the aspects must be associated to function hierarchy to enrich the description. This functional hierarchy will in turn be associated with requirements which provide concrete information regarding the concern, and link it eventually to a group of relevant parameters. On the other hand, the aspects will be associated with domain entity and design task nodes which provide a direct link between the AM information and the external models.



Stakeholder requirements definition:
-Identify stakeholders
-Elicit requirements
-Define constraints
-Document scenarios and concepts
-Establish traceability

Requirements analysis:
-Define functional boundary
-Define performance objectives
-Identify architectural constraints
-Define non-functional requ.
-Maintain traceability and baseline

Architectural design:
-Define logical architecture
-Partition system functions
-Evaluate of-the-shelf elements
-Evaluate alternative designs
-Document interfaces and integration plan

Implementation:
-Define implementation strategy
-Realize the system element
-Provide evidence of compliance
-Package, store, supply

Integration:
-Define integration strategy
-Receive elements per scheduled deliveries
-Integrate system elements
-Record integration information

Verification:
-Define verification strategy
-Requirements Verification Traceability Matrix
-Demonstrate compliance
-Analyze, record, verification and corrective actions

Transition:
-Prepare installation procedures
-Prepare operational site
-Install system
-Acceptance acknowledgement
-Document results, recomendations.

Validation:
-Define validation procedures
-Ensure system readiness
-Demonstrate conformance to stakeholder requirements
-Recommend corrections
-Attain stakeholder acceptance

Operation:
-Execute concept of oper.
-Maintain qualified staff
Obtain consumables
-Monitor operations, assess performance
-Collect stakeholder feedback

Maintenance:
-Define maintenance strategy
-Define design constraints imposed by maintenance
-Implement maintenance and logistics
-Perform maintenance
-Maintain documentation

Disposal:
-Define disposal strategy
-Impose disp. requ. constraints
-Deactivate the system
-Remove system from operational environment
-Maintain archival doc. of disposal, hazards

Figure 25. Technical processes in product life cycle, and main process tasks/activities (as specified by INCOSE)

Examples of such concerns which can be identified in the model using aspects can be found in prescribed development processes, such as the technical processes presented in the INCOSE systems engineering handbook [90], which are used to establish system requirements as a basis for development. As shown in Figure 25, eleven technical processes are considered in this approach.

3.3.5.2 Additional considerations

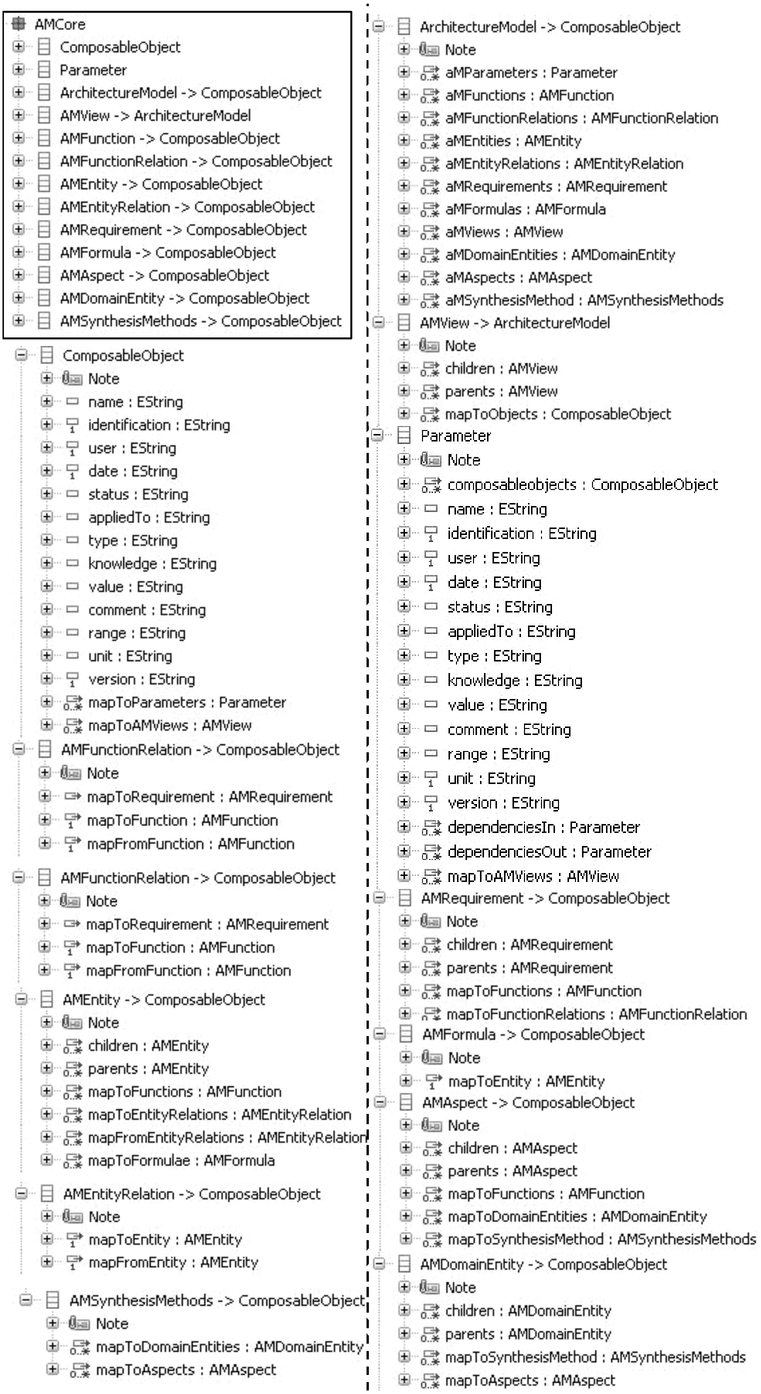
Maintaining the uniqueness of elements is necessary to correctly capture the relations between elements in different diagrams. Keeping track of the existing nodes for a big model proves to be difficult and increases the chance of creating redundant objects. There are different alternatives to aid keeping track of the existing nodes in a model. These alternatives include among others, queries, auto-completion and the use of specialized views. Queries work by allowing the user to search for a node using different methods. Auto-completion can work, for instance, by proposing a set of matching candidate objects upon object creation. Specialized views allow constructing (automatically or manually) diagrams that show existing nodes that fulfill specific requisites. Considering development simplicity, this last alternative is the main one for the present tool implementation, which quickly shows shortcuts of elements adjacent to a selection. The specialized views include a diagram listing all the behavioral layer (parameters, formulae and related entities), a hierarchical diagram showing all entities, a hierarchical diagram that shows all functions, and a diagram that shows all requirements and related parameters.

Until now, the authors have explained how to expand the AM by directly instantiating objects from the basic classes. The model has been designed with an additional instantiation mechanism in mind to facilitate reuse of (multidisciplinary) design knowledge. The main idea is to use functions as "wrappers" for information independently from their domain of origin, as domain distinctions become irrelevant at the functional level [3]. Functional information can then be used to index (with the help of function types) whole pieces of information in the AM, named 'building blocks', which later can be placed into libraries for reuse. More details and additional ideas about this can be found in a previous work of the authors [195]. Reuse saves time when building the AM and can be used to reduce modeling errors by promoting the use of automated modeling and transformations. A clear example of a building block can be found for elements that work as (in many cases ideal) connections between components, transporting data, information, or mater. The related models require the designer to focus on the parameters transported through the connector and in some other key parameters. Instead, in many cases the designer focuses on other aspects, like routing, deviating his attention from the main function the component should fulfill, and increasing the chance of creating errors. A connector building block could be configured using the key parameters above, and potentially, used together with a proper linking to external models to synthesize other routinely details.

A final point worth highlighting is that when it is necessary to create a product architecture usually the system engineer(s) or architect(s) develops a model and attempts to communicate it to other stakeholders. Most literature related to product architecture and system engineering tries to define models that the architect can use for this purpose, showing a tendency to directly address only the architect as a creator and other stakeholders are just shown the model or interviewed to gather information. Other stakeholders are not banned from contributing directly to the AM, but it becomes implicitly understood that "it is not their job". On the other hand this thesis proposes a model that can (and is meant to) be used and built through the contributions of all stakeholders, including the architect. This is consistent with the concept of the knowledge intensive design environment introduced by Tomiyama *et al* [168], however, here it is argued that the presence of a guiding architecture model is crucial to attain such a degree of cooperation.

3.3.6 Implementation

The authors developed a tool that facilitates creating correct models and enables using data transfer mechanisms. The implementation has been made over the eclipse framework [163] using the Graphical Modeling Framework (GMF). This platform provided good mechanisms for class definition and a transformation to implementations [78] that allow creating correct models and a clean, standard eXtensible Markup Language (XML) serialization.



```
AMCore
  ComposableObject
  Parameter
  ArchitectureModel -> ComposableObject
  AMView -> ArchitectureModel
  AMFunction -> ComposableObject
  AMFunctionRelation -> ComposableObject
  AMEntity -> ComposableObject
  AMEntityRelation -> ComposableObject
  AMRequirement -> ComposableObject
  AMFormula -> ComposableObject
  AMAspect -> ComposableObject
  AMDomainEntity -> ComposableObject
  AMSynthesisMethods -> ComposableObject

ComposableObject
  Note
  name : EString
  identification : EString
  user : EString
  date : EString
  status : EString
  appliedTo : EString
  type : EString
  knowledge : EString
  value : EString
  comment : EString
  range : EString
  unit : EString
  version : EString
  mapToParameters : Parameter
  mapToAMViews : AMView
AMFunctionRelation -> ComposableObject
  Note
  mapToRequirement : AMRequirement
  mapToFunction : AMFunction
  mapFromFunction : AMFunction

AMFunctionRelation -> ComposableObject
  Note
  mapToRequirement : AMRequirement
  mapToFunction : AMFunction
  mapFromFunction : AMFunction
AMEntity -> ComposableObject
  Note
  children : AMEntity
  parents : AMEntity
  mapToFunctions : AMFunction
  mapToEntityRelations : AMEntityRelation
  mapFromEntityRelations : AMEntityRelation
  mapToFormulae : AMFormula

AMEntityRelation -> ComposableObject
  Note
  mapToEntity : AMEntity
  mapFromEntity : AMEntity

AMSynthesisMethods -> ComposableObject
  Note
  mapToDomainEntities : AMDomainEntity
  mapToAspects : AMAspect

ArchitectureModel -> ComposableObject
  Note
  aMParameters : Parameter
  aMFunctions : AMFunction
  aMFunctionRelations : AMFunctionRelation
  aMEntities : AMEntity
  aMEntityRelations : AMEntityRelation
  aMRequirements : AMRequirement
  aMFormulas : AMFormula
  aMViews : AMView
  aMDomainEntities : AMDomainEntity
  aMAspects : AMAspect
  aMSynthesisMethod : AMSynthesisMethods
AMView -> ArchitectureModel
  Note
  children : AMView
  parents : AMView
  mapToObjects : ComposableObject
Parameter
  Note
  composableobjects : ComposableObject
  name : EString
  identification : EString
  user : EString
  date : EString
  status : EString
  appliedTo : EString
  type : EString
  knowledge : EString
  value : EString
  comment : EString
  range : EString
  unit : EString
  version : EString
  dependenciesIn : Parameter
  dependenciesOut : Parameter
  mapToAMViews : AMView
AMRequirement -> ComposableObject
  Note
  children : AMRequirement
  parents : AMRequirement
  mapToFunctions : AMFunction
  mapToFunctionRelations : AMFunctionRelation
AMFormula -> ComposableObject
  Note
  mapToEntity : AMEntity
AMAspect -> ComposableObject
  Note
  children : AMAspect
  parents : AMAspect
  mapToFunctions : AMFunction
  mapToDomainEntities : AMDomainEntity
  mapToSynthesisMethod : AMSynthesisMethods
AMDomainEntity -> ComposableObject
  Note
  children : AMDomainEntity
  parents : AMDomainEntity
  mapToSynthesisMethod : AMSynthesisMethods
  mapToAspects : AMAspect
```

Figure 26. Class overview (top left), and specific class definitions in the architecture model. In the Eclipse / ecore class definition the '▤' icon indicates a class. The tree expanded under it shows its attributes.

The information structure of the AM is a “flat” network of interconnected nodes contained in a main ‘ArchitectureModel’ node. ‘View’ nodes provide a link to diagrams where parts of the modes can be visualized and edited. In principle, views do not provide any hierarchy for the information, but can provide an order for construction as shown in Section 3.3.5. Figure 26 depicts the class definitions for the AM. More recent work of Woestenenk [194] provides improved internal class definitions for the AM from the programming viewpoint. It can be seen that almost all classes extend the abstract ComposableObject class. This class defines a series of common attributes such as name, user, and version, and also defines the possibility of mapping the instances to Parameter and AMView instances. These attributes result useful when analyzing model information and to “label” views interesting for different stakeholders. Many of the attributes inherited from the ComposableObject class still do not have a role for each specific class. This has been done on purpose to provide some flexibility and future extensibility to the AM. The parameter class has a similar set of attributes, and its dependency attributes can be used to represent relations between parameters such as causal dependence.

The other class definitions are portrayed in Figure 26. The role of the non-inherited attributes in the classes is clear from their names: they take care of defining specific referencing (mappings) among the classes so that the AM gets the desired structure defined in sections 3.3.1 to 3.3.4. The entity and function relation classes are not defined as simple attributes because they need attributes of themselves. All modeling elements use characteristic shapes and symbols to help the user identifying the information in the diagrams (see Figure 13).

Elements of the external communications layer contain pointers to external resources (data and models). The information reuse capabilities introduced in Section 3.3.5 depend heavily on the external communications layer and its link to the external models. At the moment, such tools have not yet been fully implemented and form part of the work of Woestenenk *et al* ([194], [195]).

3.3.7 Discussion on the capabilities of the AM

This section contains two discussions regarding contributions (Section 3.3.7.1) and some research directions (Section 3.3.7.2) which correspond particularly to the modeling language, tool, and building method proposed above, as an embodiment of the necessary support for an architecture-centric approach.

3.3.7.1 Language contribution to design information representation

This discussion revolves around the fact that (as discussed in Chapter 2) notwithstanding the existence of many methodologies related to representation of multidisciplinary information, those methodologies and the related modeling languages are not used extensively in industry. Several reasons (some of them mentioned through this thesis) may be the cause of such behavior. However, in this discussion it is pointed out that an important factor contributing to the acceptance and usefulness of “multi-domain” modeling languages lies in the nature of the language itself and, more specifically, to characteristics related to its syntax, semantics, and pragmatics. This section also clarifies related concepts expressed in previous work of the author [7].

Before discussing the particular characteristics of the AM, it is necessary to define what these characteristics describe. Referring to Fishwick [70], for a language the syntax specifies the rules of construction (structure of a model), the semantics hold the meaning of the symbols (meaning of models), and the pragmatics define the contribution of context to meaning (interaction of an interpreter with models). However, the limits between these definitions are not evident and may depend on the point of view, i.e., what is considered as the model or language.

The AM contains simple semantics (few classes of data elements) and a well structured and simple syntax (a few clear ways of linking the data elements) combined with a strong representation power (the data elements allow representing most aspects and details of a system) and the capability to focus on part of the information (choose a view). It is argued that these characteristics help in making unambiguous pragmatics, or interpretation of the models, while retaining the necessary expressive power. Analyzing these characteristics it is possible to see that the object groups defined in Section 3.2.3 roughly define data classes which can be used independently of the domain, which greatly influences the representation power of the proposal and its simplicity, addressing complexity in design. Additionally, some of the basic concepts of the implementation (e.g., unique objects, multiple views) facilitate access to the context described within the model, reducing the chances of duplicating information and of creating multiple context descriptions which disagree with each other. The stated characteristics are advantageous because they promote that the users creating the model place in it all the relevant information while at the same time information contributed by other users can be retrieved to maintain an updated and detailed image of the context (and thus restricting the possible interpretations). On the side of the disadvantages, promoting to model all the relevant information also implies that the modeling effort is greater than the required for other (more ambiguous) modeling languages. However, this is considered by the author as a necessary price to pay for obtaining formal descriptions of an extremely complex and diverse data set such as design knowledge. It can also be added that the current state of the art in digital data management and modeling tooling plays an important role in reducing such a disadvantage to a minimum. Other important characteristic is that the language implementation is intended for both human and artificial interpreters (*resp.* through graphical and XML representations), allowing supporting discussions among human stakeholders and automatic model transformations among design tools. To the best knowledge of the author, these characteristics have not been explicitly used to describe any of the studied implementations, nor have been they pointed out as advantageous for the representation of design information supporting the development processes.

Going back to the example of text-based descriptions (see Section 3.2.2.1), it can be appreciated that natural languages are characterized by very complex semantics (many words) and syntax (many possible structures for sentences, paragraphs, etc.). These characteristics provide an immense representation power to convey ideas, feelings, context and many other types of information. However, mastering the language to unambiguously and precisely represent all this information takes much time and expertise, not only for the party creating the model (for example a poem) but also for the party interpreting the model. Conversely, it is argued that the proposed language (the AM) can be mastered with far less effort than a natural language, and thus the users can convey information far more precisely than with other languages while retaining much expressive power with the ability to use different information types.

3.3.7.2 Unsolved representation limitations

As mentioned though this chapter, the AM allows representing information on different levels of granularity and from different points of view. If the views are chosen properly, a human interpreter may not have any problem in differentiating such levels within the model. However, when dealing with an artificial interpreter (e.g., a running program of some sort) these different levels of information cannot be easily distinguished, interpreted, or exploited.

Therefore, additional considerations to build a model which influence the ease of interpretation have to be mentioned. On the one hand, the proposal remains practically usable because the issues of artificial interpretation are partially addressed by making proper use of the external communications layer. In principle, the objects in this layer can be used to “label” parts of the model providing a single, completely unambiguous, description level which can be properly processed by an artificial interpreter (see Section 3.3.4). On the other hand, a more elegant solution which reduces the need for such labeling would contribute greatly to the applicability of the proposal and increase its simplicity. At this point, such considerations are taken as limitations of the proposed approach, and future work should be carried out to address them.

Improper formula set

Representing a consistent concrete behavior (like the solvable system of differential equations mentioned in Section 3.3.3) requires a complete (solvable) formula set to be mapped to a single object. However, in many cases during design, such a complete set may not be available, and instead an under- or over-constrained set may exist due to lack of information, design freedom, or overly restrictive requirements. For the purpose of this work (specially for Part II), it is considered that every concrete behavior description in the behavior layer contains a complete formula set.

Active structure and functionality

The resulting concrete behavior is affected directly by the choices in the function and structure layer because there is no restriction regarding the formulae and the (relation and decomposition) topology of those layers.

For example, the structure layer may contain not just one, but multiple “static” structural descriptions. This is possible because the structure layer does not consider information about time or the current state of the system. The ‘active structure’ is introduced to consider dynamic changes in the structure of the system. At a certain level of detail, to perform a function it is required that the system finds itself in a determined state and configuration. Therefore, the part of the structure layer mapped to a specific functionality can be considered as the active structure that implements the function. By extension, when looking at the evolution of the systems through several states one can connect them to the course of time.

This work omits the evolution of the system trough these structurally relevant states, and assume that all the structure in the AM is the active structure at any given point in time.

Varying level of granularity

The AM language can be used to describe a system any level of granularity. A single AM may contain several of such descriptions of a system, covering detailed (high granularity) functionality and structure decomposition at the part or even feature level, and concise (low granularity) descriptions at the machine level that describe the main functionality.

Additionally, the different descriptions may contain substantially different assumptions in the behavioral layer, which can lead to discrepancies among them. As an example, at a low granularity level a car can be represented as a point mass for the purpose of analyzing its motion on the road. The same car can be represented with higher granularity as a rigid body containing several point masses corresponding to the chassis and wheels. Also, the rigid body assumption can be thrown away, and the car can be represented as a group of point masses joined by elastic links, e.g., between the wheels and the chassis. As the reader can reason, the three different descriptions use parameter sets of varying size in the behavioral layer, and moreover, the different relations among the parameters can give birth to different and inconsistent behaviors and values of the parameters; e.g., the elastic assumption can introduce vibrations of the center of mass of the car that may be relevant even at the level of the first description above.

Dealing with this consideration is very important to make the model much more useful in practice because different stakeholders may be interested in modeling at different granularity levels. Several ways of formally modeling more than one granularity level in a single AM are possible. For example, different levels can be modeled in different views, or objects corresponding to different levels may be explicitly labeled. For the sake of simplicity in this work it is assumed a single consistent behavioral description within a level of granularity, and thus no particular way of dealing with multiple granularity level models is indicated.

3.4 Conclusions

The concept of an architecture-centric design approach is presented (see Section 3.2) to address the challenges identified in the previous chapter by proposing a support framework for cooperative multidisciplinary product development processes. The framework includes a tool developed during this work (the AM tool) as well as existing (domain-specific) tools currently used by development stakeholders.

The Architecture Model (AM) is a representation and tool proposal to capture design information at the architecture level (*cf.* Section 3.3), intended to support an architecture-centric design approach. Integration is supported by the model, allowing to build simulation and verification models using the data in the AM and the existing domain-specific design and analysis tools. The application of high-level function models is of paramount importance to support communication and integration. Thus, the main contributions of the AM implementation are:

- Enables building an expressive and lean representation of the product.
- A method to create and use such representation in the product development process (e.g., to support communication).
- Association of the information that is shared among the designers during the development process.
- A language and implementation which address two issues which may explain why existing languages to support modeling and using product architectures have not found a permanent place in industry. These issues are the lack of practical use of models of the architecture, and existing language characteristics which prevent the user from clearly documenting design information.
- A 'view' mechanism to model, filter, and retrieve the context of the information in the model. However, maintaining the views as the model grows presents a practical implementation issue. Future improvements for automated updating (e.g., through view subscription) and arranging of nodes in views will contribute to a tool more apt for industrial implementation.
- A way of increasing the participation of stakeholders who, normally, marginally take part in the construction of product architecture models, increasing the fidelity and usability of the documented design information.
- A model which separates (in different layers and objects) time-related behavior descriptions and exchange-related descriptions of processes (*resp.* functional and structural layers). This is crucial to obtain readable and coherent models. At first sight, modeling these aspects separately may seem obvious, but in practice requires some experience from part of the modeler (see Section 4.2).

4 Case Studies

The previous chapter described the central proposals of this thesis. This chapter documents multiple case studies which support those proposals from the side of development of the core ideas as well as from the side of testing those ideas in several scenarios. Such case studies have been respectively organized in Section 4.1 and Section 4.2. The chapter ends with a conclusions section.

4.1 Development case studies

The architecture-centric approach to design and the AM proposals have been refined from the lesions learned in several attempts to support different design processes. This section presents such attempts in two main parts. Section 4.1.1 presents the result of an industrial case study carried out to increase the understanding of product development by studying a real product, additionally finding out and verifying some of the challenges stated in the Chapter 2. Then, the germinal ideas of the proposed approach to create high level models to support design integration and the first attempts of implementation are presented and discussed in Section 4.1.2.

4.1.1 A practical study on design processes

The initial insights on the problems related to complex product design (see Chapter 2) motivated a first attempt to understand the design process and develop support tooling for a case study ([3], [5]). The prevailing idea was to improve communication among stakeholders by providing automated transfer of design data used in different design processes, and clear traceability to design specifications. Thus, the expected results were a description of the flow of data in a specific design process (i.e., the workflow), and some custom-made tools which could automate data transfer among stakeholders involved in such process in order to demonstrate the correctness of the obtained workflows.

The study involved reading project documentation, interviewing key designers and system architects, and studying the models used in the design process to follow how the values of two parameters present in the final design were obtained from design specifications and other sources of information. The practical results included two simple tools which ensured consistency of some parameter values used in a simulation model with relation to mechanical specifications, and a flowchart with an overview of the specific design process. Analysis of the interview data and the resulting flowchart lead to several conclusions which are resumed next (*cf.* Figure 27):

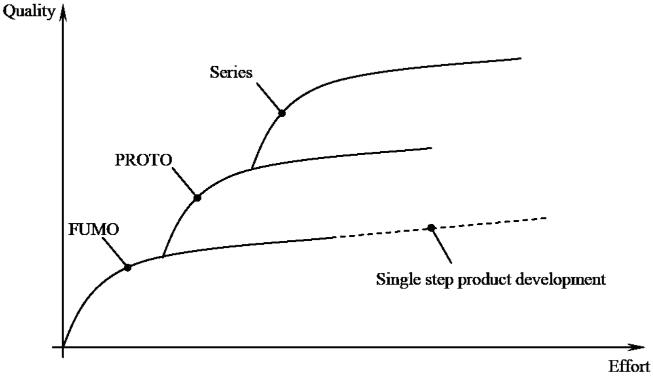


Figure 27. Evolution of quality at different stages of product development. The curves labeled 'FUMO', 'PROTO', and 'Series' correspond to prototypes present at different design stages.

- The studied process relied heavily on physical prototypes for testing modules responsible for single functions (FUMO), integrated modules (PROTO), and complete production concepts (Series).
- As shown in the figure, the incremental approach to testing reduces the required afford to obtain products with greater quality.
- More optimal solutions to design problems may be found by considering domains different to the one in which the problem was detected (e.g., increased predictability trough control rather than tight manufacturing tolerances)
- Efficiently sharing design information directly among stakeholder within a development processes for complex systems becomes unmanageable after the initial stages of design, when the design team can no longer physically work side by side. Afterwards, the available models and documentation do not efficiently support sharing or transferring information.
- Parameters and their updated values are a meaningful and abundant part of design information which can be automatically and efficiently transferred and transformed among stakeholders once a clear design process has been identified.
- Tracing parameter values up to the design requirements and specifications is very difficult if this correspondence is not documented explicitly, and such documentation is rather scarce in practice.

4.1.2 Using high-level models for integration

This section contains the main aspects, results, and learned lesions from the first implementations of high-level models for design support, resumed in sections 4.1.2.1 [4] and 4.1.2.2 ([193], [195]). The initial trial for support tooling (*cf.* Section 4.1.1) confirmed the feasibility of automatically exchanging information at the level of parameters and values. At the same time, it allowed verifying that determining how these values relate to design requirements and to design decisions is a painstaking task, especially because such information is rarely documented in a format which facilitates its (re)use. Additionally, the review of product development challenges indicated that the usage of high-level information such as functions has potential to aid in the integration tasks, but that existing implementations still face some implementation challenges. Also, these insights justify the need to implement integrated (i.e., mechatronic) design approaches, as voiced by many authors ([28], [55], [147], [172], [191]).

Based on the previous ideas, it was decided to attempt a modified implementation [4] of one of the existing techniques to represent the high-level information. The chosen method was Function-Behavior-State (FBS) ([170], [178]), as it seemed to have both high-level and detailed information together in a single model (but using different objects) while providing a link to external models (as in KIEF [199]).

4.1.2.1 An implementation of FBS using new representations

In this section the FBS model [170] is considered as a base for the function model description in the proposed framework. The FBS model was designed to be part of an integrated framework but it was not intended to be the backbone for the integration activity, and thus, some adaptation was considered necessary. This academic case-study was developed to try adapting the FBS representation to recent languages and data formats aligned with graphical representations and standardized exchange of information. Some advantages that lead to the choice of FBS are that it:

- Clearly separates design intention and objective relations between components.
- Is built to support external processing of information with other tools (e.g., qualitative reasoning [18]).
- Has been already implemented in a software tool and tested to some extent (*cf.* FBS modeler in [171]).

Another important reason to support the choice of FBS is that it differs from most system models developed at an early stage of design which are not aimed to prescribe how the systems actually behave [60]. Instead, FBS also contains information which can be used to simulate the behavior of the system from an objective point of view.

The next section provides a description of the FBS model, followed by a section explaining how the FBS primitives were represented using a recent modeling language.

The FBS model

Function-Behavior-State is a function modeling scheme created to support conceptual design in computer aided design (CAD) systems [170]. FBS aims to build a functional concept ontology [65], that is, to allow its users in creating descriptions of functions which can be reused in design. Most components of the FBS model are based on a process ontology known as Qualitative Process Theory (QPT) [71]. As specified in [65], process ontologies focus on the effects of processes over the attributes of entities, and functional concept ontologies look to develop models of devices from the subjective perspective of humans.

An FBS model (see Figure 28, left) can be divided in three parts: (1) the functions layer, (2) the behaviors layer, and (3) the states layer. Each layer is connected to the next one to form a framework that describes the functionality of a system and how to attain such functionality. Behavior and state representations are based on QPT. All the objects are stored in a knowledge base structured using the physical concept ontology [200] implemented in the Knowledge Intensive Engineering Framework (KIEF) [171], consisting of the following concepts which, with the exception of physical laws, support inheritance, i.e., they belong to a class-structure in the knowledge base.

- Entity: Represents an atomic physical object.
- Relation: Represents a relationship among entities to denote static structure.
- Attribute: It is a concept attached to an entity. It takes a value to indicate the state of the entity.
- Physical phenomenon: Designates physical laws or rules that govern behaviors.
- Physical law: Represents a simple relationship between attributes.

The next part of this section contains a brief description of the concepts and main ideas behind the three layers in FBS ([91], [170], [200]).

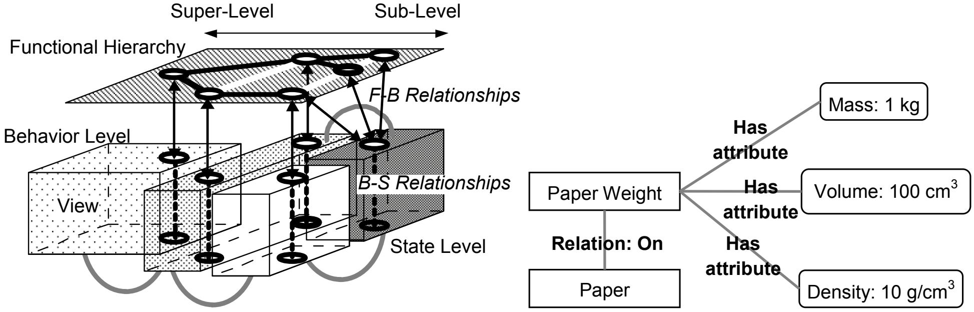


Figure 28. Scheme of FBS model (left) and state of paper weight (right). Adapted from [176]

State

To define state, first the concept of entity must be introduced. An entity corresponds to an object like a solid, a gear, or a single tooth of a gear. The choice for an entity depends on the level of detail being modeled. Entities possess attributes that describe them. Lastly, entities are connected to other entities by relations.

For modeling proposes, in FBS states and entities are treated simultaneously. A state is defined as "*a set of attributes and relations between entities*", and thus a state cannot be described without the use of entities. Figure 28 (right) depicts a state, showing several attributes of the entity "Paper Weight" and how it relates to the entity "Paper".

Behavior

First it is necessary to define physical phenomena in order to ease the explanation of behavior in FBS. Physical phenomena link a group of entities and their relations to physical laws (e.g., first law of Newton) that regulate the changes of attributes and states. These changes are called state transitions. An example of a physical phenomenon is "linear motion", which connects an entity (e.g., a solid body) and its attributes to a law (e.g., $F = m\ a$). Physical phenomena are knowledge elements that contain the Behavior-State (B-S) connections among the classes of the objects. Physical phenomena become active or inactive according to a set of enabling conditions specified by the presence of a set of entities, attributes, and relations. Behaviors constitute objective representations of what a system does. A behavior is defined in FBS as "*a sequence of state transitions over time*".

To model behavior it is possible to directly instantiate physical phenomena or groups of them. These instantiations are called physical features. Causality between involved physical phenomena can also be specified inside a physical feature. Another modeling option is to specify a behavior as a state transition table. Then other tools (the qualitative process abduction system [91] and qualitative process reasoner [176]) searches and propose candidate physical features that are able to obtain such state transitions and provide qualitative simulations.

Function

The definition of function tends to vary in the field of functional modeling, but many authors agree that the function is subjective in nature and carries the intention of design or use ([39], [65]). In FBS, function is defined as "*a description of behavior abstracted by human through recognition of the behavior in order to utilize the behavior.*" Since the function is abstracted from the behavior, the function alone is not meaningful for representing the system. Therefore, in FBS a function is represented by a tuple of function symbol and behavior that can realize the function. Function-Behavior (F-B) relations are established when a function is connected to a physical feature.

The function symbol is a text that describes the function in the form of "to do something." No further restrictions or guidelines are necessary to describe the function at this level because the function symbol itself is just intended for human recognition. Functions form a hierarchical structure that results from the decomposition of general functions into more specific subfunctions, forming a function tree [67]. Decomposition of functions is classified as either causal decomposition (i.e., into subfunctions whose execution is causally related) or task decomposition (i.e., the subfunctions can be executed independently from each other). When several functions and F-B relations have been placed in the model, the designer can proceed to connect the entities of different physical features that represent the same object. This is referred as unification of entities.

FBS in SysML

For this implementation test, the proposal mainly addresses definitional integration (see Section 3.1.1). Two different approaches were followed to "update" FBS through a re-implementation. This was possible because although FBS defines a semantic structure for the knowledge base, it does not define any data structure for it and it is not restrictive in that sense The first attempt was a very straightforward mapping of FBS concepts to graphical objects modeled in Microsoft Visio, as the use of such tools is really common in industrial practice. This allowed creating small models (with around fifty concepts or less) with relative ease, even using different files or sheets by associating the objects in different places using a unique naming scheme. However, it soon became evident that building and managing larger models became almost impossible without proper support of specialized tooling implemented directly in the modeling software (i.e., Visio and its programming interface). A less evident shortcoming was the lack of a precise definition of modeling concepts and objects for the user that is not familiar with the physical concept ontology. In other words, if the user does not have good knowledge about the definition and use of the FBS language, the resulting model will be very difficult to understand (almost meaningless) to a different user. This last point indicated the need for precise explanations of the modeling concepts and additional tool support which enforces basic syntax in the model.

With the experience of the attempt described above, the second approach started with searching recent developments of formal data and model representations. After studying approaches like the UML [124]. the related SysML [123] and STEP (ISO 10303), SysML was chosen as the base for representation. The main reason for this choice is that SysML seemed suitable to represent most of the information used in systems' design. Additionally, SysML has been successfully applied as part of an integrated design platform in works like [130] and [159]. It is also worth mentioning that part of the developing group of SysML also belongs to the group that develops STEP. Another technology related with all these representations in the eXtensible Markup Language (XML): an extensible data serialization format which used nowadays as a standard in many applications, which also promotes human readability in exchange for some terseness. At this point it was concluded that implementing FBS in SysML could get FBS in the path of standardization for both, data representation and modeling language.

The implementation principally consisted on obtaining a formal description of how FBS model concepts could be represented in SysML. Figure 29 depicts how an example of the FBS concept of physical phenomenon corresponding to rotation with one degree of freedom is represented using SysML. A discussion of the results of this second implementation approach follows. Details of the proposed mappings and other examples can be found reference [4].

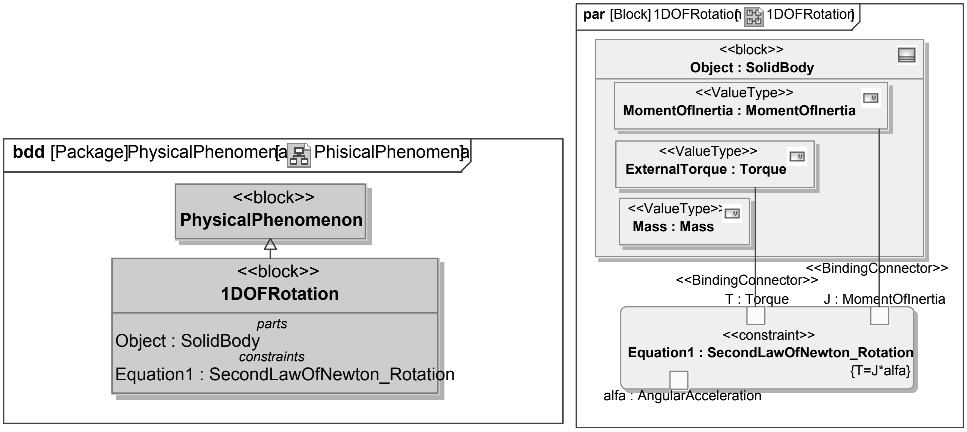


Figure 29. Physical phenomenon representation. Block representation (left) and statements definition (right)

This experience demonstrated that indeed SysML has great expressive power and that it allows formally representing the FBS syntax as a metamodel. It also reassured that FBS allows modeling the functionality, structure, and state-based representation of behavior, while providing a link among the information represented through such concepts. However, the results (i.e., resulting mappings and usage tests) also showed characteristics of the approach which do not align well with the key aspects presented earlier.

On the one hand, the FBS concepts did not contemplate explicitly and directly capturing information related to other external models/modelers (this is done through KIEF) or to requirements. Also, in FBS concept of class definition and inheritance in the knowledge base becomes intrinsically related to the definition of new knowledge and models. Though this last characteristic is not disadvantageous per se, it implies that the effort of developing a new model for a system must be coordinated with an effort to develop or maintain the class-structures in the knowledge base. Coupling these development efforts can be counterproductive for the design and modeling activities.

On the other hand, SysML also promotes the use of class structures and inheritance, coupling directly with such characteristics of FBS. It is also necessary to mention that the same variety of diagrams and objects which give flexibility and richness to SysML, in spite of clear syntax descriptions for most cases, also contribute to increase the required modeling effort by leaving to the user many choices with respect to which and how SysML objects shall be used represent some particular aspect of the system. This is also seen to a lesser extent with languages such as UML, but in that particular case, most of the terminology corresponds with well defined concepts from software engineering development. This characteristic can also be evidenced by the discussions of many SysML users (e.g., http://groups.google.com/group/sysmlforum/) in which questions like “how do I model X” or “can I represent X using Y” prevail. Besides this, like in the Visio implementation, defining the allowed FBS syntax through the extension/constraint mechanisms in SysML also requires a significant afford. A last point to discuss here is that, to this date, the modeling tools implementing SysML focus on the meta-model aspects (definition of class descriptions), while instantiation mechanisms which allow representing the systems do not seem to be widely used or well defined.

4.1.2.2 Using functions to restructure design information

This section discusses another case-study of design support, this time in an industrial context for information management ([193], [195]). The cooperating company uses an automated system to support design, which provides detailed implementation information through a set of libraries. This information was organized according to the different domains (e.g., control, electronics, and mechanics) and could be used by the designers to quickly build detailed models of systems with many components. However, the implementation still presented some issues regarding the maintenance of libraries, such as consistency verification (i.e., checking that a subsystem is not only partially described and has all the necessary representations in the libraries) and change propagation (that is, finding all the affected information when some part of the libraries changes). The goal was to analyze a sample of the libraries and propose improvements. After analyzing the data, it was found that all the libraries contained a piece of coded information described by the company as a function number. Further analysis showed that indeed these function codes related closely to the functionality of parts and subsystems (i.e., what the things are intended do).

The researchers organized the library information according to the functional information, resulting in an increased usability of the model. This was intuitively demonstrated on the one hand because the instantiation and verification of the models corresponded to the functions performed by the related components, and computationally demonstrated on the other hand because the available unfeasible choices present in the original library could be efficiently discarded, reducing the computational effort needed to verify consistency of the resulting models and providing usable choices to the designers.

This experience demonstrated the great practical value of functional information, more specifically in cases (like the one under study) of evident system modularity, that is, when functional information has almost a one-to-one correspondence with the subsystems and parts of the system. Additionally, as stated by Suh [155], the relation between functional information and the objects in the physical decomposition of the system can also be used as a measure of modularity. Nonetheless, as stated in Chapter 2, it must be highlighted that modularity is not achievable or desirable in every design situation.

4.2 Demonstration scenarios on architecture-centric design

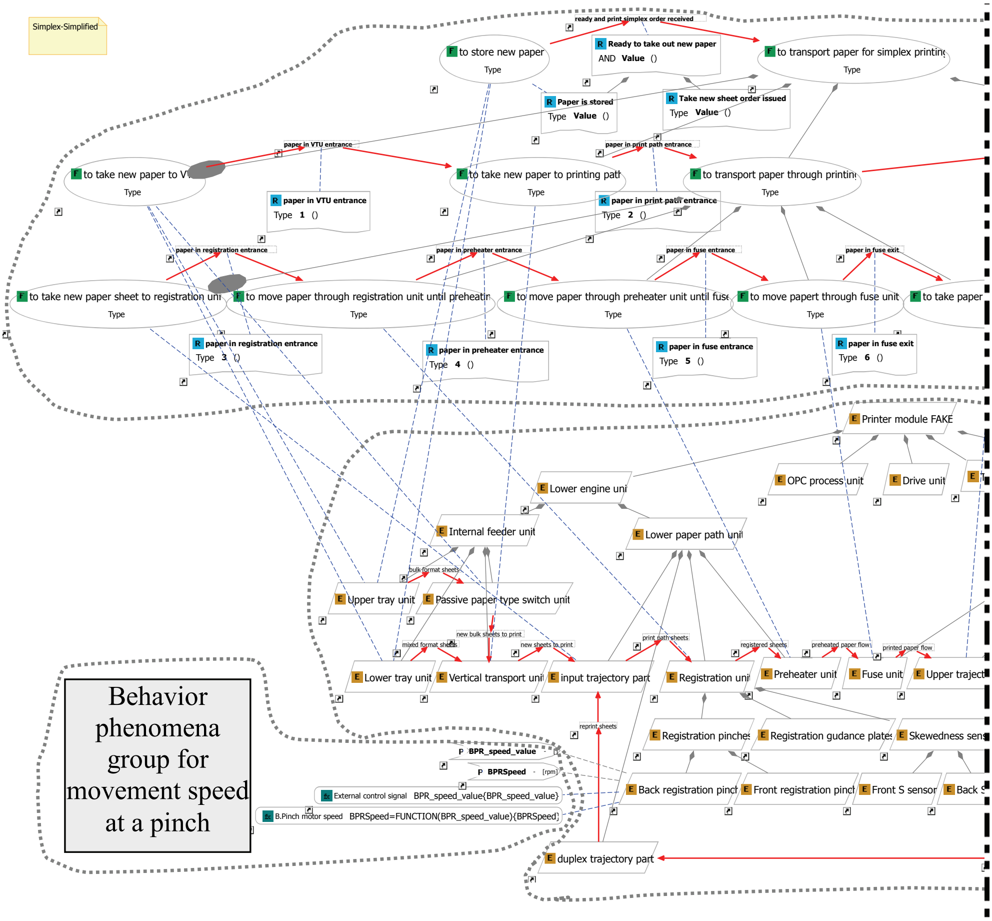
This section addresses five generic product development usage scenarios concerning the architecture-centric approach, and describes an example implementation which covers three of these scenarios. The first two scenarios illustrate the explicit use of the first three object groups (see 3.2.3). These two scenarios correspond to real industrial implementations where text-based documentation prevails and it is not possible to suddenly use model-based product development. Thus, these two scenarios correspond to a necessary transition stage towards model-based product development. The last three scenarios mainly demonstrate the practical use of the fourth group of objects (*cf.* 3.2.3) after the design knowledge has been clearly structured. These three scenarios were implemented using company data within an academic context, supporting a model-based development process. The situations supported by the tested implementation are compared to those where a "traditional" development process is used.

- In the first scenario, an architecture modeling language is used to represent and clarify information from text documents. This is useful to increase awareness about the existence of the four object groups introduced above.
- In the second usage scenario, part of the AM is used as a backbone for text-based design documentation. In this way, the structured model from the first scenario can integrate documentation related to domain-specific design knowledge.
- The third scenario consists of representing information consistently. This case involves modeling, referencing, querying, and updating unique information in a model.
- The fourth usage scenario addresses the issue of creating different views on the same product, modeling the concerns and rationale of several stakeholders. The 'views' mechanism is useful to manage complexity and support communication.
- The fifth and final scenario covers how to use a model of the architecture to support the product development chain of domain-specific tools, and how the model is expanded.

The scenarios have been ordered according to the required implementation effort/-complexity, strengthening the point that arriving to model-based process development in an industrial environment is envisioned as a gradual process. A related video can be found in this link: http://youtu.be/kib4mMzzAxE

4.2.1 First scenario: Representing clearly design descriptions

Currently, textual descriptions like the one presented in the Section 3.2.2.1 provide the base for design descriptions, sometimes, accompanying (excerpts of) domain-specific models. This entails ambiguous descriptions because of the shortcomings of textual information [179], and increases the required effort to extract shared data, because the reader accessing the information in domain-specific models must first learn the particularities of such models and domains.



Simplex-Simplified
to store new paper
ready and print simplex order received
Ready to take out new paper
to transport paper for simplex printing
Paper is stored
Take new sheet order issued
paper in VTU entrance
paper in print path entrance
to take new paper to VTU
to take new paper to printing path
to transport paper through printing
paper in VTU entrance
paper in print path entrance
paper in registration entrance
paper in preheater entrance
paper in fuse entrance
paper in fuse exit
to take new paper sheet to registration unit
to move paper through registration unit until preheater
to move paper through preheater unit until fuse
to move papert through fuse unit
to take paper
paper in registration entrance
paper in preheater entrance
paper in fuse entrance
paper in fuse exit
Printer module FAKE
Lower engine unit
OPC process unit
Drive unit
Internal feeder unit
Lower paper path unit
Upper tray unit
Passive paper type switch unit
Lower tray unit
Vertical transport unit
input trajectory part
Registration unit
Preheater unit
Fuse unit
Upper trajectory
Behavior phenomena group for movement speed at a pinch
Registration pinches
Registration gudance plates
Skewedness sensor
BPR_speed_value
BPRSpeed [rpm]
External control signal BPR_speed_value(BPR_speed_value)
B.Pinch motor speed BPRSpeed=FUNCTION(BPR_speed_value){BPRSpeed}
Back registration pinch
Front registration pinch
Front S sensor
Back S
duplex trajectory part
reprint sheets

Figure 30. Architecture-level representation of text information from the “fake” engine, using the AM (most parameters are not shown).

A structured representation of the text description from the Section 3.2.2.1 can be found in Figure 30. The object groups introduced in the Section 3.2.3, i.e., specifications, system components, and behavior phenomena, can be distinguished clearly using the different graphic representations. In the AM, additional objects have been differenced within the basic groups. For example, functions and requirements, respectively, contain qualitative and quantitative data. Notice how in Figure 30 the specifications group shows the required functionality for the paper path during a single side printing operation, and the mapping identifies the relevant components in the structure. Also, the phenomena that corresponds to setting the rotation speed of a pinch is represented by two simple formulae relating a control value and the corresponding rotational speed.

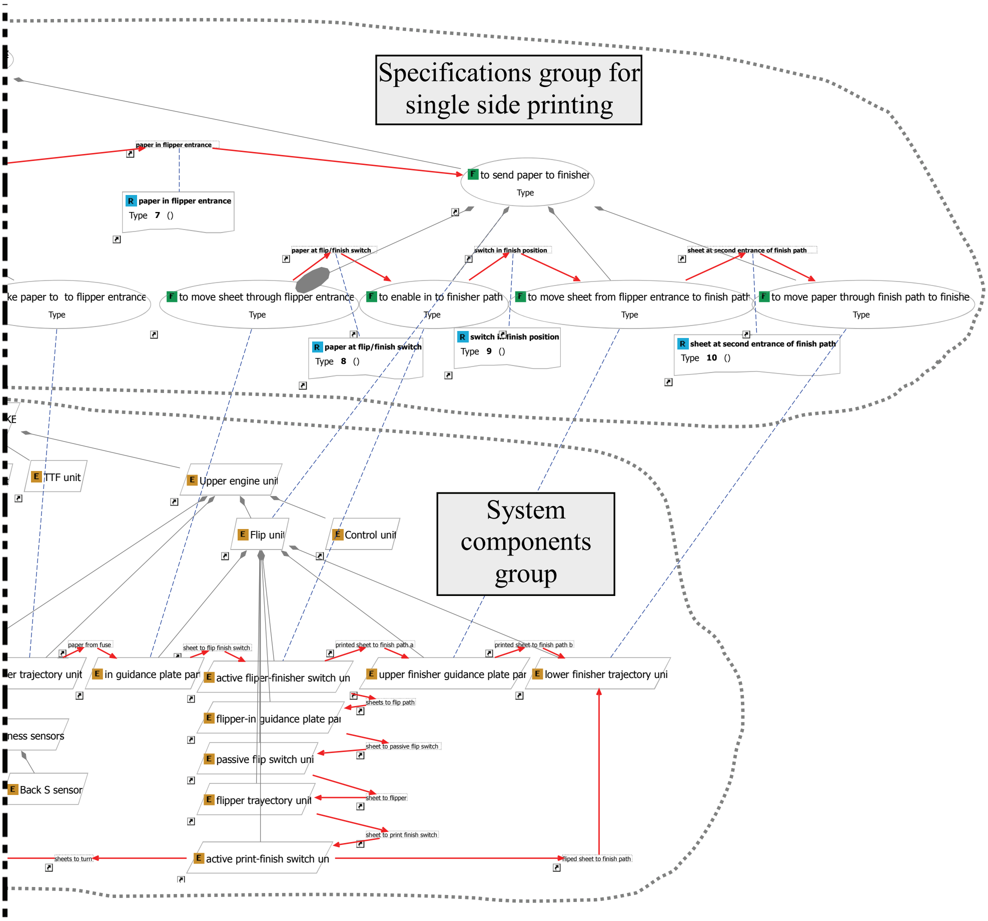

Specifications group for single side printing
paper in flipper entrance
R paper in flipper entrance
Type 7 ()
F to send paper to finisher
Type
paper at flip/finish switch
switch in finish position
sheet at second entrance of finish path
ke paper to to flipper entrance
Type
F to move sheet through flipper entrance
Type
F to enable in to finisher path
Type
F to move sheet from flipper entrance to finish path
Type
F to move paper through finish path to finishe
Type
R paper at flip/finish switch
Type 8 ()
R switch in finish position
Type 9 ()
R sheet at second entrance of finish path
Type 10 ()
KE
E TTF unit
E Upper engine unit
E Flip unit
E Control unit
System components group
paper from fuse
sheet to flip finish switch
printed sheet to finish path a
printed sheet to finish path b
er trajectory unit
E in guidance plate part
E active fliper-finisher switch un
E upper finisher guidance plate par
E lower finisher trajectory uni
sheets to flip path
E flipper-in guidance plate pa
ness sensors
sheet to passive flip switch
E passive flip switch uni
E Back S sensor
sheet to flipper
E flipper trayectory unit
sheet to print finish switch
sheets to turn
E active print-finish switch un
fliped sheet to finish path

Figure 30 (cont.) Architecture-level representation of text information from the "fake" engine, using the AM (most parameters are not shown).

This structured representation improves communication by reducing ambiguity. The tests made the involved developers at the company more aware of the existence of the different object groups (*cf.* Section 3.2.3) and their use. In particular, they could recognize the importance of functions in the specifications group as a means to introduce the purpose of designs, and of the system components group to refer to the product itself. Additionally, addressing other specific groups allows specifying more precisely how certain concerns can be verified. The language characteristics discussed in Section 3.3.7.1 also contribute to the clarity of the models.

Another point regarding the usefulness of the model is that, when creating a design description using the AM, it is easy to distinguish when consensus has not been achieved. Then the involved parties can iterate and refine the model until an agreement is reached. The model facilitates this by promoting explicit representation of knowledge, which entails sound descriptions of the context of the object under discussion. For example, for the back registration pinch in Figure 30, it is possible to request all the incoming entity relations and verify whether the speed command is provided by any relation.

It must be noted that, though the separation of the information makes clearer descriptions, the stakeholders involved in the tests required many attempts to effectively perform such separation, especially between the functionality in the specifications group and objects in the system components group. This situation happens often when the model is used to describe subsystems where functionality and implementation structure are decomposed in a very modular fashion. It was concluded that, though the ability to separate functionality from implementation objects is conceptually simple (i.e., what something should do is not what something is), as our tests evidenced, some practice is needed to acquire it. In this respect, besides practice, providing clear definitions of the meaning of modeling objects (in this case, functions and entities) is a key factor to success.

4.2.2 Second scenario: Organizing design documentation

In the original design process under study, design information often refers to a component "extracted" from a physical decomposition. Such decomposition obeys to the assembly structure of the product, and it is used as a basis in most documents and discussions. This view of the system is then polluted by directly associating to these components properties that just partially relate to them. On the other hand, the other domains have to deal with object definitions that barely correspond to their interests and models. Use of the "views" addresses this situation. An intermediate step has been taken to implement such views by using the object groups to index design documentation in the existing Product Lifecycle Management (PLM) software used at the company.

In the new implemented design process a hierarchical assembly decomposition is still used as a basis in the PLM software. However, additional decompositions are added for the functionality and other aspects (e.g., safety, stability) corresponding to the views of other stakeholders. Objects in these views can be mapped to the objects in the assembly decomposition (see Figure 31). The decompositions are represented in a tree-like format already available in the PLM system (and familiar to the company employees). The files of the geometry model in the assembly decomposition are linked to the assembly representation in the PLM. Elements in the other decompositions can be related to one or more detailing documents. This approach does not imply replacing any of the existing documentation, but provides a means of indexing documentation from different perspectives. Moreover, it provides explicit recognition of the existence of other views on the system and their relation to the components in the implementation.

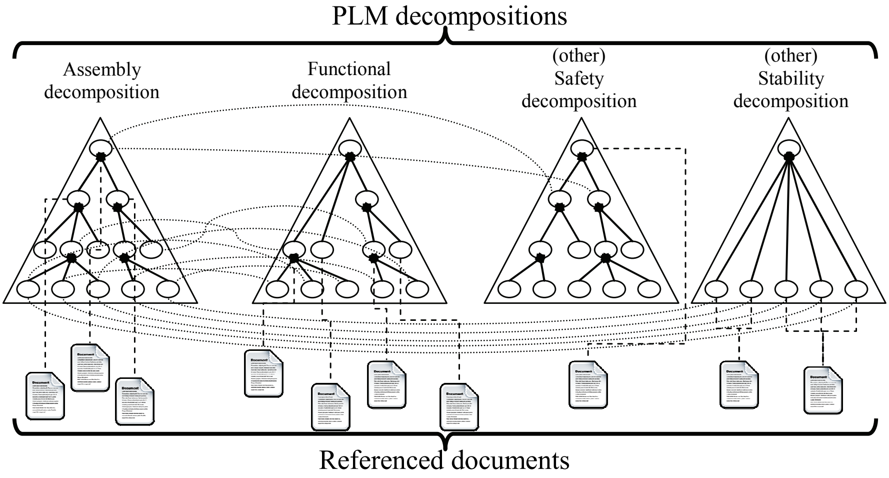


Figure 31. Organization of the multiple views to organize design documents using a PLM system

4.2.3 Third scenario: Documenting design information in a consistent model

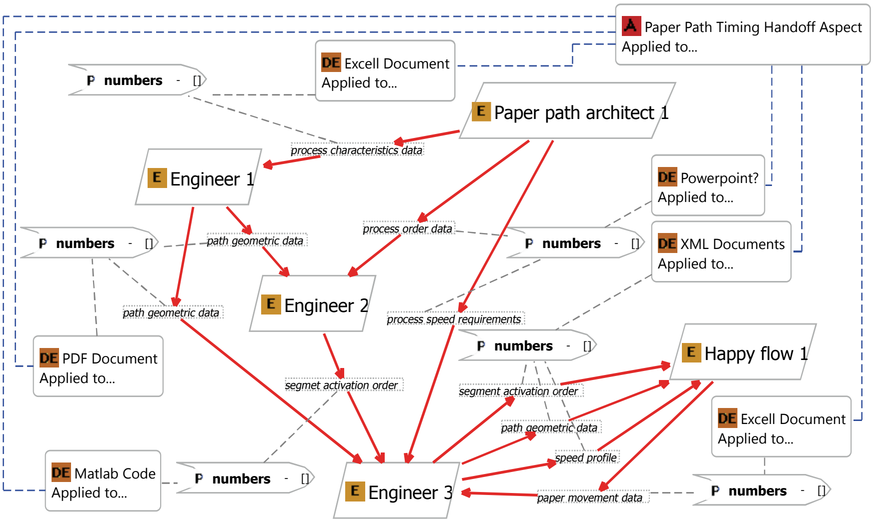


Figure 32. Manual exchange of design information to develop timing information of a print engine

Many aspects of a print engine need the models from engineers working in multiple domains to be developed. An ‘aspect’ makes reference to a common interest of a group of stakeholders, and to the set of models used to describe and verify properties in the context of such interest (see Section 3.3.4). The aspect handled in this scenario is the paper flow through the engine.

The design of the paper path is a multidisciplinary design problem involving mechanical, electronics, control, and thermal design issues. This aspect contains domain-specific information on the customer functions (single/double sided printing), customer requirements (throughput, paper sizes), geometry (component topology), electrical components (motor profiles), simulation (real-time behavior) and software (the timing algorithms and embedded software). The main functionality provided by the paper path is handling the sheets to be printed. Sub-functions include storing, feeding, positioning, heating, and transferring sheets. The input for the timing performance analysis tool, a Matlab application developed in-house named Happy Flow [19], contains data on the geometric shape of the paper path, the type and position of sensors and pinches, and the various operation modes, defined in terms of path segments through which the sheet should be moving. Currently, this data is obtained from various disciplines (*cf.* Figure 32): a dedicated two-dimensional drawing is transformed into an input file by manually indicating which parts of the drawing belong to the paper path, the location of sensors and pinches, and the start and end point of the path segments. Based on the latter information, the operation modes can subsequently be defined. This is a time consuming task that only involves data gathering and deterministic transformations. Furthermore, these steps have to be performed after every noticeable design change.

During design, this domain information is captured in scattered models that follow the sequence described in Figure 32. At each step, the models are made by specialists (using their own languages and tools) based on constraints from a text document (Microsoft Word, Visio or Excel mostly) supplied by the previous step. After models are verified against the input constraints, the information is summarized by hand in a text document and handed over to the next stage.

This approach has a number of disadvantages:

- There is no clear ownership of shared parameters, and therefore different values can exist at a given point in time bringing ambiguity about constraints.
- Manually handing documentation over is prone to consistency errors coming from transcription errors.
- The manual hand-over documentation is not versioned properly; at any stage an engineer could use obsolete values.
- There is no clear (graphical) overview of the design information flow. Figure 32 is a result of interviews during our tests, and it was not present as a mental map among the engineers at that moment. This hinders detecting the other disadvantages.
- There is no common information on the context of the design. Which aspect or functionality of the system relates to the exchanged information?
- Apart from these disadvantages, the step by step generation and interpretation of manual documentation slows down the development process, while not adding any information to the design. This issue will be handled in more detail in the section about the fifth scenario.

The AM has a mechanism that captures an abstract version of the paper path aspect described above. This is done by recognizing that the aspect is an architectural unit, as the described design process is applied for every print engine product multiple times, iterating towards a positive validation of customer requirements in relation to other aspects. The actual system perspective (behavioral and structural information) of this aspect model is handled in more detail in the fifth scenario.

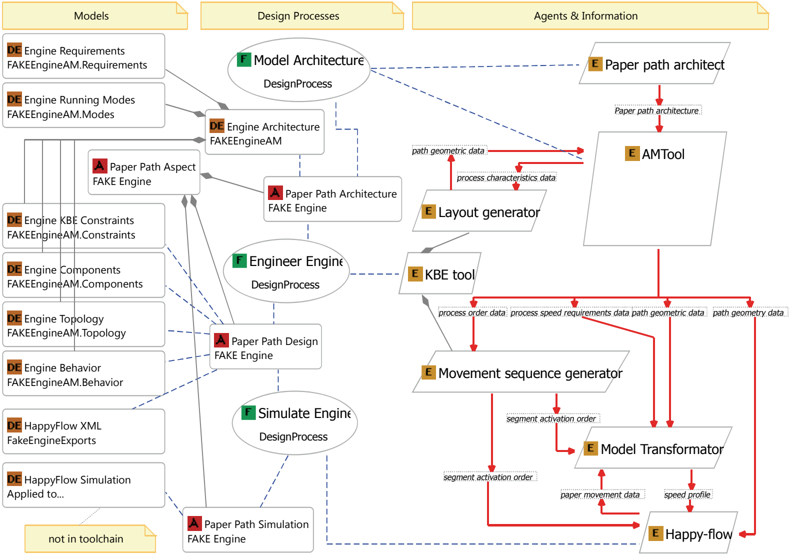

Models
Design Processes
Agents & Information
DE Engine Requirements FAKEEngineAM.Requirements
DE Engine Running Modes FAKEEngineAM.Modes
DE Engine Architecture FAKEEngineAM
A Paper Path Aspect FAKE Engine
A Paper Path Architecture FAKE Engine
DE Engine KBE Constraints FAKEEngineAM.Constraints
DE Engine Components FAKEEngineAM.Components
DE Engine Topology FAKEEngineAM.Topology
DE Engine Behavior FAKEEngineAM.Behavior
DE HappyFlow XML FakeEngineExports
DE HappyFlow Simulation Applied to...
not in toolchain
F Model Architecture DesignProcess
F Engineer Engine DesignProcess
A Paper Path Design FAKE Engine
F Simulate Engine DesignProcess
A Paper Path Simulation FAKE Engine
E Paper path architect
Paper path architecture
E AMTool
path geometric data
process characteristics data
E Layout generator
E KBE tool
process order data
process speed requirements data
path geometric data
path geometry data
E Movement sequence generator
segment activation order
E Model Transformator
segment activation order
paper movement data
speed profile
E Happy-flow

Figure 33. Separation of models, design process and designers (or automatic equivalents).

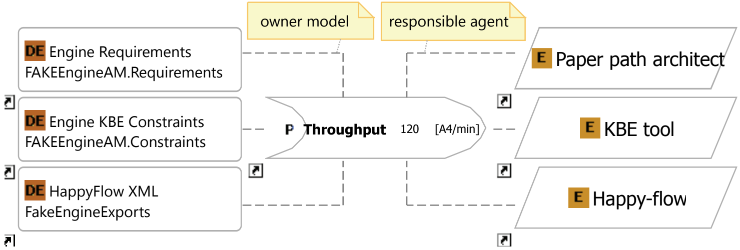

owner model
responsible agent
DE Engine Requirements FAKEEngineAM.Requirements
DE Engine KBE Constraints FAKEEngineAM.Constraints
DE HappyFlow XML FakeEngineExports
P Throughput 120 [A4/min]
E Paper path architect
E KBE tool
E Happy-flow

Figure 34. A parameter (center) shared among agents (right) and used in multiple models (left).

An information flow view of the aspect is shown in Figure 33. By modeling the aspect, it is possible to define the required hand-over between models in a parametric and reusable way. This, in effect, replaces the manual documentation by (a set of) shared models. Each stakeholder provides some part to the aspect model, and the information can be accessed on request without manual transfer. Because the aspect information is kept in a single model, there is no duplicate information, but a single updated version (this can be attained with the support of a versioning system, e.g., SVN [14]). When a shared parameter value is changed, any engineer or tool can update their models to the new value, keeping the whole aspect concurrently verified. An example is given in Figure 34. The owner of the parameter for printer throughput in this case is the engine requirements model. The parameter is stored there uniquely, while the other models reference it by shortcut (this referencing is explained in the next section). The paper path architect has ownership over the parameter value, so the “KBE tool” and the “Happy Flow tool” may not change its value.

While this model of the aspect says nothing about validation of the values or the knowledge inside the domain models, it structures the design process in such a way that (automatic) validation becomes possible (see next subsection).

The amount of information (number of nodes and relations) inside the models can grow very quickly. It is suggested to only model information exchanged between designers and tools. Even then, the amount of information can be too large to be evaluated by a single person. Therefore, the next section introduces a mechanism to limit the information that a certain stakeholder sees based on his interests

4.2.4 Fourth scenario: Addressing stakeholder interests in a shared model

As explained before, the amount of information flowing in a design process can increase very quickly. However, the figures here do not state much of the underlying information exchanged between the models and agents. This is a result from our approach to separate data from its representation. The AM is a ‘flat’ model, i.e., it is a list of elements referencing each other through attributes (see the left of Figure 35). Composition/decomposition attributes can be used to represent hierarchy without changing the flat character of the model. As such it is very easy for a computer to handle. As shown in Figure 35, stakeholders can select and create shortcuts to elements from the AM to construct their own views on the architecture. Manipulating the elements in this view will update the data in the all views based on the same AM. Referencing external architecture models is also possible through the shortcut mechanism. The AM Diagram is a representation of these elements in a human-readable form (graphs as in the right of Figure 35). Figure 32 to Figure 34 show ‘views’ on the underlying AM, with selected elements to make a point clear to the reader.

While the views are another element type of -and thus owned by- the AM, the diagrams of the models can refer to other views using shortcuts. This is another mechanism to keep the AM size manageable and manage complexity.

Because multiple views can reference one element of data, the views can be used for negotiation between stakeholders. In Figure 34 one can see that the throughput parameter is used by various stakeholders. If the Happy Flow simulation stakeholder concludes that the throughput value is unattainable, he or she could find the affected stakeholders through the attributes of the parameter object. He can then start a trade-off meeting with those stakeholders to discuss a proper value. This mechanism can be the starting point for impact analysis of design changes.

For the case study, views were facilitated to the stakeholders, such as the engineers and automated design tools, which are involved in the paper path design process. These views concern mainly behavioral (running modes, system processes) and structural (components, decompositions) views on the system, as this relates to the daily work of the engineers. The views represent an abstraction of the domain-specific models of those stakeholders, to keep a sense of familiarity with the information. An example is the separation of the physical component decomposition (used in production, assembly, and maintenance), and the functional decomposition (used in software development and simulation). These views can be used by automated tools to select a subset of the AM to work on. As such, tools can automate parts of the design process of the paper path as seen in Figure 33.

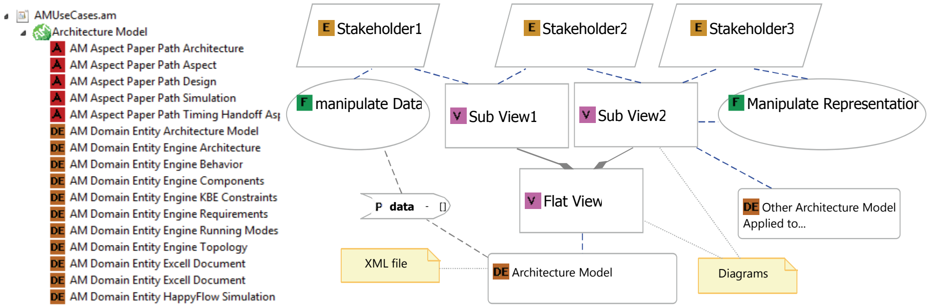

AMUseCases.am
Architecture Model
A AM Aspect Paper Path Architecture
A AM Aspect Paper Path Aspect
A AM Aspect Paper Path Design
A AM Aspect Paper Path Simulation
A AM Aspect Paper Path Timing Handoff As
DE AM Domain Entity Architecture Model
DE AM Domain Entity Engine Architecture
DE AM Domain Entity Engine Behavior
DE AM Domain Entity Engine Components
DE AM Domain Entity Engine KBE Constraints
DE AM Domain Entity Engine Requirements
DE AM Domain Entity Engine Running Modes
DE AM Domain Entity Engine Topology
DE AM Domain Entity Excell Document
DE AM Domain Entity Excell Document
DE AM Domain Entity HappyFlow Simulation
E Stakeholder1
E Stakeholder2
E Stakeholder3
F manipulate Data
V Sub View1
V Sub View2
F Manipulate Representatior
P data - []
V Flat View
DE Other Architecture Model Applied to...
XML file
DE Architecture Model
Diagrams

Figure 35. Separated data and visualization.

4.2.5 Fifth scenario: Supporting automatic information exchange among tools

The computer-readable format of the AM model allows for fast and safe model data transfer, reducing the need to copy information. Furthermore, domain-specific tools could directly use the model, allowing automated model transformations. The next discussion illustrates this through a usage scenario that involves gathering and transforming data from various design disciplines to create a simulation model used to analyze the timing performance of the paper path in a printer (*cf.* third scenario).

Firstly, since the model of the architecture presents the shared data from the various disciplines in a common format, it is much easier to automatically transform the required data into a domain-specific format, such as the one needed for the Happy Flow simulation. Secondly, the definition of the operation modes (e.g., two sided flipped, two sided not flipped) needed to run the simulation can be considered to be part of the specification of the product, and can thus be captured as a function decomposition and ordering. Mappings to components of the system then provide means to link the function ordering to the components the sheet is moving through for a specific mode, assuming this mapping is complete and unambiguous.

Additionally, there is the opportunity to generate new models based on the desired function specification and the actual components of the system. To this end, knowledge-based engineering (KBE) methods [102] were used. Assuming that the design rules of the involved disciplines are known, a software application incorporating these rules can be created, which performs the task of engineering (composing, sizing, positioning) the system defined in the product architecture. As the application would be implemented in a software language, it is possible to not only read and interpret the product architecture, but also:

- Create a possible implementation: Using stored function-component mappings, the function model can be transformed into a possible schematic design solution (see e.g., [35])
- Add new component objects: For example, if the application determines that there are two components of type A and type B next to each other, and a rule describes that there must therefore be an additional component of type C, this latter component and its relations can be automatically added.
- Determine and substitute parametric values: Based on the requirements specified in the product architecture, a possible solution to the design problem can be generated, either deterministically or by using iterative optimization methods.
- Add domain-specific objects: Based on information in the AM, the application can collect data and transform it into a model input. Similarly, the application can add domain-specific objects and parameters of its internal model of the system to the architecture, acting as another domain-specific model.

Depending on the specific problem, a combination of these approaches can be implemented. In order to determine which objects in the AM are of interest to the KBE application, each object has a 'knowledge' attribute, indicating the type of information available in the object. In this way, the KBE application can identify part of the AM as the input for model generation.

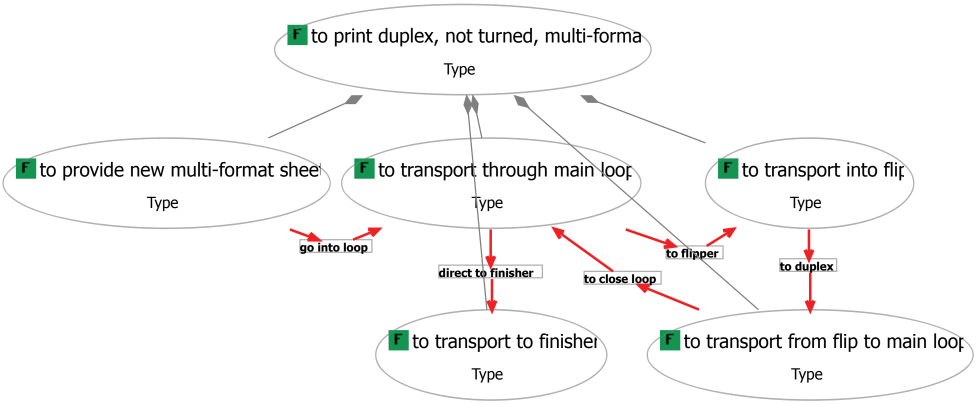

F to print duplex, not turned, multi-forma
Type
F to provide new multi-format shee
Type
F to transport through main loop
Type
F to transport into flip
Type
go into loop
direct to finisher
to close loop
to flipper
to duplex
F to transport to finisher
Type
F to transport from flip to main loop
Type

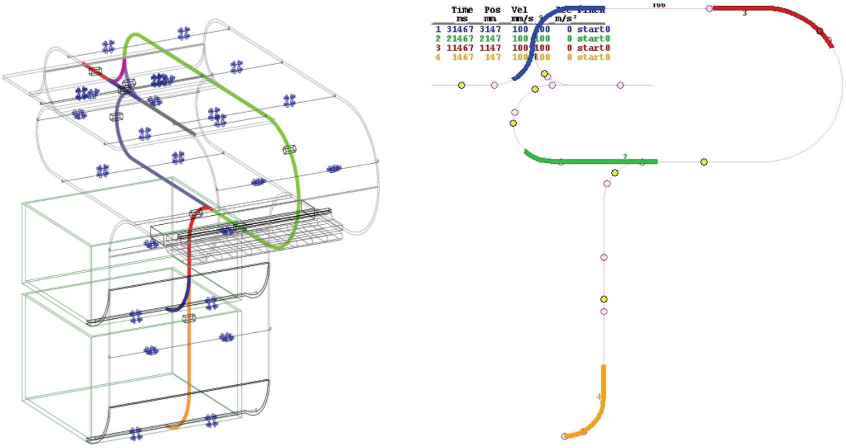
Figure 36. Definition of an operation mode in terms of function objects in the AM.

For the current use-case scenario, a prototype KBE application for the design of the paper path has been developed, which:

- Generates a conceptual geometry of the paper path, based on the component specification in the AM and a restricted set of design and engineering rules and high-level design parameters. This assumes that the system is composed of existing components, and that the set of design rules is complete.

- Determines the number and position of sensors and pinches, depending on the current geometry of the paper path, taking into account design constraints with respect to spacing and placement near junctions.
- Partitions the paper path in segments, following the specific segment definition for the Happy Flow simulation tool.
- Determines segment ordering for various operation modes, which are defined using function sequences and component mappings in the AM, see Figure 36.
- Exports component objects (sensors, pinches), domain-specific objects (segments) and parameters (segment order, shape definition, pinch and sensor positions) to the AM, containing all necessary data to generate the Happy Flow input file.

The rule-based design of the geometry relies on a parametric decomposition of the paper path in pre-defined modules, which is assumed to be common for various printer types. The left-hand side of Figure 37 shows the generated three-dimensional geometry, together with the 2D paper path split up in segments. These discipline-specific segments cross and/or split between modules, in such way that it is not possible to determine a fixed M to N mapping among them.



```
   Time   Pos   Vel         m/s²
    ms    mm    mm/s
1 31467  3147   100  100      0 start0
2 21467  2147   100  100      0 start0
3 11467  1147   100  100      0 start0
4  1467   147   100  100      0 start0
```

Figure 37. Generated geometry with segments (left) and snapshot of resulting animation (right) for the operation mode defined in Figure 36.

The exported AM objects can be transformed into the Happy Flow-specific format from within the AM tool using a custom plug-in which, again, relies on the `knowledge' attribute to collect the necessary data. For the operation mode modeled in Figure 36, a snapshot of the resulting animation is given on the right-hand side of Figure 37, with highlighted lines indicating the position of the sheets within the paper path.

Although this usage scenario relies on a specific commercial KBE modeling platform, the approach shows how a model of the product architecture can not only be used to document and communicate design decisions, but that it can also be directly applied for design-level development and analysis. Any software tool capable of handling XML-data can directly access or modify the data within the model, ensuring consistency and eliminating the need for manual data transformations.

With trends showing an increasing attention to formalized knowledge retention in the form of custom software tools to support or automate engineering tasks, the capability to capture the decisions and results of creative tasks (e.g., developing the product architecture) , in computer-readable models is vital.

4.3 Conclusions

Most of the language in the AM is based on the FBS modeling scheme, modified to better support modeling of behavior and requirements, as well as information exchange and reuse without directly relying on domain-specific classes.

The experiences presented in Section 4.1 made clear to the author that the high-level models expected to support the product development processes had a strong relation to what is commonly known as product or system architecture. This lead to steer the research efforts towards the central proposals presented in Chapter 3 of this thesis.

MBD is being offered nowadays by some commercial tools, however, its full potential is not used because it is strongly related to DSM. It is shown in this chapter that models of the product architecture can provide a backbone to integrate DSMs and fully implement MBD.

It is hard to expect that the companies suddenly adopt radically different development approaches such as migration from a document-based approach to a MBD approach. Small, intermediate steps, like the “mixed” implementations suggested in the first two usage scenarios, lead to steady changes.

PART II. APPLICATIONS TO CONTROL ARCHITECTURE GENERATION

5 Current Control Design Practices and Architecture-Centric Control Design

As stated in the introduction of this book, there in much room for improvement in support for control design practices. Part I of the thesis presents a proposal which has been developed to support design processes, while considering more specifically control design practices. Similar to [150], here the term control systems design/-development is taken in a broad sense, covering the steps from the conception of its architecture to its validation. However, the scenarios under study focus on the first stages of control design, for which it is considered that there is a higher need of supporting tools and methods. Thus, generation of control architecture, configuration, or structure are addressed directly in this part of the book. In this context, generation is understood as the transformation or synthesis of information in the AM into other models used for (control) design. The main usage scenarios supported by the tool are inspired by the area of control systems development. The reason for this is that control systems development inherently profiles as a multidisciplinary endeavor in which the benefits of improved stakeholder communication and understanding can be immediately perceived.

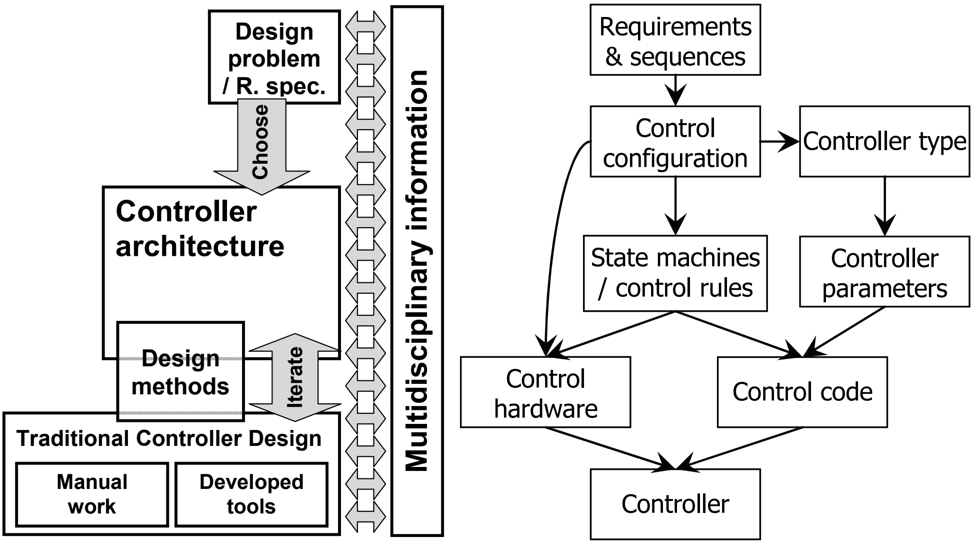


Figure 38. Elements in common control design process (left) and evolution of information (right, iterations not shown)

In a general sense, the common process of design of a controller is depicted in Figure 38 (also see Figure 12, pp. 42). After gathering information of the problem (desired behavior), a control architecture is chosen. The controller architecture contains information (see the right side of the figure) regarding the controller configuration, the type of used controllers and associated control parameters, and the behavior sequences (e.g., represented state machines by control specialists). With that choice, a set of the most used design (i.e., analysis and synthesis) methods and tools related to the specific architecture is used to manually generate a description of the controller in a formal language (automaton, state-chart, etc.). At every step, information from multiple disciplines gets added or updated, requiring to change or adjust the current design. Within such an approach, any changes in the problem specification require to regenerate the formal control description from scratch (demanding an equivalent amount of work each time), or a manual rework of the formal description (an error prone activity which also requires a considerable amount of work).

The goal of this chapter is to present a general description of how "high-level" models common to all the product development stakeholders can be used to automatically provide to the control engineer the necessary input information for the controller design task, while at the same time allows the control engineer to communicate his results and findings to other stakeholders contributing to a concurrent product development scheme. To the readers with a background in control systems design, it must be clarified that Part II addresses many methods which, many could say, do not belong to the common control systems design practice. Therefore, several terms and concepts (e.g., controllability, state space representation) can be used in rather unorthodox ways. Such situations are highlighted as much as possible to avoid confusion by providing our particular definitions in the text, which the author recommends reading carefully.

At this point, two well known control paradigms must be mentioned: centralized and distributed. Distributed control is in general less optimal than centralized control, but it is extensively used in industry because it usually requires less detailed (expensive) plant models and it is easier to tune [150]. A distributed control system can contain many layers with different purposes [145], but a general division distinguishes a regulatory layer and a supervisory layer. The regulatory layer takes care of maintaining a group of output variables under specified values (reference values) with the implementation of a feedback or feedforward loop. In turn, the supervisory layer is commonly implemented as a discrete event, timed, or state based controller that encodes the process rules and activates/deactivates (groups of) subordinate regulator loops. As there is a clear separation in design practices, these parts of control related to the tasks of regulation and supervision are treated respectively in chapters 6 and 7. Currently, formal techniques related to fully centralized control are either more related to regulatory tasks or are not that generally applicable, and therefore not addressed explicitly in this work.

After this introduction, the current practices in control design for mechatronic products and the available support are discussed in Section 5.1, justifying the need for the current proposal. Then, Section 5.2 describes more specifically how the architecture-centric approach can be applied to the generation of control architecture. Conclusions are presented at the end of the chapter.

5.1 Controller software design for mechatronic products

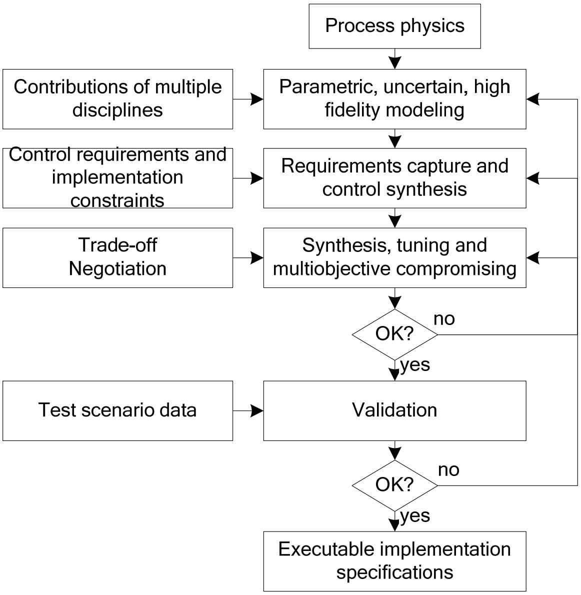

Process physics
Contributions of multiple disciplines
Parametric, uncertain, high fidelity modeling
Control requirements and implementation constraints
Requirements capture and control synthesis
Trade-off Negotiation
Synthesis, tuning and multiobjective compromising
OK?
no
yes
Test scenario data
Validation
OK?
no
yes
Executable implementation specifications

Figure 39.Control law design life-cycle, adapted from [79]

The steps in a life-cycle for control law virtual design [79] are presented in Figure 39. The diagram in the figure basically covers the conceptual design phase up to a point somewhere in the "synthesis" block. From that point on design continues until obtaining "executable implementation specifications" (e.g., of the level of Simulink block diagrams) that can be used for validation (e.g., hardware in the loop testing) and code generation. As can be observed from the figure, control specifications on the block diagram level are obtained somewhere after getting well founded ideas about the workings of the product.

Often in practice the "target" system that should be controlled is already designed or built before controller design begins [16]. Designing a process/product and its controller concurrently as a unit is justified by the fact that the success of the design depends on both of these parts. This point of view is not new to the control design practitioners, as Ziegler and Nichols [201] commented already in 1943: "*In the application of automatic controllers, it is important to realize that controller and process form a unit; credit or discredit for results obtained are attributable to one as much as the other*". However formally practicing concurrent engineering is still a challenge [133], among other reasons because concurrency requires quick and accurate flow of information among the developers. To attain this concurrency, relevant information from other disciplines must be transferred to the control designer, not at the end of the hardware design, but dynamically as design evolves [79]. Benchmark reports carried out on companies in the mechatronic industry specifically conclude that controller design for mechatronic products can greatly benefit from the implementation of an integrated product development process [28] and that challenges in such development process relate strongly to getting engineering disciplines to efficiently work together [92]. Working together implies having some common understanding (consensus), but the abundance and variety of design information hampers integration and thus arriving to a consensus. Additionally, the origins of this information can be hard to trace and (especially at early/conceptual design stages) the correlations can be unclear and fuzzy.

For controller design, the origin of some of these integration problems can partially be traced back to education, which disregards what is called here ‘conceptual controller design’ in favor of what is denoted as the ‘detailed design phase’ (*cf.* Subsection 5.1.2). The outcome of the conceptual controller design is the ‘controller structure’ [150] (i.e., the main components of the controller and their relations) and its purpose. This outcome forms big part of the controller architecture. As stated in Part I, architectural descriptions are a common base over which design information can be structured, but their current practical use is limited.

The next two subsections consider separately the conceptual and detailed design phases roughly follow the steps shown in Figure 39 and elaborating on the information required for the design of the controller. Afterwards, Section 5.1.3 discusses on how existing methods and tooling support the design steps.

5.1.1 Information for conceptual design

The high fidelity model mentioned in Figure 39 refers to the models contributed from the different engineering disciplines (including control), which represent the dynamics of the system and the steps/rules of the performed processes. Integrating all these different models into a single executable model of the system dynamics is not an easy task [79]. In practice, most of the times the control engineer manually obtains such models. Another important piece of information to consider is the possible measurements and control actions that can be taken, as technological and economic limitations play an important role in the selection/design of involved equipment.

After attaining sufficient understanding of the system and the processes it carries out, the designers have to gather requirements (see Figure 39) which specify necessary or desired characteristics of the behavior and the structure of the product. These requirements and constraints may be directly linked to controller performance (e.g., rising time, overshoot, stability), but in many cases their formulation needs a “translation” to the control domain. Part of these requirements describes “irregular” situations and disturbances (e.g., noise) under which the system should be able to perform as expected. Agreement on a requirement set and on tests to evaluate conformity is crucial to the success of the design.

The actual design task (synthesis in Figure 39) begins after this point, defining the controller architecture. The relevant phenomena involved in the system behavior and the processes dictate many of the controller design decisions for synthesis, which at this stage centers on the selection of the controller structure. Deciding which parts of the behavior are "relevant" (i.e. must be modeled and considered) requires expertise in controller design and analysis, and cooperation with other experts. The goal of analysis at this point is to obtain simplified system models, representative of the real behavior, that allow selecting suitable controller structures. At the conceptual level, selecting the controller structure concerns partitioning which tasks will be managed by different parts of the controller.

Obtaining the information mentioned above involves exchanging data among designers and keeping track of the status of the project. These activities raise the need for additional project-related functionality in the control design support tools [182]. On a different front, support can be given to compensate for the lack of expertise in controller design [107], but such an approach is not considered within the scope of this work.

5.1.2 Information for detailed design

Detailed controller design follows after the conceptual phase, going back and forth to improve the simplified models when they fail to represent important characteristics of the behavior of the system, and to change controller structures that do not result in the desired behaviors. This section provides an overview of the detailed phase of controller design, following fairly well known guidelines from literature ([15], [16], [125], [150]).

Classic control textbooks (e.g., Ogata [125], Astrom [15]) cover the 'detailed controller design' phase. Controller design is mainly shown as the process of modeling a physical system with well determined inputs and outputs, and then adjusting a set of control parameters in a transfer function to achieve the performance criteria. Other textbooks [16] explain how controller models (e.g., block diagrams, ladder diagrams) can be transformed into control software code that can be used for implementation. The importance of concurrency to deal with the challenges involved with the implementation on hardware and its verification/validation lacks a deep treatment in most cases.

Normally, a well delimited system model is not available to begin. Obtaining it takes several steps. The process requires iterating over the different steps, "jumping" when rework on a specific step is necessary. Here they are arranged sequentially:

- Decide the used type of controllers and design methods.
- Determine the values of controller parameters, obtaining stability and desired performance/robustness. This requires models that can be subject to analysis.

- Decide the type of hardware to implement the controller.
- Test and verify the controller against requirements. Controller/plant models (including physical prototypes) that can be used for simulations and tests are necessary.

The design of the controller progresses by adding the precise conditions that will trigger the control transitions along the process. Many conditions concerning error handling, safety, maintenance and other modes, etc, must be specified precisely to obtain a working controller. For the feedback/feedforward controller loops, normally the complexity of the (sub)system to be controlled and the required performance are the main drivers for deciding the type of controller, though also the choice of hardware can limit the complexity of the control algorithm. Once a regulator type is chosen, the values of its parameters can be determined. The methods to set the parameter values vary from heuristics to optimization algorithms, and often relate to the type of controller chosen.

Test and verification can be done at every point in the design. Real time and discretization requirements and constraints affect choices when designing the controller, and sometimes manage to affect choices as far/early as at the conceptual level.

5.1.3 Existing support tools/methods for controller design and discussion

This section contains an overview of the role of current tools and methods that support the controller design activities. A deeper review and discussion of the challenges to better support (conceptual) design of mechatronic products can be found in Chapter 2. This section is supported on that part of the work and on some insights by Maciejowski [107].

There is a recent awareness about the importance of concurrency in the conceptual phase of controller design, but not much is provided in terms of implementable methods and tools to support this kind of practice. CACE tools seem to focus on supporting the detailed design phase. Supporting the conceptual design phase relates strongly to the management of design activities, and as Maciejowski states, CACE tools in the future may have to deal with how best to support the work-flow of control engineers, which closely corresponds to conceptual controller design as defined here.

5.1.3.1 Detailed design support

James *et al.* described in [93] the state of the art of CACE tools in 1995. That paper reports the existence of software for comparing (PID) controllers, multi-objective optimization, improving (tuning) control, support of symbolic and numerical computation, hybrid systems representation and modeling are also mentioned [93]. Fifteen years later, most of these capabilities and other key contributions from the last decade [107] are still the focus of CACE tools (e.g. Matlab, LabView), centering on what is described here as the detailed controller design phase by supporting the (detailed) modeling and simulation of the system and controller. The detailed models (block diagrams, state machines, etc.) used there can be transformed later to code almost automatically. Challenges regarding the detailed phase of controller design are still available (e.g., control of constrained systems, hybrid systems), but still many complex systems can be controlled with the implemented techniques, and seemingly, industry is still reluctant to absorb some of the new control techniques [146] despite their value.

In spite of the view by which CACE design can usually be the first step in a systems integration problem, which should entail computer support for the exchange of information [94], many tools and methods do not provide such support. This constitutes a development niche to support tool integration through information exchange at the detailed phase, because creating a model of the system is expensive, even with model identification [107]. The work of Varsamidis [182] addresses this issue (even extending to conceptual design support), but focuses on the tight integration of control tools, and does not consider the identified need to integrate control with other disciplines. Tekin [161] also reports on such need.

Other desirable (not yet existing) characteristics of CACE tools are mentioned in [79] and include:

- Consider directly multidisciplinary physics of the plant.
- Consistently address the varied controller requirements.
- Explore the trade-off potential of controller structures with multiple tuning parameters more thoroughly.

5.1.3.2 Conceptual design support

There is a prevailing need to support the conceptual phase of controller development not covered by the CACE tools used in industry nowadays [107]. Some claim that block diagrams can be used to design the controller from scratch, but these models may not constitute the natural form of description to begin the design process [94]. Also, the "toolbox" paradigm has not yet shown to satisfy the requirements for high-level support [107]. Past efforts to produce tools that support the workflow of controller design can be grouped as follows [107]:

- **Searchable design databases** that record the technical state of a project in a model ensuring consistency, and allowing to query the model to check the current state of work. Obstacles to implement these are the variety and complexity of data and of possible queries to the model.
- **Expert systems** to capture design process knowledge and use it to guide the designer. These systems ended up limiting the possible design actions and problems.
- **Optimization problem solvers** in which design is seen as an optimization problem, and which provide a "language" to specify such problem. The

“languages” in these tools mainly developed into a sequence of predefined steps that narrowed the actions of the user.

To resume, it can be said that conceptual controller design support must overcome the problem of representing the varied design information. Finding a representation that accommodates to the design workflow is therefore crucial to build better CACE tools. As reported by Varsamidis in 1999 [182], to that date no single CACE tool addressed directly providing project-related functionality. To the best knowledge of the authors, such functionality has only been partially addressed by other tools outside of the scope of CACE tools [6]. In those cases, support for overview models is provided but there is no direct link to the domain-specific models to allow affordable integration [6]. It is necessary to explore the advantages of linking design information from domain-specific models and to provide an architectural description that enables understanding and navigating such links.

An interesting approach related to the use of architectures can be found in the work of Hayes-Roth *et al.* [82], though it focuses purely on software development. The related project sought to improve the coordination between disciplines for complex software design [93] by providing: Task decomposition of process activities, reference architectures of software models, and tools for architecture-based reuse of design.

5.2 Proposed approach

The previous section justifies the need for supporting the control development process considering a more holistic point of view. The problem of designing a control system can be misleadingly considered as a domain-specific problem. However, as seen above, the design of a controller requires input information from multiple disciplines in order to correctly implement the behavioral specifications. The previous ideas are also supported by authors like Kindler [97], who reports that the challenges for model-based software design in relation to modeling system behavior do not come from a lack of models to represent behavior but are due to a lack of concepts which allow integrating behavior models with other models (e.g., representing structure).

The proposals in chapters 6 and 7 are based on the architecture-centric approach described in Part I. As depicted in Figure 40 (se also Figure 9), the AM is used to model a formal controller specification which is machine readable and at the same time remains understandable to all development stakeholders. The generic nature of the AM does not constraint many modeling choices. In the course of the development of the work for control architecture generation it was found that additional “modeling constraints” (mainly regarding the syntax) are useful (and sometimes necessary) to obtain correct controller specifications which can be used for automatic generation of supervisory control design models. It is also worth noting that the work follows the same considerations regarding having a complete model defined in Section 3.3 and the modeling limitations as specified in Section

3.3.7.2. These modeling limitations and the information labeling choice impose most of the modeling constraints mentioned above.

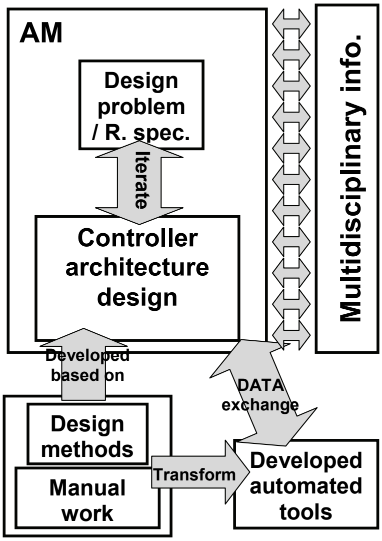


Figure 40. Elements in the proposed supervisory control design process

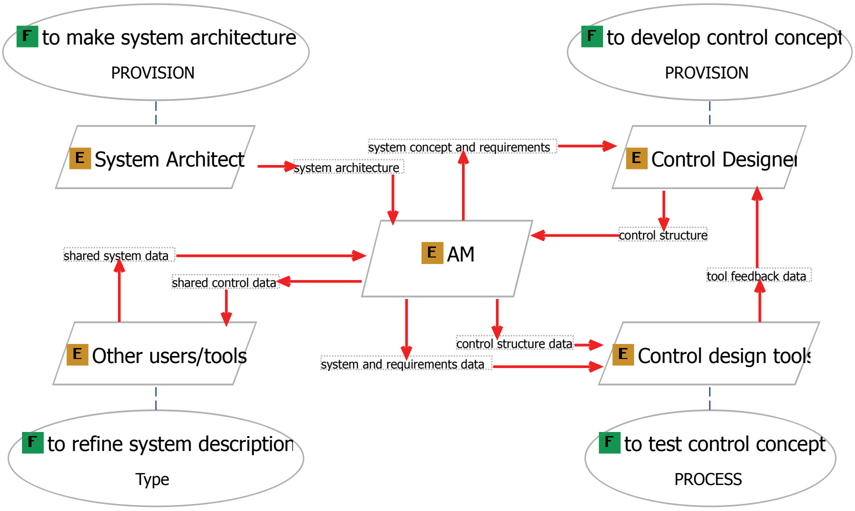


Figure 41. Scenario for control design support

The scenario for control development support is presented in Figure 41. Using the AM, system architects can provide a clear view of the system architecture which the control designer can use as input to develop the controller structure. The resulting AM can be transformed into formal representations that can be used by the control design tools to analyze and verify the specifications. Feedback from the results of the control design tool together with increasingly detailed information from other domains can be used to refine the design. In the context of this scenario, design information can be reused at almost every step to reduce required effort and time, making this proposal scalable. Previously developed design knowledge can be instantiated to describe repeated solutions, and data shared and updated through the network reducing information delay and transcription errors.

5.3 Conclusions

This chapter explains in general terms how the proposals from Part I can be implemented to specifically support controller design activities through the interaction with controller design methods and tools.

The review of existing methods and tools which specifically support control design (i.e., CACE) confirms the findings documented in Chapter 2, and shows that MBD based on DSM implementations lacks the crucial integration component necessary to properly support design activities for mechatronic systems.

- The proposed workflow fulfills the characteristics missing in the current CACE approaches presented in Section 5.1.3, supporting conceptual design by providing consistent and indexable information which can be used for searchable databases, and supporting detailed design by facilitating the exchange and presentation of multidisciplinary design information used to build the plant models and for evaluating design tradeoffs.

6 Regulatory control structure

A regulatory controller takes care of maintaining a group of variables under control following a reference, and in general it is implemented as feedback and/or feedforward control loop. Its structure comprises identifying the state variables and selecting an appropriate group of measurements, manipulated variables and controlled outputs. In general, an “appropriate group” allows obtaining a stable behavior and following the references under the performance specifications.

This chapter focuses on the regulator layer of control, and more specifically, on its configuration. The control configuration problem [150] refers to the task of selecting appropriate inputs and measurements for the system and grouping them in individual control loops that can attain the following goals:

- Stabilize an unstable plant
- Reject disturbances

Track reference changes

In this chapter, it is shown how the input information from the AM can be (semi)automatically transformed to be used in powerful and light “linear structured systems” (LSS) analysis and synthesis techniques ([48]-[52], [61]) which can be used to tackle part of the control configuration problem, possibly extending to other domains. Figure 42 depicts an overview of such transformation process, where the behavior layer of the AM is parsed to extract a parameter network which in turn can be transformed into a LSS representation (in graph or matrix form) using additional information from the other AM layers.

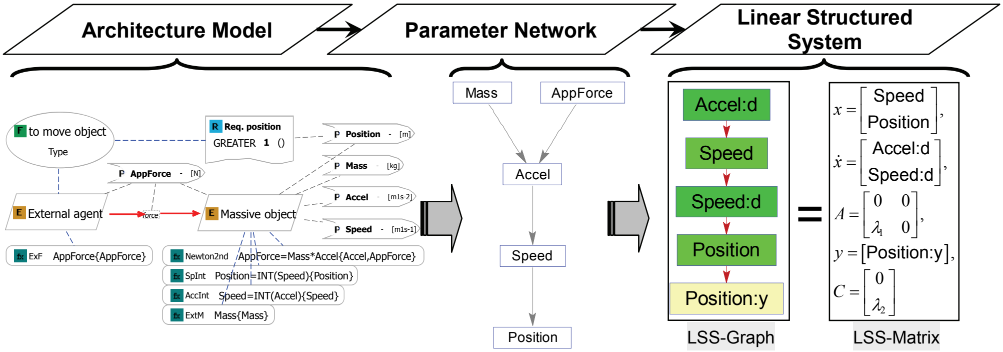

Architecture Model
Parameter Network
Linear Structured System
F to move object
Type
R Req. position
GREATER 1 ()
P Position - [m]
P Mass - [kg]
P AppForce - [N]
P Accel - [m1s-2]
P Speed - [m1s-1]
E External agent
force
E Massive object
fx ExF AppForce{AppForce}
fx Newton2nd AppForce=Mass*Accel{Accel,AppForce}
fx SpInt Position=INT(Speed){Position}
fx AccInt Speed=INT(Accel){Speed}
fx ExtM Mass{Mass}
Mass
AppForce
Accel
Speed
Position
Accel:d
Speed
Speed:d
Position
Position:y
LSS-Graph
=
$x=\begin{bmatrix}\text{Speed}\\ \text{Position}\end{bmatrix}$,
$\dot{x}=\begin{bmatrix}\text{Accel:d}\\ \text{Speed:d}\end{bmatrix}$,
$A=\begin{bmatrix}0 & 0\\ \lambda_1 & 0\end{bmatrix}$,
$y=[\text{Position:y}]$,
$C=\begin{bmatrix}0\\ \lambda_2\end{bmatrix}$
LSS-Matrix

Figure 42. Overview of AM to LSS transformation (top) and the corresponding models for a simple example (bottom)

The chapter is structured as follows: First, the general aspects of the control configuration problem are explained in Section 6.1. Section 6.2 describes which kind of information relevant for the regulator configuration problem can be extracted from the AM. Section 6.3 briefly presents related techniques of LSS analysis, and Section 6.4 explains how to extract the control-relevant information from the AM and transform it into an LSS description which can be analyzed as explained in Section 6.5. An example case study demonstrates the concepts from previous sections in Section 6.6, while a second case study in Section 6.7 is used to discus how the results may extend to other domains (for co-design). Section 6.8 finishes the chapter with a brief discussion of the results and conclusions.

6.1 The controller configuration or I/O problem

This section gathers material from the work of Skogestad *et al* [150] and from the experience of the authors, dealing with the most general aspects of the control configuration problem. The purpose is to quickly introduce the most general audience into the control domain and the configuration problem. The terminology from [150] has been modified in order to better match some of the terms in the works of Commault *et al* [49], used in section 6.5.

6.1.1 The regulator and its parameters

The regulatory controller configuration problem seeks to define the structuring/-decomposition of each regulator. First a model for the regulator is described. Each regulator can be generically described as shown in Figure 43. Five important signal types that relate to the regulator can be distinguished:

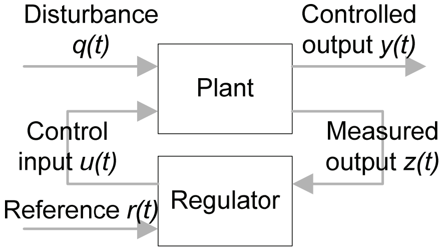


Figure 43. Generalized plant model

- Controlled outputs, $y(t)$, consist of those signals of the plant which are to be kept under certain values to achieve the system goals.
- Control inputs, $u(t)$, are the signals which are manipulated by the regulator through actuators.
- Measured outputs, $z(t)$, refer to signals coming from the plant which are measured and sent to the controller.
- Feedback controllers typically take in consideration reference signals, $r(t)$, that can be compared to some of the (measured or estimated) controlled outputs to

compute the control inputs based on the comparison (i.e., error, $e(t) = y(t) - r(t)$); in this sense, feedforward regulators do not use such reference signals.

- Plant disturbances, $q(t)$, include all input parameters to the plant which are not part of the control inputs. Thus, $q(t)$ includes process inputs which are not under direct control of the regulator and may be subject to variability. If present, plant model uncertainties and measurement noise are modeled explicitly in the AM and therefore are not considered in this category.

The 'signals' above are represented by 'parameters' in our models. Of course, the models also require a representation of the plant itself. A sixth type of parameters not mentioned above are the state variables, denoted by $x(t)$. State variables are used to identify the current state of the system and are inherent to it, though state variables can be represented in different reference frames.

The first problem to tackle when dealing with the control configuration problem is the selection of such parameters. A big part of the Input/Output (I/O) selection problem [180] can be included within such scope. This problem is often solved by using the experience of the designer or using methods that rely heavily on the analysis (e.g., by simulation) of numerical data to quantify the interactions among parameters and other properties like observability and controllability. These techniques try to formalize more "general" solution guidelines. As an example, refer to the guidelines from Chapter 10 in [150], which uses methods based on the computation of condition numbers and the relative gain array. In this context one can pose the question: how to solve the control configuration problem at an early stage of design (where it should be initially posed) in which normally no (detailed) values are available and possibly many design alternatives are still under discussion? The techniques discussed in the sections ahead address this challenge.

Normally, $y(t)$, $r(t)$, $q(t)$, and $x(t)$ form part of a well posed configuration problem. Surely, posing the problem may also be considered part of the problem itself, as it involves important decisions that affect how the whole system is constituted. Such parameters act as the input information to select $z(t)$ and $u(t)$. In general, the selection of the latter parameters should achieve the following:

- Ideally, vales in $y(t)$ should be independent of values in $q(t)$. If this is not the case, the problem of disturbance rejection must be tackled and a part of $u(t)$ should affect the system ($x(t)$) in such way that changes of values in $q(t)$ affect as little as possible the values in $y(t)$.
- Values in $x(t)$, $y(t)$, and $q(t)$ should be known, either directly from $z(t)$ or indirectly by computation from a part of $z(t)$. The part of $z(t)$ used for a purpose should, ideally, be robust and reliable, e.g., the part of $z(t)$ used to quantify $y(t)$ should be independent from $q(t)$ and other disturbances.
- Changes (within range) of values in specific parts of $u(t)$ should efficiently affect values in $y(t)$ in a specific way, i.e., complying to $r(t)$.

6.1.2 Control configuration elements

In addition to selecting the right parameters for the regulator, it is necessary to determine how these parameters relate to each other by selecting the links inside the controller and which will be the role of the regulator in the control scheme, possibly also conveying a design sequence. This is done before specifying the controller type (PI, PID, LQG, etc.) and gain values. Based on different properties of the problem, several choices with different uses can be mentioned at this point:

- The level of knowledge of the system and the disturbances: One has “good knowledge” when one can accurately predict system behavior. Feedforward (feedback) control elements can be used when very good (not so good) knowledge of the system and the disturbances is available.
- The coupling of systems: When subsystems are coupled (physically or by knowledge or measurement limitations), decoupling elements are used to be able to control a subsystem independently from other subsystems. In some cases, this structure appears as if the controlled outputs of a subsystem are considered as disturbances in another subsystem.
- The coupling of dynamics in time: When a system shows dynamics at significantly different time scales (the significance depends on the problem), configurations such as cascade control can be used to handle and tune the control of the dynamics separately. In cascade control the controllers responsible for the faster dynamics are “tuned” first, and subsequently considered as part of the plant for the following, slower, controller.
- The system dynamics can be accurately represented with linear models or not: These affect the configuration choices in multiple ways. However, here it is noted that together with our level of knowledge of the system, the nature of the dynamics affect the choice (including all the range in between) of using more simple regulators managed by a supervisor or selector, or using more complex regulators without any supervisor. Having multiple control objectives can also be considered a non linearity, as the controlled plant “changes” depending on the objective.

As it can be seen, the different regulators could need to be designed (tuned) in a certain order according to the chosen configuration. This is especially true in the case of cascade controllers (though it can happen in other configurations), where the regulators in the “inner loops” should be tuned before the regulators in the outer loops.

6.2 Parameter network

The parameter network is a causal dependencies graph [121] between parameters of a system. It is composed of parameter vertices and directed causal relation edges (*cf.* Figure 42, bottom-middle). An edge between two parameters indicates that the parameter at the beginning of the edge influences the value of the parameter at the end of the edge. The concept of causality taken in this work is explained in detail by Nayak [121].

The parameter network can be modeled directly, but this is an error prone activity when many parameters and relations must be considered. Therefore here the information in the behavior layer is used to facilitate modeling and allow reusing the information. The behavior layer contains ‘formula’ objects representing the effects of (natural or artificial) phenomena. In this work, the formulae are used to define causality relations among parameters. Therefore, the formulae only need to capture which parameters affect the value of every parameter in the behavior layer and, as done by Nayak in [121], do not necessarily specify more detail regarding how (proportional, inverse, exponential, etc.) they affect each other.

To be able to build the parameter network from formulae, the latter must be causally oriented [121]. In the context of building a linear structured system description, causal orientation obeys to the causality that explains the behavior of the system. Considering as a simple example Newton’s second law “Force equals mass times acceleration”, the value of the mass is not determined by the other terms and thus the corresponding formula can only cause the force or the acceleration. See formula ‘Newton2nd’ in Figure 42 (bottom-left) to see how this information is represented in the AM. Each formula can be chosen to cause a single parameter, and therefore, in cases where several parameters could be caused, causality can be determined by analyzing a complete set of equations. This analysis can be done using the causal ordering algorithm presented by Nayak [121], where a perfect bipartite matching is searched between the set of parameters and the set of equations that can cause them. Such algorithm has been implemented in Matlab together with an algorithm to read the data from the serialized AM.

In the AM, a complete formula set is mapped to each individual entity or entity relation (see Figure 42, bottom-left), allowing to build a parameter network for each object in the structure layer. Then, the individual parameter networks can be joined simply by using the shared unique parameters. To reduce the modeling effort, it is possible to implement more complex algorithms to build the parameter network by, for example, simultaneously ordering the equations of several entities, which reduces the number of equations needed to indicate that parameters are just ‘passed’ to an entity and caused by another one. The simple parameter network in Figure 42 shows how straightforward is the process to extract the parameter network from the formula sets: for the entity ‘Massive object’, the mass and the applied force are determined externally, then the acceleration becomes caused by those two parameters as formula ‘Newton2nd’ indicates, and the two formula specifying integration relations cause the speed and position parameters.

The parameter network can also be obtained from other physical models which are able to represent the causal orientation of parameters, e.g., bond graphs [95] (see also [58] for direct structural analysis using bond graphs). The main reason to choose using the information of the structural layer in the AM instead of other existing models is the ease of use and implementation. That is, the AM language is very simple and implementing a modeler that allows filtering and querying the information in the AM format did not require much effort. In that way it was easier to build the model of the structure natively and connect it to the other information in the AM rather than using an external model that had to be parsed and connected. Other important reasons are:

- It is possible directly to add requirements and other relevant information coming from different disciplines in the AM to define the control problem
- The AM structure representation is more flexible (though less formal) than other representations.
- Some existing models are easily understood mainly just by domain experts.
- Unnecessary details pertaining to domain-specific models (e.g., detailed mathematic equations) can be easily filtered out from the AM.
- To the best knowledge of the authors, very few models are intended to seamlessly represent behavior coming from both, natural/physical phenomena (as in mechanics or electrics), and from man-made/artificial phenomena (as it is present in software systems). This also translates into a breach between the representation of continuous and discrete behavior. Bond graph models [95] as used in physics-based control [149] and some models for hybrid systems [158] partially address these representation gaps.

6.3 Linear Structured Systems

A LSS corresponds to the 'structure' of a state-space description of a linear continuous-time time-invariant system [61]. To refer to the LSS this chapter adopts the form $\dot{x} = Ax + Bu + Eq$, $z = Hx$, $y = Cx$, and $u = Fz$, where the capital letters indicate the structure matrices relating the parameter vectors. By structure, it is implied that, for each matrix, it is only known which elements of the matrices are fixed to zero and which ones are not (also called free parameters), i.e., except for zero, the values inside the matrices are not known! Such representation allows analyzing a system and obtaining well founded guidelines about which control configurations lead to desirable control properties (*cf.* Section 6.1.1), though normally they do not allow obtaining a direct answer about which configuration should be chosen. An important advantage of LSS over other (value-dependent) techniques is that it requires substantially less detailed data, analytical effort, and computational power [180]. It is common that the structure matrices are represented as graphs and that graph theory tools (e.g., path finding algorithms) are used to verify the structural properties. This facilitates visualization and interpretation of the results. These characteristics make the method a good candidate to match control-related information even at the early stages of design, where architectural descriptions start developing.

It follows a presentation of the desirable control properties that can be structurally verified using the LSS analysis, including a simplified description of how each test is performed. A more complete description will be given in Section 6.5. Though these methods do not coincide with the common methods to determine these properties numerically, they are funded over equivalent (but less detailed) theoretical basis (e.g., zero structure at infinity). For more tests (invariant zeros, fixed modes, fault detection, etc.) and further details the reader is referred to the comprehensive work on the topic by Dion *et al* [61], to the earlier work of Reinschke[140], and other available papers ([48]-[52]).

Observability: Determines whether one can find out the state of the system with the current sensors (measuring $z(t)$). It is verified by checking paths between parameters representing state variables and measured outputs.

Controllability: Verifies if it is possible to affect the state of the system with the current actuators (affecting $u(t)$). It is tested by checking paths between parameters representing control inputs and state variables.

Disturbance rejection: More specifically refers to the problem of disturbance rejection by measurement feedback and the test documented by Commault *et al* in [49]. It basically deals with detecting whether the disturbances can be detected fast enough (i.e., are dynamically close, *cf.* Section 6.1.1) and the controlled inputs can correct such disturbances before they reach the controlled outputs.

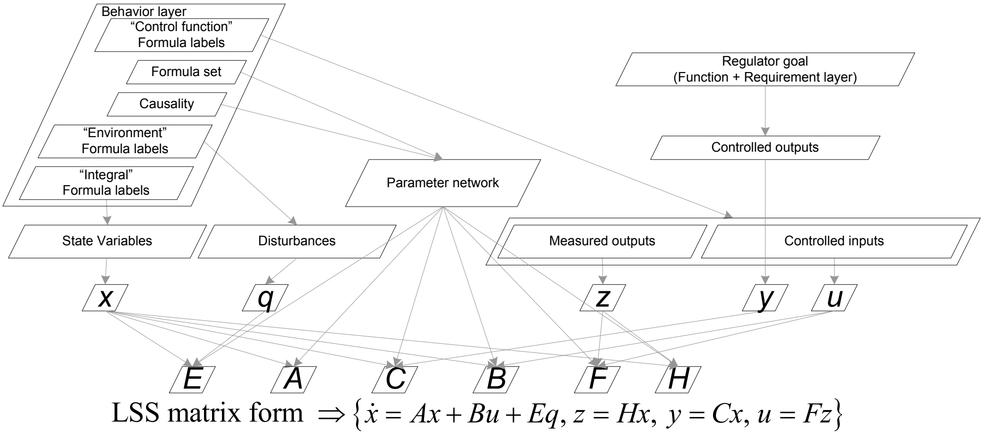
I/O grouping and pairing: Consists of trying to pair inputs that are better fit to control certain outputs with such outputs, and of grouping inputs and outputs meant to perform similar control tasks.

The subsequent section details how the information in the AM can be matched and transformed into an LSS description, performing what is coined here as parameter classification, and gives some examples of the tests mentioned above.

6.4 Parameter classification process

The LSS description necessary for analysis can be obtained directly from models such as the state space description of the system or its equivalent system of equations. Normally, the control engineer either receives such models from other stakeholders or has to develop it by himself. In the first case, the control engineer can treat the model mainly as a black box, and little communication will be present among the stakeholders. While this may be acceptable when dealing with well known systems, for systems under development it can mean that the modeling considerations are not transmitted effectively to the control engineer, reducing his ability to detect and solve many problems. On the other hand, the second case implies that the control engineer will have to look for the model information and assumptions by himself. This situation can be acceptable when dealing with simple systems, but more complex systems can demand much effort from the control engineer, not only to get enough understanding for building an acceptable model, but also to update the model when changes from other disciplines affect its validity.

The approach followed here is an important contribution of this work. It is proposed to let each stakeholder build his /her part of the model, using a simple modeling language common to all stakeholders (that is, the AM). Then the control engineer can query for the necessary data and automatically obtain a model suitable for LSS analysis, with the additional advantage that he can gain better insight on the system by querying the modeling platform to the extent he wishes. This section describes the process to query and identify the relevant data in the AM and to generate an LSS description (see Section 6.3) from it.



$$\Rightarrow \{\dot{x} = Ax + Bu + Eq, z = Hx, \ y = Cx, \ u = Fz\}$$

Figure 44. AM (top) and LSS (bottom) data correspondence for the parameter classification process

Figure 44 provides a more detailed view of the last step of the transformation process introduced in Figure 42. For this case, each formula is labeled directly by identifying certain keywords in it (e.g., 'INT' in some of the formula in Figure 42, bottom-left) which identify the appropriate parameter type in the context of LSS. Then, the parameter network is used to find how the relevant parameters causally relate to each other. First it is useful to focus on the 'plant' or the portion of the system that does not include the controller. Thus, the formulae corresponding to the controllers (labeled as 'CONTROL FUNCTION') are not used. The state variables present in a dynamic system can always associate with the presence of their corresponding derivatives. Therefore, from the formulae labeled as 'INT' one can identify the parameters corresponding to the state variables and their derivatives, respectively as the caused and causing parameters. The simple case from Figure 42 (bottom-left) depicts how some formulae are labeled with 'INT'. As seen in Figure 42 (bottom-right), position and speed are identified as state variables, and speed and acceleration as their derivatives. The parameters corresponding to derivatives of other parameters have been named ending with a ':d' for the sake of convenience (*cf.* Figure 42, bottom-right).

From the control perspective, disturbances correspond to all the system inputs which will not be controlled by the regulator. A set of disturbances can be identified as the causing parameters from the formulae labeled as 'ENVIRONMENT'. Considering only the formulae labeled as 'CONTROL FUNCTION' it is possible to obtain the measured outputs as the set of causing parameters, and the controlled inputs as the set of caused parameters. For simplicity our simple example in Figure 42 does not contain a suggested controller structure. Since the parameter ‘AppForce’ is not caused by a formula labeled as a control function, the system does not identify it as a control input. Identifying the controlled outputs is based on the premise that it is not possible to control any parameter without specifying some desired characteristics about how it should behave. Therefore, the model must contain the required characteristics for all such parameters as requirements mapped to them. The set of parameters constrained by valid requirement objects determines the potential controlled outputs. However, since requirements are also used to specify acceptable characteristics of disturbances (i.e., the operation conditions), one must subtract from the previous set those parameters that belong to the disturbances set. For our simple example in Figure 42, the position is correctly identified as a controlled output. For clarity, when a state variable is both a measured and a controlled output, dummy nodes have been created for the controlled output case, adding ‘:y’ to the original state parameter name. The requirements definition for set of controlled outputs can be effectively used as the description of the reference values for control.

At this point, the implemented tool has identified all the different types of parameters required for the LSS model. However, the structure matrices also have to be built. To do this, the network has been simplified so that it contains only the classified parameters and their connections, i.e., each node corresponding to a parameter not present in the sets found with the labeling shown above is replaced by direct connections among its adjacent nodes. In this implementation, it has been decided to traverse the network to find each path connecting a pair of parameters, and then simplify it as a single causal relation. The results for our simple example can be seen in Figure 42 (bottom-right).

Section 6.6.3 illustrates how the parameter classification was done for the bigger example case, giving further detail about the parameter classification process. The analysis of control properties is explained in Section 6.5.

6.5 LSS analysis

From above it can be seen that the result is that, starting from an AM, the developed tool is able to interpret the data and generate a LSS description. The resulting LSS is used here to analyze a prototype controller regarding controllability, observability, and disturbance rejection. Afterwards, the tool also presents some advice to modify and improve the prototype controller regarding I/O pairing. Examples of this analysis are provided in Section 6.6.

Several equivalent approaches can be used to test the different control properties, and in this work uses some of the most recent methods ([49], [51], [61]) which transform the original LSS into bipartite graph descriptions which can be efficiently traversed to perform the tests introduced in Section 6.3.

The observability and controllability tests respectively verify each state variable reaches at least one measurement (output connected) and can be reached by at least one control input (input connected). This can be verified by tracing paths from/to each state variable to/from the nodes marked as outputs/inputs in the LSS graph. Additionally, the controllability and observability tests include verifying whether the system is irreducible. Irreducibility of the system matrix can be proven using LSS techniques as shown by Reinschke in [140].

The measurement feedback disturbance rejection test can result positive or it can indicate that, with the selected configuration, the problem is not solvable because not enough measurements to appropriate state variables are available. This "appropriate" set of state variables (called here 'I*') is such that, an inaccessible disturbance affecting directly these states can be rejected by state feedback, provided that the disturbances do not affect directly the outputs [49]. Unfortunately, the test does not indicate exactly which measurements set should be taken.

The analysis of the I/O pairing can be addressed by solving the problem of non-interacting control [61] for the cases where equal size sets of inputs and outputs have been defined. The pairing of inputs and outputs is done by finding minimum length paths connecting inputs and outputs. The existence of such implies that the non-interacting control problem is solvable.

6.5.1 Additional experimental methods

Additional clues for measured outputs and controlled inputs have been found during the tests and are presented here as part of our contributions. However, no formal proof has been developed yet. Furthermore, from the practical point of view, the output suggestions may contain unfeasible solutions for the implementation (e.g., because of cost, or implementation difficulty).

Some I/Os can be suggested depending on the state variables relations and the identified controlled outputs. The first step consists on finding minimum-weight paths from the state variables to the controlled outputs in a similar fashion to that of the observability test. The resulting paths (resp. 'Pso') can be used to suggest possible measured outputs at the end of them, and possible control inputs at the beginning of them. For our simple example from Figure 42, the system suggests to measure the position and to actuate through the acceleration. Checking for the parameters that affect the suggested input in the parameter network (for our simple example, the mass and applied force) can provide more clues to select the control inputs.

If disturbances are known, this suggestion can be further expended to cope with the disturbance rejection problem. Each of the suggested set of control inputs can be expanded along the Pso paths to the state variables which are right after the state variables affected by the disturbances. The measured output combinations can also be expanded. The minimum-weight paths (resp. Pqi) are defined as going from the disturbances to the nodes of the set I* that connect to state nodes out of such set (this subset is called ‘I*f’). New measured outputs can be taken from the states in such paths.

Boarding the I/O matching problem through the non-interacting control approach can help pairing single inputs and outputs. However, following the same lines of thought as our previous suggestions, inputs or outputs can be clustered in bigger groups according to the metrics of path length (i.e., inputs with similar path lengths to a certain output can be considered similar) or other heuristics like coincidence with a path affected by a certain disturbance (i.e., an I/O path with a Pqi path). Other possible suggestions to cluster inputs and outputs may come from the subjective definition of the function nodes and their attached requirements by, for example, treating controlled outputs as a group when they must comply with requirements mapped to a single function. Thus, the latter suggestion would take into account the desired configuration proposed by the designer.

6.5.2 Control configuration interpretation of the LSS analyses

In addition to the previous analyses, which mainly deal with selecting an appropriate set of inputs and outputs (see Section 6.1.1), some light can also be shed over the problem of choosing certain control configuration elements like the ones introduced in Section 6.1.2.

- Feedforward elements can be implemented for complete rejection of disturbances which can be quickly measured or whose behavior is known in advance. This can include, for example, inertial or gravitational effects over a known mass. In this sense, the specific measurements and control inputs added for the disturbance rejection problem may be implemented as feedforward controllers.
- In general, decentralized/decoupling control elements can be implemented whenever the non-interacting control problem can be generically solved.
- Use of a cascade control is greatly affected by the speed of its composing parts, therefore, LSS techniques do not seem appropriate to aid in the selection of cascade controllers.
- The supervisors (selectors) are meant to deal with specific nonlinear particularities or different operation modes depending on specific values. Since these details cannot be appreciated in the LSS representation studied here, the analysis methods presented in this chapter do not support such choices.

6.6 Example case study: demonstration of transformation to LSS

The sample case studied here corresponds to the well known problem of control of a distillation column ([61], [150]), in this particular case, with thirteen stages. This section documents on the creation of an AM suitable for the extraction of a parameter network and the resulting LSS representation. The obtained LSS model is analyzed to make several suggestions for the design of the regulatory control structure.

6.6.1 AM for a distillation column

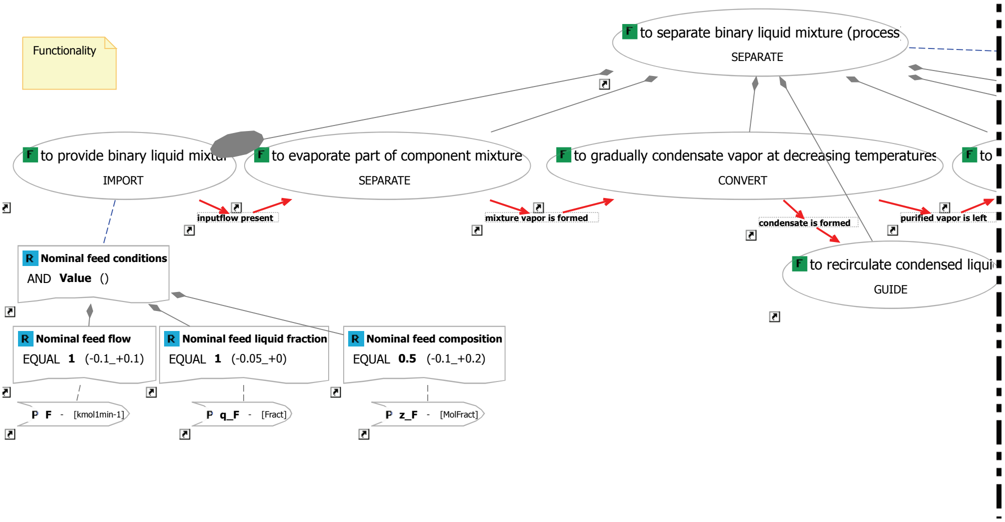

Functionality
F to separate binary liquid mixture (process
SEPARATE
F to provide binary liquid mixtu
IMPORT
F to evaporate part of component mixture
SEPARATE
F to gradually condensate vapor at decreasing temperatures
CONVERT
F to
inputflow present
mixture vapor is formed
condensate is formed
purified vapor is left
F to recirculate condensed liqui
GUIDE
R Nominal feed conditions
AND Value ()
R Nominal feed flow
EQUAL 1 (-0.1_+0.1)
R Nominal feed liquid fraction
EQUAL 1 (-0.05_+0)
R Nominal feed composition
EQUAL 0.5 (-0.1_+0.2)
P F - [kmol1min-1]
P q_F - [Fract]
P z_F - [MolFract]

Figure 45. Functional decomposition for distillation column

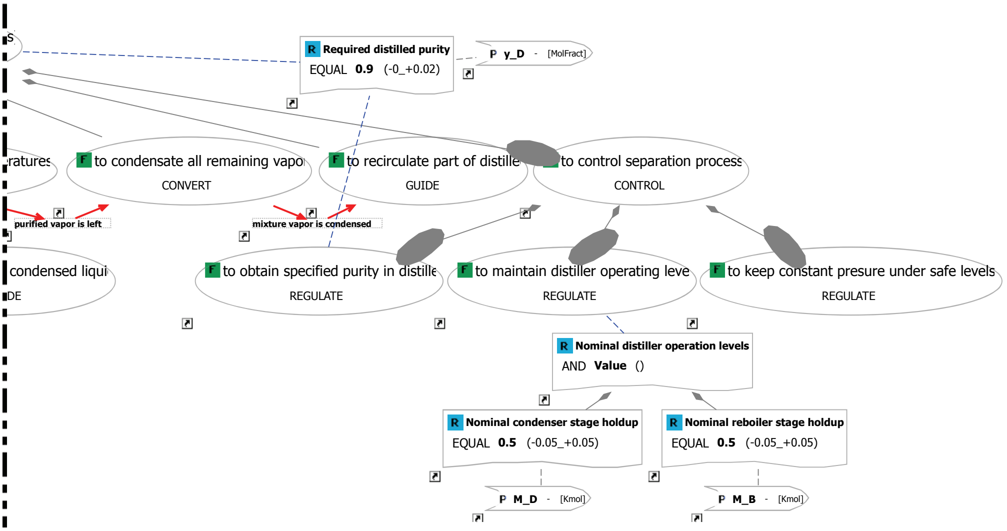

s
R Required distilled purity
EQUAL 0.9 (-0_+0.02)
P y_D - [MolFract]
ratures
F to condensate all remaining vapo
CONVERT
F to recirculate part of distille
GUIDE
to control separation process
CONTROL
purified vapor is left
mixture vapor is condensed
condensed liqui
DE
F to obtain specified purity in distille
REGULATE
F to maintain distiller operating leve
REGULATE
F to keep constant presure under safe levels
REGULATE
R Nominal distiller operation levels
AND Value ()
R Nominal condenser stage holdup
EQUAL 0.5 (-0.05_+0.05)
R Nominal reboiler stage holdup
EQUAL 0.5 (-0.05_+0.05)
P M_D - [Kmol]
P M_B - [Kmol]

Figure 45. (cont.) Functional decomposition for distillation column

This section contains a sample AM consisting of a distillation column with thirteen stages and a prototype controller, from which it is possible to derive a ‘parameter network’ that provides the raw material for control structure analysis. Initially, a brief description of the functionality may be sketched by a stakeholder such as the process designer or the system architect. Figure 45 documents that the purpose of the process is to separate a binary liquid mixture, with a certain required purity specified by the distillate concentration parameter ‘y_D’. The main functionality decomposes to describe in more detail how it is achieved, for example, by evaporating part of the mixture and gradually condensing the vapor. More detailed requirements specify the required conditions for the feed flow and operation levels.

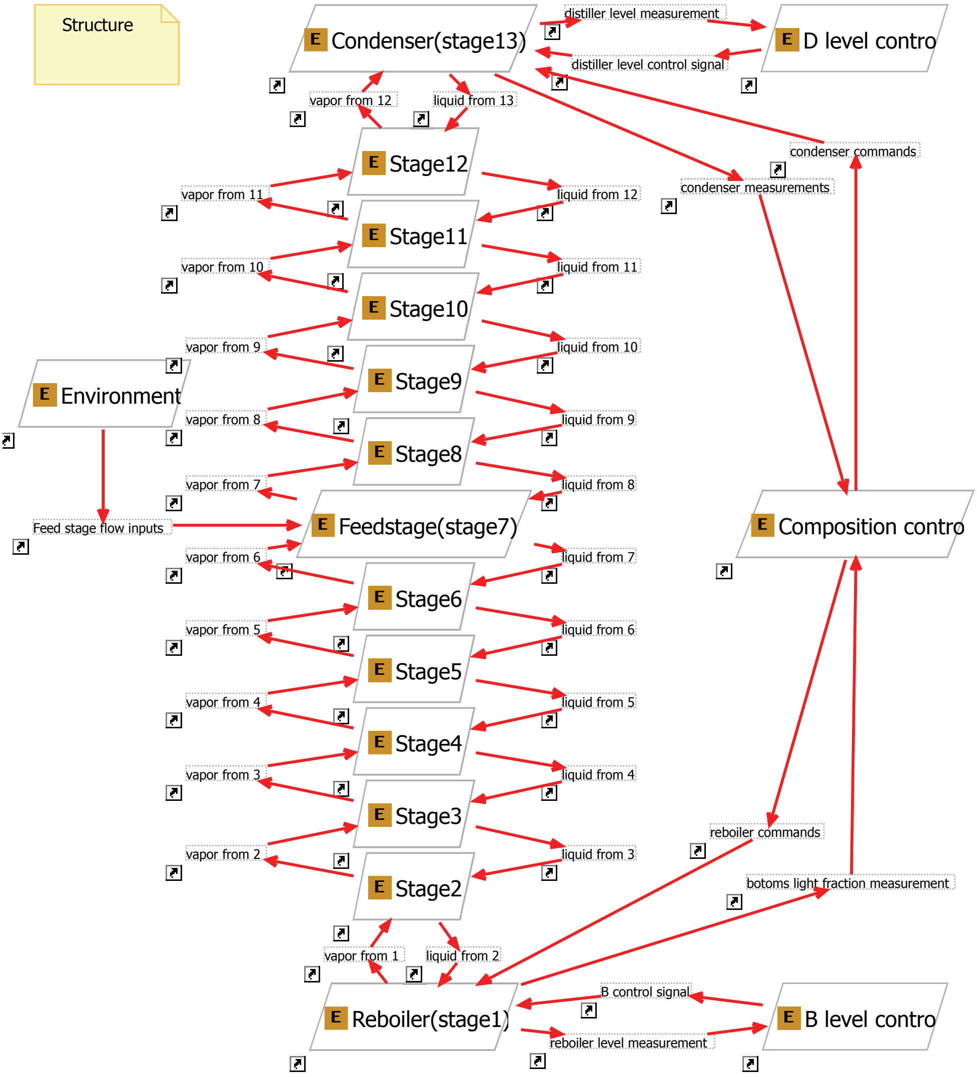


Figure 46. Separation column structure (parameters are not shown)

A stakeholder in charge of developing the implementation concept (e.g., process engineer) can model the structure diagram of the column (see Figure 46), which shows that the environment provides an inflow of mixture to be separated, while the stages orderly exchange vapor and fluid. Three separate regulators that exchange signals with the column take care of maintaining the levels at the reboiler and condenser stages, as well as of guaranteeing the composition of the distillate. At this point it is possible to indicate which parts of the structure take care of implementing the functionality by creating mapping links between the function and entity nodes. For space reasons all these mappings are not shown in here, and an example is presented later in Figure 48.

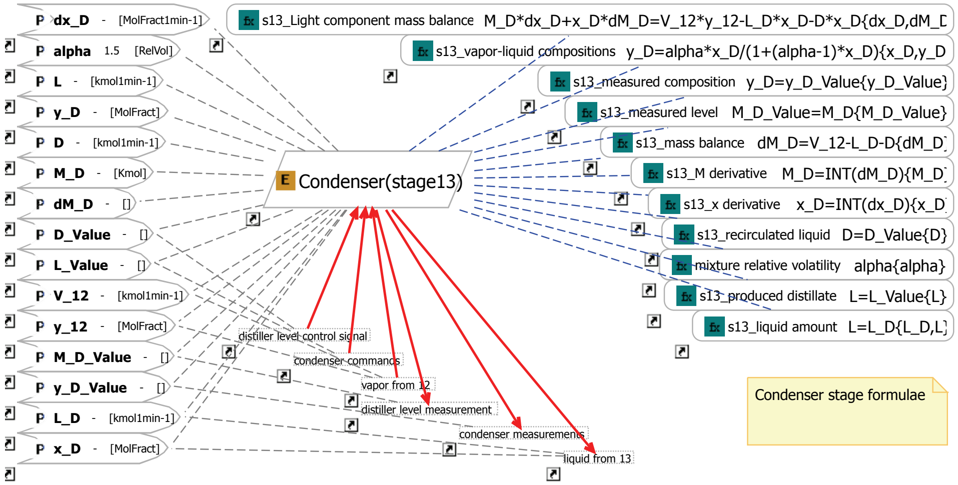

P dx_D - [MolFract1min-1]
P alpha 1.5 [RelVol]
P L - [kmol1min-1]
P y_D - [MolFract]
P D - [kmol1min-1]
P M_D - [Kmol]
P dM_D - []
P D_Value - []
P L_Value - []
P V_12 - [kmol1min-1]
P y_12 - [MolFract]
P M_D_Value - []
P y_D_Value - []
P L_D - [kmol1min-1]
P x_D - [MolFract]
E Condenser(stage13)
fx s13_Light component mass balance M_D*dx_D+x_D*dM_D=V_12*y_12-L_D*x_D-D*x_D{dx_D,dM_D
fx s13_vapor-liquid compositions y_D=alpha*x_D/(1+(alpha-1)*x_D){x_D,y_D
fx s13_measured composition y_D=y_D_Value{y_D_Value}
fx s13_measured level M_D_Value=M_D{M_D_Value}
fx s13_mass balance dM_D=V_12-L_D-D{dM_D
fx s13_M derivative M_D=INT(dM_D){M_D
fx s13_x derivative x_D=INT(dx_D){x_D
fx s13_recirculated liquid D=D_Value{D}
fx mixture relative volatility alpha{alpha}
fx s13_produced distillate L=L_Value{L}
fx s13_liquid amount L=L_D{L_D,L}
distiller level control signal
condenser commands
vapor from 12
distiller level measurement
condenser measurements
liquid from 13
Condenser stage formulae

Figure 47. Diagram containing the formulae for an entity, and its relation to parameters and entity relations.

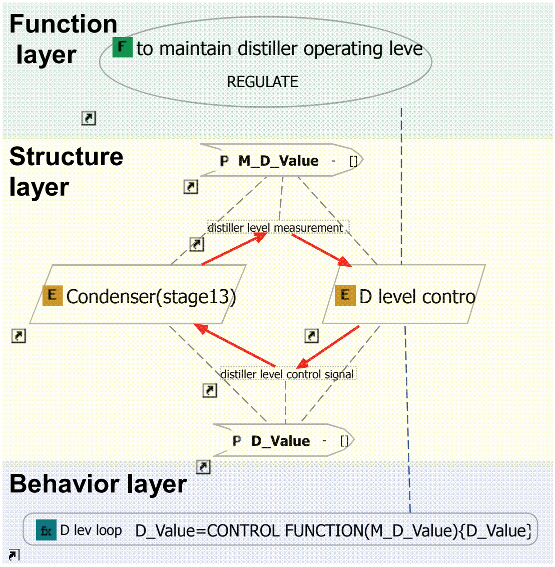

Function layer
F to maintain distiller operating leve
REGULATE
Structure layer
P M_D_Value - []
distiller level measurement
E Condenser(stage13)
E D level contro
distiller level control signal
P D_Value - []
Behavior layer
fx D lev loop D_Value=CONTROL FUNCTION(M_D_Value){D_Value}

Figure 48. View spanning over several model layers, showing objects directly related to the distilled level controller ("D level control")

For each entity and entity relation one can define a set of parameters, and formulae to define the relations among parameters. An example of this part of the model can be seen in Figure 47. At the modeled level of granularity, fifteen parameters define the state of this entity, and only a few constant values (e.g., the relative volatility 'alpha') have been set. Eleven formulae have been modeled explicitly to relate these parameters, while other four formulae are implicitly modeled for the parameters that relate to incoming entity relations, specifying that the values of these parameters are defined externally to the entity, and completing the formulae set. The derived formulae could also be modeled explicitly, and this implicit mechanism has been implemented to reduce the modeling effort. The formulae for each stage of the distillation column are similar from stage to stage, and the detailed formulation for the can be found in [150]. This information will be transformed into the parameter network. Other entities are described in a similar fashion.

An example of how stakeholders (such as the control engineer) can view parts of the model is depicted in Figure 48. The controller is meant to maintain the operating level in the distiller (mapped requirement is shown in Figure 45) by taking a measurement of the current distiller holdup and returning a control input that regulates the distilled reflow 'D', to the condenser stage valve (the detail of the valve is not modeled here). The formula 'D lev loop' indicates that the two parameters relate to each other through a generic control function, and that the parameter 'D_Value' is the caused parameter.

6.6.2 Obtaining the parameter network

Considering the limitations explained in Section 3.3.7.2, the distillation column system offers a single, fixed active structure that does not vary with its state (it is considered that the structure does not change during operation), and it is possible to focus on a description on a single level of granularity corresponding to a consistent set of equations. Since a fixed active structure is considered, the required behavior specified by the functions and function relations will not be used. After verifying that all entities count with a complete formula set and appropriate relations, the model can be parsed as explained in Section 6.2 to build the parameter network (*cf.* Figure 49), containing 120 parameters and their causal relations. As it can be seen in the figure, many intertwined causal relations exist in this fairly simple system.

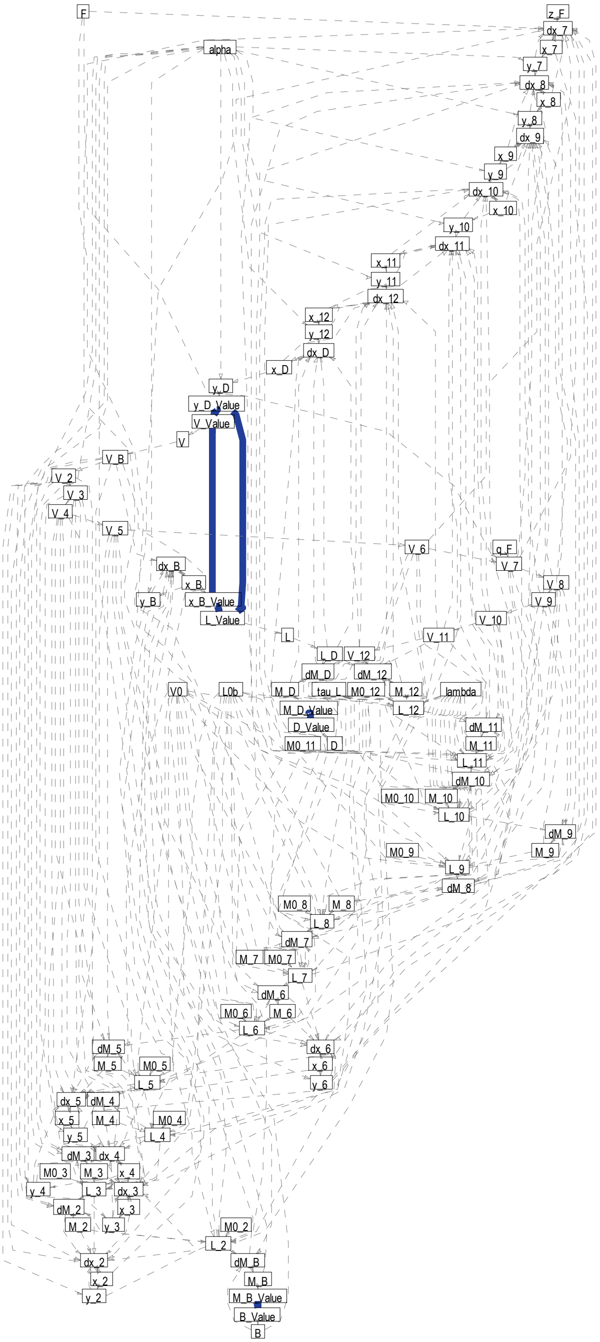


Figure 49. Parameter network for the 13 stage distillation column. Thick continuous lines represent relations coming from formulae in the controller definition.

6.6.3 Parameter classification

Starting from the parameter network (*cf.* Figure 49) and the formula type information classify the interesting parameters (see Section 6.4). Twenty six parameters are correctly identified as state variables, corresponding to the mass holdups ‘M_#’ and the liquid compositions ‘x_#’ for each of the thirteen stages (identified here with ‘#’). With a different set of equations, the vapor compositions ‘y_#’ could be used instead of ‘x_#’ because these variables are coupled (algebraically related). To avoid wrongly identifying coupled variables as independent ones (in case their respective derivatives figure in the formulae) the implemented algorithm checks for the existence of algebraic relations among state variables by looking for any path between them which does not contain a parameter that is the derivative of the other state variable. For our example, the input feed flow ‘F’, its composition ‘z_F’, and its liquid fraction ‘q_F’ are placed in the disturbances set, based on the ‘ENVIRONMENT’ labels.

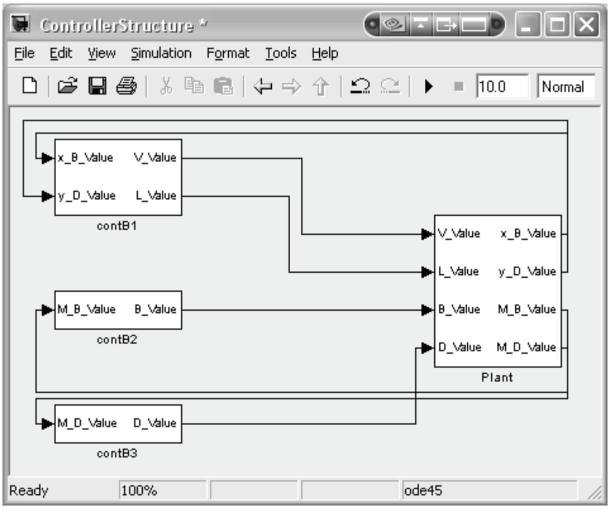


Figure 50. Block diagram representation of the original prototype controller structure in a domain-specific tool

Using the control formulae labels, the measured outputs are identified as the parameters corresponding to the measured values of the boiler and condenser mass holdups (‘M_B_Value’ and ‘M_D_Value’), and to the boiler and condenser compositions (‘x_B_Value’ and ‘y_D_Value’). The plant is to be controlled with the control inputs of reflux ‘L_Value’, boilup ‘V_Value’, distillate ‘D_Value’, and bottoms ‘B_Value’, which determine the value of the corresponding physical flow parameters ‘L’, ‘V’, D’, and ‘B’. The controlled outputs correspond to the parameters of condenser and reboiler holdups (‘M_D’ and ‘M_B’), and to the distillate composition ‘y_D’. A block diagram representation of the modeled controller structure can also be built automatically (see Figure 50). This is a simple transformation from the AM to a domain-specific tool.

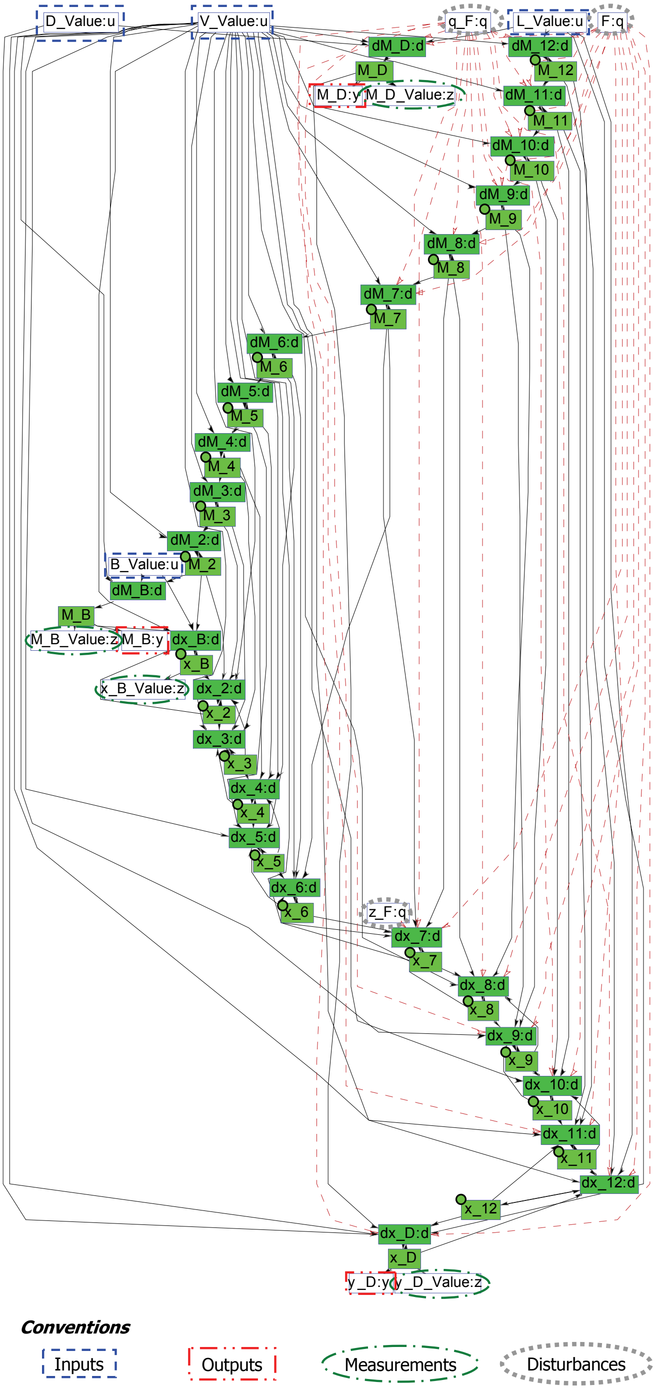


Figure 51. Graph representation of the LSS model of the distillation column. It shows the state variables, their derivatives (:d), disturbances (:q), measured outputs (:z), controlled outputs (:y), and control inputs (:u). Dashed lines represent influences from disturbances.

The result of the parameter classification process (*cf.* Section 6.4) is shown in Figure 51, which can be interpreted as the equivalent set of structure matrices in the LSS.

6.6.4 LSS analysis

The prototype controller structure is shown in Figure 50, and the corresponding LSS appears in Figure 51. The result of the measurement feedback disturbance rejection test is that, with the prototype configuration, the problem is not solvable because not enough measurements to appropriate state variables (belonging to I*) are available. In the current situation, two disturbances ('q_F' and 'F') affect directly the reboiler and condenser level outputs, and thus, should be measured directly for effective action. For the purpose of continuing the analysis this last fact is ignored from this point on. The algorithm indicates that at least two different measurements (sensors) have to be taken from the I* set, whose nodes are marked in Figure 51 with a dot '●' on their top-left corner. It is possible to see that only one measurement is taken from the suggested set (over 'X_B'). As in the last example provided in [49], the necessity for another measurement may rise from the fact that the original sensors measure states under the combined effect of the disturbances, and thus, enough measurements need to be taken to discern which disturbance is acting and in which amount in order to respond accordingly.

For the I/O pairing in our example, the minimum length paths indicate the following matching: 'V' to 'y_D'; 'L' to 'M_D'; 'B' to 'M_B'. Since there are more control inputs than controlled outputs, the unmatched input are simply not considered. There are no shorter paths connecting inputs and outputs, though there are other possible paths of the same length, which means that the non-interacting control problem is solvable.

The additional tests presented in section 6.5.1 are performed in our example to suggest a controller. The Pso paths indicate a suggested controller in which measuring the state variables 'M_B', 'M_D', and 'X_D' provides observability, while controllability is attained by actuating through the derivatives of state variables 'M_B', 'M_D', and 'M_12'. Therefore, the suggestion is to control the levels at the distiller and reboiler directly, while the composition is controlled by affecting the holdup at the stage right before the distiller.

Considering the disturbances, it can be proven that there is no need for any additional control inputs to solve the DR problem in the case of the original prototype structure. For the new suggested controller, the control inputs set suggested previously from Pso paths would require an input to affect directly derivative of the output 'X_D'. To expand the measured outputs the Pqi pats suggest that it suffices to measure the mass holdup in stage two 'M_2' and the composition at stage twelve ('X_12') in the original prototype controller. The same measurements also have to be added in the suggested controller.

6.7 Example case-study: Co-design using analysis results

A co-design (or concurrent design) approach between the different disciplines is important to efficiently attain integration, and companies struggle to achieve designing through such an approach [28]. Co-design [59] is enabled when the influences of the different subsystems among one another are made evident to the eyes of the designers. This section, complements is the proposal earlier in this chapter by defining a design workflow which assumes a common representation scheme of the physical and control architectures of a system in addition to the availability of LSS analysis methods, and which also features an external tool to specify the physical architecture of the system. Therefore, the results of such analysis may suggest modifications of either physical or control architecture (or both architectures) to improve the system performance. With help of design and analysis tools along with the workflow, a designer could systematically improve the original sensor-actuator configurations by modifying either the physical architecture or control architecture (or combination of them). The section also briefly explains the employed tools and algorithms, which are extension of the work by the authors about computational supports for system architecting ([11], [99]).

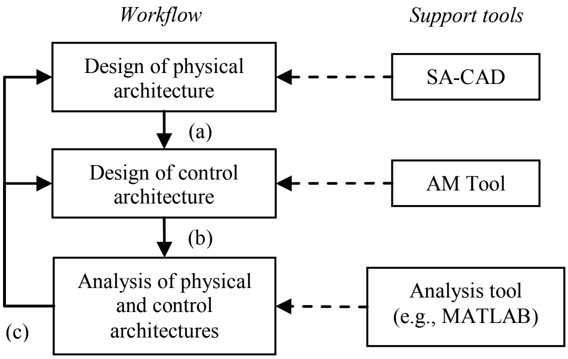
The next section explains in more detail the proposed design workflow, the employed representation of the physical and control architectures of a system, and tools to build a system model following the representation and support the workflow with the model, ending with an explanation of how the analysis results can be applied to physical decomposition. After that, a section demonstrates the implemented method through the example case study, where physical and control architectures are co-designed. More details on this specific approach can be found in the original source [8]

6.7.1 Proposed Design Workflow and Modeling

Figure 52 shows the proposed design workflow and tools to support co-design of the physical and control architectures of a system. The workflow mainly consists of iteration of the design and analysis of the architecture of a system described in physical and control domains.

First the initial physical architecture of a system is described using the SA-CAD as a set of subsystems, which realize desired behaviors at the system level in order to meet required functions [100]. The SA-CAD system is used to assist developing the physical architecture and its decomposition. After modeling these concepts, it is possible to import them into the AM. Then, the control architecture of a system is defined as specified in the previous sections. At this point, the proposed LSS analysis can be performed, but additionally, some insights regarding the physical decomposition can also be gained.

On the one hand, the insight gained from the LSS analysis may not only have effect on the controller architecture, but also on the physical architecture of the system. In the most common scenario, the analysis results show that the current actuators or sensors are not appropriate to obtain desirable control properties. In this case, a change on the physical architecture has to be considered and assessed.


Workflow
Support tools
Design of physical architecture
SA-CAD
(a)
Design of control architecture
AM Tool
(b)
Analysis of physical and control architectures
Analysis tool (e.g., MATLAB)
(c)

(a) The architecture of physical system

(b) Integrated architecture of physical and control systems

(c) Suggestions of modifications about physical and control architectures

Figure 52. The proposed Workflow and support tools

On the other hand, it is evident that changes originating from the physical architecture can also affect the design of the controller. Deciding to change, add, or merge any component may have an immediate effect on the system (model) used as a base for controller design.

6.7.2 Architecture co-design example

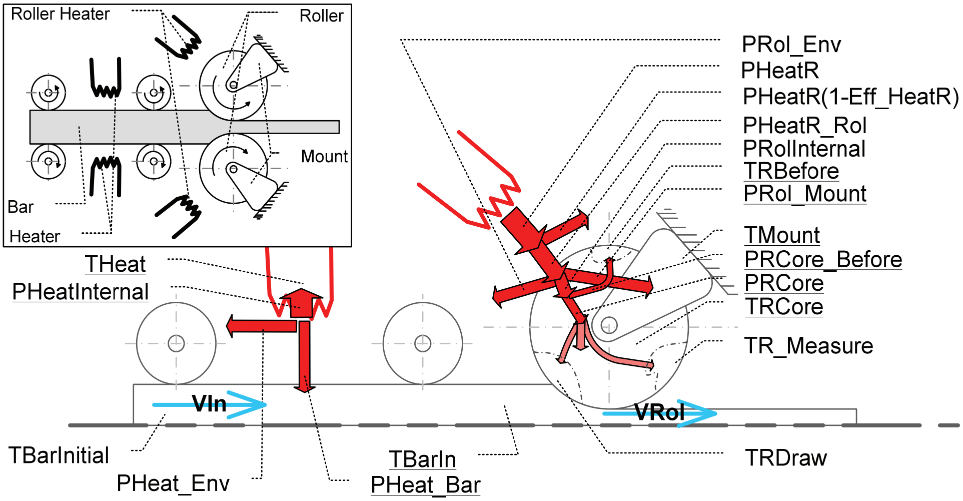


Figure 53. Schema showing the main components of the wire-drawing machine (top left) as well as the main parameters describing the heat flow. The thick arrows depict the heat flow.

The case study shown in this section concerns the structural property of sensor-actuator configurations (also known as Input/Output configurations in control theory [180]) of a simplified wire drawing machine. It is required that the changing temperature of the environment does not affect the quality of the production. The basic parts and the main parameters describing the heat flow in the machine are presented in Figure 53. The heater takes care of warming up the metal bar to be drawn, while the roller heater warms-up the drawing rollers. Some heat flows are modeled based on first principle equations (see underlined parameters in Figure 53), i.e., using differential equations to describe the dynamics. Other parameters are modeled with mathematical relations based on experimental data. The heat power transferred between the drawing rollers and the metal bar is not modeled. The original physical decomposition and phenomena that explain the different flows are modeled in SA-CAD using several entities and expressions, as shown in Figure 54. As explained in the section "parameter network and system structure", for the purpose of our analysis, the expressions relating parameters to each other (*cf.* Figure 54,) only require capturing the concept of causal relations which enable constructing our parameter network. More detail of the case study will be explained when necessary to illustrate the co-design process. Other details of the case study, which do not contribute to such explanations, are omitted to maintain the focus.

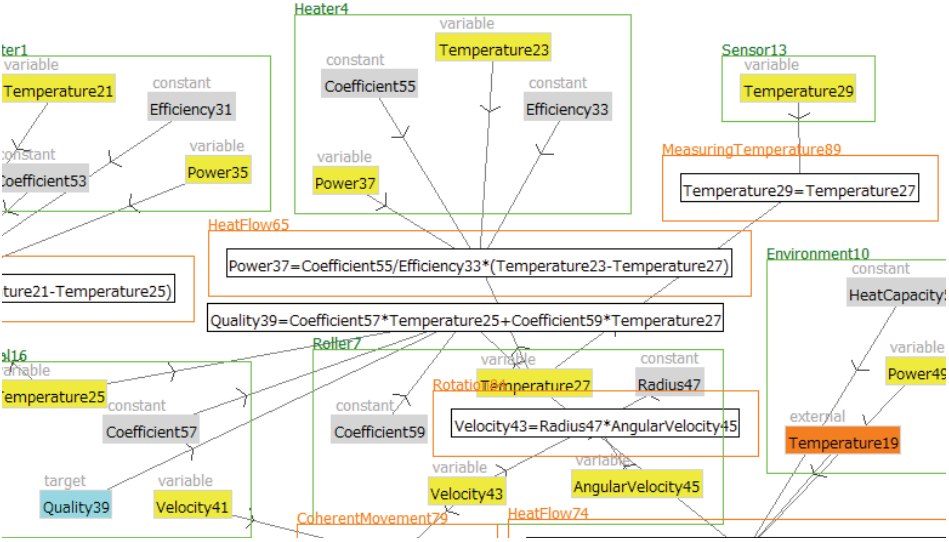


Figure 54. Fragment of physical decomposition model made in KIEF-SA-CAD

The SA-CAD model is then parsed and loaded as part of an AM model. As can be seen in Figure 55 and , SA-CAD entities are assigned to AM entity objects, SA-CAD expressions are assigned to AM-formula objects, and SA-CAD parameters are assigned to AM-parameters. The mappings among the objects are preserved. The AM tool is used to further expand the model by adding functions and requirements that indicate those parameters which are of interest fro the drawing process (*cf.*

Figure 55). In this case, it is required to maintain the quality of the drawing process by estimating a ‘DrawQualEstimation’ parameter that has to be kept within certain values. Other parameters to keep under control are the speeds of bar feed ‘Vin’ and drawing ‘VRol’. Additionally, knowledge about an initial controller architecture is added by modeling a controller object, labeling its attached formulae nodes (i.e. behavior equations), and labeling parameters from the environment that act as disturbances (e.g., the environment temperature).

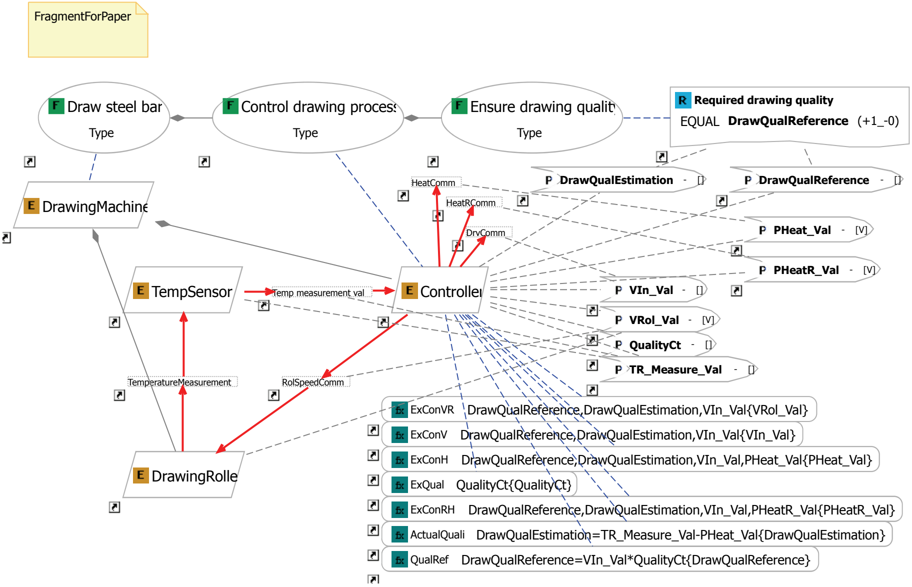

FragmentForPaper
F Draw steel bar
Type
F Control drawing process
Type
F Ensure drawing quality
Type
R Required drawing quality
EQUAL DrawQualReference (+1_-0)
E DrawingMachine
P DrawQualEstimation - []
P DrawQualReference - []
HeatComm
HeatRComm
DrvComm
P PHeat_Val - [V]
P PHeatR_Val - [V]
E TempSensor
Temp measurement val
E Controller
P VIn_Val - []
P VRol_Val - [V]
P QualityCt - []
P TR_Measure_Val - []
TemperatureMeasurement
RolSpeedComm
E DrawingRoller
fx ExConVR DrawQualReference,DrawQualEstimation,VIn_Val{VRol_Val}
fx ExConV DrawQualReference,DrawQualEstimation,VIn_Val{VIn_Val}
fx ExConH DrawQualReference,DrawQualEstimation,VIn_Val,PHeat_Val{PHeat_Val}
fx ExQual QualityCt{QualityCt}
fx ExConRH DrawQualReference,DrawQualEstimation,VIn_Val,PHeatR_Val{PHeatR_Val}
fx ActualQuali DrawQualEstimation=TR_Measure_Val-PHeat_Val{DrawQualEstimation}
fx QualRef DrawQualReference=VIn_Val*QualityCt{DrawQualReference}

Figure 55. Fragment of AM with additional requirements and control information

The resulting AM is then parsed by the external tool to perform the LSS analysis. The first step includes an interpretation of the parameter network (see Figure 56), and its subsequent reduction to an LSS graph (*cf.* Figure 57).

The resulting LSS graph (*cf.* Figure 57) contains the state variables and derivatives corresponding to the parameters modeled using first principles (see Figure 53). The graphical representation of this simple system allows seeing some of its characteristics at a glance. The state variables form two independent clusters because, as explained earlier, the modeling equations do not consider the heat transfer interaction between the drawing rollers and the metal bar. Variables from both clusters form loops. The LSS does not contain one the specified control inputs, ‘VRol_Val’, corresponding to the speed of the drawing rollers. This happens because ‘VRol_Val’ does not affect any parameters corresponding to state variables, as it can be seen in Figure 56. All the outputs and the measurement depend on a single state variable from ‘cluster1’, ‘TRBefore’, corresponding to the temperature of the roller at a point before heating, because all the outputs are actually values computed from the single measurement of such state variable.

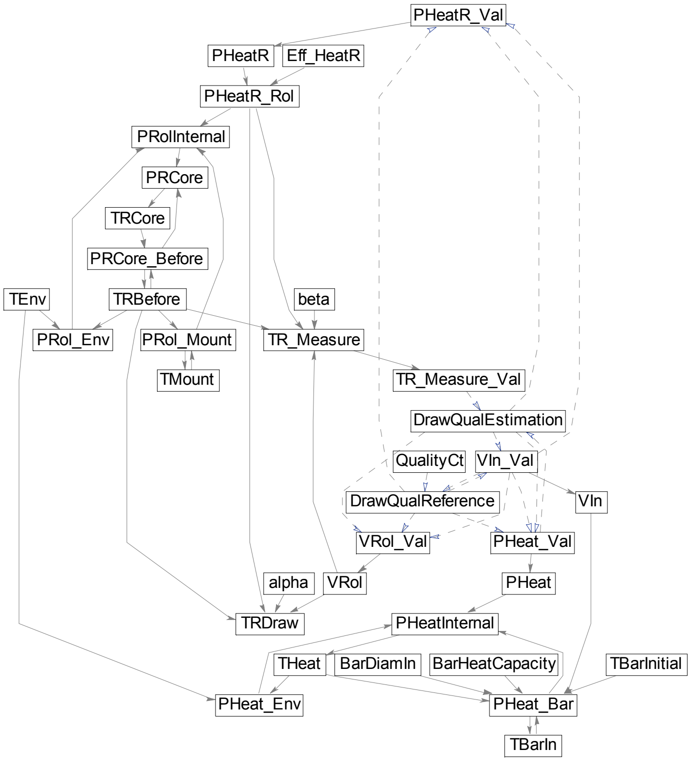


Figure 56. Parameter network extracted from the AM. Controller edges are dashed.

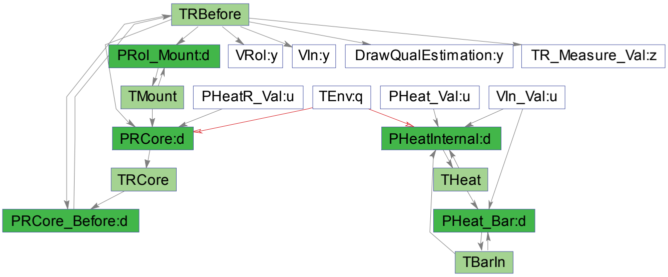


Figure 57. LSS graph depicting the state variables, their derivatives (:d), disturbances (:q), measured parameters (:z), controlled outputs (:y), and manipulated parameters (:u). Left state variable cluster is 'cluster1', and the right one is 'cluster2'.

Running the LSS analysis it is possible to verify that the system is generically controllable, but that it is not generically observable. This can be intuitively verified in Figure 57, as the inputs are able to affect all the state variables, and as the only measurement just captures the information from one of the clusters ('cluster1'). From the side of disturbance rejection, the result indicates that it cannot be achieved unless the value of the disturbance is known a priori, that is, measured directly. It can be seen that the disturbance directly affects the same state variable as it does the control input in 'cluster1', and therefore, if the disturbance is not known, the controller has to wait until the disturbance reaches the measurement and the outputs before it can act through the 'PHeatR_Val' input.

Another model of the drawing machine is built following the same procedure as before, to increase the physical insight in the analysis. This time, the physical phenomena models are modified to include first-principles descriptions for the measured temperature 'TR_Measure' and the temperature of the drawing roller at the drawing point 'TRDraw'. This resulting LSS graph is shown in Figure 58. This illustrates how changes in the model of the physical plant (or our modeling assumptions) can affect the design of the controller.

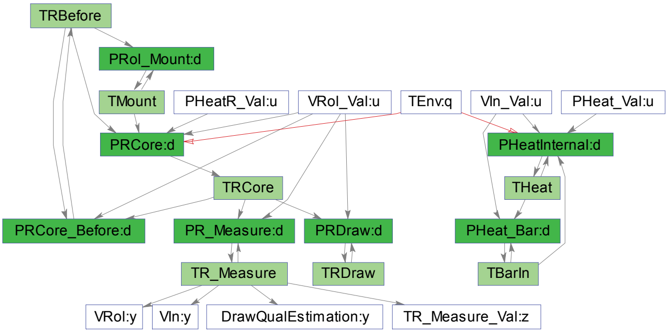


Figure 58. LSS graph of the modified model where some experimental relations are replaced by first-principles explanations. Conventions are as in Figure 57.

The new LSS shows again state variable structure forming two clusters. The new model shows the influence of 'VRol_Val' on the temperatures. Also, the outputs and measurement are related to the new state variable corresponding to the measured temperature value. The results of the LSS analysis for observability and controllability are the same. The result for disturbance rejection is the same, i.e., disturbance rejection is not generically possible, but analysis differs. The system suggests to measure of 'TRCore' to obtain the generic disturbance rejection property, so direct knowledge of the disturbance is not required.

An obvious change from the physical decomposition point of view is the existence of sensors. Both models suggest that taking different measurements is required to obtain outputs which are completely resilient to the changes in the temperature of the environment.

As the example shows, changes in the physical architecture (model) can be reflected immediately in the control analysis using the proposed method. For example, the actuators could be merged to have a single heater that radiates over the metal bar and the drawing rollers, or to have the drawing rollers alone moving the bar. For the sake of brevity, such changes will not be explored in this work.

6.8 Conclusions

This chapter explains and shows through examples how to capture relevant information for the regulatory control structure definition problem in an architecture-level model that can be used by other design stakeholders.

The proposed approach aims to improve communication practices for design information by allowing each stakeholder to contribute with the information from his/her field of expertise to a common model that, in this case, can be used by the control engineer to perform design activities early in the product development process.

Specifically, it is also explained how information in the AM can be automatically interpreted by an external tool, and proposed a method to transform that information into a structural (LSS) representation. The transformation and mapping does not serve only the practical purpose of automation, but also shows how information from a specific domain maps to a model representing the interests of multiple stakeholders. That information is highly valuable for the purpose of development process understanding, improvement, and management.

The LSS representation is used to verify the existence of important control properties in the system under study, such as controllability and observability, using existing methods from literature. Extensions to the analysis of the LSS representation are proposed and tested, which allow making suggestions for the solution to the regulatory control configuration problem. A brief interpretation of the LSS analysis is extended to advice on the selection of control configuration elements. The proposed extensions are based on the authors' experiences and preliminary tests, and no formal theoretical support is provided in the present chapter. Future work should address formalization of such proposals.

A method for co-design of physical and control architectures of mechatronic systems at the early stage of product development is proposed. Section 6.7 explains and demonstrates through an example how the proposed method of explicit modeling of system relations can be used to perform co-design in two different domains. The method is supported by an implementation on a set of tools that allow modeling the physical architecture and the control architecture, as well as analyzing data extracted for such models to support the co-design activities. Future work includes verification of the proposed method in industrial cases, and possibly, further integration of the tools on which the method has been implemented.

The performed analysis used in this work is based on parametric information. This information does not depend exclusively to any domain, but rather can represent information from all domains. In this specific case, the analysis can be used to support both, control stakeholders and physical decomposition stakeholders. Furthermore, the work shows that the used analysis technique effectively supports design activities from both disciplines, and retains an intuitive level that makes it generally applicable.

7 Supervisory control structure

This chapter revolves around the design of supervisory controllers. The goal of supervisory control is to ensure a designed process behavior within constraints (e.g., safety or cost), embedding the process rules. It can be built hierarchically ([15], [150]), reflecting the structure of the processes it coordinates. The supervisory task is separated from the regulatory task by defining the regulatory task only as directly ensuring that a controlled variable or parameter remains close to a desired reference value, in the context of a single set of continuous system dynamics declared as a state or operation mode. The supervisor can take care of providing the reference signals for the regulatory controllers, but in many cases the references are computed by a separate optimization layer [150], or internally by the regulatory controllers. The supervisory controller structure is a representation of the hierarchy and process rules mentioned before. In resume, the supervisory task involves deciding which actions (including regulatory actions) should be activated according to a well defined system state to arrive to another state.

This work proposes formal transformations from an AM specification to other models used for supervisory controller design. The target models were chosen because of their potential applicability using information available from the first stages of controller design. Heemels *et al* note [84] that not many design methodologies for hybrid systems controllers exist. The chosen transformations support a proposed design process flow which constitutes another contribution in this chapter. In addition, these transformations impose additional constraints to the syntax of the function layer of the AM model (*cf.* Section 5.2), which appear to ensure an unambiguous and complete specification of the desired behavior.

This chapter is structured as follows: Section 7.1 documents the analysis that led to the chosen target models for controller design. Section 7.2 walks through the process of building a correct problem and design specification using the AM. Section 7.3 has proposals for algorithms to perform the transformation from the AM to the chosen target models. Some conclusions are presented at the end of the chapter.

7.1 Supervisory and hybrid control methodologies

The goal of this section is to highlight the main characteristics of a few methods that cover a wide spectrum of representations and techniques related to for supervisory controller design available in literature, and not to perform an exhaustive review of literature (for that see [84]). Based on such review, target models for which to develop transformations from the AM are chosen. The methods in literature cover different aspects related to the control of mechatronic systems (which for us implies the need for hybrid system models). Our interest falls over the techniques that present a combination of the following characteristics:

- Formal representation
- Controller verification
- Controller synthesis
- Available implementation

The next subsections resume the reviewed methods, and section 7.1.5 discuses on the selection of appropriate target models.

7.1.1 Hybrid automatons

From a point of view, many systems can be seen as continuously evolving on a certain mode, and when certain conditions are met, it changes to another continuous mode. Such a view can be modeled by a mixture of differential equations for the continuous modes and an automaton ruling the transition among modes. Though different formal descriptions exist, this work considers the one proposed by Tabuada in [158] as a sextuple consisting of: (1) a set of states; (2) a set of initial states; (3) a set of inputs (events); (4) a transition relation which determines the change between states under a certain input; (5) a set of outputs; (6) an output map which determines the output that corresponds to each state.

Though many analysis techniques simply simulate the model [114], Tabuada [158] also covers a rather popular view of controller synthesis and analysis when hybrid automata are chosen for representation. The techniques under this view mainly use the concept of bisimulation. Bisimulation is based on the principle that some system models can be found to have equivalent states and events. Then, this equivalence is extended to other properties of the system models such as being non-blocking. Thus, design parts from a description of the system and a description of the behavior specification in terms of automata which will be later compared and analyzed for controller synthesis.

7.1.2 Behavior-based control

A fast way of defining behavior based control is to compare it to other controller architectures as done by Mataric ([109], [110]), who distinguishes three types of control architectures with different characteristics:

- Purely reactive: Direct coupling between sensors and actions. No world model. Look-up for each set of sensor readings.
- Planned: Centralized world model to verify sensory information. This includes mixing of planed actions for decision making and reactive actions for low level behavior (hybrid architecture).
- Behavior based (in the middle): A collection of concurrent behaviors without a central arbiter that decides when to execute them. A similar definition, called subsumption architecture, is provided as well by Brooks [32].

From the applications described in literature, this methodology poses as a good candidate for problems including control in unstructured environments. The paper [122] presents an architecture which allows specifying sequences of actions in addition over other. The lowest level goals take care of basic safety and motion tasks such as avoiding obstacles and moving ahead. These low level tasks are used or inhibited to perform more complex tasks.

A global design strategy for is defined in three steps: (1) specify behaviors as the goals of the system; (2) break-down behaviors into observable disjoint actions that serve as sub-goals which reduce the current distance of the system to the goal; (3) define the actions in terms or system actuation, in which the change in state (actuation) must be smaller when the required precision is high. However, no algorithmic methods seem to be available for controller synthesis, as the research in this matter is mainly aimed at learning systems.

7.1.3 State-tree structures

Discrete-event models consider that the system can be represented by a discrete set of states and transitions that take place at discrete points in time (events) [38]. Cassandras and Lafortune [38] have documented some of the best known design methods associated with these models, including some algorithm developed after the work of Ramadge and Wonham [136]. The basis of such techniques is the analysis of the model topology using graph-based techniques which are used to design supervisory controllers by verifying the (co)reachability of states. A recent approach to these methods developed by Ma *et al* [106] called State-Tree Structures (STS) is discussed next.

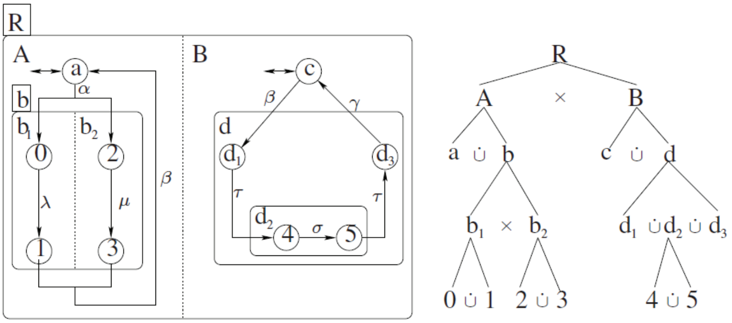


Figure 59. STS model (left) and its state tree (right) [106]

STS is based on the work of Ramadge and Wonham, but additionally proposes a leaner representation of the state space. The model (see Figure 59) uses a hierarchical and concurrent representation of the states of a system through the combination of state trees (which resume the possible states of the system) and the corresponding automata representing local behavior, denoted “holons”. Starting on a root state (“R” in the figure) the state tree (Figure 59 right) is composed of nodes representing either composite parallel/serial superstates (AND/OR, indicated respectively by $\times / \dot{\cup}$) and simple states which do not decompose (the leafs of the tree). The only constraints to construct correct state trees are that it should be a tree, and that the simple states should be components (children) of OR superstates [106]. This last constraint just reflects that “AND sub-trees” (a sub-tree containing just AND superstates) can be assimilated as simple states related to many parameters or as additional descriptions of other OR states. The leafs of the tree represent partial state descriptions, and the complete state of the system is defined by having one “active” simple state node for each OR superstate at the top (wide end) of the tree. A holon is declared for each OR super-state (a state composed of mutually exclusive states). In other words, a holon represents the transition structure of sub-tree rooted on an OR superstate (*cf.* Figure 59 left). The holon contains nodes which represent the states and directed edges representing transitions between the simple states under a certain event. Events (and their associated transitions) are either controllable (depending on the output of the controller) or uncontrollable (only depending on the plant). The control functions resulting from the synthesis literally specify the states that should correspond to enabling the controllable events in order to obtain the desired behavior. We do not detail about additional constraints related to the events as these can be obtained by construction or circumvented through event renaming [106]. Additionally, it is worth mentioning that Ma has developed a tool available to the public (STSLib), which allows reading STS descriptions in a certain format and provides the basic rules of the synthesized controller.

7.1.4 Hybrid programs

The research visited in this section again considers the “modes and events” point of view introduced in section 7.1.1. However, in this case the approach for representation and, foremost, for design is based on logical analysis and theorem proving. This relatively new work by Platzer is called logical analysis of hybrid programs [132].

Systems are modeled using the Hybrid Program (HP) language (called “differential dynamic logic”) which can be used to represent discrete and continuous dynamics, aggregated as sets. These in turn can be composed with a “control structure” using other operators. The resulting model looks much like a computer program, thus the hybrid programs can represent hybrid systems directly. Hybrid programs have been designed to formally prove system properties that can be modeled via logic formulae of the type $\psi \rightarrow \gamma$, where ψ is a starting precondition for execution (or guard) and the formula γ can take a rich structure. Platzer presents formulae γ that includes state dependent modes of operation “$[.]\varphi$” and “$\langle . \rangle \varphi$”, temporal operators “□” and “◇”, and universal “$\forall$” and existential “$\exists$” quantifiers. Below are a number of examples with a corresponding interpretation [132]:

- $\psi \rightarrow [\alpha]\varphi$, represents that ψ holding implies that all states reachable by system α satisfy formula φ.

- $\psi \to \langle\alpha\rangle\varphi$, for the post condition, there is at least one state reachable by α that satisfy formula φ.
- $\psi \to [\alpha]\,\Box\,\varphi$, formula φ is satisfied for every point of the trace of every run of system α.
- $\psi \to [\alpha]\,\Diamond\,\varphi$, formula φ is satisfied for at least one point of the trace of every run of system α.
- $\psi \to \exists \mathrm{p}[\alpha]\langle\beta\rangle\varphi$, there exists a parameter p such that for all possible behaviors of system α there is an action of system β such that φ holds.

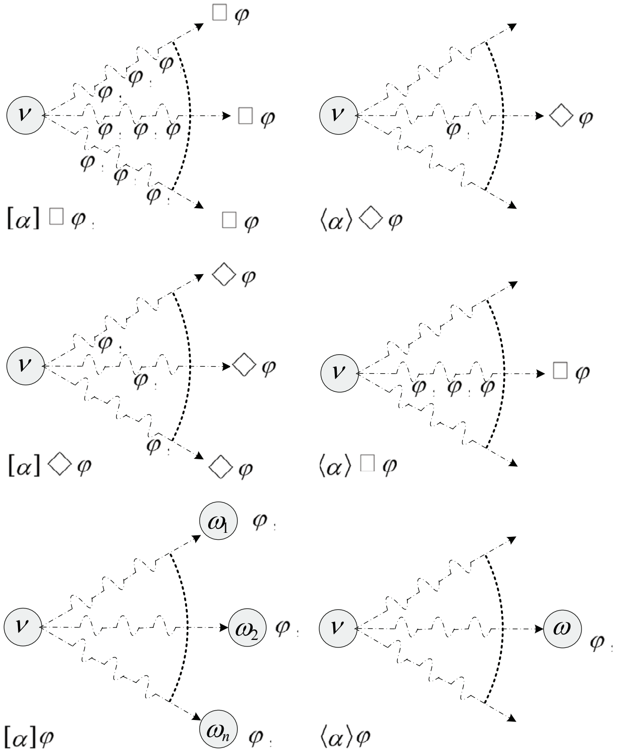


Figure 60. Graphical representation of traces in hybrid programs (adapted from [132])

Figure 60 illustrates various combinations of these operators, where each jagged path leaving (arriving to) the state ν (ω) represents a trace of evolving α. The process of proving the formula $\psi \to \gamma$, may simply lead to verify its validity, but may also provide conditions (value ranges of parameters) under which the formulae remains valid. This is very interesting from the point of view of control synthesis and requirements refinement. Therefore, Platzer suggests the following design approach called iterative refinement process:

- Find a controllable state region by symbolic decomposition of the uncontrolled system dynamics.
- Successively add partial control laws and again use symbolic decomposition to add parametric constraints. Repeat this step until the system can be proven safe.

- Proof liveness by demonstrating that the system is not overconstrained.

At this point is also worth noting that Platzer has developed a software implementation of the logical analysis algorithms, called KeyMaera, and that this tool is available to the public.

7.1.5 Analysis and choice of target methodologies

The previously reviewed approaches provide interesting possibilities. By looking at their characteristics and their potential to fit our purposes one can highlight that:

- The trend for design using hybrid automata is tightly related to the concept of bisimulation, by which the hybrid system is approximated to a finite-state system for which analysis techniques can be applied and then extrapolated to get and/or refine a controller for the hybrid system.
- Behavior based control approaches do not provide a clear set of formal analytical methods which enable proving properties of the controlled system. However, they may be good candidates for synthesis of controllers using other techniques which relate more to artificial intelligence, such as learning algorithms. Also it is worth noting that architectures like the one presented in [122] start resembling hybrid automatons of hierarchical controllers with an intense communication among the controller components.
- STS provides powerful design techniques which have been implemented in a tool allowing synthesis of non-blocking supervisors. However, the methods do not address most concerns related to the hybrid nature of systems.
- Hybrid program techniques provide allow verifying properties of a hybrid model and refining transition conditions, and also have been made available in a tool. However, the potential for synthesis of supervisors is not apparent.

From there, it is possible to arrive to some conclusions discussed next. The line of thought related to hybrid automata is tightly dependent on much numeric analysis and information, thus drifting apart from our purpose to support controller design from the first stages of the development process. Behavior-based control methods lack much of the necessary formalism for automatic generation. Although the STS and HP techniques do not provide complete solutions, they seem to complement each other. Therefore, here it is chosen to implement transformations which support design process described in Figure 61. The proposed design process starts with an AM that can be transformed to an STS model from which a non-blocking supervisor can be synthesized. The result is placed back in the AM by adding the synthesized transition conditions (i.e. the control laws). The model then requires a more specific description of the system dynamics so that it can be transformed to the HP representation for further analysis and refinement of transition conditions. In other words, the idea is to integrate the chosen design methods by using the AM tool as a base model, under the premise that the AM allows representing multidisciplinary design information in an orderly fashion. It must be added that the tools provided by the authors of the two chosen methods also present an advantage to implement the proposed approach.

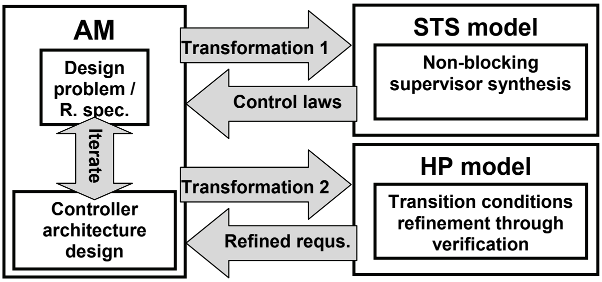

AM
Design problem / R. spec.
Iterate
Controller architecture design
Transformation 1
Control laws
Transformation 2
Refined requs.
STS model
Non-blocking supervisor synthesis
HP model
Transition conditions refinement through verification

Figure 61. Proposed design process for supervisory control generation

The reader may notice that Petri net methods have not been considered in this small review. Petri nets are not considered here because though they provide a good set of analysis algorithms [119], supervisory control synthesis based on Petri nets must still overcome fundamental challenges. As explained by Cassandras and Lafortune in [38], Petri nets constitute a more powerful modeling formalism than finite-state automata but do not pose as good candidates for extending the algorithmic component of supervisory control theory to non-regular languages. Thus, automaton-based methods (i.e., STS) are considered here to be a valid equivalent subset of Petri nets for which synthesis is more clearly defined.

7.2 Modeling supervisory control architecture

In general terms, the supervisory control architecture is part of the system architecture and can be modeled as presented in Chapter 3.Furthermore, specifically for the subject of interest of this chapter, it is pointed out that other authors [45] also report that techniques for analysis and control of hybrid systems are quite limited at present and that many important problems, including modeling problems, remain to be addressed. The questions introduced in Figure 14 (pp. 46), “what will the system do?, “how will the system do what it is required?”, and “How is the system designed and by whom?”, have to be addressed for the particular case of supervisory control.

7.2.1 Basic information and modeling objects

“What” the system aims to achieve is described as a goal in terms of an abstract state, which can be made concrete by adding parametric information. The state itself is represented by means of a requirement node (since it is a required goal) which specifies (either directly or through its requirement sub-tree) a set of parameters and constraints over the values of such parameters. The intended behavior of a system can unambiguously be represented by a sequence of transitions (triggered by events) between a set of states, modeled through ‘function relations’ and their related requirements. When a function reaches a goal, this implies a change in the parameter values which entails a change of state in the system or its perceived environment. Therefore, starting from a specific state, a new reached state can be determined.

"How" goals are achieved is specified by represent the things that implement the functionality and how they are related. Concrete representation of the entities is achieved when their properties are unambiguously described by parameters and their values. The objective description of behavior is achieved through the use of formulae and their associated structure. Each formula specifies a relation among specific parameters, making the description concrete. The language for writing such relations is not specified in our proposal. However, in this chapter it corresponds to expressions used in the differential temporal dynamic logic (dTL) [132].

More specifically, besides the previous description, the supervisory control architecture mainly involves deciding which parameters and events are visible to a controller or directly affected by a controller. This also includes deciding how controllers relate to each other through events or parameters. In normal conditions, a supervisor must be able to identify all relevant system states through its inputs, and affect all relevant system parameters in order to achieve the desired sequence of states in the system. Thus, for the discussions we assume that each particular supervisor is responsible for an observable and controllable system or subsystem.

Turning again our attention to Figure 14, the question regarding how the system is designed remains to be addressed. As explained in Section 3.3.5.1, in this chapter also aspects and domain entities related to the design of supervisory control are defined. Specific examples are given in the following sections.

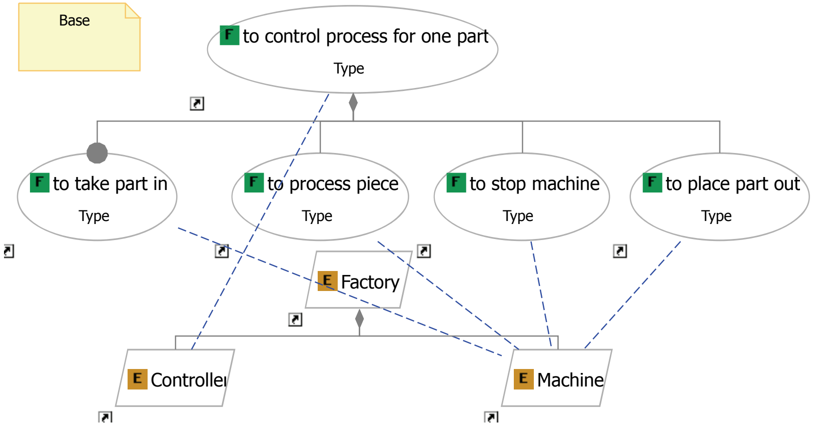


Figure 62. Basic architecture for a small factory

7.2.2 Modeling architecture

The aspect-based modeling scheme introduced in Figure 24 (pp. 56) is used as a reference to build the AM in this use case. The first step to model the architecture as stated above requires specifying the system components as entity nodes and the things these components do as function nodes. Both object types can be decomposed to add more detail to the architectural description. In step two, links are added between the two object types to show how functions and entities relate. The resulting model is shown in Figure 62. As shown next, more detail can be added to the model, following steps three through six in Figure 24, either from the side of the entities or from the side of the functions.

As part of step three, entity relations are added to further detail the structure of the system. In our example, we realize how the system also relates to the environment, and add this to the model. At this point, the model contains mostly subjective information. In step four, objective information is added by introducing some parameters that can be used to characterize the state of the system. The parameters must be mapped to the corresponding entities and entity relations. The resulting structure is depicted in Figure 63. The parameters indicate the number of pieces sensed at the machine (“Load”), the current status of the processing (“Progress” and “Speed”), and the control command (“LoadCommand”) used to force a new part in the line. Other parameters of interest for the behavior include the capacities of the physical systems represented by the entities. It can be seen how the model unambiguously describes, for example, that the loading command is handled by the machine once the controller has set it. Specifying such details in text form requires detailed descriptions and good writing skills [7]. Continuing to step five on the side of entities, formulae can be added to represent the relations among parameters, as shown in Figure 63 (top-right corner). The modeling is completed on the structural side performing step six, by mapping the formulae to the entities and entity relations. Here, it is specified how the value of parameter “Progress” changes according to the value of the parameter of “Speed” of the process. The AM formalism does not place any particular restriction to the syntax used in the formulae with the exception that formulae can be interpreted as relations among parameters. More details regarding formatting of these formulae will be given in the section explaining the model transformations.

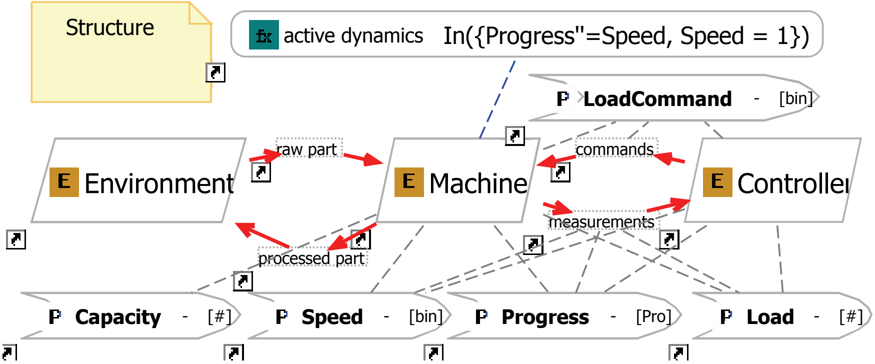


Figure 63. System structure and relevant parameters

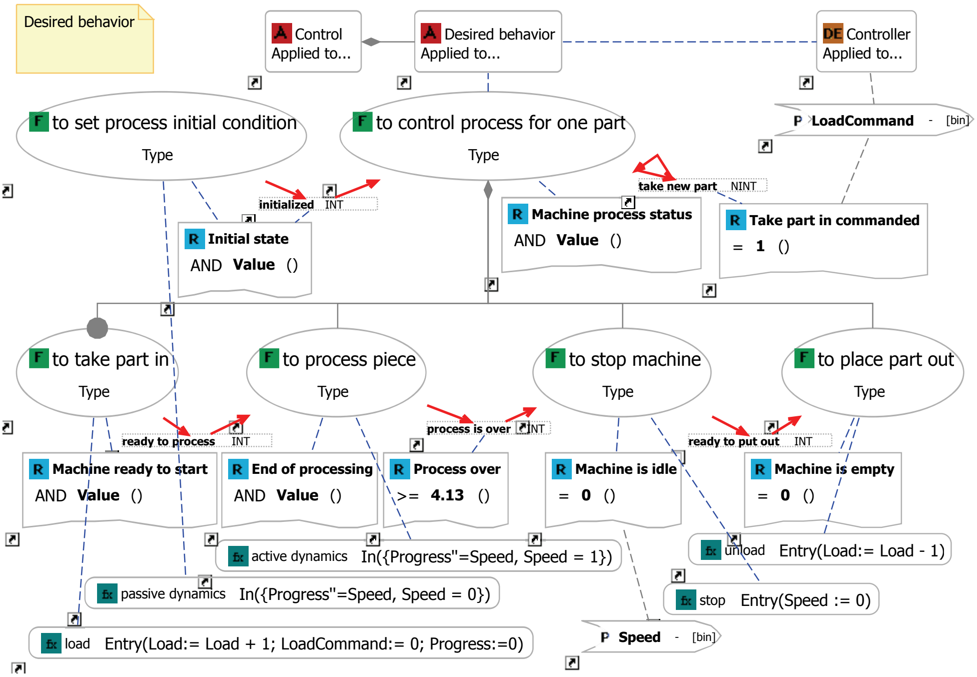

Desired behavior
A Control Applied to...
A Desired behavior Applied to...
DE Controller Applied to...
F to set process initial condition Type
F to control process for one part Type
P LoadCommand - [bin]
take new part NINT
initialized INT
R Initial state AND Value ()
R Machine process status AND Value ()
R Take part in commanded = 1 ()
F to take part in Type
F to process piece Type
F to stop machine Type
F to place part out Type
ready to process INT
process is over INT
ready to put out INT
R Machine ready to start AND Value ()
R End of processing AND Value ()
R Process over >= 4.13 ()
R Machine is idle = 0 ()
R Machine is empty = 0 ()
fx active dynamics In({Progress"=Speed, Speed = 1})
fx unload Entry(Load:= Load - 1)
fx passive dynamics In({Progress"=Speed, Speed = 0})
fx stop Entry(Speed := 0)
fx load Entry(Load:= Load + 1; LoadCommand:= 0; Progress:=0)
P Speed - [bin]

Figure 64. Processing sequence for the small factory

Up to this point the modeling has focused in rather static descriptions from the system and what it does. However, the intended behavior of the system remains to be modeled. For that matter we carry out steps three to six from the side of functions. The functions are used to explain the order of the processes that the system must implement, together with requirements which will provide objective goal descriptions for the functions. As indicated at the start of this section, the mechanism for ordering the functions is to place function relations between them describing function execution precedence (step three form the side of functions in Figure 24). The parameters modeled in step four can be used together with requirements to specify objective descriptions of the functionality, completing step five. The state descriptions are just partial, because specifying the complete state of the system potentially requires defining value (ranges) for many parameters, which entails much modeling effort (later we explain how we deal with this issue). The label in the bottom-left corner of each requirement indicates either the type of constraint over a parameter (e.g., =, <, >) relative to the value specified in the label at the bottom-middle (possibly in terms of other parameters), or the type of relation among children requirements (i.e., OR, AND) for composite requirements. By mapping requirements to functions and function relations step six is carried and the modeling process is completed. The resulting diagram specifying the function of carrying the desired processing sequence (and the related formulae) is depicted in Figure 64. In this example, the (partial) state related to “stop machine” is specified in terms of parameter “Speed” by the requirement “machine is idle”. The behavior under study is specified by a main function node “to control process for one part”. The sub-tree of this node contains all the functions that have to be executed in order to carry out the process. The function “to set process initial conditions” ensures a well defined initial condition or state from which the behavior under study can be analyzed. To indicate which children function(s) should start executing when their parent is activated, it is possible to use the ‘start function’ connector represented by the gray circle on top of the function node in Figure 64.

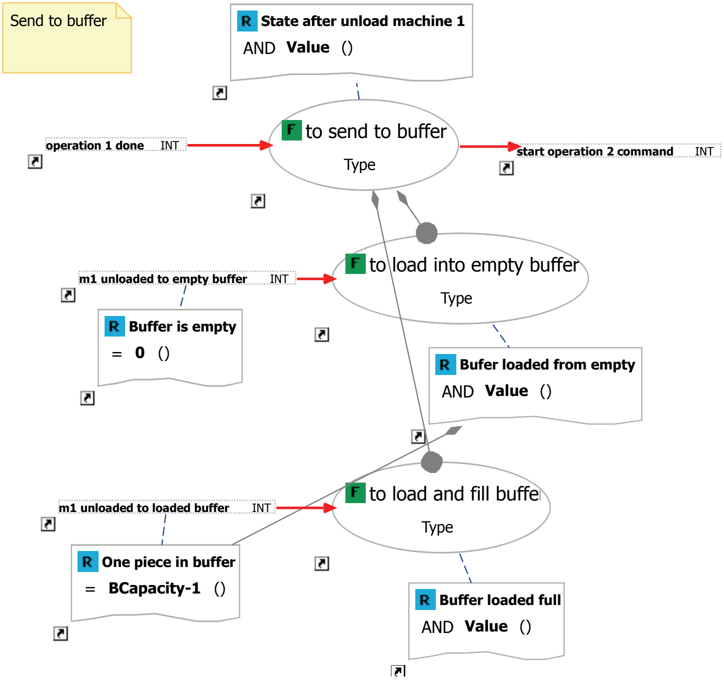


Figure 65. Use of open function relations as preconditions

Yet another way of using the function relations is to have them connected only from one end or ‘open’, i.e., connected only incoming/outgoing to/from a function node. Through this construct, it is possible to add detail to incoming/outgoing function relations associated to the parent of the present function, working effectively as an additional precondition for transition in/out of it. This can be used to represent a choice upon starting the execution of a composite function as shown for example in Figure 65 (this example does not correspond to the previous example), where the requirements associated as an event to the function relation “operation 1 done” are complemented by (united to) the requirements associated to the open input transitions “m1 unloaded to empty buffer” (requirement “buffer is empty”) or “m1 unloaded to loaded buffer” (requirement “one piece in buffer”). This solution is preferred over the use of a "dummy function" (which could not be mapped to any entity) preceding both children which would act as a choice point in the model.

As demonstrated by the works on formal language identification [57], specifying desired (allowable) and undesired (forbidden or constraining) situations helps reducing the detail of a description based on only one type of situations, as enumerating all the desired or undesired evolutions may be too cumbersome. From a more strict control perspective, desired behaviors partially specify the sets of allowed states the system can reach, while undesired behaviors specify the sets of states the system should not reach. Thus, we also model desired and undesired behaviors in the model by using several descriptions like the ones in Figure 64. Following this view, functions (together with requirements) can be used to represent system "invariants", or conditions (partial states) that the system should never reach or that should always comply to (in the case of desired behaviors).

The complete state space is represented in a compact way by using the hierarchical structure of requirements (*cf.* Figure 66) in the same manner as done by the state tree in section 7.1.3. In this way, the full state space of a system can be represented very economically. Requirements in the STS can also be used as part of the description of other requirements (see Figure 66). The modeled requirements can be used efficiently to link the subjective functional description to objective goals, states, and events. The main function node maps to a main requirement node which is parent to all the requirements that can be used to specify the behavior under study. For our example, the main requirement node is "machine process status" (see Figure 64 and Figure 66)

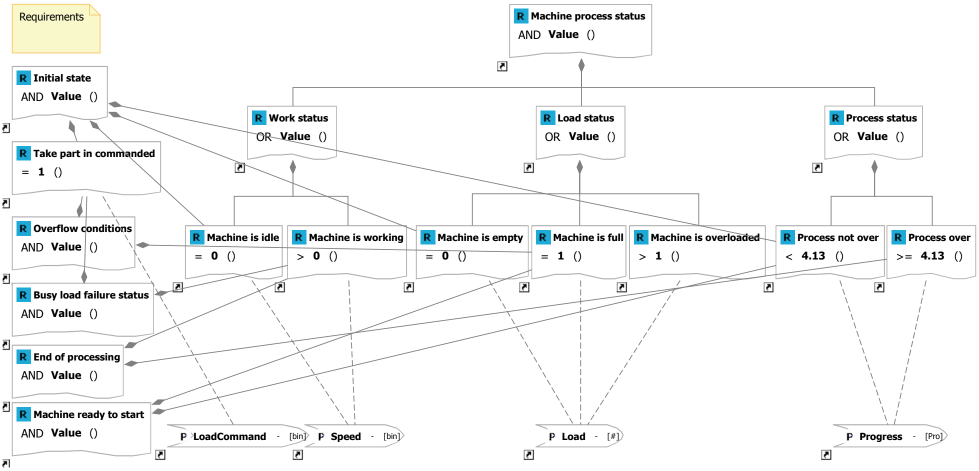


Figure 66. State-tree structure for the processing sequence (tree on left) and other requirements (lined up at right). Mappings to parameters not shown.

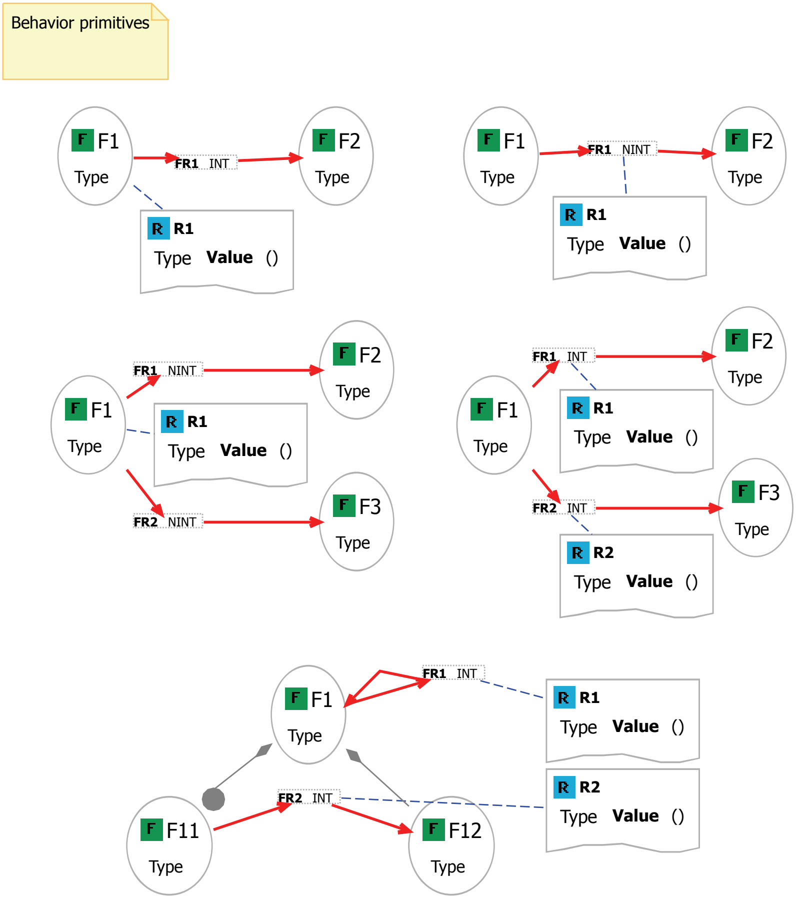


Figure 67. Some possible transition structures used to model intended behavior: (a) interrupting transition after R1; (b) non-interrupting transition on R1; (c) non-interrupting synchronized transition on R1; (d) interrupting transition choice on R1 or R2; (e) "reset" of F1 on R1. Note that the goal requirements of functions are only show for F1 in a and c to simplify this view, but other requirement goals still should be present in the model.

Now follows a more detailed explanation of how function relations can be used together with requirements to precisely indicate transition conditions and the effects of such transitions, including parallelism and choice. Some of the possible transition structures for the model are represented in Figure 67. The combinations come from representing a transition under an event:

- associated with a goal requirement of the function preceding a function relation (as in Figure 67 a and c), or associated with a separate requirement (as in Figure 67 b, d, and e)
- defined as interrupting ("INT" in Figure 67 a, d and e) or non interrupting ("NINT" in Figure 67 b and c)

If several function relations are associated with the same requirement node, then the activation of the subsequent functionality will be synchronized by the event (*cf.* Figure 67 c). Association with different requirements implies a choice (see Figure 67 d). Additionally, we allow placing a function relation as a self-loop of a function (Figure 67 e). The multiple descriptions covering desired and undesired behaviors and the non interrupting transitions allow specifying concurrency in the behavior.

It is necessary to ensure consistency as the different objects are decomposed. For the purpose of the transformations proposed in this paper is particularly important that the events involved in transitions between functions at different decomposition levels are not conflicting, i.e., do not include conflicting requirements with constraints such as “A>0” and “A<0”. More generally, a parameter may be constrained by several goal requirements of functions. To decide on the possible range of values that it can take, simply determine it from the different requirements. To be consistent, no two requirements can constrain it to be equal to more than one value; neither can two requirements point it to be both greater and smaller than a certain value.

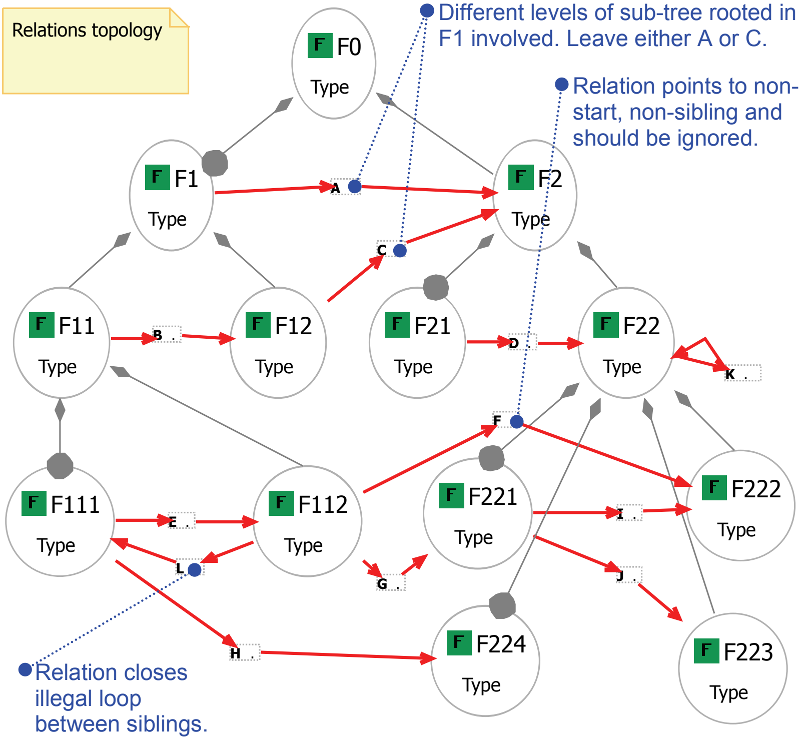

Relations topology
Different levels of sub-tree rooted in F1 involved. Leave either A or C.
Relation points to non-start, non-sibling and should be ignored.
Relation closes illegal loop between siblings.
F0
F1
F2
F11
F12
F21
F22
F111
F112
F221
F222
F223
F224
Type
A
B
C
D
E
F
G
H
I
J
K
L

Figure 68. Relation topologies in the AM. Conflicts are indicated with notes and dotted lines.

Considering the language tradeoffs established before, we introduce some additional constraints to the modeling. To simplify parsing the model part representing the desired behavior, we do not allow placing function relations which form loops between children of the same function. This facilitates identifying the "end sub-functions" among the children of a single function and entails more readable graphical models. Another simplifying modeling constraint involves disallowing to place (function and entity) relations among different "detail levels" in a hierarchy. This corresponds to only allowing relations among children of the same parent (siblings), and among non-siblings under the condition that no two nodes in the same decomposition sub-tree at different levels relate to elements in the same level of another sub-tree. Examples of these constraints are depicted in Figure 68. Similar rules also apply for entities, but detailing this part is out of the current scope. Additionally for the functions, there must be consistency with relation to the labeled start functions.

7.3 Model transformations

After presenting all the necessary information to model using the AM towards designing a supervisory control, we proceed to detail how this information corresponds to the one in the chosen target models: STS and HP. For both transformations we first make a better description of some points in the target models and the related design methods. For complete descriptions the reader can consult the original works of Ma [106] for STS and Platzer [132] for HP. Then, we explain the mappings and transformation process starting from an AM. For space reasons, we emphasize on the description of the transformations and mappings rather than showing the detailed development for our example case.

7.3.1 AM-STS mappings

Recalling Section 7.1.3, we can identify two main parts in the STS model: the state tree and the holon. Though in fact the holon contains all the information from a certain state tree, we treat them separately to facilitate identifying the correspondence to the information in the AM. The synthesis problem takes as inputs the STS model and the list of states we do not want to the system to reach. The result is a set of control functions or rules, specified as binary decision diagrams (Boolean functions). Once available, these results can be directly represented through the requirements corresponding to the controllable transitions in the AM.

Though represented graphically in Figure 59, the STS model is serialized as two formatted text files for the STSLib software which describe the system ("sts") and the control problem, called the logical specification ("spec"). These forms provide additional information. The spec file contains lists of undesired behaviors and illegal states which are used to pose the control problem. The sts file contains a description of the state tree, the holons, the initial state, the marker states, and the memories. The last two terms are further explained here. Marker states reflect important states of the system (e.g., marking the state of completion of a task), and are relevant because from them depend the results of reachability and coreachability tests used in the methodology. Normally holons are constructed to represent the full or uncontrolled behavior of the system; memories are simply holons which are interpreted as specifying fully desired or controlled behavior of the system (i.e., specification automata in [136]), entailing that all the unspecified transitions will be considered as part of the undesired behavior.

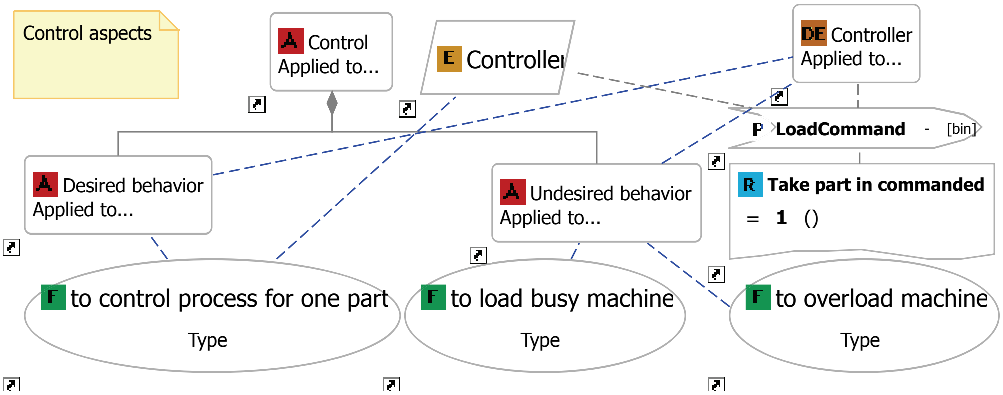


Figure 69. Labeling of AM information for the proposed transformations

Passing to the AM-STS correspondence, first we define the aspects to label the AM information relevant to the transformation. That is, we declare aspects which map directly to the top functions specifying desired and undesired behavior. From these function nodes the rest of the information can be identified in the model. One possible way of modeling this in our example is depicted in Figure 69. The figure also shows how the domain entity "controller" (top-right), corresponding to the controller we intend synthesizing using our STS model, is mapped to the control aspects and at the same time to the parameters relevant to this end. This last mapping allows correctly identifying which states and events depend from the controller being synthesized. For this synthesis approach, we declare as controllable events those related with requirement nodes which map to the controllable parameters (requirement "take part in commanded"). Since observability is assumed in our STS synthesis, there is no need to identify the parameters related to sensing. It may seem like the choices for aspects in the example do not cover every possible situation, as for example, there could be other physical constraints relevant to supervisory control such as a machine capacity or a maximum carrying weight. A possible way to address those situations is to add more aspects which specifically label such conditions. For the example of the capacity of the plant components, the information could be labeled in the AM using an additional child aspect, which in turn maps to a domain entity and to the capacity parameters. Nonetheless, a better approach seems to be adding such conditions as undesired behavior descriptions. In such way, the aspects of desired and undesired behavior can cover all constraints over the behavior without the need for many aspects. In this case, the undesired situation of overloading the machine covers the plant capacity consideration.

An evident relation can be seen between the state tree in STS and the requirements tree which represents the system states in the AM. As shown in Figure 70, the requirements tree from Figure 66 can actually be directly parsed as a valid state tree if the constraints introduced in this section are respected, or it can be preprocessed to eliminate the "redundant" AND-component-simple states. For simplicity, the figure uses the initials in the names of the referred requirements (e.g., MW for "machine in working")

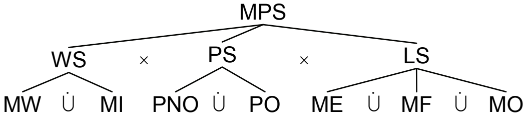


Figure 70. State tree for the factory example

We now describe how to form the transition structure in the holons. This is accomplished by matching the transition requirements associated to the function relations in the desired behavior descriptions to the transition events used in the holons. This can be done because the requirements mapped to the function nodes correspond to partial states in the state tree. Yet, one must be careful and consider the possible complete states of the system when a certain function is active so that a true trace of the system can be reconstructed. To do this and to correctly capture the modeled concurrency, we interpret the desired behavior specifications as Petri nets in which each function node is a place, and the function relations can be translated to several transitions according to their characteristics (see next paragraph). The capacity information can be added directly to the places of the resulting Petri net, effectively bounding and reducing the number of states it represents. The initial marking of the net is obtained from the requirement which sets initial conditions associated to the main function(s). The reachability graph of the Petri net contains all the possible system states and transitions. One possibility to obtain this reachability graph is to use transformation tools such as those developed within the Compositional Interchange Format project [46].

Figure 71 depicts the equivalent Petri net transitions for the transition structures presented before in Figure 67. The cases (a)-(e1) correspond directly, while the case (e2) in this figure corresponds to the situation in which the looping function relation "FR1" in (e1) is considered as non-interrupting. In the case in which FR1 in Figure 67-e corresponds to a transition after F1 stops executing (e.g., is not mapped to a requirement), the Petri net transition in Figure 71-e labeled R1 would correspond to the appropriate end-event of F1. Such event would probably be a combination of the end-events of the children of F1. Notice that some transitions in Figure 71 do not follow the standard Petri-net representation and are depicted with a hollow rectangle; here we call these 'OR transitions'. An OR transition fires with any (largest possible) marking of the places attached to its incoming arcs. If the OR transition is interrupting, it will have an outgoing arc to each start children function of F1 and will add a token to those places when firing. If the OR transition is non-interrupting, it will not consume the tokens of the incoming places (this has been represented in Figure 71 by placing additional outgoing arcs to F11 and F12). Therefore, and OR transition is in fact representing multiple transitions on the same event which cover all the possible markings of the places of the incoming arcs. The resulting Petri net for the desired behavior of the small factory example it shown in Figure 72 (names of nodes represented by their initials). The Petri net can represent multiple levels of functionality simultaneously by, for example, adding a place and corresponding links to transitions for the "control process for one part" function node. Only the most detailed level is presented in Figure 73 for simplicity.

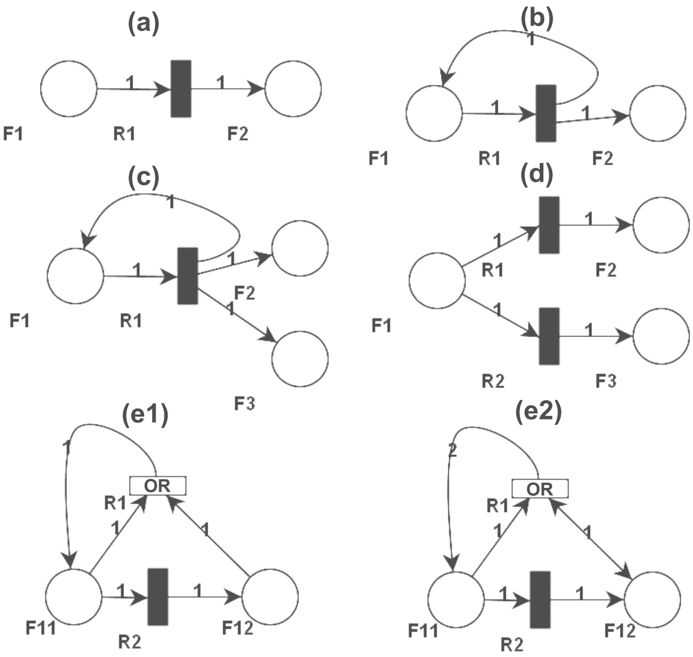


Figure 71. Equivalent Petri net transitions for function relations in Figure 67. The "white" transitions in (e1) and (e2) correspond to an OR transition. For simplicity, input/output transitions to each Petri net are not depicted (without them the nets are blocking).

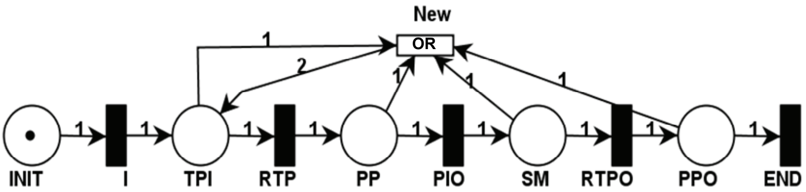


Figure 72. Resulting Petri net for the desired behavior in the small factory

The forbidden states to build the logical specification can be obtained from the undesired behavior specifications in a similar manner to the holon's transition structure, taking care of joining both reachability graphs and extracting the illegal states as the nodes of the undesired behavior specifications which do not mach nodes in the desired behavior. The concept of memories is not used in the proposed transformation, but it could also be included. Other sources of information for the forbidden states can be found on the capacities of the subsystems and the invariants related to undesired behavior (*cf.* section 7.2.2), as these directly indicate illegal states due to physical constraints.

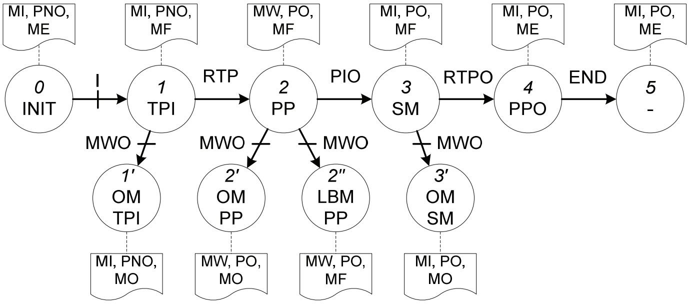


Figure 73. Resulting automaton for the factory example

The resulting reachability graph is shown in Figure 73 (again, just the initials of the referred nodes are used), where the numbered circles indicate the active functions in each state, with the attached symbols indicating the corresponding goal-state requirements. The transitions correspond to the function relations. The “END” transition indicates the change to a state where no functions are active. The horizontal line crossing states zero to five is obtained from the desired behavior graph, while the “primed” states correspond to the undesired behavior. It can be seen that all the transitions leading out of the desired behavior are controllable (crossed by a bar in the figure), because their requirements relate to the controllable parameter “LoadCommand”. A note regarding the use of plant capacities here is necessary at this point. In this work, we used a rather simple algorithm to obtain the reachability graph of the resulting automaton which only handles bounded Petri nets. Adding place capacities allows us to ensure such property by construction, simplifying the present implementation. However, here the restriction in the overflow undesired behavior can also be used to this end, detecting the violating states and bounding the reachability graph of the Petri-net. Thus here, the resulting holons would not include the illegal states which violate the plant capacities. Therefore, to deal with this issue in the STS framework, we would have to include in the holon the descriptions the states and transitions specified in the undesired behavior. The logical specification will then ensure that these states are not reached.

The only remaining input information for synthesis through STS methods is the list of marked states. A simple and safe approach includes considering every state a marked state. However, here we also propose marking those states related to end sub-functions at the lowest level (adjacent to the leafs) of functional decomposition in the desired behaviors. This is justified because these states effectively represent the completion of the different functions at the highest level of detail available. For the case of self-looping functions, the states associated to the starting functions should also be added, ensuring that the non-blocking property of the system can be checked properly. To finish, the result of the STS synthesis is that a new part should be fed when the machine is idle becomes empty. This result can be taken back to the AM by complementing the requirement of taking a part in with these two requirements.

7.3.2 AM-HP mappings

As the design using HP will be used to verify and refine the parameter values (possibly associated with events) in the controlled system, we assume that the AM has been updated with the results of the STS design step explained in the previous section, that is, the rules for the controllable events are modeled in the AM. Also, the labeling of the AM information is the same as the one specified for the STS transformations.

At this point we recall the definitions from Section 7.1.4 to define the basic concepts in HP. The remaining information on the model mainly concerns the description of the possible structures of a hybrid program that can be interpreted by the KeyMaera tool. The formula '$\psi \rightarrow [\alpha]\varphi$' introduced in Section 7.1.4 sets the first level of the structure in the HP. The initial conditions of the system indicated as the precondition 'ψ' are placed in the first section, the dynamic system description 'α' is placed in the subsequent section, and the postcondition 'φ' holding the properties we wish to verify is placed in the last section. All the information in the sections is represented as HP formulae, for which Platzer provides a precise description. Next we provide a short account of the basic syntax. Discrete dynamics are represented by instantaneous assignments ':=', and continuous dynamics are modeled using differential or differential-algebraic equation systems '=' and evolution constraints '>', '<', '≥', '≤'. The sets of such statements are built using a conjunction operator '∧'. The available operators for modeling the control structure include: ';' for sequential composition,'*' for repetitive execution, '∨' for nondeterministic choice, and '?' for tests.

While 'ψ' and 'φ' have a relatively straightforward structure, 'α', which contains all the system dynamics, can be structured in many possible ways. One of such structures (and our choice for this work) is precisely a direct representation of hybrid automata. Now we describe the mappings between information in the AM and the HP. The resulting HP for the presented example is shown in Figure 74, and the next explanations can be followed through the figure.

The preconditions 'ψ' in the HP hold a direct relation to the initial conditions specified for our desired behaviors in the AM. It suffices to take all the leafs in the requirements tree which compose the initial state, translate them to instantaneous assignments, and join them through a conjunction. The postcondition 'φ' information can be found directly on the invariants declared in the AM (*cf.* section

7.2.2), taking care of adding a negating operator to the ones related to undesired behavior descriptions. The whole formula for 'φ' can be constructed similarly to the one for 'ψ', but there is also the possibility to verify each invariant independently if desired.

For simplicity, we have chosen to structure 'α' as a hybrid automaton. Thus, the transition structure can be obtained from a Petri net built as explained for the STS transformation, greatly simplifying this transformation. The resulting reachability graph can be directly associated to active/inactive states and their associated functions. However, we also have to attach the corresponding dynamics associated with the AM formulae mapped to every function. Depending on the syntax used to describe the formula, we may need parse the formulae information and transform it into valid HP statements. To simplify this, and as the AM does not restrict the formatting in the formulae, here we chose that our AM formulae descriptions abide the HP syntax, adding some labels for entry, in, and exit, which indicate when the dynamics should be executed.

```
\problem {
     \[ R Load, MaxLoad, LoadCommand, State, Progress, Speed; \]
     (
          (Load = 0 ∧ LoadCommand = 1 ∧ Progress < 4.13 ∧ Speed = 0 ∧ State = 0) /* Initial condition */
          ->
          \[
               (/* Automaton */
                    (?(State = 0); /* Initial state INIT*/
                         ({Progress' = Speed, Speed = 0, !(Progress < 4.13 ∧ Speed = 0) })/*Passive
                         dynamics*/
                         V
                         (?(Load = 0 ∧ LoadCommand = 1 ∧ Progress < 4.13 ∧ Speed = 0); State := 1;
                         Load := Load + 1; LoadCommand := 0; Progress := 0) /* Transition to State 1 */
                    )
                    V
                    (?(State = 1); /* State Take Part In*/
                         /*No internal dynamics, entry executed in State 0 transition*/
                         (?(Load = 1 ∧ Progress < 4.13); State := 2 )/* Transition to State 2 */
                    )
                    V
                    (?(State = 2); /* State Process Piece*/
                         ({Progress' = Speed, Speed = 1, !(Progress >= 4.13) }) /* Internal dynamics:
                         Processing piece */
                         V
                         (?(Progress >= 4.13 ); State := 3; Speed := 0 )/* Transition to State 3 */
                    )
                    V
                    (?(State = 3); /* State Stop Machine*/
                         /*No internal dynamics, entry executed in State 2 transition*/
                         (?(Speed = 0 ); State := 4; Load := Load - 1 )/* Transition to State 4 */
                    ) V
                    (?(State = 4); /* State Place Piece Out*/
                         /*No internal dynamics, entry executed in State 3 transition*/
                         (?(Load = 0); State := 5 )/* Transition to State 5 */
                    )
                    V
                    (?(State = 5); /* Default end state after 4... nothing happens here*/
                         ?0 = 0 /* This always evaluates to true */
                    )
               )* /* The dynamics are repeated indefinetly */
          \]
          ( !(Load > 1) ∧ !((Speed > 0) ∧ (LoadCommand = 1) ) ) /* Forbidden situations of machine
          overflow and loading busy machine*/
     )
```

Figure 74. Resulting HP code (comments enclosed in '/**/')

It can be seen in Figure 74 how the states numbered from zero to five in Figure 72 have their proper place in the automaton, separated from each other by or symbols.

In each state, the internal dynamics are evaluated, followed by the possible transitions in that state (this order could be reversed as well). The first section of the code provides a variable declaration followed by the specification of the initial conditions. The last part of the code contains the negated (marked by "!") undesired states as the invariants to be verified. After loading the resulting file in the Keymaera tool, it can be successfully verified that the forbidden situations are not reached.

7.4 Conclusions

Though this work just deals with transformations to two models, the choice of these models was done attempting that the basic information placed in the proposed AM is sufficient for transforming into any other model that requires the same kind of input, i.e., supervisory control design models. As shown in the European research initiative MULTIFORM [118], albeit differences in representation power, many formalisms for the description of behavior have equivalent transformations among each other.

The representation of behavior addressed through the architecture model is a valuable contribution to the field of formal behavioral representation, and can be used in other developments, e.g., the work of Cheung *et al* [42].

It must be emphasized that the current proposal for modeling of behavior does not pretend to replace the multiple formal behavior specifications found in literature and practice such as Petri nets and hybrid automata. The current proposal rather looks to fill the gap between data modeled with such representations and other sources of information which are relevant to the control design process, while at the same time searches to remain human and machine readable and understandable to (and thus usable by) a wide variety of stakeholders.

Concepts modeled here through the AM such as "state" and "behavior" may be perceived as specific to the domain of control. However, it is argued that these concepts are shared among many stakeholders, though probably using different names like "behavior" and "process flow", depending on the context in which they are addressed. Modeling such information is thus of great added value for the whole design process and not only for controller design.

Discussion

Apart from the intensive review of literature which led to identify more clearly several challenges in current complex product development practices, this thesis contains three main contributions: (1) a framework concept for supporting product development, (2) a language proposal for the representation of design information at the architecture level, and (3) an controller design workflow which can be supported by automation and considers the early stages of development. Additionally, the concepts in the proposals are proven through implementations. The next paragraphs support these contributions pointing out advantages, comparing to existing work, and highlighting limitations of this work while showing possible research directions for the future.

Using the classification in [190], the AM and its implementation could be defined as a model-based, awareness tool and a collaboration infrastructure. It also addresses indirectly the searchable design databases efforts (see Section 5.3). One differentiating factor with respect to previous efforts is the successful use of functional information and other abstract objects to build an architectural description that integrates dissimilar design information and cross-cutting concerns. Representation of cross cutting concerns cannot be properly done in component oriented architectural descriptions [73]. Additionally, the AM formalizes design practice information to allow systematic analysis from the point of view of engineering design research, effectively contributing to bridge the practice/research gap that exists in design [139].

The proposed model and implementation (that is, the AM and AM tool) cover other desirable characteristics lacking in current tools (*cf.* sections 2.3, 3.2.2.1, and 5.1.3):

- An architecture-level representation of the system to navigate information and identify its creators and stakeholders.
- Based on the use of functions (*cf.* Chapter 3), it allows associating multi-domain, varied, dissimilar information, identifying units of reusable knowledge, and representing decomposition of processes (the things the system should do).
- Modeling of physics and other behavioral concerns through parameters and parameter relations (formulae) (see Section 3.3.3).
- Permits requirement definitions from all stakeholders, which can be related to external models and traced back to associated functionality or implementing entities (see Section 3.3.1).
- Supports control software generation parting from high-level information (*cf.* Part II), which also opens perspectives for more “intelligent” control implementations that can deal with irregular situations and changing operation environments.

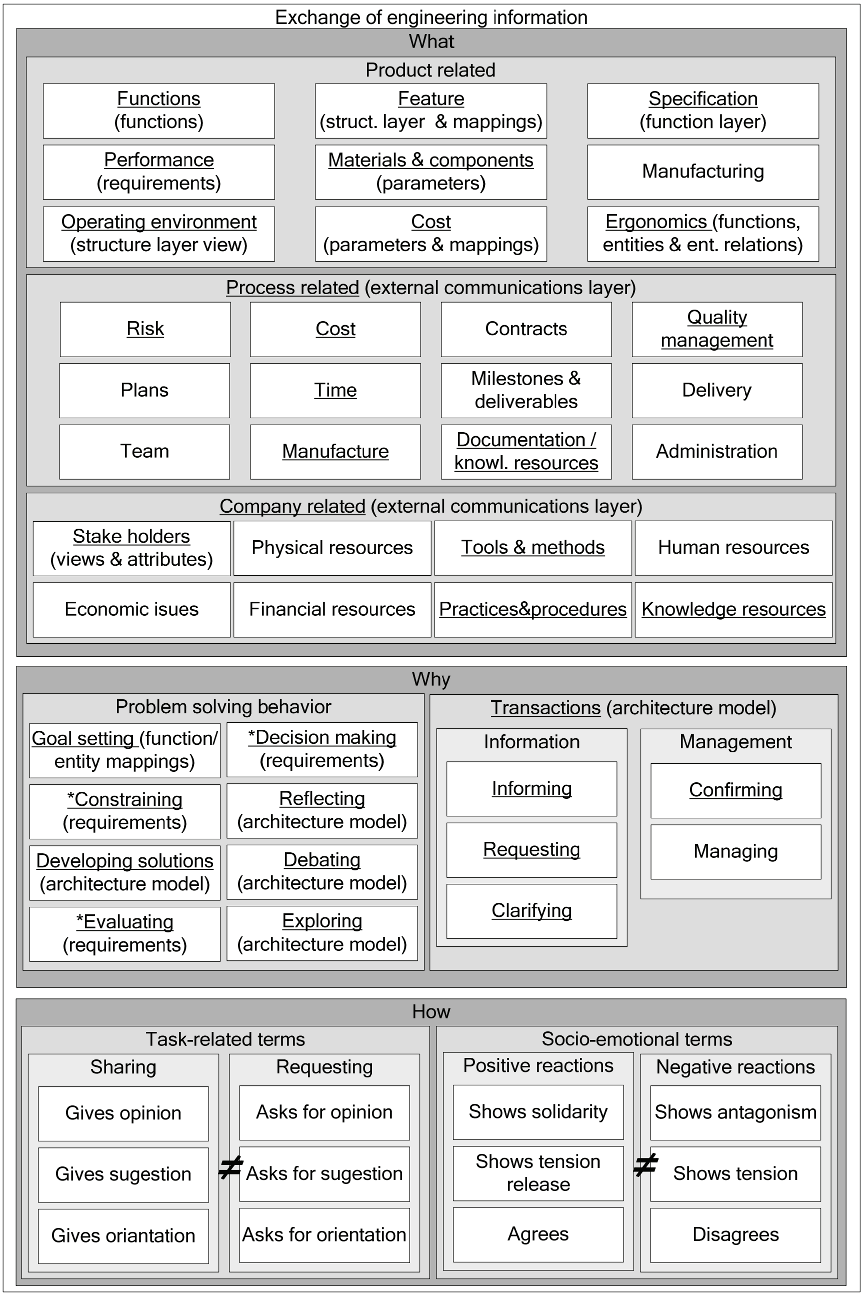

Exchange of engineering information
What
Product related
Functions (functions)
Feature (struct. layer & mappings)
Specification (function layer)
Performance (requirements)
Materials & components (parameters)
Manufacturing
Operating environment (structure layer view)
Cost (parameters & mappings)
Ergonomics (functions, entities & ent. relations)
Process related (external communications layer)
Risk
Cost
Contracts
Quality management
Plans
Time
Milestones & deliverables
Delivery
Team
Manufacture
Documentation / knowl. resources
Administration
Company related (external communications layer)
Stake holders (views & attributes)
Physical resources
Tools & methods
Human resources
Economic isues
Financial resources
Practices&procedures
Knowledge resources
Why
Problem solving behavior
Goal setting (function/ entity mappings)
*Decision making (requirements)
*Constraining (requirements)
Reflecting (architecture model)
Developing solutions (architecture model)
Debating (architecture model)
*Evaluating (requirements)
Exploring (architecture model)
Transactions (architecture model)
Information
Informing
Requesting
Clarifying
Management
Confirming
Managing
How
Task-related terms
Sharing
Gives opinion
Gives sugestion
Gives oriantation
Requesting
Asks for opinion
Asks for sugestion
Asks for orientation
≠
Socio-emotional terms
Positive reactions
Shows solidarity
Shows tension release
Agrees
Negative reactions
Shows antagonism
Shows tension
Disagrees
≠

Figure 75. Classification of engineering information exchange, adapted from [188]. Underlined concepts are addressed by the AM through the elements inside parenthesis. For the terms preceded by an asterisk (*), the AM provides a definition basis (i.e. input information) and computations are performed by external models/tools.

To focus on the information exchange, we come back to the discussion in Section 1.2. Figure 75 presents an overview of the developed classification terminology in the work of Wasiak *et al.* [188], and how these communication aspects are addressed by the AM. In that way, the AM proposal can be used to represent most information covering:

- The “what” topics regarding the product and the involved stakeholders, also adding a link to the knowledge resources, tools and methods, and practices.
- The “why” purpose of the designed product used for problem solving.

“How” the content is expressed falls out of the scope of the AM, as it does not represent (neither pretends to replace) in any form the socio-emotional aspects of communication.

The next points detail how the objects in the architecture model map to the pieces of exchanged information in Figure 75.

- Function nodes clearly show what the product must do, corresponding to the “functions”. The functional description is enriched progressively by decomposing functionality, and function relations allow specifying composite behaviors.
- Requirement nodes and descriptions represent the “specification” and desired “performance” of the product. Requirement nodes can also be used to constraint any resource (e.g., “cost”), and the actual resource usage of a unit or part can be represented straightforwardly by parameters attached to the entities that represent such parts.
- “Features” that specify what achieves a “function” are defined by the entities and the structural layer, together with its mappings to the functional layer.
- “Materials” and other characteristics of the product or process can be represented using parameters.
- The “operating environment” can be modeled in the structural layer.
- The “ergonomics” which cover user interaction with the product, are defined partially as functions and partially as the relations between the entities of the product and the entities that identify the user.
- “Stakeholders” can be identified in the model by finding the creating users of the objects (including views) through attributes (such as the user attribute).
- “Company and process related” information such as “Knowledge resources”, “tools and methods”, and “practices and procedures” are not represented directly in the architecture model. However, as long as there are external models containing such information, the external communications layer can provide links to such models. It is worth noting that, in any case, the AM can provide as an input (or receive as output) shared data related to the external models, but does not provide any means of computing or processing such data natively.
- The architecture model itself represents the “developing solutions”, as the implementations residing in the structural layer, for the functional layer, evolve. Attributes such as version or modification date can help managing such aspect.

- Other information related to provide reasons within the "problem solving behavior", can also be represented in the architecture model. The progress and aims of the design termed as "goal setting" can be seen in the model when analyzing the functions which have and have not been achieved, i.e., mapped to the structure layer.
- "Constraining", "evaluating", and "decision making" are covered by the link to external models (which allow evaluation by the stakeholders) through the external communications layer, passing through the parameters, to the entities they belong, to the functions they fulfill, and ending in the requirement nodes that map to those functions. Another path to the requirements can be found through analysis of the parameter network and the direct relation of causing parameters to the requirements.
- Most of the "transactions" are handled directly through the architecture model by direct request of the user.

It must be added that the implemented tool is a prototype intended to prove the proposed concepts, and that even though it has been used on industrial case-studies, a more robust implementation would be necessary to handle models at industrial scale, specially because the implemented tool shows issues regarding memory usage and fast visualization of large graphs. Other developments that would improve the tool relate to query and instantiation mechanisms which speed up the process of search and retrieval of design information. At this point, besides manual creation of shortcuts, the tool mainly provides one automated mechanism to shortcut data which is directly related to a certain object (data nodes adjacent to a selection), which allows visualizing the general concept with relative ease. Another aspect to investigate is the choice of appropriate visualization methods for the model. Visualization of models is important to facilitate understanding and appeal of the model, which strongly influence the decision of using a model or not using it.

Capturing the architecture information requires additional work from all stake holders. The goal is that designers use the model to consider and share their first (and subsequent) design assumptions before developing detailed domain-specific models. Afterwards, time can be saved by allowing reuse of previously modeled designs and by speeding up data updates and model synthesis on domain-specific models. It must me noted that, the required effort to build an AM is justified by the need to have common information in a "stakeholder-neutral" format, and that these effort may not be paid back if the intention is to document and use knowledge within a closed (domain-specific) audience which already shares well established and formal means of communication. For example, it makes no sense to create an AM that will be used only by the control specialists when they already count with specialized means of communication and models understood unambiguously by all their members.

Since the proposed approach proposes the integration of other models, caution must be taken not to confuse it with multi-modeling approach (see chapter 14 of [77]), which focuses on integrated model execution and simulation and maintains a domain-specific focus. Differences among the current proposal and the seminal work of FBS have been already commented in Chapter 3. Other related works are presented in [36], [37], [157], and [183].

The modeling philosophy in this work is close to that one described by the fine systems engineering material by Buede [36]. However, both works disagree at several points. Buede insists that decompositions to describe the architecture (for the equivalents to requirements, functions, and entities) should form directed trees, so that each new decomposition level forms a partition (the implications are analyzed in [36]). That restriction is not imposed here, as it is considered that it limits the ability of the AM to represent different views. Additionally [36] seems to make use of the "transformation" point of view for functions rather than the "intention" description point of view taken here.

A related work has been developed at the University of Padeborn ([37], [75]). That work uses similar constructs to the ones presented in this proposal, but offers more constrained applications as it specifically focuses on a controller architecture for self-optimizing systems. This is evident in the larger amount of specific modeling diagrams (more complex syntax) and the use of, what is consider here, domain-specific elements such as disturbance parameters and shape. Another important point is that there, contrary to our work, functions seem to play an accessory role for documentation.

The work of Szykman *et al* [157] also proposes and shares many valuable ideas with the proposals in Part I. Although it has relatively simple semantics for its core, it is specifically stated that the purpose was not aiming to a minimal core representation and that the set of object classes (including, e.g., shape and material) tries to follow some traditional representations in design. Though it is not argued that the AM proposal is minimal, it has been made evident (see Section 3.3.7.1) that there is a conscious strife for the simplification of the representation language to increase its applicability and popularize it use. An additional difference is the device-centric [65] focus of this model, in which most information revolves around the representation of the real objects and the function has a smaller role.

The proposed framework has many similarities with the CORE tool [183]. One fundamental difference is that the proposed framework aims at the integration of software tools used by the designers for design (in addition to representing multidisciplinary information), supported on the AM implementation. Another fundamental difference is that the proposed framework leaves analysis, verification, and simulation to the specialized tools that can be integrated instead of dealing with such computations directly as in the case of CORE.

The use of SysML has been promoted in works such as the one of Peak *et al* ([129], [130]), showing many advantages of integrating information. However, our experiences with SysML (see pp.70) showed certain characteristics that do not align well with the proposed architecture-centric approach, such as direct implementation of inheritance in the language, and lack modeling guidance and simplicity.

It can also bee seen that the architecture centric approach has common ideas with service-oriented architectures (SOA) [192], and may have certain applicability in that area. For example, the proposed representation may be tested as the shared format necessary for SOA, providing links to services through the external communications layer.

As stated in Section 3.3.7, the proposed language implementation does not consider class descriptions in order to reduce the modeling effort by simplifying the intrinsic language semantics. This has an immediate repercussion in the capacity to reduce modeling effort using the concept of inheritance in the instantiated data elements. However, this does not influence the capacity to use class information or "typing" in the data (i.e., creating a specific ontology), because this information can be placed outside the AM (with more appropriate tools such as OWL [186]) and indexed in the AM data elements using attributes such as "type" (*cf.* Section 3.3.6). This also expands the possibility to interpret the data in external tools, while keeping a lean shared information model, and may and help coping with the identified representation limitations of the language (*cf.* 3.3.7.2) by, for example, helping to identify different detail levels using external ontological information. Other possible uses of the class information can be seen in works like [80] and [157], and exposed are placed as future research directions in [11] and [195], e.g., explicit use of functional information in computational synthesis of (control) design knowledge.

A clear research direction is to make more extensive use of the representation proposal to create many examples representing different kinds of systems (e.g., agent-based systems) and situations and, linked to an increase in formalization and refinement of the required modeling steps (as done with control design in Part II), to gradually build a knowledge base of reusable models. Refining the modeling steps requires special attention must in the way design processes are carried by analyzing how the user places information into the model and the relation to the specific design processes. This research can also help identifying what level of proficiency and knowledge of the system is required to build and maintain an appropriate architecture-model, as it is evident that a user with no connection to the development or use of a certain system (not a stakeholder) can have many problems in these regard.

Ulrich [174] has identified architecture analysis and comparison as an important research direction. However, first it is necessary to have a common modeling language to represent the product architecture. A model with the characteristics proposed here can constitute a basis for such research.

A possibility to be explored lies on the function's potential to carry the design intention of systems, which could be used to guide or filter the results of a qualitative reasoning process towards the interesting alternatives, thus, increasing the potential of qualitative reasoning and naïve physics techniques. References [1] and [170] propose to carry out the task of defining the function by using the physical principles (or physical phenomena) that are intentionally used to achieve the function; then, the identified basic principles could be used to do some filtering by choosing the results in which they appear in action.

From the side of control design, besides obvious implementations to use other design methodologies and formal representations, an open research question remains for defining in detail the integration of regulatory and supervisory synthesis approaches. However, studying both procedures it can be seen that they mainly hold in common information in the behavioral layer, and that since the formulae syntax used in the HP for the supervisory control can also be interpreted from the point of view of causality, as it is required for the analysis carried out for regulatory control. Thus, both proposed design methods are not incompatible but further research has to look for deeper relations and interactions to form a unified design approach.

Conclusions

Current development practices of complex and multidisciplinary products face many challenges to produce truly integrated systems in an efficient way. The challenges relate to integration of the work of stakeholders, which can be partially supported by integration of the information they share. However, the variety and amount of multidisciplinary information involved in design make of its integration a complex problem.

The reviews and discussions show that most current supporting tools and methods just address the challenges partially, and that the domain-specific models supporting such methods have intrinsic handicaps when stakeholders from different domains have to work together.

The architecture-centric approach to design seeks overcoming the reviewed challenges by proposing the use of architecture-level information to handle the complexity of the information integration problem. The proposal provides greater insight into why product architecture can effectively support design, and what are the desirable characteristics of a model which allows representing architecture and using such representation in design. These characteristics respond mainly to the need for providing an overview of the current state and goals of design, separating different types of information, and interlinking the shared information in design. An integral part of the proposal is the need to consider systems architecting activities while taking into account that all stakeholders provide valuable knowledge to them.

The proposed AM language and tool offer support for the architecture-centric approach. This is challenging because it requires representing information from a wide variety of sources and models. To capture such information, the AM is built with abstract, high-level representations, such as functions and behaviors, which by virtue of their (generic) nature, can accommodate to a wide range of modeling situations. Additional contributions from these proposals relate to the characteristics of the AM language and tool, which aim to simplify the semantics and syntax of common representation and allow retrieving and reusing modeled data to maintain an overview of the design context. This is partially possible because the domain-specific semantics for the modeled information (corresponding to classes, ontology, or taxonomy) are kept separate of the proposed common model to represent architecture. Developing such domain-specific semantics can greatly improve the usability of the modeled information, but does not form part of the research documented in this thesis.

The main concepts in the proposals have been tested in cases inspired by the control design domain, taken as a representative example of the need to integrate information from many domains. This book also compiles contributions to the control domain, including the identification of the need and the available methods to support controller design starting at the conceptual phases, and providing an implementation framework based on the architecture-centric approach which allows integrating design processes.

As seen through the thesis, the external communications layer provides key elements for the connectivity of the architecture model to the design practice (also necessary for automation), by enabling formal representation of the exchange channels. However, these developments do not belong the focus of the present thesis and are part of the work of Woestenenk ([193], [194], [195]).

References

[1] Akman V, P. J. W. ten Hagen, and T. Tomiyama, "A fundamental and theoretical framework for an intelligent CAD system," in *Computer-Aided Design archive*, Volume 22, Issue 6 (July/August 1990), pp. 352-367.

[2] Alink T, Eckert C, Ruckpaul A, Albers A, 2011. Different Function Breakdowns for One Existing Product: Experimental Results. In Design Computing and Cognition '10, Gero JS (Ed.), Springer, pp. 405-424.

[3] Alvarez Cabrera AA, Erden MS, Foeken MJ, Tomiyama T, 2008. High level model integration for the design of mechatronic systems. In proceedings of IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications. Beijing, China. pp. 387-392.

[4] Alvarez Cabrera AA, Erden MS, Tomiyama T. On the potential of Function-Behavior-State (FBS) methodology for the integration of modeling tools. Proceedings of the CIRP design conference, 2009: 412.

[5] Alvarez Cabrera AA, Foeken MJ, 2008. Introductory study on automated data transfer between modeling tools for controller design. Internal project report, internship at Phillips Applied Technologies.

[6] Alvarez Cabrera AA, Foeken MJ, Tekin OA, Woestenenk K, Erden MS, De Schutter B, van Tooren MJL, Babuška R, van Houten FJAM, Tomiyama T. Towards Automation of Control Software: A Review of Challenges in Mechatronic Design. Mechatronics, 20(8), pp. 876-886. <http://dx.doi.org/10.1016/j.mechatronics.2010.05.003>

[7] Alvarez Cabrera AA, Foeken MJ, Woestenenk K, Stoot G, Tomiyama T, 2011. Modeling and Using Product Architectures in Industrial Mechatronic Product Development: Experiments and Observations. In Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011. August 28-31, 2011, Washington D.C.

[8] Alvarez Cabrera AA, Komoto H, Tomiyama T, 2011. Supporting co-design of physical and control architectures of mechatronic systems. In Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011. August 28-31, 2011, Washington D.C.

[9] Alvarez Cabrera AA, Lopes GD, Tomiyama T. An Architecture-Level Supervisory Controller Specification for Automatic Generation, *DRAFT to be submitted for publication. August 2011.*

[10] Alvarez Cabrera AA, Tomiyama T. Architecture-Level Representation and Analysis of Regulatory Controller Configuration for Complex Mechatronic Systems, *DRAFT to be submitted for publication. June 2011.*

[11] Alvarez Cabrera AA, Woestenenk K, Tomiyama T, 2011. An Architecture Model to Support Cooperative Design for Mechatronic Products: A Control Design Case. Mechatronics, 21(3), pp. 534-547. Online <http://dx.doi.org/10.1016/j.mechatronics.2011.01.009>

[12] Amerongen J van, Breedveld P. Modeling of Physical Systems for the Design and Control of Mechatronic Systems. Annual Reviews in Control 2003; 27: 87–117.

[13] Amerongen J van. Mechatronic Design. Journal of Mechatronics 2003; 13(10): 1046-166.

[14] Apache, "Apache Subversion". On the WWW, January 2011. <`http://subversion.apache.org/`>.

[15] Astrom, K., and Hagglund, T., 1995. PID Controllers: Theory, Design, and Tuning. 2nd ed. Instrument Society of America.

[16] Auslander DM, Ridgely JR, Ringgenberg JD. Design and Implementation of Real Time Software for Control of Mechanical Systems. 2002.

[17] Avigad G, A. Moshaiov, and N. Brauner, "Towards a general tool for mechatronic design," [online] in *Proceedings of 2003 IEEE Conference on Control Applications*, 2003, CCA 2003, Vol. 2, pp. 1035-1040.

[18] Barr A, Cohen PR. The Handbook of Artificial Intelligence. Vol. 4, Chapter 21. Los Altos, CA, USA: William Kaufmann, Inc.; 1989.

[19] Beckers JMJ., Heemels, W.P.M.H., Bukkems, B.H.M. and Muller, G.J., 2007, "Effective Industrial Modeling for High-Tech Systems: The Example of Happy Flow," In *Proceedings of 17th Annual Symposium of INCOSE*, San Diego, CA, US.

[20] Berends JPTJ, Tooren MJL van, Schut EJ. Design and implementation of a new generation multi-agent task environment framework. In: 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 4th AIAA Multidisciplinary Design Optimization Specialist Conference. Schaumburg, IL, USA; 2008.

[21] Bishop RH *Ed*. The mechatronics Handbook: Mechatronic system control, logic, and data acquisition. CRC press. 2nd Ed, 2008.

[22] Blessing LTM, Chakrabarti A. DRM, a design research methodology. London, UK: Springer-Verlag; 2009.

[23] Boehm B. A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Software Engineering Notes 1986; 11(4):14-24.

[24] Bonnema GM. FunKey architecting - An integrated approach to system architecting using functions, key drivers and system budgets. Ph.D. thesis. University of Twente. Enschede, The Netherlands, 2008.

[25] Borches, P.D., 2010, *A3 Architecture overviews*. Ph.D. thesis, University of Twente. Enschede, The Netherlands.

[26] Bosch J, 2002. Architecture-centric software engineering. In ICSR-7. Gacek C. (Ed.), Lecture Notes in Control Science 2319, pp. 347-248.

[27] Boucher M, 2009, System Engineering: Top Four Design Tips to Increase Profit Margins for Mechatronics and Smart Products," Aberdeen Group, Boston, MA.

[28] Boucher M, Houlihan D. System design: new product development for mechatronics. Boston, MA, USA: Aberdeen Group; 2008.

[29] Bracewell R, Sharpe J. Functional descriptions used in computer support for qualitative scheme generation - "Schemebuilder". AI EDAM Journal - Special Issue: Representing Functionality in Design 1996; 10: 333-346.

[30] Bradley DA, "The what, why and how of mechatronics'" in *Engineering Science and Education Journal*, Apr. 1997, Vol. 6, Issue 2, pp. 81-88.

[31] Braspenning NCWM, Boumen R, Mortel-Fronczak JM van de, Rooda JE, 2011. Estimating and quantifying the impact of using models for integration and testing. In Computers in Industry, 62(1), pp. 65–77.

[32] Brooks RA, 1986. A robust layered control system for a mobile robot. In IEEE journal of robotics and automation, RA-2(1), pp. 14-23.

[33] Brown SL, Eisenhardt, K.M., 1995, “Product Development: Past Research, Present Findings, and Future Directions,” Academy of Management Review, **20**(2), pp. 343-378.

[34] Browning TR. The many views of a process: Toward a process architecture framework for product development processes. Systems Engineering 2009; 12 (1):69-90.

[35] Bryant CR, McAdams, D.A., Stone, R.B., Kurtoglu, T., Campbell, M.I., 2005, “A Computational Technique for Concept Generation,” In *Proceedings of IDETC/CIE 2005*, Longbeach, CA, US.

[36] Buede DM, 2000. The engineering design of systems: Models and methods. Wiley.

[37] Burmester S, Giese H, Munch E, Oberschelp O, Klein F, Scheideler P, 2008. Tool support for the design of self-optimizing mechatronic Multi-agent systems. *Int J Sfttw Tools Technol Transfer*; 10:207-222.

[38] Cassandras CG, Lafortune S, 2008. Introduction to discrete event systems. Springer.

[39] Chakrabarti A., Bligh, T. “An approach to functional systhesis in mechanical conceptual design. Part I: Introduction and knowledge representation,” *Research in engineering design*, 6(3), pp. 127-141.

[40] Chandrasekaran B. “Representing function: Relating functional representation and functional modeling research streams,” AIEDAM, 19(2), pp. 65-74.

[41] Chandrasekaran B. Functional representation: A brief historical perspective. Applied Artificial Intelligence 1994; 8: 173-197.

[42] Cheung KS, Chow PKO, 2008. A Petri-net approach to refining object behavioral specifications. In *Informatica*; 33, pp. 221-232.

[43] Childers SR, Long JE. A concurrent methodology for the system engineering design process. Unpublished green paper; 1994. < http://www.vitechcorp.com/support /papers.php>

[44] Chmarra M, Alvarez Cabrera AA, Van Beek TJ, D’Amelio V, Erden MS, Tomiyama T, 2008. Revisiting the Divide and Conquer Strategy to Deal with Complexity in Product Design. In proceedings of the MESA08 conference, 2008.

[45] Christofides PD, El-Farra NH, 2005. Control of nonlinear and hybrid process systems. Springer.

[46] CIF project, 2011. CIF: The Compositional Interchange Format for Hybrid Systems. Project website < http://se.wtb.tue.nl/sewiki/cif/start >

[47] Citherlet S, Clarke JA, Hand J. Integration in building physics simulations. Energy and Buildings 2001: 33; 451-461.

[48] Commault C, Dion JM, Benachene M, 1993. Output feedback disturbance decoupling graph interpretation for structured systems. Automatica 29 (6) 1463-1472.

[49] Commault C, Dion JM, Do TH. The disturbance rejection by measurement feedback problem revisited. In proceedings of the American Control Conference, 2010.

[50] Commault C, Dion JM, Hovelaque V, 1997. A geometric approach for structured systems: Application to disturbance decoupling. Automatica 33 (3) 403-409.

[51] Commault C, Dion JM, Van der Woude JW, 2002. Characterization of generic properties of linear structured systems for efficient computations. Kybernetika 38(5) 503-520.

[52] Commault C, Dion JM, Yacoub Agha S, 2008. Structural analysis for the sensor location problem in fault detection and isolation. Automatica 44, 2074-2080.

[53] Controllab Products B.V. 20-sim. <http://www.20sim.com>.

[54] Craig K, De Vito M, Mattice M, La Vigna C, Teolis C. Mechatronic integration modeling," [online] in *International Conference on Advanced Intelligent Mechatronics, 1999. Proceedings. 1999 IEEE/ASME*, Sept. 1999, pp.1032 – 1037, Atlanta, GA.

[55] Craig K. Mechatronic system design. ASME newsletter; 2009. <http://files.asme.org /asmeorg/NewsPublicPolicy/Newsletters/METoday/Articles/17845.pdf>

[56] Cutkosky MR, Engelmore RS, Fikes RE, Genereseth MR, Gruber TR, Mark WS, *et al.* PACT: An experiment in integrating concurrent engineering systems. Computer 1993; 26(1), pp. 28-37.

[57] Damas C, Lambeau B, Dupont P, van Lamsweerde A, 2005. Generating annotated behavior models from end-user scenarios. In IEEE transactions of software engineering, 31(12), pp. 1056-1073.

[58] Dauphin-Tanguy G, Rahmani A, Sueur C, 1999. Bond graph aided design of controlled systems. Simulation practice and theory 7: 493-513.

[59] De Michielli, G., Gupta, R.k., 1997, "Hardware/Software Co-design," Proceedings of the IEEE, **85**(3), pp. 349-365.

[60] Derelöv M., 2008, "Qualitative modeling of potential failures: On evaluation of conceptual design," *Journal of Engineering Design*, 19(3), pp. 201-225.

[61] Dion JM, Commault C, Van der Woude J, 2003. Generic properties and control of linear structured systems: a survey. Automatica 39 (7) 1125-1144.

[62] Dolk DR, 1993. An introduction to model integration and integrated modeling environments, [online] in *Decision Support Systems*, 10(3), pp. 249-254.

[63] Dolk DR, Kottemann, JE, 1993, Model integration and a theory of models, *Decision Support Systems*, 9(1), pp. 51-63.

[64] Dynasim A.B. Dymola – Dynamic modeling laboratory; 2008. <http://www.dynasim. se/index.htm>.

[65] Erden MS, Komoto H, Van Beek TJ, D'amelio V, Echavarria E, Tomiyama T, 2008. A review of function modeling: Approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*; 22(2), pp. 147-169.

[66] Erixon G., von Yxkull, A., Arnstrom, A., 1996, "Modularity–the Basis for Product and Factory Reengineering," Annals of the CIRP, **45**(1), pp. 1–6.

[67] European Cooperation for Space Standardization. Space engineering – Functional analysis (E-10-05A); 1999. <http://esapub.esrin.esa.it/pss/ecss-ct05.htm>.

[68] Ferrarini L, Carpanzano E. A structured methodology for the design and implementation of control and supervision systems for robotic applications. IEEE Journal of Control Systems Technology 2002; 10(2): 272–9.

[69] Ferretti G, Magnani GA, Rocco P. Virtual Prototyping of Mechatronic Systems. Annual Reviews in Control 2004; 24: 192–206.

[70] Fishwick PA, 2007. The languages of dynamic system modeling. Chapter in Handbook of Dynamic Systems. Ed. Fishwick PA. Taylor & Francis Group

[71] Forbus KD, 1984. Qualitative process theory, *Artificial Intelligence*, 24(3), pp. 85-168.

[72] Friedenthal S, Moore A, Steiner R. A practical guide to SysML: The systems modeling language. Morgan Kaufmann OMG press. 2008.

[73] Fuentes L, Pinto M, Sanchez P, 2008. Generating CAM aspect-oriented architectures using Model-Driven Development. *Information and Software Technology* 50: 1248-1265.

[74] Fujita K., Yoshida, H., 2004, "Product Variety Optimization Simultaneously Designing Module Combination and Module Attributes," Concurrent Engineering, **12**(2), pp. 105–118.

[75] Gausemeier J, Frank U, Donoth J, Kajl S, 2009. Specification technique for the description of self-optimizing mechatronic systems. In Res. Eng. Design; 20, pp. 201-223.

[76] Geoffrion AM, 1989. Reusing structured models via model integration, *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, 1989*, Vol.III: Decision Support and Knowledge Based Systems Track, pp. 601-611.

[77] Gray J, Tolvanen JP, Kelly S, Gokhale A, Neema S, Sprinkle J, 2007. Domain-specific modeling. Chapter in Handbook of Dynamic Systems. Ed. Fishwick PA. Taylor & Francis Group.

[78] Gronback RC, 2009. Eclipse Modeling project: a domain-specific language (DSL) toolkit. The eclipse series. Adison-Wesley.

[79] Grübel G. Perspectives of CACSD: embedding the control system design process into a virtual engineering environment. IEEE International Symposium on Computer Aided Control System Design, 1999.

[80] Güroğlu S, 2005. An evolutionary methodology for conceptual design. PhD thesis, Midel East Technical University.

[81] Harel D. From play-in scenarios to code: an achievable dream. IEEE Computer 2001; 34(1): 53–60.

[82] Hayes Roth F, Erman LD, Terry A, Hayes Roth B. Domain-specific software architectures: distributed intelligent control and management. IEEE Symposium on CACSD, 1992.

[83] Heck B, Wills L, Vachtevanos G. Software technology for implementing reusable, distributed control systems. IEEE Control Systems Magazine 2003; 23(1): 21-35.

[84] Heemels WPMH, De Schutter B, Lunze J, Lazar M, 2010. Stability analysis and controller synthesis for hybrid dynamical systems. In Phil. Trans. R. Soc. A13, 368(1930), pp. 4937-4960.

[85] Hirtz J., Stone, R., McAdams, D., Szykman, S., Wood, K., 2002, "A functional basis for engineering design: Reconciling and evolving previous efforts," *Res. Eng. Des.*, 13(2), pp. 65–82.

[86] Huckle T. Software Bugs – Software Glitches: Collection of software bugs. (Website) 2010. <http://www5.in.tum.de/~huckle/bugse.html> consulted in June 2011.

[87] Hunt J. MACE: A system for the construction of functional models using case-based reasoning. Expert Systems with Applications 1995; 9(3): 347-360.

[88] IEEE, 2007. International standard (ISO/IEC 42010 - IEEE 1471): Systems engineering and software engineering – recommended practice for architectural description of software-intensive systems.IEEE.

[89] INCOSE. Survey of Model-Based Systems Engineering (MBSE) Methodologies. Technical report number INCOSE-TD-2007-003-01. 2008. Online in < http://www.incose.org/ProductsPubs/pdf/techdata/MTTC/MBSE_Methodology_Survey_2008-0610_RevB-JAE2.pdf >

[90] INCOSE. Systems Engineering Handbook: A guide for system lifecycle processes and activities. Cecilia Haskins (Ed.). INCOSE-TP-2003-002-03. Version 3. June 2006.

[91] Ishii M., Tomiyama, T., Yoshikawa, H., 1993, "A synthetic reasoning method for conceptual design," *IFIP World Class Manufacturing '93*, Amsterdam, pp. 3-16.

[92] Jackson CK, 2006. Simulation Driven Design Benchmark Report. Boston, MA, USA: Aberdeen Group; 2006.

[93] James J, Cellier F, Pang G, Gray J, Erik Mattsson E. The state of computer-aided control system design (CACSD). IEEE Control Systems; 15(2): 6-7. 1995.

[94] Jobling CP, 1996. Advances in computer aided control systems design. IEE Colloquium on Advances in Computer-Aided Control System Design (Digest No: 1996/061).

[95] Karnopp DC, Margolis DL, Rosenberg RC, 2006. System Dynamics: Modeling and Simulation of Mechatronic Systems, 4th ed. New York, NY, USA: Wiley.

[96] Kelly S, Tolvanen J-P. Domain-Specific Modeling: Enabling Full Code Generation. Hoboken, NJ, USA: Wiley-IEEE Computer Society Press; 2008.

[97] Kindler E. Model-based software engineering: the challenges of modeling behaviour. In proceedings of BMFA'10, June 2010, Paris, France.

[98] Knowledge Based Systems Inc. IDEF Family of Methods website. <http://www.idef.com/>

[99] Komoto H, Tomiyama T, 2010. Computational support for system architecting, Proceedings of International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. ASME, DETC 2010–28683.

[100] Komoto H, Tomiyama T, 2011. Multi-disciplinary system decomposition of complex mechatronics systems, Annals of the CIRP, **60**(1), (provisionally accepted)

[101] Krishnan V., Ulrich, K.T., 2001, "Product Development Decisions: A Review of the Literature," Management Science, **47**(1), pp. 1-21.

[102] La Rocca G, Van Tooren MJL, 2007. Enabling distributed multi-disciplinary design of complex products: A knowledge-based engineering approach. Journal of Design Research 2007; 5(3): 333–352.

[103] Lattanze AJ, 2005. The architecture-centric development method. School of Computer Science Technical Report CMU-ISRI-05-103, Carnegie Mellon University.

[104] Levis A, 1993, "National Missile Defense Command and Control Methodology Development," contract data requirements list A005 report for US Army contract MDA 903-88-0019, delivery order 0042. George Mason University, Center of Excellence in Command, Control, Communications, and Intelligence, Fairfax, VA.

[105] Lu RX, De Silva CW, Ang Jr. MH, Poo JAN, Corporaal H, 2005. A new approach for mechatronic system design: Mechatronic design quotient (MDQ), in *Conference on Advanced Intelligent Mechatronics, Proceedings, 2005 IEEE/ASME International*, July 2005, pp. 911-915, Monterey, CA.

[106] Ma C, Wonham WM, 2005. Nonblocking supervisory control of state tree structures In the series Lecture Notes in Control and Information Sciences. Springer.

[107] Maciejowski JM. The changing face and role of CACSD. In proceedings of the 2006 IEEE conference on CACSD. 2006.

[108] Martin JN. Systems Engineering Guidebook – A Process for Developing Systems and Products. Boca Raton, FL, USA: CRC Press; 1997.

[109] Matarić MJ, 1992. Behaviour-based control: main properties and implications. In Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems.

[110] Matarić MJ, 1997. Behaviour-based control: examples from navigation, learning, and group behavior. In J. Expt. THeor. Artif. Intell. 9, pp. 323-336.

[111] Mathur N. Mechatronics – Five design challenges and solutions for machine builders. Instrumentation Newsletter 2007; 19 (2): 6-7. < http://zone.ni.com/devzone/cda/pub/p/id/145>

[112] McDonough, W., 2005, On Cradle to Cradle Design. In TED2005 (speech transcription). On the WWW, May 2011. <http://www.ted.com/talks/william_mcdonough_on_cradle_to_cradle_design.html>

[113] McMahon CA, Caldwell NHM, Darlington MJ, Culley SJ, Giess MD, Clarkson PJ. The Development Of A Set Of Principles For The Through-Life Management Of Engineering Information. 2009. <http://www.bath.ac.uk/idmrc/themes/projects/kim/kim40rep007mjd10.doc>

[114] Mosterman PJ, 2007. Hybrid dynamic systems: modeling an execution. Chapter in Handbook of Dynamic Systems. Ed. Fishwick PA. Taylor & Francis Group.

[115] Mosterman PJ. HyBrSim – A Modeling and Simulation Environment for Hybrid Bond Graphs. <http://moncs.cs.mcgill.ca/people/mosterman/papers/ jsce01/p.pdf>

[116] Muller G. System Architecting. Eindhoven, The Netherlands: Embedded Systems Institute; 2009.

[117] Muller GJ. CAFCR: A multi-view method for embedded systems architecting, Ph.D. thesis. Delft University of Technology. Delft, The Netherlands. 2004.

[118] MULTIFORM consortium, 2008. Integrated multi-formalism tool support for the design of networked embedded control systems MULTIFORM. Project website <http://www.multiform.bci.tu-dortmund.de/>

[119] Murata T, 1989. Petri nets: Properties , analysis and applications. In proceedings of the IEEE, 77(4), pp. 541-580.

[120] National Institute of Standards and Technology (NIST), 1993. Integration definition for function modeling (IDEF0). <http://www.idef.com/pdf/idef0.pdf>.

[121] Nayak PP, 1995. Automated Modeling of Physical Systems. In the series Lecture Motes in Artificial Intelligence. Springer.

[122] Nicolescu M, Matarić MJ. A hierarchical architecture for behavior-based robots. In proceedings of AAMAS'02, July 2002, Bologna, Italy, pp., 227-233.

[123] Object Management Group. OMG Systems Modeling Language, V1.0; 2001. <http://www.omg.org/cgi-bin/apps/doc?formal/07-09-01.pdf>.

[124] Object Management Group. Unified Modeling Language, V2.2; 2009. <http://www.omg.org/spec/UML/2.2/>

[125] Ogata K. Modern Control Engineering. 3rd edition. Prentice Hall. 1997.

[126] Pahl G, Beitz W, Feldhusen J, Grote KH, 2007. Engineering Design: A Systematic Approach, 3rd ed. London, UK: Springer London Limited.

[127] Parallax Inc. The Boe-Bot robot. <http://www.parallax.com/tabid/411/Default.aspx>. Website consulted in June 2010.

[128] Paredis C, Diaz-Calderon A, Sinha R, Khosla PK. Composable models for simulation-based design. Engineering with Computers 2001; 17: 112-128.

[129] Peak RS, Burkhart RM, Friedenthal SA, Wilson MW, Bajaj M, Kim I, 2007. Simulation-based design using SysML: A parametrics primer. In: Proceedings of INCOSE International Symposium, San Diego, CA, USA.

[130] Peak RS, Burkhart RM, Friedenthal SA, Wilson MW, Bajaj M, Kim I, 2007. Simulation-based design using SysML: Celebrating diversity by example. In: Proceedings of INCOSE International Symposium, San Diego, CA, USA.

[131] Perrin K. Digital prototyping in mechatronic design. Project Mechatronics (website); 2009. <http://www.projectmechatronics.com/2009/07/13/digital-prototyping-in-mechatronic-design/>

[132] Platzer A, 2010. Logical analysis of hybrid systems. Springer.

[133] Portioli-Staudachera A, Van Landeghemb H, Mappellic M, Redaelli CE. Implementation of concurrent engineering: a survey in Italy and Belgium. ROBOT CIM-INT MANUF; 19: 225–238. 2003.

[134] Process Systems Enterprice Limited. gPROMS Advanced Process Modeling and Process Simulation. <http://www.psenterprise.com/gproms/index.html>

[135] QFD Institute. QFD Institute home page. <http://www.qfdi.org/>

[136] Ramadge PJ, Wonham WM, 1989. The control of discrete event systems. In Proceedings of the IEEE, 77(1), pp. 81-98.

[137] Ramos-Hernandez DN, Fleming PJ, Bass JM. A novel object-oriented environment for distributed process control systems. Control Engineering Practice 2005; 13: 213–230.

[138] Rash JL, Hinchey MG, Rouff CA, Gracanin D, Erickson J. A requirements-based programming approach to developing a NASA autonomous ground control system. Artificial Intelligence Review 2006; 25(4): 285–297.

[139] Reich Y, 2010. My method is better! Res Eng Design 21: 137-142.

[140] Reinschke KJ, 1988. Multivariable control, a graph-theoretic approach. In the series Lecture notes in control and information sciences 108. Springer-Verlag.

[141] Ridley MW, 2010, When Ideas Have Sex. In TEDGlobal2010 (speech transcription). On the WWW, January 2011. URL <http://www.ted.com/talks/matt_ridley_when_ideas_have_sex.html>

[142] Rodenacker W., 1971, *Methodisches Konstruieren*, Springer-Verlag, Berlin.

[143] Rzevski G. On conceptual design of intelligent mechatronic systems. Mechatronics 2003; 13: 1029-1044.

[144] Sakao T, Umeda Y, Tomiyama T, Shimomura Y. Generation of sequence-control programs from design information. IEEE Expert, 1997; 12.

[145] Scattolini R, 2009. Architectures for distributed and hierarchical model predictive control – a review. Journal of Process Control 19, 723-731.

[146] Schlacher K, Kugi A, 2001. Automatic control of mechatronic systems. Int. J. Appl. Math. Comput. Sci. 11(1): 131-164.

[147] Schöner HP. Automotive mechatronics. Control Engineering Practice 2004; 12: 1343-1351.

[148] Shapiro J. Mechatronics design faces two challenges – and two solutions. Electronic design (website); 2008. < http://electronicdesign.com/Articles/Index.cfm?AD=1& ArticleID=18068 >

[149] Sharon AK, Hogan N, Hardt DE, 1991. Controller design in the physical domain. Journal of the Franklin Institute 328(5-6), 697-721.

[150] Skogestad S, Postlethwaite I. Multivariable feedback control. Wiley. 1996.

[151] Sohlenius G. Concurrent Engineering. CIRP Annals 1992; 41 (2): 645-655.

[152] Sosa ME, Eppinger, S.D., Rowles, C.M., 2004, "The Misalignment of Product Architecture and Organizational Structure in Complex Product Development," Management Science, **50**(12), pp. 1674-1689.

[153] Stevens R, Brook P, Jackson, 1998. System Engineering: Coping With Complexity. Prentice Hall Europe.

[154] Stone R, Wood K, 2000. Development of a functional basis for design. *ASME Journal of Mechanical Design*, 122(4): 359–370.

[155] Suh NP, 1990, *The Principles of Design,* Oxford University Press, Oxford.

[156] Synopsys. Saber Mixed-Signal, Mixed-Technology Simulation. <http://www.synopsys.com/Tools/SLD/MECHATRONICS/Saber/Pages/default.aspx >

[157] Szykman S, Fenves SJ, Keirouz W, Shooter SB, 2001. A foundation for interoperability in next-generation product development systems. In Computer-Aided Design, 33, pp. 545-559.

[158] Tabuada P, 2009. Verification and control of hybrid systems. Springer.

[159] Tactical Science Solutions Inc., 2007, *Quicklook final report*, [online] (http://www.tacticalsciencesolutions.com/files/05-30-07%20Quicklook%20Final%20Report%20v1.19.pdf)

[160] Technical University of Berlin. SMILE - The Simulation Environment for Scientific Computing. <http://www.smilenet.de>

[161] Tekin OA, Babuska R, Tomiyama T, De Schutter B. Toward a flexible control design framework to automatically generate control code for mechatronic systems. In proceedings of the American Control Conference, June 2009.

[162] Thane H. Safe and Reliable Computer Control Systems: Concepts and Methods. Technical report No. TRITA-NMK. Mechatronics Laboratory; The Royal Institute of Technology, Sweeden, 1996.

[163] The Eclipse Foundation. Eclipse modeling project main page.<http://www.eclipse.org/home/categories/index.php?category=modeling>

[164] The MathWorks. Simscape; 2009. < http://www.mathworks.com/products/simscape/?s_cid=HP_FP_SL_Simscape>

[165] The Modelica Association. Modelica and the Modelica Association; 2008. <http://www.modelica.org>.

[166] Thompson HA, Ramos-Hernandez DN, Fu J, Jiang L, Choi I, Cartledge K, *et al.* A flexible environment for rapid prototyping and analysis distributed real-time safety-critical systems. Control Engineering Practice 2007; 15: 77–94.

[167] Tomiyama T, Kirayama T, Takeda H, Xue D, Yoshikawa H, 1989. "Metamodel: A key to intelligent CAD systems," in *Research in Engineering Design*, 1(1) , pp. 19-34

[168] Tomiyama T, Kiriyama T, Umeda Y, 1994. "Towards Knowledge Intensive Engineering," In *Computer aided conceptual design, proceedings of the 1994 Lancaster international workshop on engineering design CACD '94*, Sharpe, J., Oh, V., eds., Lancaster University, Lancaster, UK, pp 319-337

[169] Tomiyama T, M. Bonnema, First workshop on complex systems architecting, best practices and new development, unpublished.

[170] Tomiyama T, Umeda Y, 1993. A CAD for functional design," in *Annals of the CIRP'93*, 42(1) pp. 143-146.

[171] Tomiyama T, Umeda Y, Ishii M, Yoshioka M, Kirayama T, 1996. Knowledge systematization for a knowledge intensive engineering framework. In: Tomiyama T, Mantyla M, Finger S, editors. Knowledge Intensive CAD: Volume 1, Chapman & Hall, p. 55-80.

[172] Tomizuka M. Mechatronics: from the 20th to the 21st century. Control Engineering Practice 2002; 10: 877-886.

[173] Toom A, Naks T, Pantel M, Gandriau M, Indrawati. Gene-Auto: an automatic code generator for a safe subset of Simulink/Stateflow and Scicos. In: 4th European Congress on Embedded Real Time Software. Toulouse, France; 2008.

[174] Ulrich KT, 1995, “The Role of Product Architecture in the Manufacturing Firm,” Research Policy, **24**(3), pp. 419-440.

[175] Ulrich KT., Eppinger, S.D., 2000, *Product design and development*, 2nd ed., Irwin McGraw-Hill.

[176] Umeda Y, Ishii M, Yoshioka M, Shimomura Y, Tomiyama T, 1996. Supporting conceptual design based on the function-behavior state modeler. AIEDAM 10(4): 275-288.

[177] Umeda Y, Tomiyama T, Yoshikawa H, Sakao T, Shimomura Y, Tanigawa S; Mita Industrial Co., Ltd., assignee. Method of automatically creating control sequence software and apparatus therefore. US patent 194,064. 1994 Feb 9.

[178] Umeda Y, Tomiyama T. FBS modeling: Modeling scheme of function for conceptual design. In: Workshop on Qualitative Reasoning about Physical Systems. Amsterdam, The Netherlands; 1995, p. 271–278.

[179] Van de Laar P, Punter, T., Eds., 2011, *Views on Evolvability of Embedded Systems*, Springer.

[180] Van de Wal M, De Jager B, 2001. A review of methods for input/output selection. Automatica, 37, 487-510.

[181] Van Eck D, McAdams, D., Vermaas, P., 2007, “Functional decomposition in engineering: A survey,” *Proceedings of the ASME 2007 IDETC/CIE*, Las Vegas, Nevada, USA.

[182] Varsamidis T, Hope S, Jobling CP. Use of a prototype CACE integration framework based on the unified information model. IEEE Int. Symposium on Computer Aided Control System Design, 1999.

[183] Vitech corporation. CORE software website. <http://www.vitechcorp.com/products/Index.html>

[184] Voskuijl M, La Rocca G, Dircken F. Controllability of blended wing body aircraft. In: Proceedings ICAS of the Intern.council of the Aronaut.Sciences including the 8th AIAA Aviation Techn., Integr. and Operations Conf. Edinburg, UK: 2008.

[185] VRS ROPAX. Virtual Reality Ship systems project webpage. <http://www.vrs-project.com/index.phtml>

[186] W3C, 2004. OWL web ontology language recommendation. < http://www.w3.org/TR/owl-features/>

[187] Wang L, Shen W, Xie H, Neelamkavil J, Pardasani A. Collaborative conceptual design - state of the art and future trends. Computer-Aided Design 2002; 34: 981-996.

[188] Wasiak J, Hicks B, Newnes L, Dong A, Burrow L, 2010. Understanding engineering email: the development of a taxonomy for identifying and classifying engineering work. Res Eng Design 21: 43-64.

[189] Weilkiens T. Systems engineering with SysML/UML: Modeling, analysis, design. Morgan Kaufmann OMG press. 2007.

[190] Whitehead J. Collaboration in software engineering: A roadmap. In Proceedings of the International Conference on Software Engineering 2007, Future of Software Engineering (FOSE'07), p. 214-225.

[191] Wikander J, Törngren M, Hanson M. The science and education of mechatronics engineering. IEEE Robotics & Automation Magazine 2001; 8(2): 20-26.

[192] Wikipedia. Service-oriented architecture. Online in < http://en.wikipedia.org/wiki/ Service-oriented_architecture>. Consulted in January 2011.

[193] Woestenenk K, Alvarez Cabrera AA, 2009. Vanderlande case study report. Internal project report, cooperation with Vanderlande Industries.

[194] Woestenenk K, Alvarez Cabrera AA, Bonnema GM, Tomiyama T, 2011. Capturing design process information in complex product development. In Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011. August 28-31, 2011, Washington D.C.

[195] Woestenenk K, Alvarez Cabrera AA, Tragter H, Bonnema GM, FJAM van Houten, Tomiyama T, 2010. Multidomain design: integration and reuse. In Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2010. August 15-18, 2010, Montreal, Quebec, Canada.

[196] Wood W, Dong H, Dym C, 2004. Integrating functional synthesis. *AIEDAM*; 19(3): 183-200.

[197] Xu Y, Zou H, "Design principles for mechatronic systems based on information content," [online] in *Proc. IMechE*, Vol. 221, Part B, pp. 1245-1254, March 2007.

[198] Yen JY, and R. J. Lee, "A solid modeling based mechatronics approach to machine tool servo design," [online] in *Proceedings of the 2004 IEEE International Conference on Control Applications*, 2004, Vol. 1, pp. 730-735.

[199] Yoshioka M, Sekiya T, Tomiyama T, 2001. An integrated design object modeling environment - pluggable metamodel mechanism -. *Turkish Journal of Electrical Engineering and Computer Sciences*; 9(1): 43-62.

[200] Yoshioka M, Umeda Y, Takeda H, Shimomura Y, Nomaguchi Y, Tomiyama T, 2004. Physical concept ontology for the knowledge intensive engineering framework. Advanced Engineering Informatics; 18(2): 69-127.

[201] Ziegler JG, Nichols NB, 1943. Process lags in automatic control circuits. Transactions of the ASME 1943; 65: 433-444.

[202] Zieliński C, W. Szynkiewicz, K. Mianowski, and K. Nazarczuk, "Mechatronic design of open-structure multi-robot controllers," [online] in *Mechatronics*, 11(8), December 2001, pp. 987-1000.

Index

Curriculum vitae and list of publications

Professional profile

Mechanical engineer interested and focused on the areas of systems design, automation and control, manufacturing processes planning and analysis. Research experience on problems in industrial design processes. Experience with CAD/CAE software and programming in various languages.

Studies

PhD Delft University of Technology (TUDelft), Intelligent Mechanical Systems (IMS) group, Department of Biomechanical Engineering (BioMechE), Faculty of Mechanical, Maritime, and Materials Engineering (3mE),.

Start date, October 2007 – End date, October 2011

Thesis: Architecture-Centric Design: Modeling and Applications to Control Architecture Generation

Topics: Systems design and control theory, methods, and applications; architecture modeling; mechatronic design.

Project: Automatic generation of control software for mechatronic systems

Erasmus Mundus Master of Mechanical Engineering (EMMME) scholarship, Universidad Politécnica de Catalunya (ETSEIB) and INSA Lyon

Start date, September 2005 - End date, July 2007

Thesis: Position and force control in a didactic test bench of a hydraulically powered airplane’s aileron

Topics: Automation, robust control, PID control, hydraulic system's modeling

Mechanical engineering, Universidad Nacional de Colombia

Start date, February 1999 - End date, October 2004

Thesis: Adaptation of the manufacture for a functional silk screen printing automatic carousel press prototype

Topics: Automation, manufacturing, QFD, silk screen printing

Professional experience

Finished product plant engineer

Organization: ALFAGRES S.A..

Webpage: http://www.alfagres.com/

Start date 4/2005 end date 6/2005

Functions: As a new member in the engineering crew, my duties consisted on collaborating in the design and development of improvements and new procedures, mainly in the area of finished products, for the ALFAGRES plant. Early retirement to dedicate myself to study with the EMMME scholarship award.

Development engineer

Organizations: Universidad Nacional de Colombia, CEIF, COLCIENCIAS, Ind. Derjor Ltda.

WebPages: http://www.unal.edu.co/, http://www.ceif.unal.edu.co/,

http://www.colciencias.gov.co/, http://www.empresario.com.co/derjor/

Start date 7/2003 end date 10/2004

Functions: Development and construction of an automated station for a multi-station silk-screen printing press, as a part of the investigation project "Technologic design and development of the automation for multi-station silk-screen printing press, thermoforming machine and UV curing machine." (COLCIENCIAS code: 1101-08-11032). Project awarded with the second place at the IV symposium of industrial automation (Pontificia Universidad Javeriana, Bogotá, Colombia, August 27 / 2004).

Development Engineer

Organization: Universidad Nacional de Colombia.

Webpage: http://www.unal.edu.co/

Start date 2/2003 end date 11/2003

Functions: The work consisted in the design and construction of a braking device that uses the parasitic currents (Eddy currents) phenomenon, conceived for its use in testing equipment for small motors (less than 3hp). This work was carried along other two students, in the class of "Machine Design" of the "Universidad Nacional de Colombia". Project awarded with the first place under the category of scientific contribution in the XIV MACHINE AND PROTOTYPE SHOW (Universidad Nacional de Colombia, February 2004).

Publications

- Alvarez Cabrera AA, Erden MS, Foeken MJ, Tomiyama T. High level model integration for design of mechatronic systems. In: Proceedings of IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Beijing, China; 2008, p. 387-392. (http://dx.doi.org/10.1109/MESA.2008.4735736).
- Chmarra MK, Alvarez Cabrera AA, Van Beek T, D'Amelio V, Erden MS, Tomiyama T. D&C vs. C&D: The Divide and Conquer Strategy to Deal with Complexity in Product Design. In: Proceedings of IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Beijing, China; 2008, p. 393-398. (http://dx.doi.org/10.1109/MESA.2008.4735679).

- Foeken MJ, Voskuijl M, Alvarez Cabrera AA, Van Tooren MJL. Model Generation for the Verification of Automatically Generated Mechatronic Control Software. In: Proceedings of IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Beijing, China; 2008, p. 275-280. (http://dx.doi.org/10.1109/MESA.2008.4735662).
- Alvarez Cabrera AA, Erden MS, Tomiyama T. On the Potential of Function-Behavior-State (FBS) Methodology for the Integration of Modeling Tools. In: Proceedings of CIRP design 09 conference: competitive design, Cranfield, UK; 2009, p. 412-419. (http://www.cranfield.ac.uk/sas/cirp-design/).
- Alvarez Cabrera AA, Foeken MJ, Tekin OA, Woestenenk K, Erden MS, De Schutter B, Van Tooren MJL, Babuška R, Van Houten FJAM, Tomiyam T. Towards automation of control software: A review of challenges in mechatronic design. In: Mechatronics. (http://dx.doi.org/10.1016/j.mechatronics.2010.05.003).
- Foeken M, Alvarez Cabrera AA, Voskuijl M, Van Tooren MJL. Applying Knowledge-Based Engineering to Control Software Gereration. In: Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2010. August 15-18, 2010, Montreal, Quebec, Canada. (http://www.asmeconferences.org/IDETC2010/index.cfm).
- Woestenenk K, Alvarez Cabrera AA, Tragter H, Tomiyama T, Bonnema GM. Multi Domain Design: Integration and Reuse. In: Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2010. August 15-18, 2010, Montreal, Quebec, Canada. (http://www.asmeconferences.org/IDETC2010/index.cfm).
- Alvarez Cabrera AA, Foeken MJ, Woestenenk K, Stoot G, Tomiyama T, 2011. Modeling and Using Product Architectures in Industrial Mechatronic Product Development: Experiments and Observations. In Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011. August 28-31, 2011, Washington D.C.
- Alvarez Cabrera AA, Komoto H, Tomiyama T, 2011. Supporting co-design of physical and control architectures of mechatronic systems. In Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011. August 28-31, 2011, Washington D.C.

Certified knowledge

- Technische Universitat Munchen, spring school of systems engineering (2010)
- ECQA certified European innovation manager trainer course (2009)
- The University of Tokyo Global Center of Excellence for Mechanical System Innovation (GMSI), summer camp - engineering and its role in society (2009)
- Presentation skills course (2008)

- TOEFL score 270/300
- ECAES (Colombian higher education quality examination) test score 71/100 (2003)
- Basic knowledge of LabVIEW 7 Express and data acquisition (National Instruments seminary)
- Data storing with LabVIEW and Field Point supervisory control (National Instruments seminary)
- Basic pneumatics, electro-pneumatics and industrial communications knowledge (Festo seminary)
- Basic PLC programming (Festo seminary)
- Positioning systems basic knowledge (Festo seminary)
- Basic sensors knowledge (SENA course)
- Piping design with PDS 3D (TIPIEL/TECHNIP course)
- Quality management and ISO 9000 standards (SENA course)
- Virtual course for creation of technology-based industries (SENA course)

Acknowledgements

First and I'd like to thank my dear project colleagues, Aydin, Krijn, Maarten, and Mohsen. Together we developed, searched, wrote, and proofread most of the original material which has been compiled in Part I of this book. Then, I'd like to thank my promoter, Professor Tomiyama, first for providing the opportunity to participate in the project, second for his cunning insights on key research decisions which undoubtedly have shaped my work, and third for providing a consistent work environment in so many different ways. Also thanks to my office colleagues, Erika, Hitoshi, Magda, Suphi, Thom, and Valentina, which not only provided valuable discussions and advice, but also their friendly support through this journey. I also extend my gratitude to all other office colleagues in TUDelft, and to all the project members in academia and industry who provided additional advice and feedback on the multiple case studies. Without all your contributions this work would have not been fruitful, nor finished on time.

Last, I would like to thank all my family and my "extended family" back at my home country, in Delft, and through the entire globe: the amazing network of family, friends, and acquaintances which I have been so lucky to count on through all my life. Just to mention some, thanks to all of you, mom, dad, and siblings, Marta, Angelita, Pablo, Friedy, Ivan, Jorge, Steffie, Layla, Juan, Liz, Aurelie, Marcelo, Bea, Sergio, Claudia, Andreia, Donata, Fabiola, Alberto, Patrice, Milene… and to anyone whom I may forget to mention ;-)