CONSISTENCY, INTEGRATION, AND REUSE IN MULTI-DISCIPLINARY DESIGN PROCESSES

THROUGH AN ARCHITECTURE MODELING FRAMEWORK

PROEFSCHRIFT

ter verkrijging van

de graad van doctor aan de Universiteit Twente,

op gezag van de rector magnificus,

Prof. Dr. H. Brinksma,

volgens het besluit van het College voor Promoties

in het openbaar te verdedigen

op donderdag 6 maart 2014 om 16.45 uur

door

Krijn Woestenenk

geboren op 3 januari 1980

te Lochem

Dit proefschrift is goedgekeurd door de promotor Prof. Dr. Ir. F.J.A.M. van Houten en de assistent promotor Dr. Ir. G.M. Bonnema.

Consistency, Integration, and Reuse in Multi-Disciplinary Design Processes

Through an Architecture Modeling Framework

PhD Thesis

By Krijn Woestenenk at the Department of Engineering Technology (CTW) of the University of Twente Enschede, the Netherlands.

Enschede, 6 maart 2014

De promotiecommissie:

Prof. Dr. G. Dewulf	Universiteit Twente,voorzitter
Prof. Dr. Ir. F.J.A.M. van Houten	Universiteit Twente,promotor
Dr. Ir. G.M. Bonnema	Universiteit Twente,assistent promotor
Prof. Dr. Ir. B.R.H.M. Haverkort	Universiteit Twente
Prof. Dr. R.J. Wieringa	Universiteit Twente
Prof. Dr. T. Tomiyama	Cranfield University
Prof. Dr. J.J.M. Hooman	Radboud Universiteit Nijmegen

Keywords: System design, system architecting, stakeholders views, design automation, multidisciplinary research

ISBN: 978-90-365-3614-1 DOI: 10.3990/1.9789036536141

Cover image by Krijn Woestenenk

Printed by Ipskamp Drukkers

This work has been published using LaTeX 2ε and the Twentethesis documentclass.

Duurt even, maar dan heb je ook wat...

Summary

The development of modern (mechatronic) systems demands close cooperation between experts from multiple engineering disciplines. These disciplines each speak their own engineering language and can even have conflicting views of what the system is. This leads to a complex situation that impedes the communication between these experts, as well as the integration of design and analysis tools.

The main 'pressures' in improving multi disciplinary design are the improvement of system performance, time-to-market and cost. This can be improved by facilitating communication between stakeholders, modeling architectural concerns and other important design information, and facilitating capturing and reusing design knowledge across disciplines. These pressures, however, cannot be subject to direct scientific research, as experiments cannot be easily realized: We cannot let multiple companies perform multiple design processes to compare methods that aim to improve the pressures. Instead, the pressures are 'transformed' in the issues of **consistency**, **integration** and **reuse**, which can be evaluated through case studies. As such, the hypothesis for this thesis is: The consistency, integration and reusability of multi-disciplinary design processes can be improved by facilitating communication between stakeholders, modeling architectural concerns and other important design information, and facilitating capturing and reusing design knowledge across disciplines.

To help this communication, multiple stakeholders need to have a shared model to: share design decisions, create common understanding of the design process, share deliverables (workflow), and keep this shared information consistent. In such a model, stakeholders can make their own views to create models for their aspects of the system. The information in these views can be connected to other views, and even reused by copying or referencing. This facilitates the stakeholder gathering the information from multiple sources to do his job, or allow exposing his concerns so he can influence activities performed by experts of other disciplines.

The argument of this thesis is: there is a lack of a common language that can describe both the system architecture and information flows in the design process. Such a language should enable us to model the shared information between the designers and their models and the tools that they use. There is no unifying modeling language to exchange information across disciplines, across design process phases and across levels of granularity. This means we lack the means to make a model of all design aspects, especially the ones that cut across the design disciplines. We lack the means to model how information from the mono disciplinary design tasks is handed over to subsequent design tasks. And we lack the means

to provide a big picture of how mono disciplinary design information is connected to the abstract system wide concerns such as functionality, requirements and decomposition. The **Architecture Modeling (AM) Framework**, developed during research for this thesis aims at providing a solution to these points.

There are recurring modeling concepts used in a large set of methods. Think of designers' goals, requirements and constraints for their design tasks, and abstract system functionality and composition. These concepts will be abstracted into a new language, to be used to facilitate 'dialogue' between people from various disciplines. The resulting dialogue can be documented in **Architecture Models**, AMs, using a diagram editor also developed in this research.

The **Architecture Modeling Language** is an amalgamation of known modeling concepts, practical industrial research , and a lot of iterative programming of test applications in various languages. The (preliminary) end result is a practical tool that can be used to model much of the information in multi disciplinary design processes normally left to the tacit understanding of the people involved. Many of the modeling concepts in the AM language are similar to often–used concepts occurring in other methods (e.g. `Function`, `Parameter`, `Requirement`). The AM language casts a wider net than these individual methods by prescribing the syntax between these often–used concepts. As such, AM language can be used to apply many modeling methods directly, and create synergy between these methods as a bonus. As an example, think of a functional model coupled with an constraints model. Where the functional part states behaviors of a system, and the constraints part states the customer wishes for this behavior in the form of `Requirements`. Tunable `Parameters` and the `Entities` model what components execute these behaviors. Also, more thought is given to modeling the design process. Most methods focus on modeling systems and their behavior. While the AM language can be used for this, we also added concepts to define the design process itself in the form of `Design Tasks` and `Domain Entities`.

The amount of concepts in the AM language also implies that we need to specify multiple `views` per model, as putting in too many symbols in one `view` will make it unreadable. To ensure that the views will not become isolated models, the views are a representation of an underlying consistent and integrated data model, with relations to and from concepts in different views. This makes sure that hand over of important design information between stakeholders is formalized, and that information is stored consistently in a single location. This enables not only stakeholder/user specific views to exchange information across disciplines, across design process phases and across levels of granularity, but also the formulation of subsets of the model for specific aspects of design. In contrast to many existing methods – where a view often has a specialized goal (UML Class or Sequence diagram, IDEF0 to 14) – an AM `View` can contain references to all other concept types. Such a shared model can then be used in two roles, the meta model for the abstraction of other design information, and the reference model for the categorization, review and exchange of design information.

The AM language is usable by both humans and computers for the following reason; Let the humans do what they are good in – creative ideation, anxious to get results, and adaptive reasoning about a problem – and let computers do what they are good in – consistent and indefatigable work. Compensate human sloppiness and tendency to boredom

for difficult and long tasks, and compensate the computer's 'dumbness' and 'narrowmindedness' for only being able to work with a preprogrammed set of algorithms. As the AMs are machine readable, they can be used to support automation of information flow and information transformation to and from external sources – effectively partially automating the design process.

Partial automation becomes possible by the separation of automated design knowledge, model data, and model representation. This is a paradigm often used in software engineering, but is not often applied in design and (systems) engineering models, where the models are constructed to facilitate a specific engineering method (e.g. CAD/CAM, Matlab). Effectively, by not specifying the appropriate design or engineering method (or viewpoint) for each part of the model (or view), we are separating the design method from the model and the modeling language. This means knowledge bases – reusable design knowledge in the form of software algorithms – can be developed for a partial model, while the rest of the model will be ignored by this knowledge base.

In most other modeling methods that would be silly, why make a model of things not used in the method? However, as the AM can be used by multiple methods, this becomes a strength: automating and reusing knowledge when possible, and keeping the model 'freeform' and flexible when appropriate. The AM language barely contains prescribed mathematical/algorithmic methods. This has been a choice to prevent the 'Model of Everything' dilemma. The only mathematical constructs prescribed are sets and (un)directed graphs. This makes it possible for software to manipulate any AM model in terms of navigation, serialization, representation and editing, but does not prescribe the application of the model. However, knowledge can be easily automated in separate knowledge bases to extend the applicability of the AM framework to automate specific design processes, as demonstrated in the case studies of this thesis. The result is an application area from informal diagrams like mind maps, to automated design specification generation, and any point, and mix of points, in between.

Samenvatting

De ontwikkeling van moderne mechatronische systemen vereist nauwe samenwerking tussen experts van meerdere ontwerpdisciplines. Deze disciplines spreken allemaal in hun eigen 'taal' en soms hebben ze zelfs conflicterende standpunten over wat het systeem is of moet zijn. Dit leidt tot een complexe situatie die de communicatie tussen deze experts in de weg staat en voorkomt dat ontwerpgereedschappen met elkaar geïntegreerd kunnen worden.

De belangrijkste speerpunten in het verbeteren van multi disciplinaire ontwerpprocessen zijn de verbetering van de kwaliteit van een systeem, de time-to-market en de kosten. Deze kunnen worden verbeterd door het faciliteren van de communicatie tussen betrokkenen, het modelleren van systeem architecturen en ontwerpbeslissingen, en het faciliteren van kennismodellering en hergebruik tussen de ontwerpdisciplines. De speerpunten zelf echter, kunnen niet zomaar onderzocht worden in een wetenschappelijk kader: We kunnen niet meerdere bedrijven meervoudig een ontwerpproces laten doorlopen om methodieken te vergelijken. Daarom vertalen we de speerpunten in de onderliggende problemen met consistentie, integratie en hergebruik, welke we wel kunnen evalueren door middel van case studies. De hypothese van dit boek is dan ook: De consistentie, integratie en hergebruik in multidisciplinaire ontwerpprocessen kunnen worden verbeterd door het faciliteren van communicatie tussen betrokkenen, het modelleren van systeem architecturen en andere ontwerp informatie, en het faciliteren van kennismodellering en hergebruik tussen de ontwerpdisciplines.

Om deze communicatie te faciliteren moeten de belanghebbenden een gedeeld model hebben om: ontwerpbeslissingen te delen, een gezamenlijke begripsvorming over het ontwikkelingsproces te creëren en gedeelde informatie consistent te houden. In zo'n model kunnen belanghebbenden hun eigen interpretaties modelleren voor de systeemaspecten waar zij belang bij hebben. Deze informatie kan dan binnen dat model gedeeld worden door het te verbinden met informatie van andere belanghebbenden of door informatie op meerdere plekken te gebruiken. Een belanghebbende kan op deze manier van meerdere bronnen de informatie vinden die hij nodig heeft, en, vice versa, zijn belangen kenbaar maken zodat andere experts er rekening mee kunnen houden.

Het uitgangspunt voor dit boek is: Er is een behoefte aan een gemeenschappelijke taal die zowel de systeem architecturen als de informatie stromen in een ontwerpproces kan modelleren. Zo'n taal moet ons in staat stellen om de gedeelde informatie tussen belanghebbenden en hun modellen en hun ontwerpgereedschappen te kunnen vastleggen. Er is geen geünificeerde modellering taal om informatie te delen tussen ontwerpdisciplines, tus-

sen ontwerp fases of tussen verschillende niveaus van detaillering. Hierdoor kunnen we geen modellen maken van alle aspecten van het ontwerp, met name in het geval wanneer deze aspecten door meerdere disciplines gedefinieerd worden. We hebben geen modellering taal om de overdracht van informatie van ontwerptaak naar ontwerptaak vast te leggen. Tevens missen we de mogelijkheid om een overzicht te verkrijgen over hoe discipline specifieke informatie past in het grotere geheel van de systeem architectuur, zoals de gevolgen voor functionaliteit, ontwerp eisen of systeem structuur. Het **Architecture Modeling Framework**, ontwikkeld gedurende het onderzoek voor dit boek, probeert voor deze punten een oplossing te geven.

Verschillende modellering concepten komen voor in meerdere ontwerpmethodes. Denk hierbij aan ontwerpdoelen, eisen en voorwaarden voor ontwerptaken, systeemfuncties en component structuur. Deze concepten vangen we in een nieuwe taal, om de 'dialoog' tussen mensen uit de verschillende disciplines te faciliteren. Deze dialoog kunnen we dan vastleggen in **Architecture Models** met een editor die ook in dit onderzoek is ontwikkeld.

De Architecture Modeling Language is een synthese van veel voorkomende modellering concepten, praktijkonderzoek in de industrie, en het programmeren van testapplicaties. Het voorlopig eindresultaat is een praktisch gereedschap dat we kunnen gebruiken voor het modelleren van veel van de informatie in multidisciplinaire ontwerpprocessen die normaal alleen in het hoofd van de betrokkenen blijft zitten. Veel concepten van de AM language komen overeen met concepten die worden gebruikt in andere methodieken (`Function`, `Parameter`, `Requirement`). De AM language gooit echter een wijder net uit dan deze individuele methodieken door de mogelijke relaties tussen al zulke concepten vast te leggen. Derhalve kun je de AM language gebruiken om veel van die methodieken direct te gebruiken, maar ook synergie tussen de methodieken te creëren. Denk bijvoorbeeld aan een functioneel model, gekoppeld aan een randvoorwaarden model. In het functionele gedeelte kan je het systeemgedrag modelleren en in het randvoorwaarden gedeelte kun je de klantwensen van dit gedrag vastleggen in de vorm van `Requirements`. `Parameters` en `Entities` kunnen modelleren welke componenten het gedrag uitvoeren. Tevens is er meer aandacht gegeven aan het modelleren van het ontwerpproces. De meeste methodieken focussen op het modelleren van het systeem of het systeemgedrag. De AM language kan hiervoor gebruikt worden, echter, we hebben ook concepten toegevoegd om het ontwerpproces zelf te beschrijven in de vorm van `Design Tasks` en `Domain Entities`.

De hoeveelheid concepten in de AM language maakt het noodzakelijk om meerdere `Views`(weergaven) in een model te kunnen modelleren, omdat een model met teveel symbolen in een enkele weergave onleesbaar wordt. Om te voorkomen dat een view geen geïsoleerd model op zichzelf wordt, zijn views deel van een onderliggend consistent en geïntegreerd datamodel, met relaties van en naar andere views. Op deze manier wordt de overdracht van informatie tussen belanghebbenden geformaliseerd en wordt de informatie eenduidig en centraal opgeslagen. Dit maakt het mogelijk dat betrokkenen de informatie in hun view kunnen delen met anderen, dat een view een verzameling informatie kan bevatten uit allerlei disciplines, ontwerpproces fases en detailniveaus, maar ook doorsnijdingen van het model voor specifieke aspecten van het systeem. In tegenstelling tot veel andere methoden – die per viewtype een bepaald doel hebben (UML klassediagram, IDEF0-14) –

kan een AM `View` alle mogelijke concept types bevatten. Zo'n gedeeld model kan vervolgens op twee manieren gebruikt worden: Als meta model voor de abstractie van andere ontwerpinformatie en als referentie model voor het categoriseren, beheren en uitwisselen van ontwerpinformatie.

Deze AM language is leesbaar voor zowel mensen als computers, en wel om de volgende reden; laat mensen doen waar ze goed in zijn – creatief ideeën formuleren, een goed resultaat willen hebben, snel schakelen en beredeneren om problemen op te lossen – en laat computers doen waar zij goed in zijn – consistent en onvermoeibaar werken. Compenseer menselijke slordigheden en neiging tot verveling bij lastige en lange taken, en compenseer de domheid van computers die alleen voorgeprogrammeerde algoritmes kunnen afdraaien. Omdat de AMs door de computer kunnen worden gelezen, kunnen ze gebruikt worden om informatie uitwisseling en transformaties van en naar externe bronnen te bewerkstelligen – op deze wijze kan men een deel van het ontwerpproces automatiseren.

Gedeeltelijke automatisering van ontwerpprocessen wordt mogelijk door het scheiden van de automatische ontwerp algoritmes, de model data, en de model representatie. Dit is een paradigma dat veel gebruikt wordt in software engineering, maar niet vaak wordt toegepast in ontwerp en engineering modellen, waar de modellen worden gemaakt voor en door een specifiek engineeringproces (bv. CAD/CAM, Matlab). In AMs specificeren we echter niet voor welk doel(belanghebbende) een bepaald stukje van een model (view) moet worden gemaakt. Zo scheiden we de ontwerpmethodiek van het model en van de modellering taal. Op deze manier kunnen we knowledge bases ontwikkelen – herbruikbare ontwerpkennis, gevangen in algoritmes – die alleen op bepaalde stukken van een AM zullen inwerken, terwijl de rest van het model met rust wordt gelaten.

In veel andere methodieken zal dit een onzinnige werkwijze zijn, waarom zou je een model maken van zaken die niet binnen een bepaalde methodiek vallen? Echter, omdat AMs door en voor meerdere methodieken gebruikt kunnen worden, wordt deze eigenschap een sterk punt: automatiseer processen wanneer mogelijk en houd het model flexibel en informeel wanneer dat beter uitkomt. De AM language zelf bevat nauwelijks mathematische of algoritmische bewerkingsmethoden. Dit was een keuze om het 'Model Van Alles' probleem te voorkomen. De enige mathematische constructen in de taal zelf beslaan sets en (un)directed graphs. Dit maakt het voor software mogelijk om elk AM model te manipuleren voor bijvoorbeeld navigatie, serialisatie, representatie en bewerking, maar zegt niets over de toepassing van het model. Echter, het is vrij gemakkelijk om losse knowledge bases te programmeren die de functionaliteit van het AM framework uitbreiden om specifieke ontwerpprocessen te automatiseren, zoals wordt gedemonstreerd in de case studies van dit boek. Het resultaat is een toepassingsgebied van informele diagrammen zoals mindmaps, tot aan het automatisch genereren van ontwerpspecificaties en elk punt en mix van punten daartussenin.

Preface

Thank you for reading this thesis. In this preface, I would like to introduce myself. I was born in 1980 in Lochem, a town in the Achterhoek (the Back Corner). I studied Mechanical Engineering at University of Twente in the Netherlands, where I did my masters research on the abstract information structures underlying design processes. During my studies, I often helped out the university by building automatic design tools. Examples are the automatic generation of springs and gears in SolidWorks assemblies, and optimized geometry and finite element models for aluminum extrusion dies.

After that, I had a short career as a software engineer, in which capacity I helped build tooling to generate industrial plant geometry and bills of material from functional models. However, I still wanted to go back to the university. I got this chance when Hans Tragter of the university of Twente asked me to become a PhD student for the Automatic Generation of Control Software for Mechatronic Systems project. In this project, I could bring my knowledge of building software tooling for knowledge based engineering, and learned a lot from my colleagues about mechatronic systems. The result of this period in my life, you are reading now.

The four years of being a PhD student passed quickly, and I found I did not have enough time to finish my thesis. As a result, I had to finish it in combination with my new and current job at Demcon Advanced Mechatronics. Demcon is a high-end supplier of technology and dedicated as a company to the development and production of mechatronic applications. Focus areas are high-tech systems and medical devices. Demcon is specialized in the integration of several disciplines (including mechanics, electronics, informatics, control engineering, physics, optics, materials science and sensor technology). In other words, a very interesting company for someone with my background.

At Demcon, I started out as a mechanical engineer. However, it soon became apparent that I could be more useful in another role. Currently, my function is business analyst. In this capacity, I work to facilitate information flows throughout the company. This mainly concerns the deployment of product data management software for the engineers, and enterprise resource planning software for the logistics, project management, and finance departments. The remainder of my time, I spend in automatic model transformations to reduce the workload for engineers, and increase the consistency and quality of design documentation. I also help out with the IT, supporting the system administrator of the company.

What I found during these years, is that ICT is still rapidly changing the way we find, use, and share information. While everyone uses a computer, only a few people use it to

build macros, or small software tools to make processes easier, faster, or clearer. People still tend to think inside the box of their own disciplines and (software) tooling, while a lot of synergy can be created by looking at the integration of information flows. This thesis is an explanation of how I think information flows can be more effective. After reading it, I hope you have picked up some awareness of the issues, and hopefully, some ideas for possible solutions.

November 2013,

Krijn Woestenenk

Contents

1. Introduction

This chapter discusses the positioning of this thesis. We will introduce the Automatic Generation of Control Software for Mechatronic Systems Project project, of which this research was a part. An outline of the thesis will conclude the chapter.

Design of modern (mechatronic) systems is becoming increasingly complex. Companies must continuously reduce time-to-market while increasing the cost, quality, diversity, and functionality of their products (§2.1). As a result, more and more specialists from various disciplines are needed to develop such products. Think of systems engineers/architects that develop a viable concept, mechanical, electrical and software engineers to develop a working system, production engineers to make a system manufacturable, and purchasing and procurement specialists to make a system a marketable product (§2.3).

These disciplines each speak their own engineering language and can even have conflicting views of what the system is. This leads to a complex situation that impedes the communication between these experts (figure 1.1). As a result, it is difficult to keep the information flowing throughout the design process **consistent**. Because each domain has its own specialists, design tools, and modeling languages, and every new generation of products has a greater product variety and added product functionality, we see an increase in the complexity of **integrating** the design work into a working system. Furthermore, time-

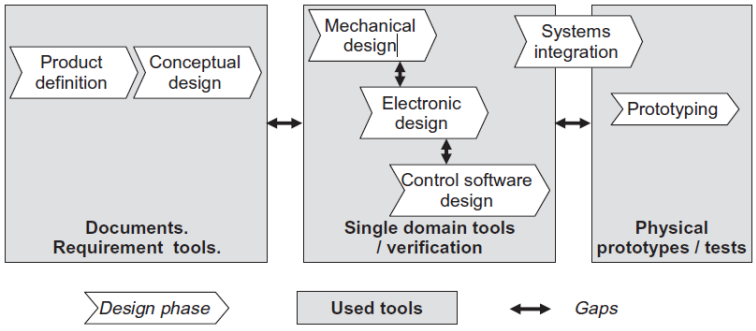


Figure 1.1: Simplified representation of a multi disciplinary design process. With isolated automation inside disciplines. The gaps represent manual data transfer. From research project proposal [Tomiyama et al., 2008].

Figure 1.2: Modern mechatronic products, requiring multi disciplinary design approaches. An Océ print engine, a Vanderlande baggage handling system, an ASML ultraviolet lithography concept.

to-market constraints drive the need for **reuse** and standardization of design artifacts and knowledge into reusable design objects, to speed up the development process (§2.4).

To find out the reasons for difficulties in design processes, and to propose possible mitigations, we will explore multi disciplinary designing in both state of the art modeling techniques and the industrial practice (chapter 3).

The argument of this thesis is: there is a lack of a common language that can describe both the system architecture and information flows in the design process. Such a language should enable us to model the shared information between the designers and their models and the tools that they use to make and use these models. There are recurring modeling concepts used in a large set of other methods. Think of designers' goals, requirements and constraints for their design tasks and the models they use, and abstract system functionality and composition. These will be abstracted into a new language, to be used to facilitate 'dialogue' between people from various disciplines. The resulting dialogue will be documented in models using a diagram editor also developed in this research (chapter 4).

This language thus is used by both humans and computers; Let the humans do what they are good in – creative ideation, anxious to get results, and adaptive reasoning about a problem – and let computers do what they are good in – consistent and indefatigable work. On the flip side compensate human sloppiness and tendency to boredom for difficult and long tasks, and compensate the computer's 'dumbness' and 'narrowmindedness' for only being able to work with a preprogrammed set of knowledge.

Models constructed using this language should enable integration of information flows, (goal) overview and consensus for the design process, and consistency of the used information. As the models are machine readable, they can be used to support automation of information flow and information transformation – effectively automating the design process partially(chapter 5).

1.1 Automatic Generation of Control Software for Mechatronic Systems

This research was conducted in the context of the "automatic generation of control software for mechatronic systems" AGCSMS project. This title could be deceptive, because the project goal is not to develop another code generator. The project originally aimed to automate the design process *in order to* generate control software. The software generation itself can then be done with existing tools. Automation across design disciplines requires an abstraction that can be interpreted by all disciplines. An abstract language must fill the logical gap between the disciplines, and an abstract meta and or reference model must provide the disciplines with the instructions of how to build domain-specific models with existing tools.

The project team had expertise in topics including function modeling, multiple model management, qualitative physics, mechatronics features, product development processes, knowledge based engineering, model generation, model and knowledge based control. This project's aim was to develop a set of prototype tools and a framework with which a interdisciplinary product development team can (almost) automatically generate control software for mechatronics machines. This framework is shown in figure 1.3.

In collaboration with industrial partners, the project demonstrated automatic generation of models, beginning with functional information defined at a very early stage of the development and ending with validation of the generated models. Its ultimate goals were to demonstrate the feasibility of improved product development practice in which control software can be developed in a concurrent manner and the quality of software can be improved. In this sense, it is a project to establish a best practice in mechatronics product development.

Finally, the project achieved the development of a set of prototype tools to implement the developed method. For this reason, we also have a dissemination (valorization) plan in which industrial partners will validate the tools and methods. This plan also foresees in further development of the framework in association with other academic and industrial partners.

Within this project, I have mainly assumed the role of bringing together the high abstraction modeling techniques of one project partner [Alvarez Cabrera, 2011], with the design automation of another [Foeken et al., 2010]. This led to a focus of research on issues that are normally out of scope for specific research disciplines: the consistency, integration and reuse in multi disciplinary design processes. As an answer, much of the delivered framework, its modeling language, and its tooling, and two of the case studies, §5.4 and §5.5, were my contributions to the project, as well as developing much of the tooling around the case study in §5.6. These topics will be discussed from the perspective of consistency, integration and reuse, as is represented in the thesis layout in the next section.

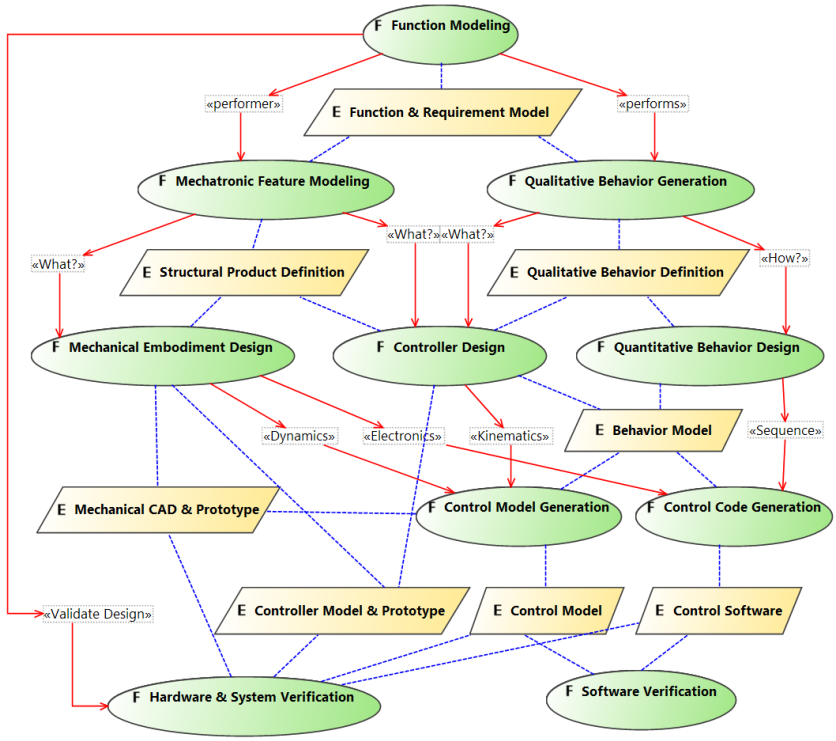


Figure 1.3: The 'Tomiyama Tree', the guiding figure in the AGCCSMS project. It shows the goal of building a mechatronic system from high level, highly abstract information. This figure is made with the Architecture Modeling tool, developed in this research.

1.2 Thesis Outline

The thesis outline is depicted in figure 1.4 and table 1.1. There we see both the structure of the thesis and a short clarification. We will follow this reasoning:

In this research we consider modern (mechatronic) product development, which has System Performance, Time-to-Market, and Cost as the key drivers for companies (§2.1). We will identify the design (and engineering) process as the part of product development where we can best induce improvements in these key drivers (§2.2). Then, using both literature and case study observations (discussed later), a set of issues and challenges is deduced that obstruct improvement of product development (§2.4). These issues are reshaped in a hypothesis (§2.8).

Of course, many individual issues have been identified and addressed before by others. However, there are not many examples of taking a step back and looking at their interconnectedness. In a literature study, we will make an inventory of available methods and tools for tackling the issues (§3.1 to §3.6). We will attempt to make a generalization of many of the reviewed methods, and identify possible gaps between them (§3.7).

These gaps are then the requirements for a proposed extension to the body of methods and tools. This will consist of a generic modeling approach and a set of tools to facilitate this modeling and apply these models in the design process (§4.1 to §4.4). The result is a framework that is intended to fill the gaps between existing methods and tools, as well as to address the formerly defined issues (§4.5).

Where the framework is the (preliminary) end result of the research, we will also discuss its creation (§5.1) and validation (§7.4) process. In a set of iterative case studies (§5.4 to§5.6), the industry-as-a-lab method for scientific research was applied (§5.2). These case studies advanced the insight into real-world issues and challenges (§2.4), while expanding on the framework tooling and its modeling capacities (§4.2 and §4.3). In the discussion, the framework efficacy and novelty will be reviewed (chapter 6). An analysis of the chosen methodology and wider implications of the framework in terms of the hypothesis will conclude the thesis (chapter 7). As this thesis is part of a wider research effort, we will end with the possible next phase of framework development and further research opportunities (chapter 8).

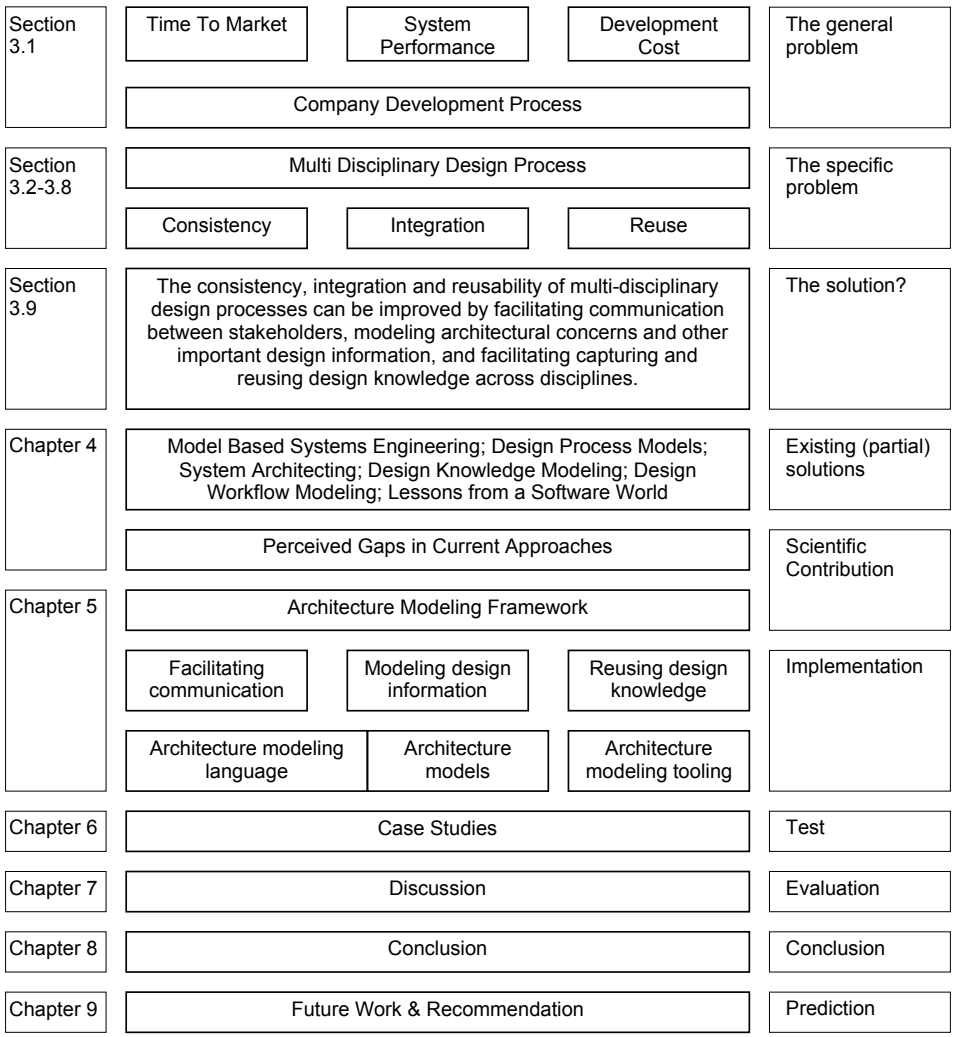


Figure 1.4: Overview of the main topics in this thesis.

Table 1.1: Thesis content summary and overview

Location and Goal	Summary
2. Problem definition What is the thesis context?	Modern (mechatronic) product development. System Performance, Time-to-Market, and Cost. The design process is key to improving overall product development. Definition of a multi disciplinary design process.
2. Problem definition What are the issues?	Consistency of information flow in design process. Integration of design information from various disciplines. Reuse of design information and knowledge.
2.8. Hypothesis How can issues be resolved?	Facilitating communication between stakeholders. Modeling architectural concerns and other design information. Facilitating capturing and reusing design knowledge across disciplines.
2.9. Scope What do we do?	In scope: See hypothesis, model information (flow) of a design process. Outside scope: Non-formal modeling. Outside scope: Specific algorithms for specific problems. Outside scope: Algorithm (knowledge) modeling.
3. State of the Art How did others do it?	Model Based Systems Engineering Design Process Models. System Architecting. Design Knowledge Modeling. Design Workflow Modeling. Software Modeling and Development.
3.7. Gaps in state of the art What is the scientific contribution?	Circumvent 'model of everything' dilemma. Define small set of concept types usable in multiple design methods. Model exchanged information across disciplines, design process phases and levels of granularity. Multiple views per shared model, based on stakeholder concerns. Formalize hand over, between humans as well as between software tools. Separate the design methods from the model and the modeling language. Automate knowledge reuse when possible, keep model 'tacit' and flexible when appropriate.
4. Framework What does this research add?	Architecture Modeling Framework. Architecture Modeling Language to facilitate communication. Architecture Models to model design information. Architecture Modeling Tooling to reuse design knowledge.
5. Case studies How does framework work in practice?	Industry-as-a-lab method. Structural Model of Baggage Handler. Software Generation of a Lithography System. Paper Path Design of Printer Copier.
6. Discussion Does the framework work?	Comparison with other solutions, and coverage of other issues. Reflection on chosen research method. Applicability of framework in other situations.
7. Conclusion Test hypothesis	Improve consistency with multi disciplinary model in which to track the 'who, what, when, where, why, by what means' design information. Improve integration with multi disciplinary model that maintains overview and context between the views containing stakeholder specific design information. Improve reuse with multi disciplinary model that facilitates (partial) construction of the design specification by applying knowledge bases. Other possible framework embodiments.

2. Problem Definition

Modern product development becomes an increasingly difficult and complex activity. In this chapter we will see that system performance, time-to-market and cost are the pressures that drive this increase in difficulty, and that managing the design process is a way to mitigate the pressures. An analysis of how to improve system performance, reduce time-to-market and reduce development cost by means of the design process will show that this is, among other things, an issue of guarding consistency, facilitating integration, and reusing design information, which will be the main topic of this thesis. We will continue with laying some definitions for design and design processes, and what is complex about them, to clarify the terminology of the next chapters. Then follows a more in depth analysis of the relation between system performance, time-to-market, and cost versus consistency, integration, and reuse. We will enumerate a set of issues surrounding consistency, integration, and reuse as these were perceived by research at industrial partners and literary sources. The chapter will close with a hypothesis on how we can resolve these issues, and a scope definition.

2.1 The Relation Between Design Process, System Performance, Time-to-Market, and Cost

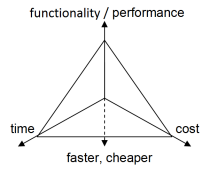
As we can see around us nowadays, increasingly complex systems are being developed in industry (hybrid cars, smart phones, computers, etc.). These provide more and more functionality, needing more and more engineering disciplines and people to develop – both in the design process and in the manufacturing process. This phenomenon gives rise to a trend in the development companies: the large number of people and disciplines involved in the

Table 2.1: Top 5 company pressures to improve mechatronic product development [Boucher and Houlihan, 2008]

Pressure	**Response**
Shorter product development schedules	69%
Increased customer demand for better performing products	44%
Reduced development budgets	25%
Accelerated product customization	20%
Increased requirements to incorporate electronics and software into product	16%

	Essential	Not essential	Not sure
Works as it should	98	1	1
Lasts a long time	95	3	2
Is easy to maintain	93	6	1
Looks attractive	58	39	3
Incorporates latest technology	57	39	4
Has many features	48	47	5

(a)



(b)

Figure 2.1: What is system performance? (a) Quality as observed by the customer is a good indicator of system performance (from [Ullman, 2003]). (b) More functionality or better performance of a system costs more and takes more time to develop.

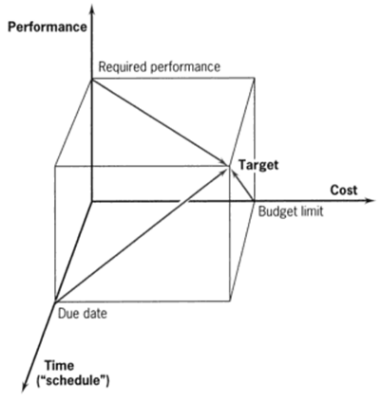


Figure 2.2: There is a space within which projects need to be conducted, bounded by performance, cost and time (from [Meredith and Mantel, 2008])

design process leads to an increase in the complexity of the product *and* in the complexity of the design process. For a company, the increase of the complexity accumulates with pressures by the market demand: People want a better product, they want it cheaper, they want it tailored to their needs, and they want to be the first to have it (see table 2.1 or the introduction of [Ullman, 2003]).

These pressures can be summarized in an organizations' desire to increase system performance (figure 2.1(a)) while at the same time reduce the time and cost of system development (figure 2.1(b)). [Meredith and Mantel, 2008] shows the relation between project or system performance, lead-time or time-to-market, and project cost as they apply to project development (figure 2.2). This performance of the project (or the delivered system or product) is measured as the agreement of the project with the requirements that the organization and the customer have.

In this thesis we see the development process as a combination of all activities needed to deliver a product or project, such as ideation, business analysis, project staffing and scheduling, intellectual property investigation, beta testing, marketing, (technical) design, engineer-

ing, manufacturing, logistics, commercialization of prototype, etcetera... Within this development process, we discern the design process as the process where requirements are translated into a design specification. This means that for this thesis, engineering is part of the design process, as are other aspects, such as design for manufacturability, life cycle analysis and design for maintainability, etcetera. (A definition for the design process follows in §2.3)

Somehow, organizations need to mitigate the 'pressures' of system performance, time-to-market and cost as expressed in table 2.1. This can most effectively be done at the design process of the larger development project, because, in this design process, the customer requirements are translated into specifications for the construction of the system that will fill these requirements, which means the performance of a system is to a large extent determined at the design process (see figure 2.5). Furthermore, the cost of the design process is often low compared to the other development stages (see figure 2.3(a)) while the decisions taken in the design process have huge implications for the total cost of the total development process (see figure 2.3(b)).

Time-to-market is often an issue for generating revenue. Reducing lead time of the development process by a better design process can put the product on the market before the competition, leading to increased revenue, because the customer will accept a higher price for the system, and thus a higher profit margin for the organization. Later, with more competitive products on the market, the sales will be spread over more suppliers, thus lowering revenue (figure 2.4). In that case, an organization needs to increase system performance as specified in figure 2.1(a), repeating and evolving this cycle of product development. One way to keep ahead of the competition is thus a relatively better development process, and in particular a better design process.

Focusing more attention on the design process also means managing various non customer requirements – such as in Design for X – early on in the development process. This should make sure the project leads to a manufacturable, maintainable, evolvable, etcetera, system, and one that performs better than that of the competition. In figure 2.5 this is plotted in a number of curves. Design decisions taken early on have a big impact on the system performance and capabilities. When these decisions are later proven to lead to unfeasible or undesirable system characteristics, it can be very costly to make changes. Therefore, a design process should aim to take the right decisions early on, verifying those decisions by checking the various requirements.

2.2 *Guarding Consistency, Facilitating Integration, and Reusing Design Information*

Resuming from the previous section: the design process is the key in reducing time to market and cost, and increasing system performance. Analysis of three well known methodologies will show that this is, among other things, an issue of guarding consistency, facilitating integration, and reusing design information. First, a meta analysis from [Boucher and Houlihan, 2008] (see table 2.2) shows the most important strategies that successful or-

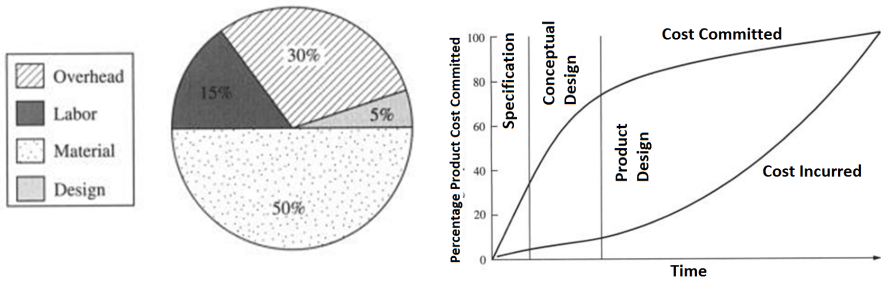


Figure 2.3: Relation between the design process and system cost (from [Ullman, 2003]). In (a) We see that design is a relatively small contribution to the total system cost (from ford motor company). However, in (b) we see that much of the system total cost is already committed in the design process. This means that when we want to influence system cost, it has to be done at the design process stage.

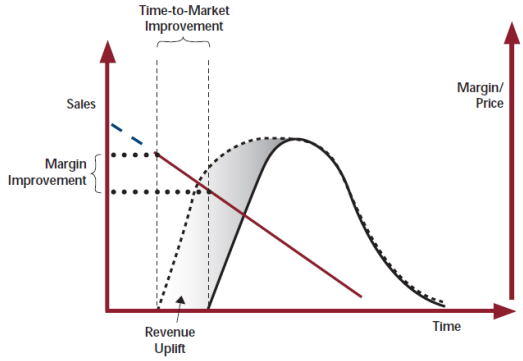


Figure 2.4: Relation of time-to-market and revenue adapted from [Proud and Wetzer, 2003], which states "By improving time to market, the sales curve shifts up and to the left, and profit margin is increased through price and expanded market share."

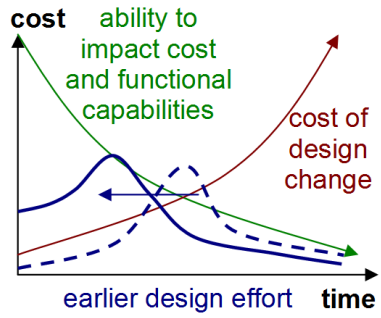


Figure 2.5: Take good design decisions early. Reduce the lead time of the design process, by doing more analysis early on, and doing more testing on models instead of prototypes (adapted from [MacLeamy, 2004] and [Ullman, 2003]).

Table 2.2: Top 5 actions Best-in-Class companies apply to improve mechatronic product development [Boucher and Houlihan, 2008]

Actions	Response
Improve communication and collaboration across disciplines	71%
Increase visibility into status of requirements	49%
Increase ability to predict system behavior prior to testing	46%
Implement or alter new product development processes for a multi-disciplinary approach	43%
Increase real time visibility of product Bill of Materials throughout the development process	39%

ganizations have employed to improve mechatronic product development in response to the performance, time-to-market, and cost pressures they are facing (presented in table 2.1). The actions mainly aim at better communication: **Integrated** communication of information across disciplines and project progression. **Consistent** control of the process by controlling system behavior, measuring requirements, and continuous insight in the status of the process deliverables (Bill of material in table 2.2, but could be other models).

In [Mulbury Six Sigma, 2011]: "DFSS stands for Design For Six Sigma – an approach to designing or re-designing a new product and/or service for a commercial market, with a measurably high process-sigma (ed. low and centered variability) for **performance** from day one.". Design For Six Sigma is a methodology to increase control over the quality of the development process by controlling the design process better. It is a continuous process of 'Define, Measure, Analyze, Improve, Control' of the developed systems (or components thereof). To be able to practice Six Sigma, the development process should be measurable, and feed back results to the design process. In the design process, the information from the various measurements should be **integrated** with the (multi-disciplinary) causes of these measurement results. Any improvements in the design should thus lead to better measurement results later on. However, it is preferable to make the right decisions *before* the product or system is developed. For this purpose six sigma has tools such as QFD – Quality Function Deployment, FMEA – Failure Mode Effect Analysis, DOE – Design Of Experiment, and simulation techniques. Therefore, Six Sigma mainly aims at improvement of the **consistency** of the design specification with the customer requirement.

Lean Product Development [Walton, 1999]: "Lean is the search for perfection through the elimination of waste and the insertion of practices that contribute to reduction in **cost** and **schedule** while improving **performance** of products." An important aspect of getting the whole process lean, is the reduction of waste in the design process, as this is where many important decisions for generating waste are taken, and because the design process itself may not always be that lean itself. This is reflected in [Noor, 2007] (or in a short presentation of [Badgerland Users Group, 2007]) expressed in the following points:

- Hand-offs: cause information losses, accountability and ownership losses.
- External quality reinforcement: inspection by others takes away ownership of the quality and adds waiting time for inspection.

Table 2.3: Top Business Pressures and Strategies for Digital Product Development [Aberdeen Group, 2007]

Business Pressures		**DPD Strategies**	
Demand for more products	53%	Increase engineering efficiency and throughput	69%
Decreased product development budgets	51%	Streamline entire product development	64%
Requirements for higher complexity products	28%	Decrease manufacturing errors and cost	26%
Increased customer sensitivity to product quality	24%	Improve product quality and/or performance	22%
Increased competition	16%	Earlier verification of product with costumers	16%

- Waiting: wait on others to supply data, decisions, resources, etc... major source of long queues and long lead times.
- Transaction waste: Non-value adding information is exchanged. For example, unused documentation, long negotiations, bringing in experts on the problem too late.
- Re-invention waste: Solving the same problem repeatedly, no reuse of existing solutions. Often caused by lack of learning or knowledge transfer.
- Weak schedule discipline: dangerous in concurrent design, worsens task inter-arrival time variation and thus lengthens cycles, and ultimately lead time.
- High process and arrival variation: Causes long queues and thus lead time.
- System over-utilization: as utilization reaches capacity, cycle times balloon non linearly, causing high process and arrival variation.
- Large (information) batches: create variation in process capacity and task inter-arrival times.
- Redundant tasks: repeated across design process steps, while nothing has changed.
- Stop and go tasks: frequent shut down and set up diffuses attention, eg. engineer with multiple concurrent tasks.
- Unsynchronized concurrent work: concurrency can shorten lead time, but if results diverge, can lengthen lead time, check result with goals.

As the lean development points show, much depends on good communication between the participants in the process. Important points are: **integrate** the information flow as much as possible to prevent waste (transaction/hand-off/redundancy) and provide the means to control the flow (wait/stop go/concurrent), and thus keep the information **consistent** (quality/redundancy). **Reuse** information where possible (directly to prevent transaction waste, or re-invention, indirectly by knowledge transfer to others).

Digital Product Development (DPD) is the use of computers in the product development process. DPD tools are for example Computer Aided Design/Drafting (CAD), Computer Aided Engineering (CAE), Computer Aided Manufacturing (CAM), Project Data Management (PDM), and Enterprise Resource Planning (ERP). As can be seen in table 2.3 from [Aberdeen Group, 2007], the application of Digital Product Development aims at a shorter **time-to-market**, product development at lower **cost**, and a better **performance**. DPD is used mainly in two distinct ways: First, to **reuse** specialized knowledge – such as in CAD, CAE and CAM, where the tools provide algorithms that replace complex manual design tasks,

Table 2.4: Reasons for using Digital Product Development [Aberdeen Group, 2007]

Key Business Value Findings
Best in class manufacturers hit their revenue, cost, launch date, and quality targets for 91% or more of their products. By building one fewer prototype than all others, top performers gain between a 13 to 99 day time to market advantage and spend between USD. 7,600 and USD. 1,200,000 less in development costs depending on product complexity. By executing 5.4 fewer change orders than all others, top performers gain a 51 day time to market advantage and spend between USD. 8,000 and USD. 32,000 less in development costs depending on product complexity. Best in class performers have a 16% higher rate of design reuse than laggards.
Implications & Analysis
Top performers are 19% to 22% more likely than laggards to digitally prototype a product's performance in all phases of product development. Top performers are 34% more likely to use digital communications between engineering and manufacturing instead of paper. Top performers are 35% more likely to assess a product's manufacturability prior to design kick-off. Top performers are 43% more likely to use electronic forms and twice as likely to use lifecycle states and workflows to notify development stakeholders that information is ready to use.

and users define reusable components to design quickly and consistently. Second, to **integrate** information flows and keep that information **consistent** – such as in PDM and ERP, where the tools keep track of the resources, models and documentation in the design process. Also according to the empirical study from [Aberdeen Group, 2007], using DPD will lead to products more **consistent** with customer requirements, as more aspects of the product can be tested digitally before they are physically prototyped. [Aberdeen Group, 2007] further shows that DPD is an effective way to achieve a good system performance, time-to-market, and cost in table 2.4.

This section showed the main 'pressures' in improving multi disciplinary design are the improvement of system performance, time-to-market and cost.The pressures, however, cannot easily be subject to direct scientific research, as experiments on measuring these pressures are not possible (see also chapter 5 Case Studies): We cannot let multiple companies perform multiple design processes to compare methods that aim to improve the pressures. Instead, the pressures are transformed in the issues of Consistency, Integration and Reuse, which *can* be 'measured' through case studies. This is defendable because, as the referenced literature shows, there is a clear relation between the system performance, time-to-market, and cost and the consistency, integration, and reusability in the design process. However, some more explanation on these relations will be necessary, and will be resumed in §2.5 Ensure Consistency to §2.7 Facilitate Reuse. Before going into more detail on consistency, integration and reuse, the next section will define some important terminology that will be used for the rest of the thesis.

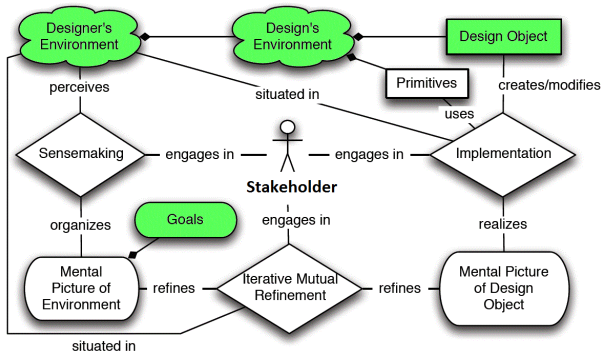


Figure 2.6: A stakeholder in a design process adapted from [Ralph and Wand, 2008]. The green items are shared between the stakeholders and should be modeled in a central location.

2.3 A Definition for a Complex Design Process

2.3.1 Defining the Design Process

Most systems are too big to be understood by a single person within a reasonable timespan. This is why many people from many disciplines and even multiple companies are needed to design a system within a profitable time-to-market period. All these actors involved in the design process, the engineers, managers, suppliers, designers, customers, etcetera, we will define as **stakeholders** in the design process (see figure 2.6). They are stakeholders, because apart from wanting a functional system, they can have other 'stakes' and/or concerns in this process. Finding out, modeling and analyzing what these stakeholders want, is called requirements engineering [Nuseibeh, 2000]. The measure in which the various system or system design aspects will correspond to the stakeholder requirements will therefore determine the system's performance.

The stakeholders within a design process must work together to construct an understanding of a (not yet existing) system in a design. [Ralph and Wand, 2009] distills a definition of design and designing from 29 definitions used in other literature. They define a design as *"(noun) a specification of an object, manifested by an agent, intended to accomplish goals, in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to constraints; (verb, transitive) to create a design, in an environment (where the designer operates)"*. In this thesis, the word 'design information' will be used to refer to these concepts in general, the specific definitions are as follows:

- *Design object*: The artifact, system or process being designed.
- *Agent*: The subject of the design object, how it is perceived by the designers (or software design tools).
- *Primitive components*: Indivisible parts, elements, software code, energy, thoughts or ideas that assemble the design object or are transformed to the design object.

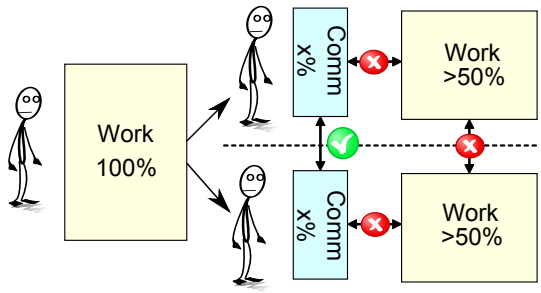
- *Specification*: Physical, symbolic and/or mental representation of a design object's structural properties, and the composition of the primitives.
- *Goal*: Informal or explicit goal, purpose, or objective for which the design is made.
- *Environment*: Organization that designs, or the environment where the design object will function.
- *Requirements*: The expected or desired properties or behaviors of the design object. (note, this thesis splits this in functions and requirements)
- *Constraints*: For the design agent: time and resources (Also mental faculties) to be spent on the design, for the design object: laws of physics, or other knowledge rules that limit the way the specification can be built.

2.3.2 Communication in the Design Process

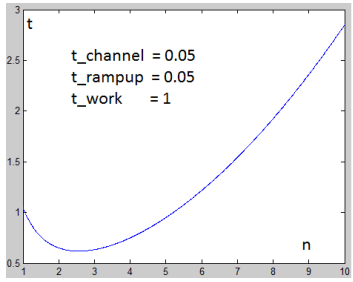
As stated before, to create a design, multiple stakeholders are needed to design a system. When a problem becomes too big to be understood by one person, the work is divided between more persons. When a problem becomes too complicated to be understood by one person, specific work tasks are assigned to more persons. An organization will decompose a design process into design tasks that can be handled by individual designers and engineers [Mosterman et al., 2005]. These individuals have specific skills that enable them to build a good specification of the design object, using primitive components from their discipline (e.g. mechanical engineers use materials and geometric features, software engineers use classes and protocols, electrical engineers use electrical diagrams and components). Often, but not always, the resulting specification will be formalized in a model or other type of document. As the specification of the system becomes clearer over time, these models grow in detail, and reduce in abstraction.

This process will lead to a need for coordination of tasks between the cooperating persons and communication of task results. The more the work is divided and specialized, the more organization, communication and coordination is needed – 'communication overhead'. This is explained in Brooks's law in The Mythical Man–Month [Brooks, 1975], where the number of coordination channels expands with the number of workers (see 2.7(b)). Brooks's law is determined on the premise of adding workers to an existing software project. However, a design process can be seen as a comparable type of project, where people need to learn the current status of the project from other people all the time. This ramp up time, spent on people learning and integrating with a team, is another source of 'communication overhead'.

In figure 2.7(a) this is depicted as splitting a single task into two. Because of more ramp up time of the task and more models that need to be built, the separate tasks will cost more time than the single task (given equally skilled workers). Then there will be additional need for communication, which will be added to the task time. Think of meetings and documentation to coordinate, synchronize, and share the tasks and their results. Improving this communication is therefore the most important action an organization can take to improve the overall multi-disciplinary design process (according to table 2.2). However, this statement does not explain *how* this improvement can be achieved. According to the Wikipedia



(a)



$$t_{work} \xrightarrow{\text{divide under}\atop\text{n workers}} \begin{bmatrix} \frac{n\cdot(n-1))}{2}\cdot t_{channel} \\ n\cdot t_{rampup} \\ \frac{t_{work}}{n} \end{bmatrix}$$

(b)

Figure 2.7: Workers and Communication Overhead. (a) Dividing work between stakeholders will necessitate communication overhead. (b) A formula of the communication overhead, adapted from [Brooks, 1975]. More workers cause more communication channels. They need to be introduced into the process, which causes ramp-up time.

site on Brooks's law [Wikipedia, 2012], the following points can mitigate the communication overhead:

- A correct project schedule will prevent the extra coordination effort from unforeseen delays (more time but less risk).
- Better specify roles of workers beforehand (stakeholder view on the design process).
- Continually check if the results of the individual design tasks still fit into the goals of the integrated design (check with requirements).
- Tools for (software) development and documentation (a framework as will be introduced in chapter 4)
- Design patterns or other reusable design information. These simplify the distribution of work, because the entire team can do its part within the framework provided by that pattern. The design pattern defines the rules that the stakeholders follow, simplifies communication through the use of a standard language, and provides consistency and scalability. Patterns can be automated in knowledge bases.
- Correct decomposition of design tasks. Closely related with the role specification of point two. Can be captured in a shared reference model.
- Social and political aspects of the work climate. Perhaps the most important tool to communicate is human face to face language, the stakeholders should be comfortable in their work climate.

Referring back to the figure 2.7(a), the communication between stakeholders is supported by methods such as emailing, meetings, project documentation, project management, (phone) conversations, etcetera (vertical v arrow). This means a stakeholder has to 'manually' translate the required information from a model into something communicable. These methods are often informal, and do not have a consistent link with the underlying models or work, in terms of the up-to-date version, or consistent values, units, or rationale of the information (the horizontal x arrows). Furthermore, there is often no 'meta' language to translate the work directly (vertical x arrow). So when someone wants to have information of someone else's task, then the information needs to be requested through the informal system, which takes time, as the other person must stop his or her own task to build a document or email to 'package' the requested information. A shared meta model for this information network could be an answer to this problem.

2.3.3 Complexity in the Design Process

Resuming, improved communication is needed between the actual design work and the communication and coordination methods, and between the different design work tasks, models and tools. These improvements must take into account a complex set of interactions between stakeholders and a complex set of types of information that is to be exchanged about the system under design.

We can say we make models in order to increase, formalize and communicate our knowledge of the system, and thus reduce the complexity of the system – if one fully comprehends the system, it is no longer complex, as complexity is a measure of understanding,

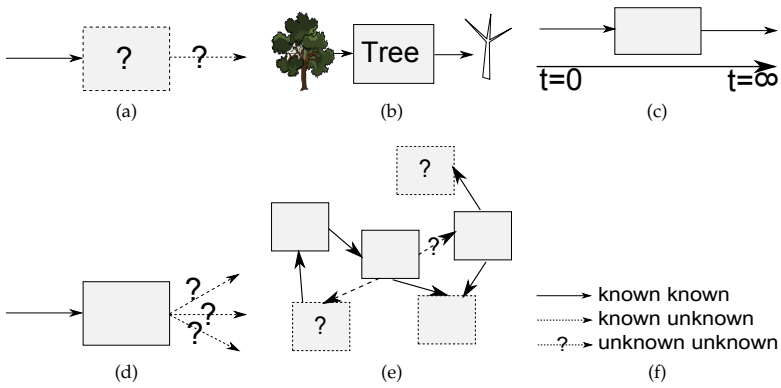


Figure 2.8: Complexity types. (a) Algorithmic complexity as the lack of knowledge to solve a problem. (b) Algorithmic complexity as the inadequate/simplified modeling of a complex problem. (c) Algorithmic complexity as the effort it takes to solve a problem. (d) Deterministic complexity as an unpredictable result/behavior of a problem/system. (e) Aggregate complexity as an emergent network of interrelated entities. (f) legend.

not only an attribute of a system. However, what is complexity? Finding a definition for complexity seems to be a complex task in itself [1]. Whereas complexity is the topic of multiple fields of research in itself, this thesis does not aim at giving a complete literature review on complexity. Rather, we shall look into some appropriate complexity types encountered in modeling design process information.

Mostly, complexity seems to be defined as a measure of interactions and relations between entities that make up a system. However, this definition is mostly used in terms of models of systems. As these models are chosen to represent a certain level of granularity and scope, we could say complexity is a measure of the modeler's understanding of the system. A grain of sand may not be complex to a phone manufacturer, but it is to a geotechnical engineer, whereas a phone is just a simple device for said engineer, and thus not so complex from his perspective. [INCOSE, 2006] reflects this definition as follows: *"complexity can be considered as a measure of how well knowledge of a system's component parts explains the system's behavior and also by the number of mutually interacting and interwoven parts, entities or agents"*. In more detail, [Manson, 2001] defines three types of complexity, which can be used in the context of design: Algorithmic, Deterministic and Aggregate complexity.

Algorithmic complexity is the measure of effort required to solve (mathematical) problems about the system. This complexity is relevant to design processes in a number of ways:

- *Knowledge*(figure 2.8(a)): First off, what do we know about the system at a given stage in the design process? Is this knowledge accessible for formal analysis or is it tacit knowledge in someones head? Donald Rumsfeld said it best: "We know there are known knowns: there are things we know we know. We also know there are known

[1] Definition of Complexity in Webster online dictionary: 1: something complex 2: the quality or state of being complex, sic.

unknowns: that is to say we know there are things we know we don't know. But there are also unknown unknowns – the ones we don't know we don't know." One cannot make a calculation on something undefined, so a design process is complex when the knowledge on how to design something is lacking, or when it is unknown what problem needs to be solved.

- *Solvability*(figure 2.8(b)): A design problem is complex if a solution that was derived from applying mathematics or algorithms does not properly relate to the system in the actual world. Think of linearization of non-linear behavior or incomplete or inadequate modeling techniques to describe the empirical real world with mathematical models.
- *Effort*(figure 2.8(c)): A design problem is complex when it takes a lot of effort to derive a solution. This can be because problems are big, such as finite element models for computers or filling a database with manual documents for humans, or because problems need iterative solving, such as design optimization (given a parametric definition of a system, what is the optimum value of every one of these parameters given certain constraints?).

Deterministic complexity is a measure of knowing how the outcome will change when initial conditions change, in other words, how chaotic it is (figure 2.8(d)). A double pendulum is such a deterministic complex system, as the output (how it behaves) will vary greatly with small changes in input. The design process can also be deterministically complex, as a design decision can have unknown results on the design specification, and thus on the system under design. Risk management and requirements management are focused on modeling the cause of this kind of complexity, in order to make designs more predictable and thus less deterministically complex. We can see a team of designers and engineers working on a problem as a deterministically complex system, as a design process would probably not result in the same design when done twice. However, the chaos theory does mention 'strange attractors' [Manson, 2001], which, in the case of the design, are the requirements: The design will gravitate towards fulfilling the set of requirements over time (this is the reason changing requirements in the middle of a project is a bad idea).

Aggregate complexity is a measure of emergent behaviors resulting from the interactions among system entities (figure 2.8(e)). The design process can be seen as an aggregate complex system, as the relations between designers and other stakeholders tend to emerge, evolve and disappear during the design process. The design itself is also aggregate complex, as the system will get more components (parts/entities) and relations and thus emergent behavior needs to be taken into consideration (for example, overheating of a system caused by heat generated by processor units that were added to increase dynamic performance). This will then necessitate changes in the design specification.

Considering this, we can separate the complexity of the system under design from the complex system of companies, designers, manufacturers and engineers and their tools that carry out the design, manufacturing, sales and maintenance process. In this thesis we will call the system under design **the system**, and the means of designing such a system, **the design process**. A literature review paper of [Chivagomez, 2004] also recognizes a design process as a complex adaptive system, where "heterogeneous agents which interrelate with

each other and with their surroundings, and are unlimited in their capabilities to adapt their behavior as a result of their experience. In each system, each agent is different from the others, and its performance depends on the other agents and the system itself, which influence its behavior." This field of complexity theory also allows for the explanation of self organization, creativity and learning [Anderson, 1999], but that will be out of scope of this thesis.

What is in scope of this thesis is the development of a 'tool' that will reduce the aforementioned complexities by modeling the 'known knowns' and 'known unknowns' as soon as they become apparent: to make models of design information and relations, abstracted towards a level of granularity that all stakeholders can understand [Hoover, 1991], and can be used to facilitate communication and coordination throughout the design process. The development of such a tool is discussed in chapter 4. The remainder of this problem definition chapter will focus on the further analysis of what is needed to improve communication in multi disciplinary design processes, to find the required capabilities of such a tool.

2.4 Sources for Analysis of Consistency, Integration, and Reuse

In order to find grounds for improvements in multi disciplinary design, this thesis has two main sources. First, the focus group of the case studies and the 'Automatic Generation of Control Code Software for Mechatronic Systems' (AGCCSMS) project. Secondly, the body of scientific literature. The AGCCSMS project, of which this thesis is a result, dedicated a number of papers on the issues in complex design processes, which in turn quote a number of review papers (see deliverables list A). As could be read in the introduction, the AGCCSMS project first focused on the generation of control code of mechatronic systems. Why was that so difficult? As the project progressed, we found out that the main reasons can be seen in a broader sense than 'just' for mechatronics design.

These issues were extracted during our cooperation with academic and industrial partners. This focus group consists of experts from academia and industry that were interviewed during the AGCCSMS project. This industry-as-a-lab approach (chapter 5) led us to pragmatically extend functionality of our tooling to cover more of the scope of the project: generating control software from abstract architectural information. However, it also led to the recognition that the issues were not limited to the modeling of mechatronic systems. Much of the topics in this thesis were derived from a large number of discussions with industrial, and academic partners. A quick survey of travel declarations shows approximately 94 meetings outside of Twente (the author's work location), approximately 10 meetings in Twente, and approximately 10 video conferences. External visits were about 43 company visits and 55 project meetings, where most of these meetings lasted about four hours or more, and were often return visits, conducted with the same people. These discussions, meetings and interviews can be categorized in three main categories: workshops and research project feedback meetings, conferences and academic discussions, and company interviews and case studies:

2.4.1 Workshops and research project feedback meetings

- *Participants:* Around 60 individuals; professors, PhD students, industrial system architects, engineers, technology directors
- *Main concerns:* Consensus making, Building and communicating system architectures, Communication and versioning of design decisions and design knowledge, Knowledge transfer of project to market, Automatic software generation

The workshops and research project feedback meetings were specifically intended for user feedback on the applicability and valorization of the developed tooling of the AGCCSMS project. As the tooling evolved, it gave the focus group a hands-on experience on how the members of the AGCCSMS project formulated complex design processes, so evolving the discussion from an abstract terminology issue to a practicality issue of what tooling functionality works and does not work. The workshop is discussed in more detail in appendix C.

2.4.2 Conferences and academic discussions

- *Participants:* Around 30 individuals; (assistant) professors, PhDs , PhD students
- *Main concerns:* System engineering and architecting, Model & knowledge based engineering, Mechatronics design, Human factor in communicating complex design information, meta & reference modeling.

The conferences and academic discussions rooted our research project within the academic context. It supplied us with the state-of-the-art of the research fields involved with multi disciplinary designing. Much of the terminology used in this thesis is a result of lengthy debate of what certain words actually mean (function, system, complex). Also, it led us to focus and position our academic efforts in the perceived 'gaps' of the state-of-the-art. The academic discussions and conferences lead to the published papers in the deliverables list A.

2.4.3 Company interviews and case studies

- *Participants:* Around 40 individuals; Engineers and managers in multinational high-tech companies: ASML, Mathworks, Océ, Philips, Vanderlande, Embedded Systems Institute
- *Main concerns:* Multi-view system architecture, Traceability ownership and communication of design decisions, Model & knowledge based engineering, Consistency in design information, Innovation vs. reuse.

Company interviews and case studies were conducted to develop and test our ideas in an industry-as-a-lab setting. It led us to discover a number of concerns that industry found were not addressed by their design technology. This, in turn, led us to focus specifically on those concerns. Were these issues addressed in available technology or academic

approaches, or could we add something to the state of the art? The cases are explained in detail in chapter 5.

The issues found in this empiric research are explained in the next sections of this chapter: 2.5 Ensure Consistency, 2.6 Enable Integration, and 2.7 Facilitate Reuse. Where they are further supported with literature references to surveys and reviews of existing approaches.

2.5 Ensure Consistency

2.5.1 Current Situation

In the dictionary, consistency is defined as: (1) Agreement or accordance with facts, form, or characteristics previously shown or stated. (2) Agreement or harmony between parts of something complex; compatibility. (3) The state or quality of holding or sticking together and retaining shape. (4) Conformity with previous attitudes, behavior, practice, etc.

In design process terms this will be: (1) The design specification must model the system under design in its environment. (2) The parts of the design specification do not contradict each other. (3) The design specification must comply with the requirements and goals of the design throughout the design process. (4) Constraints of new design tasks must be in accordance with what is specified in existing models.

Figure 2.9: Some system use cases are not verifiable in a design process www.museumofunintendeduse.com

The relation of consistency to system performance, time-to-market and cost (see figures 2.4 and 2.5) is not obvious. The system performance will be dependent on how good the system, once finished, adheres to the wishes of the customer. In the design process we will try to predict and work towards a good system performance, however, the system is not finished yet, so measurements on a prototype are not possible. To mitigate this, the models that capture the design specification need to adhere to the requirements collected from the customer. This means we can verify the design specification at any given time in the design process by measuring the adherence to the requirements. The design of the system will be valid if the design specification also describes the system performance correctly. However, often this is not knowable, as the wishes of the customer will change in response to what the system does, as well as unforeseen behaviors of the system that will emerge during the design process (see figure 2.9). Therefore all we can say is that the design specification (including prototypes) must be consistent with the requirements and goals of the design. This continuous consistency can only be the case if the consistency is guarded at every step of the design process, which means the hand-over of information between design tasks must

be consistent, and that the results of the design tasks must be verified against the requirements.

The time-to-market and design cost could be extended by focusing too much on working consistently and adhering to requirements, as this constant checking will cost time not directly invested in designing a system. However, this will only be the case if everything would go well the first time. The cost and time needed to rework inconsistencies in a design could greatly exceed the cost of working consistently the first time around. As can be seen in this analysis on design changes in civil engineering [Chang et al., 2011], finding out design errors in the production phase is expensive: "The redesign costs are from 2.1% to 21.5% and on average 8.5% of the construction change cost, equivalent to the fee of a new design project."

2.5.2 Issues

By interviewing industrial partners (§2.4) and reviewing literature, the AGCCSMS project has found a number of issues and challenges related to multi disciplinary design. These challenges, concerning consistency of (exchanged) design information, are summarized in table 2.5. The more extensive description can be found in appendix B.1.

2.5.3 Resolution

The issues around consistency of complex design processes seem to be guided mainly by the way information is exchanged by the human stakeholders in the organization, and by the modeling of this exchanged information. Unmentioned in the listed issues is the consistency in terms of the first point of the definition: "The design specification must model the system under design in its environment", and will therefore be out of scope of this thesis. This type of consistency is about making correct models of behavior, such as finite element models, physical simulations, dynamic analysis etcetera (a matter of algorithmic complexity as stated in §2.3). In this thesis we will assume that such models can be made, and we will represent them as Domain Entities (see the Architecture Modeling Language 4.1 section).

We can summarize the consistency issues in the following goals: Continually check the consistency of a design by verifying the design process results to the system requirements. Aspect models, containing design information from multiple disciplines, provide the means for this checking. Therefore, make a model of information exchange between the design tasks, make a model of the shared design information between models, and make a model of the relation of the shared information to the system performance criteria and other design aspects, to come up with a measurable evaluation. Provide this information to the stakeholders in stakeholder-specific views, so they can maintain overview and context of their work. In summary, make a model of the answers to the "who, what, when, where, why, in what way, by what means" questions in the design process.[2]

[2] Already used as a way to obtain arguments in classical rhetorics in 100 BC, Hermagoras of Temnos was quoted in De Rhetorica from pseudo-Augustine: "Quis, quid, quando, ubi, cur, quem ad modum, quibus adminiculis"

Table 2.5: A summary of consistency issues in multi disciplinary design processes. See appendix B.1 for details.

Index	Issue	Goal
1	Big Picture	Show how information is shared between stakeholders, design goals and design tasks.
2	Overview of the Design Information Flow	Define how and what information is exchanged.
3	Provide Design Information in a Context	Describe the input output kind of information for design tasks.
4	Prevent Ambiguity	Store design information centrally and uniquely.
5	Check the Requirements	Follow adherence of design aspects to requirements throughout the design process.
6	Tacit Understanding	Provide platform for developing mutual understanding and agreement.
7	Modeling Design Decisions	Capture decisions and provide rationale.
8	Provide Traceability	Define the where/who/when/why for important design information.
9	Ownership of Shared Design information	Define stakeholder ownership as attribute of any design information.
10	Consensus Making	Provide platform for discussions such as negotiation, compromises and trade-offs.
11	Versioning	Versioning on model content in the context of other design information.
12	Allow for Emergent Properties	Facilitate augmenting and sharing of (new) views and models between stakeholders.
13	Automate Hand-over Generation	Provide machine and human readable common modeling language.
14	Do Not Duplicate Information	Store information centrally and uniquely.

The stakeholders work in various design disciplines, build models in different languages, and build different models in different phases of the design process. To maintain consistency, these sources of information must somehow be integrated into a shared model, which, as we will explore in the next section, is a challenge in itself.

2.6 Enable Integration

2.6.1 Current Situation

Integration is the connection of different parts into a whole. In the design process, this integration has two faces: (1) The system under design has functionality that is performed by multiple entities (parts/components/objects...), that are separated by both topological distance and by the design disciplines needed to design these entities. And/Or, an entity performs multiple functions. As such, system functionality is the integrative aspect of systems. (2) The design process is performed by an organization of stakeholders and their tools that are separated by both distance and by their design disciplines. This organization works toward a joint goal (of making money or other organizational goals) and making a design that answers the requirements of the customer. As such, the goals and the requirements are the integrative aspect of the design process.

How does integration relate to system performance, time-to-market and cost? In modern mechatronic products, for example, the design of the function performer (the system) is still often split into the software, hardware and electronics parts, that become design tasks for the various disciplines (source: [Boucher and Houlihan, 2008]). Ideally, these parts can be designed independently of each other, on the condition that the interfaces between the parts and the constraints of the design task are correctly defined in advance. This means the time-to-market can be shortened by working concurrently, instead of waiting for the other disciplines to finish their work first (often hardware before software). Unfortunately this concurrent work is not so easy to achieve, given the consistency and integration issues stated in this chapter.

The system performance – how well does the system meet the customer requirements – is measured in various aspects cutting across disciplines (throughput, cost, maintainability...). So the performance of systems is for a large part engineered across the interfaces between design disciplines, which often is a gray area of 'ownership', and lacks a lot of the mature tooling available in the domain specific disciplines (such as CAD, office tools, or ERP). Ideally, an integrated model should provide insight into the connectivity of the system performance to the various parts of the system, thus enabling the system performance to be measured, and, eventually, managed.

The cost of a system could be reduced by making a fully integrated system, where there are no unnecessary interfaces and there is no duplicate or redundant technology. In contrast, maintenance and assemblability aspects could reduce the cost over the life cycle by making modular system components – such as software upgrades, replaceable battery packs – that can be easily replaced or upgraded. In both cases, the enabler for the low

cost is a well thought through design, that focuses on the integration of functionality in single components on one hand, and the integration of interfacing components on the other hand.

2.6.2 Issues

By interviewing industrial partners (§2.4) and reviewing literature, the AGCCSMS project has found a number of issues and challenges related to multi disciplinary design. These challenges, concerning integration of design information, are summarized in table 2.6. The more extensive description can be found in appendix B.2.

2.6.3 Resolution

What should an integrated design process look like? The issues indicate that it should be based on the integration of the shared information in the design process, not an integrated model of the system under design. A system modeling language that can describe everything in mathematical and executable form does not exist, as it would encompass all human knowledge of all different disciplines. Even if it would exist, the resulting 'model of everything' is not practical as it would require too much input and it would probably be too big to comprehend, or oversee. Even then, the tool using the language would replace all existing specialized modeling tools, necessitating everyone in an organization to switch to the new tool, which they will not like. Therefore integration of the design process must focus on the stakeholder, modeling the resources the stakeholder needs to understand the system and design task (in that point of the design process) from his or her point of view.

Ideally, stakeholders (using their tools) should work in parallel, and know what to do, given that they know their goals, requirements and constraints. They know what models they use as input, and what models they need to deliver to a next stage of the design process. Design decisions cannot be taken from the point of view of a single domain, as often they will have an impact in other domains. If problems are detected or design decisions are taken, the infrastructure needs to be in place to quickly localize the 'neighbors' of the design process that are also affected. Also, in reverse, stakeholders have the infrastructure to find out who made a certain design decision, that could affect their design task.

Apart from modeling the design tasks, goals, requirements, constraints, models and the workflow of the various stakeholders, we must also make a model of the broad outline of how the system works, the system architecture. This information should be captured in a discipline-independent language through a process of parameterization of the design specifications: a language to abstract discipline specific models and information in a manner that it can be integrated in a single reference model or meta model. This means also cross cutting aspects – concerns that need to be tackled across various disciplines – can be modeled. This integrated model can then be used for a concurrent engineering approach with a highly integrated design strategy – a design based on multi-disciplinary objective, requirements and constraint evaluation in terms of the parameterization of external models. These objectives

Table 2.6: A summary of integration issues in multi disciplinary design processes. See appendix B.2 for details.

Index	Issue	Goal
15	Communication Distance	Close distance by modeling shared information in a shared language and a shared model.
16	Shared Data Model	Define a shared model of the shared design information that is also machine readable.
17	Model and Communicate System Architectures	Define a language that can define architecture descriptions.
18	Multi-View System Architecture	Allow partial views on a shared model. Define view independent language.
19	Multi-Composition	No single view takes the lead; allow for multiple decompositions of a system, in various disciplines or in other stakeholder concerns
20	Multi-Mapping	Define a language that can connect design information of different types.
21	Capture Design Decision	Parameterize a decision and define the context in terms of stakeholder or system goals, design aspects and discipline specific models.
22	Design Process Tasks	Define a concept type to model the workflow in a design process, the stakeholder who performs the task, the aspect the task concerns, and the models involved in the task.
23	Simple Shared Data Language	Define a language with a small set of well known concept types, must be machine readable.
24	Abstract and Reference Model Content	Define domain entities as pieces of abstracted information used in the design disciplines.
25	Support Parametric Definition	Define a parameter concept type to parameterize design information.
26	Support Parametric Integration	Define the context of parameters, define the possible occurrence of parameters in multiple disciplines, design aspects and stakeholder views.
27	Mechatronics Design	Define a language that can model the basic behavior and structure of mechatronic systems.
28	Align to Functions	Use the functionality of a system as the central system description, as opposed to the physical decomposition of mechanical parts.

and constraints can in turn also be expressed as parameters of the design, thus forming an integrated model of the design. Such a parametric definition can be used to automatically synthesize and analyze new parts of the design or to generate new external domain specific models. This is a process of design knowledge reuse that is in itself a topic of this thesis, as we will see in the next section.

2.7 Facilitate Reuse

2.7.1 Current Situation

In general, most tools that are used to design are already a form of reuse. Software tools such as a 3D CAD modeler 'knows' a lot about shapes and how they can be connected mathematically; physical behavior simulators 'know' how a dynamic system responds when an input is changed. More basically, textbooks and standards contain pre-defined solution strategies (how many teeth does my gear need under these loads?) and embodiments (what are the dimensions of my standardized gear?) for numerous problems. We can here spot two basic types of reuse. First, the reuse of a certain piece of design specification directly, and second, the reuse of knowledge to construct a piece of design specification for a specific context or application or to analyze a piece of design specification. However, both are ultimately used to construct the design specification.

Within tools we often see toolboxes or libraries that can be used to quickly assemble new models from existing templates or design primitives. These templates are instantiated into their models and configured for the new context. The tool can then perform analysis on models so constructed, as the tool contains the specific knowledge to interpret and work with the semantics of the model (see §4.1 for further explanation). As will be reviewed in the state of the art chapter (§3.4), more advanced tools are able to actually generate the models they need to analyze first, based on a set of requirements the user puts in. These **knowledge based engineering** tools are often used to reuse specific design knowledge in a fast and consistent way. Reusable design objects do not only cover structural artifacts of the system such as models for bills of material, geometry or software classes, they can also be developed for models for behavior analysis, simulation, verification, and validation as is shown in, for example, the multi-representation architecture of [Peak et al., 2007b]. Basically, reuse concerns generating and transforming models. As such, reuse is one of the cornerstones of model-based engineering.

Within organizations, certain architectural concepts are also often reused, as they will be part of any system the organization designs. Think of the paper path of a printer/copier, the engine of a car, or the database of a website. While the embodiment of these concepts will be completely different from system to system, the 'raison d'être' for it being there will not change (much), the people who design it will not change (much), the knowledge of how to make it will not change (much), the tools that are used to make a model will not change (much), and the technology and knowledge needed to design it will not change (much).

How does integration relate to system performance, time-to-market and cost? Time-to-market constraints drive the need for reuse and standardization of design artifacts and knowledge into reusable design primitives. According to [Gautam et al., 2007], reuse can attack a number of wasteful design activities compared to designing from scratch: There is no re-inventing the solution, therefore, associated errors and consistency problems are evaded. Redundant design tasks are prevented, for example, validation and verification for a certain design aspect can be standardized and reused too. Reusable primitives prevent multiple versions of equal solutions, limiting problems when integrating the primitive into a larger design specification. The hand-off of information is pre-determined by the interface definition of the primitives, which further reduces transaction waste and waiting time compared to a situation where a designer has to uniquely identify and explain how the design primitive should be integrated. Finally, because of the fact that the reusable primitives are so well-defined, they can often be supported by automation, stored in libraries, and implemented automatically, greatly reducing design time.

The user does not need to have the knowledge of how the reusable primitive was constructed – or the reasoning behind the primitive – to be able to use it effectively. This means the development of the reusable primitive is only done once, saving design time each time the primitive is used, saving the cost involved in this time. Because the user is not required to have all the knowledge on the reusable primitive, the organization can specialize people more, thus design a broader or more complex set of functionality in systems. Or they can spend more time in optimization or exploring more design options. All of this will ultimately increase the system performance.

However, reuse and specialization can also cause cost problems. Designing reusable primitives is costly and will require continuous maintenance and tooling development. This must be done by software engineers, who are often not part of the core business of the company. This setting up of a new design tooling department, independent of a single project, is often an unpopular decision for the higher management. A danger of making knowledge reusable can be the 'brain drain' of the skill from the organization, especially when these reusable tools and primitives are created by external software developers: When someone retires, or the organization switches software supplier, there could be nobody left who understands the reusable primitives or design rules.

2.7.2 Issues

By interviewing industrial partners (§2.4) and reviewing literature, the AGCCSMS project has found a number of issues and challenges related to multi disciplinary design. These challenges, concerning reuse of design information, are summarized in table 2.7. The more extensive description can be found in appendix B.3.

Table 2.7: A summary of reuse issues in multi disciplinary design processes. See appendix B.3 for details.

Index	Issue	Goal
29	Meta Modeling	Use a central abstract model to generate detail models
30	Reference Modeling	Use a central model to keep track of status and location of design information
31	Model Multi-Domain Information	Abstract domain specific information into a shared model.
32	Defining Reusable Design Primitives	Use functionality of a system as a basis for reusable design primitives
33	Defining Reusable Design Knowledge	Store knowledge in software algorithms, store the constraints in a central meta model store implementations in external models.
34	Optimization	Use a central model to keep track of optimization targets across multiple models.
35	Impact Analysis of Reusable Design Primitives	Keep track of the relations between primitives to those in other domains.
36	Lean Models	Put the minimum of information in the model, but not less.
37	Use Existing Tools	Define a language that can describe information from the various tools.
38	Platform Development and Support	Define a framework that can be used to develop models of multiple product versions.
39	Freedom vs. Speed vs. Consistency	Define a language that has a minimum of predefined semantics, but allows for custom addition of it, and has a predefined consistent syntax.

2.7.3 Resolution

Reusability should be supported in multi disciplinary design. The basis for this is a language that enables the abstraction of design goals, requirements, and constraints, as well as the parametric definition of the system that should attain these goals within the stated requirements and constraints. This language must not contain the semantics of the design problem, as this will make the language too big; there is no limit to which domain specific knowledge could be inserted. Therefore the language should be accessible and generic enough to be usable by different external 'knowledge bases' that contain the semantics needed to interpret the models.

To make it possible to reuse design information across design domains and disciplines, the information must be stored in a central model. This model is a meta model in the sense that it specifies the way other models need to be constructed, and a reference model in the sense that it specifies where the needed information (input, output, knowledge, models, etc.) is stored.

The design primitives from the various disciplines must be made reusable in a concurrent way (see also the Baggage handler case study 5.4). This means a set of reusable primitives can be implemented across multiple domains according to a single input. To enable this, the reusable primitives must be aligned to either a (domain independent) functional goal of the system or a (cross discipline cutting) design aspect of the system. Where possible, the manner in which the primitives are implemented in domain specific models must be determined by the system requirements, to ensure that the various tools optimize towards the same goals of the design process.

Keep in mind that not all design knowledge of a company can – or must – be made reusable. As was stated in 2.7.1, there is a trade off between the effort of making knowledge reusable and the gains of reusing the knowledge. Reusability should therefore be implemented modularly.

2.8 Hypothesis

In the previous sections we saw that the main 'pressures' in improving multi disciplinary design are the improvement of system performance, time-to-market and cost (table 2.1). And that they can be improved by by facilitating communication between stakeholders, modeling architectural concerns and other important design information, and facilitating capturing and reusing design knowledge across disciplines (table 2.2).

These pressures, however, cannot be subject to direct scientific research, as experiments on measuring these pressures cannot be realized: We cannot let multiple companies perform multiple design processes to compare methods that aim to improve the pressures. Instead, the pressures are 'transformed' in the issues of consistency, integration and reuse, which can be evaluated through case studies. As such, the hypothesis for this thesis is:

The consistency, integration and reusability of multi-disciplinary design processes can be improved by facilitating communication between stakeholders, modeling architectural concerns and other important design information, and facilitating capturing and reusing design knowledge across disciplines.

The first part of the hypothesis reflects the research question: How can we improve consistency, integration and reusability in multi-domain design processes? This chapter provided a set of issues in these categories, and the possible resolutions. From the issues described above the following 'metrics' are distilled that measure what needs to be done to cause improvement:

- Consistency is improved with a multi disciplinary model in which to track the 'who, what, when, where, why, in what way, by what means' design information.
- Integration is improved with a multi disciplinary model that maintains overview and context between the views containing stakeholder specific design information.
- Reuse is improved with a multi disciplinary model that facilitates (partial) construction of the design specification by applying knowledge bases.

What such a model will look like will be the topic of the literature review in chapter 3. In that chapter, we will see that the current state of the art will leave gaps, as no single method addresses the three topics at once.

For this hypothesis to be tested, we need a language to facilitate communication between stakeholders, a tool to make a model of architectural concerns and other important design information and a facility to plug in design reuse applications. The combination of these things we will call the **Architecture Modeling Framework**, which will be the topic of chapter 4. In this framework, we will address the perceived gaps that are stated as a conclusion in a state of the art review in chapter 3. The goal of the framework is to address the issues stated in this chapter.

The resulting framework will be used to find out if the hypothesis holds by describing a number of case studies that were aimed at improving consistency, integration and reuse through facilitating communication between stakeholders, modeling architectural concerns and other important design information, and facilitating capturing and reusing design knowledge across disciplines. These 'experiments', as described in chapter 5, were conducted mainly in the environment of the industrial partners of the ACCSGMS project.

If the framework helps to facilitate communication between stakeholders, make a model of architectural concerns and other important design information, and facilitate capturing and reusing design knowledge across disciplines, we can assume that the hypothesis is valid.

2.9 Scope

What is and what is not within scope of this thesis is determined along the lines of the design process definition, and the complexity definition in this chapter. In the state of the

art chapter 3 we will find the state of the art of the topics that *are* in the scope defined in this section. Here we focus more on what is *not* in scope.

In a broad sense, this thesis focuses on the tools that we (can) use to *make a model of* information in design processes. This already dismisses a large part of available methods and tools where modeling and formal modeling may not be the best approach to facilitate communication between stakeholders, capture architectural concerns and other important design information, and facilitate capturing and reusing design knowledge across disciplines. There could be situations where informal communication works better, as in brainstorming for ideation, or explaining a problem in a face to face meeting, or a Microsoft Powerpoint presentation to inform project progress to a group of people. When formal communication is desired to help engineer a system, we speak about model based systems engineering, which is the topic of literature section 3.1. The research in this thesis applied model based systems engineering with a framework – the architecture modeling framework of chapter 4 – that was developed during the AGCCSMS project. In the definition of a design process in section 2.3, we find many of the required modeling concepts that we must employ in an architecture modeling framework:

- *"Specification of an object, manifested by an agent, using a set of primitive components subject to constraints"* will be the topic of literature review in the design knowledge modeling section 3.4 and the lessons from a software world section 3.6.
- *"intended to accomplish goals, in a particular environment, satisfying a set of requirements"* will be the topic of literature review in the design process model section 3.2 and the system architecting section 3.3.
- *"to create a design, in an environment (where the designer operates)"* is covered in the literature review on workflow modeling 3.5, and in the perceived gaps section 3.7.

The complexity definition of section 2.3 further constrains the scope of the thesis. For algorithmic complexity, this implies that we will not specify how all possible design knowledge is to be modeled, we will not explain the best way to solve a design problem, and we will not explain how to solve large problems efficiently. What will be in scope is a method to parameterize both the input and output of problems, and how to specify the constraints and requirements of problems. To find out how to do this, literature on the subject is studied in the design knowledge modeling section 3.4. In the architecture framework these problems are modeled in a meta model (section 4.3) with a modeling language (section 4.1), and, when automation is possible, solved with a – case specific – tool (section 4.4). That we can actually use the framework to help in algorithmic complex situations will be the topic in the Paper Path case study (section 5.6).

For deterministic complexity, we will not provide the best way to reduce risk in the design process, or how to check requirements effectively or how to predict how design changes effect the design specification. What will be in scope will be methods to make a model of the relations between requirements and various design aspects and system functionality, as will be the topic of the system architecting literature section 3.3. In the architecture modeling framework this kind of information is modeled as a reference model (section 4.2) with a modeling language (section 4.1). Such a reference model should provide a basis to perform

risk management and design change management, but this will only be demonstrated in one case study, the baggage handler case (section 5.4).

Finally, for aggregate complexity, we will not provide the best way to construct new design information based on the existing information, or a way to predict emergent behaviors from a given situation. What will be in scope is the language specification to enable modeling of these kind of design activities (see section 3.7). First, the language should be able to build a meta model (section 4.3), which can be used to automatically construct new models, or alter existing models through pluggable knowledge bases (section 3.4 and 4.4). Second, the language should be able to build a reference model (section 4.2) that provides the means to find the resources to do specific analysis, and automate these analyses if they are specified in knowledge bases. Both the meta model as the reference model types were topics in the paper path case study (section 5.6).

3. State of the Art

To improve on the current multi disciplinary design process, we need to know the current state-of-the-art in modeling design process information. First we review the tools at our disposal to facilitate communication between stakeholders, make a model of architectural concerns and other important design information, and facilitate capturing and reusing design knowledge across disciplines. Then we discuss what is missing in these tools so that they do not provide a complete solution for improving consistency, integration, and reusability in multi disciplinary design processes?.

The problem definition (chapter 2) discussed the issues in multi disciplinary design. From these issues a set of requirements was deduced. This chapter will try to reconcile the requirements with available methods from industrial practice and academic literature in a number of different sections. The literature is selected based on its industrial usage or on relevance to the development of a framework, the **Architecture Modeling Framework**, that is described in the next chapter.

In this literature review, we search for common concepts in popular methods (e.g. function, parameter). In the next chapter, these concepts will be synthesized into a new modeling language that can be used to facilitate communication in multi disciplinary design processes. Therefore, to limit the amount of duplication between this chapter and the next, the methods discussed will be augmented with parenthesized words in (**bold**) that represent an equivalent concept or attribute in the architecture modeling framework of the next chapter. Note that the following sections also contain a number of references to websites of communities using certain methods, instead of referencing peer-reviewed literature directly. This is done because these (community) websites reflect the state of the art in those methods, and offer a better source for further reading.

3.1 Model Based Systems Engineering

Systems engineering is the “interdisciplinary approach and means to enable the realization of successful systems" [INCOSE, 2006]. As such it is the discipline that handles the relation between the design process and the system under design. The research of this thesis can therefore be placed in the systems engineering discipline. To be more specific, this thesis aims at coming up with a modeling framework that can be used in multi disciplinary design. In the scope definition, the focus was put on those parts of the design process where

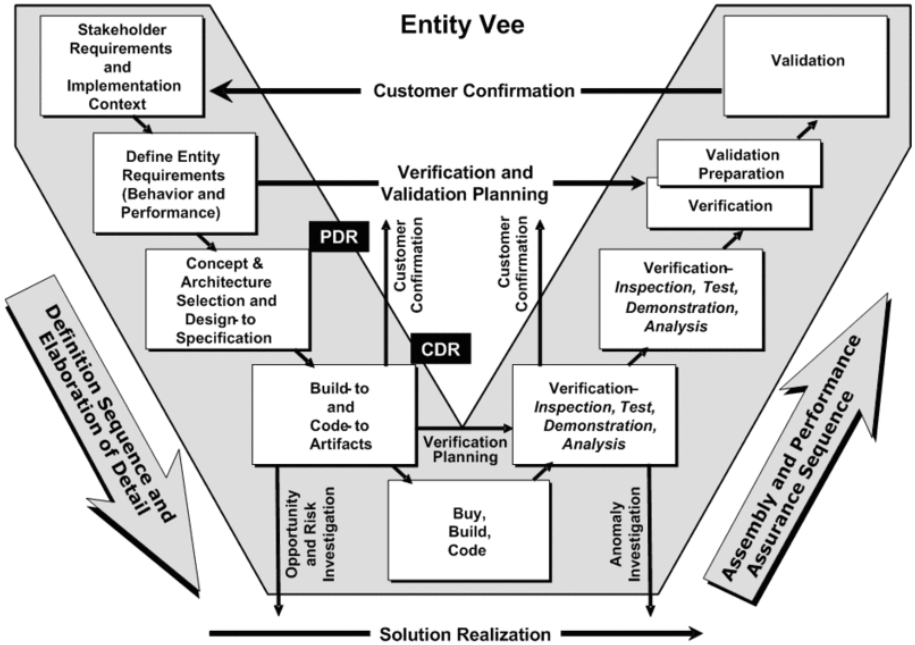


Figure 3.1: The Vee model is often used to visualize the development process. This variant of [Forsberg et al., 2005] fits the systems engineering process. (PDR: Preliminary Design Review, CDR: Critical Design Review)

information was exchanged through models, rather than exchanged through non-formal communication. This means this thesis can be further positioned in the Model Based Systems Engineering (MBSE) subdiscipline.

The International Council on Systems Engineering [INCOSE, 2006] defines MBSE as "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. MBSE is part of a long-term trend toward model-centric approaches adopted by other engineering disciplines, including mechanical, electrical and software. In particular, MBSE is expected to replace the document-centric approach that has been practiced by systems engineers in the past and to influence the future practice of systems engineering by being fully integrated into the definition of systems engineering processes."

The research in this thesis applies MBSE through the framework introduced in chapter 4. In this framework we exchange and store the design process information of a multi disciplinary design process. In [Estefan, 2007] we find a survey of methodologies that show what we need to perform MBSE. These methods, described below, are applied to a systems engineering process that can be roughly depicted in a V shaped model as in figure 3.1.

Telelogic Harmony SE [Hoffmann, 2008]: The Telelogic Harmony SE process assumes model (**domainentity**) and requirements (**requirement**) artifacts (**entity**) are available, which

are maintained in a centralized model/requirements repository (**architecture model**). It can then (1) perform identification and derivation of the required system functionality (**function**); (2) Identify associated system states and modes (**parameter**); (3) Allocate system functionality / modes (**function**) to a physical architecture (**entity**).

INCOSE Object-Oriented Systems Engineering Method [Friedenthal et al., 2008]: "Uses the OMG SysML language (see §3.6) to describe the relationship (**entityrelation, composition**) among the physical components (**entity**) of the system including hardware, software, data and procedures (**domainentity**). The system nodes define the distribution of resources. Each logical component (**function**) is first mapped to a system node to address how the functionality is distributed (**mapping**). Partitioning criteria are applied to address distribution concerns such as performance, reliability, and security (**aspect**). The logical components are then allocated to hardware, software, data, and manual procedure components (**domainentity**). The software, hardware, and data architecture are derived based on the component relationships (**parameter + knowledge bases**). The requirements (**requirement**) for each component are traced to the system requirements and maintained in the requirements management database." Quote by [Estefan, 2007].

IBM Rational Unified Process for Systems Engineering for Model-Driven Systems Development [IBM, 2012b]: Defines Roles ('who?') – A role defines a set of related skills, competencies, and responsibilities (**user attribute**); Defines Work Products ('what?') – A work product represents something resulting from a task, including all the documents and models produced while working through the process (**domainentity**); Defines Tasks ('how?') – A task describes a unit of work assigned to a role that provides a meaningful result (**designtask**). It also uses a number of standard architectural views to perform recurring tasks tasks such as partitioning of system (**entity + composition**), and product logical decomposition (**function + composition**).

Vitech Model-Based System Engineering Methodology, State Analysis [Vitech Corporation, 2012]: This methodology has four tenets to follow in MBSE: " (1) Make a model of the problem and the solution space via modeling 'language' (**architecture modeling language**); This helps facilitate model traceability, consistent graphics, automatic documentation and artifacts, dynamic validation and simulation, and promotes more precise communication. (2) Utilize a MBSE system design repository (**reusable primitives**). (3) Engineer the system horizontally before vertically (abstract design specification must be finished before advancing to more detail, see baggage handler case study 5.4). (4) Use tools (**architecture modeling tooling**) to do the 'perspiration stuff' and your brain to do the 'inspiration stuff' (reuse design knowledge)." Quote by [Estefan, 2007].

This Vitech MBSE approach uses a System Definition Language (SDL, see figure 3.2) that reflects some lessons that were also learned during the ACCSGMS project. As [Estefan, 2007] says: "Such an 'SDL' has a number of uses such as providing a structured, common, explicit, context-free language for technical communication serving as a guide for requirements analysts, system designers, and developers, and providing a structure for the graphic view generators, report generator scripts, and consistency checkers." This method shows that a single language can indeed be useful in MBSE. Some important generic prerequisites for such a language need to be defined in order to effectively structure, exchange and store

SDL Language*	English Equivalent	MBSE Example
Element	Noun	• **Requirement:** Place Orders • **Function:** Cook Burgers • **Component:** Cooks
Relationship	Verb	• **Requirement** basis of **Functions** • **Functions** are allocated to **Components**
Attribute	Adjective	• Creator • Creation Date • Description
Attribute of Relationship	Adverb	• **Resource** consumed by **Function** • Amount (of Resource) • Acquire Available (Priority)
Structure	N/A	• Viewed as Enhanced Function Flow Block Diagram (EFFBD) or FFBD
*Mapped to model element property sheets in Vitech CORE®		

Figure 3.2: Vitech MBSE System Definition Language (SDL) [Estefan, 2007]. Layering semantics upon abstractions.

information. In the ACCSGMS project, these prerequisites are roughly based on the first and second design axioms from [Suh, 1998], and are extended with criteria for (engineering) information structures by [Lutters, 2001].

- *The independence axiom* – Avoid overlapping, redundant and contradicting information, model information in discrete information objects (**basicobjects**).
- *The information axiom* – Minimize information content, make a model of what information is needed, but not more (and not less). To determine what is needed, think of a 'push-pull' mechanism; instead of stakeholders 'pushing' information into the model (such as in progress reports or design documentation or model repositories), stakeholders should only add information when asked, or pulled, by other stakeholders.
- *The communication axiom* – Minimize the need for communication. Information is pulled from the model by the user when it is needed, not pushed toward all users by the provider when information becomes available.
- *Independence of structure and instantiation* – During a design process, information objects are instantiated and being related to each other into a network (**basicobject, mapping, dependency, composition**). These networks can help conceptualize the system (system architecture) but can also have other tasks, such as finding ownership, or act as a basis for model generation (see §3.4). Therefore the network does not have to mirror the (hierarchical) structure of the system under design (**composition**). Indeed, there is no predefined way in which to make the information networks, as it may not even be a system that needs to be described.
- *Signify flexibility* – Information content can be added and changed throughout the design process, without losing functionality of the existing network. For example in evolving from a conceptual model into a model-generating meta model, the conceptual model part can still be used to, for example, keep track of requirements.
- *Independence of subjective perception* – The information content can be represented in multiple contexts **(mapping)**, and from multiple viewpoints (**view**).

- *Unequivocal methods* – The tooling for editing the information network are generic for all possible information content of that structure.
- *Independence of levels of aggregation* – There is no difference in the use of the information network on different levels. Sub-sets are operated the same as super sets (**composition**). Thus, the level of detail (system, system of systems, components) is independent of the used modeling mechanism.
- *Avert dominant hierarchies* – Hierarchies are often orderings of information objects based on the design discipline that uses the hierarchy. There can be multiple hierarchical views on a multi disciplinary information network (**view, composition**). Therefore the system hierarchy must not be 'leading' in the network hierarchy.
- *Transparency* – Different types of information should be recognizable – by humans and by software tools – without knowing its content, thus enabling systemic modeling. This necessitates a specialization of information objects **(concepts in architecture modeling language)**.
- *Enable navigation* – The information content can be reviewed, interpreted and changed with ease. A user can get from one information entity to another over every possible path through the network. This necessitates the network to be formed in cross-referenced graphs (directed or non directed) and sets of information objects (**mapping**, **dependency**, **composition**).

3.1.1 Required Capabilities to Perform System Engineering

The MBSE definition, examples, and information management rule set lead us to specifications for a framework that can capture and exchange information from all over the design process, and do so in a way that keeps the amount of (exchanged) information to a minimum. MBSE is an activity that requires the modeling of a number of important aspects (see also figure 3.1). Together with the topics of the problem definition this leads to a list of required modeling capabilities:

- Make a model of the relations between the results of the design process and the system requirements for design verification.
- Make a model of the shared design information between models centrally and uniquely.
- Make a model of the stakeholder goals, requirements and constraints for their design tasks and the models they use.
- Make a model of the external models as parametric abstractions, where the parameters are shared by the stakeholders.
- Make a model of how external information is connected to the abstract system functionality and composition.
- Make a model of the design aspects that cut across the design disciplines, and the models and requirements they have.
- Make a model of the system functionality, design aspects, system composition, and the parameterization thereof.
- Make a model of the parametric definition of the system that should attain these goals.

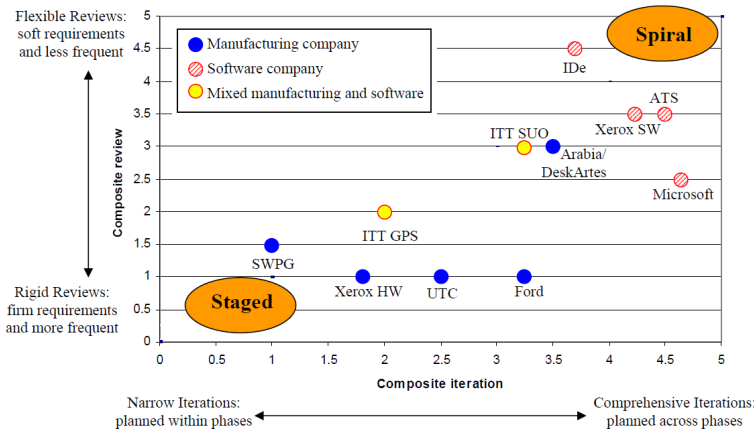


Figure 3.3: Between a spiral desigm process model, and a staged design process model, different companies have different needs for conducting planning and design review [Unger and Eppinger, 2006].

3.2 Design Process Models

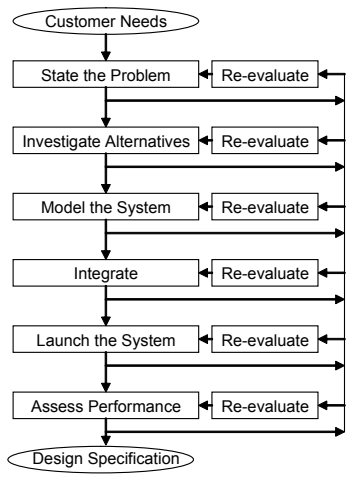
Design process models globally show the intended relation between planning and review of the design process, they can be used to plan design reviews and risk assessments, and perform design iterations based on their results [Unger and Eppinger, 2011]. There are two extremes in these design process models: the staged, or waterfall process such as in [Pahl et al., 1996], and the spiral process such as described in [Boehm, 1988]. Other process models are described by sources as the SIMILAR process in [Bahill and Gissing, 1998], the CAFCR process in [Muller, 2011a] or the mechanical design process in [Ullman, 2003]. In [Unger and Eppinger, 2006] we find when a certain type, spiral or staged or in between, was more convenient: Companies that develop systems with more easily integratable components tend to implement a spiral design process model, as this allows for quicker design iterations. This is mostly the case in software intensive systems. In companies where the focus lies more on physical prototyping to check requirements, it is of greater importance to implement the requirements right in a single well managed step, such as in a staged process. The results of this study are represented in figure 3.3.

Specifying the requirements of a design process is a crucial activity, as those requirements determine what the organization designs, and provide the metrics on measuring the system performance. The specification of the correct requirements is a discipline of its own, called requirements engineering [Nuseibeh, 2000], but that is out of scope of this thesis. However, modeling of these requirements, and how they are connected to other concerns, is in scope.

A design process model can be used at various time scales or levels of granularity: it could be on component level, at a conceptual phase, or it could be at system integration level at detailing phase. This is a question of what 'the system' is. A supplier of an airplane

Table 3.1: SIMILAR design process (Source: [Bahill and Gissing, 1998] and [Bahill et al., 1996])

(S)tate the problem	Understanding customer needs Stating the problem Discovering system requirements Validating requirements
(I)nvestigate alternatives	Defining performance criteria Exploring alternative concepts Sensitivity analysis
(M)odel the system	Functional decomposition System modeling System design Risk management Reliability design
(I)ntegrate	Design and manage interfaces System Integration
(L)aunch the system	Configuration management Project management Documentation
(A)ssess performance	Prescribing tests Conducting design reviews Total system test
(R)e-evaluate	Improve quality



passenger chair will view the chair as 'the system', while the airplane itself could be seen as 'the system' too, or indeed the textile fabric that is used in the chair. Within organizations, a system architect could view the total design of a system as 'the system', while a mechanical engineer could view the finite element model of the system as 'the system'. Using different design process models for different parts of the design process, spiral or staged, is one option. Alternatively, the SIMILAR process model seems an interesting choice for covering both spiral and staged modes of process. SIMILAR [Bahill and Gissing, 1998]: "State the problem, Investigate alternatives, Model the system, Integrate, Launch the system, Assess performance, and Re-evaluate." is shown in the figure 3.1. Not all steps in this design process model have to be executed in every case or at every design stage. However, it could be a good idea to give these steps a 'standardized' place in the design process, so that the process steps can be managed uniformly.

3.2.1 Required Capabilities for Managing the Design Process

The design process model provides the global layout and time of design activities. In this thesis, we will not take into consideration activities such as work planning or scheduling, as these are not directly in scope of consistency, integration and reuse. What is vital however is using the requirements to specify the definition for the design specification (see §3.4), and verify the design specification by checking whether the requirements are still being met or not. The requirements should therefore be traceable to any piece of design information defined in an architecture model (see Architecture Model as a ReferenceModel 4.2). As a short requirement:

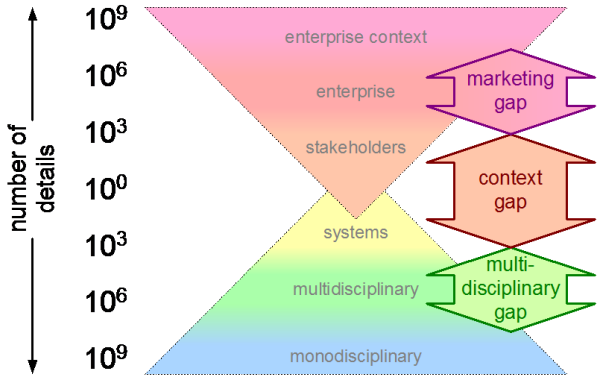


Figure 3.4: The detail pyramid from [Muller, 2011b], showing the information or translation gaps between the different granularities and abstractions.

- Make a model of the relations between the results of the design process and the system requirements to continually verify design.

3.3 System Architecting

"System Architecting is a means to create systems efficient and effective, by supplying overview, by guarding consistency and integrity, and by balancing" [Muller, 2011b]. For this architecting, a wide array of tools is available – however, it remains to a large extent an activity based on heuristics and experience, as the sheer amount and variety of information, and the complexity thereof is not readily modeled in any one model. Indeed multiple models are needed to optimally convey information of various types, at various times and in various representations, according to the wishes and tasks of the stakeholders.

This is graphically represented in figure 3.4, where an architect should match all kinds of requirements and details from all levels of the pyramids, as Muller says: "From needs and requirements to design: decomposition, interface definition, allocation, concept selection, technology choices anticipating engineering needs and constraints. Capturing all information that is required for logistics, manufacturing, legislation, maintenance, life-cycle support". Note: The figure also indicates why it is unlikely that there could be a model that captures all design information – while the order of magnitude in the figure is probably qualitative, one can imagine that a model of 'even' 10E6 details is pretty difficult to manage, fill or read.

System architecting thus requires a rather broad set of information. However, for consistency, integration and reuse, we will need to make a model of at least essential design information needed for architecting. And to make a model, we need a language that can

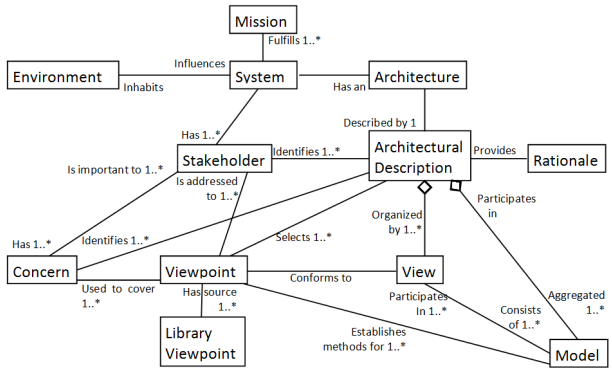


Figure 3.5: The IEEE1471 architecture description definition

represent this information. The remainder of this section will therefore filter some important concepts for such a language.

According to [Hilliard, 2007] on the IEEE1471 architecture website, "architecture (is) the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution (where: fundamental = essential, unifying primitives and principles; system = application, system, platform, system-of-systems, enterprise, product line,... ; environment = context, developmental, operational, programmatic, ...)" It is important to recognize that every system has an architecture, but that this architecture is not often explicitly modeled. The explicit representation of architectures are models called architecture descriptions.

IEEE1471 works with a number of concepts, which should be addressed in an architectural description (see figure 3.5). A system (**entity**) performs a mission (**function**) in its environment (**requirements**). Stakeholders have concerns (**aspect**) about the system, which are related to a partial view (**view**) of the system, and can be modeled (**domainentity**) from that viewpoint (**architecture modeling language**). A view is a description of the entire system from the perspective of a set of related concerns. A view is composed of one or more models (**domainentity**). A viewpoint determines the resources and rules for constructing a view (**knowledge base**).

SASD, SADT [Mylopoulos, 2004]: Structured Analysis, Structured Design Technique uses activity boxes (**function**) and object boxes (**entity**) to make a model of information flow. Each box gets an input and output flow, as well as a control flow (**mappings**). This modeling language was further expanded into the Integrated Definition Models IDEF0 to IDEF14 [KBSI, 2011]. These fourteen IDEF formats have not been fully developed, but are still in wide use. IDEF0 adds mechanisms flows (**entity**) to the concepts of SADT. IDEF1 to IDEF5 add concepts such as entities (**entity**), connections (**mappings**), categories (**type attribute**), keys (**identification attribute**), attributes (**parameter**), processes (**function + function relation**), state transitions (**function relation + requirement**), classes and objects (**concept**

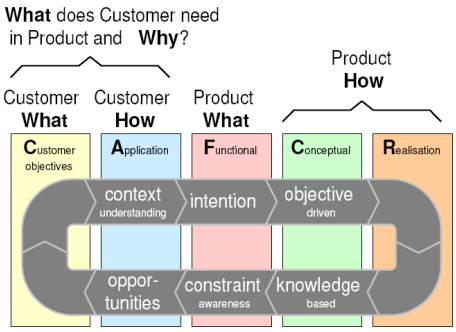


Figure 3.6: The CAFCR model puts the attainment of customer qualities first in the design process, these qualities should be continually traced throughout the design process [Muller, 2011a].

reuse), and ontology definitions (**architecture modeling language**). IDEF and SASD are also used in the United States Department of Defence Architecture Framework, DoDAF, discussed below.

CAFCR - Customer Objectives (**requirement**), Application (**aspect**), Functional (**function**), Conceptual (**function, entity, aspect**), Realization (**domainentity**) - focuses on the reason a company makes a system in the first place, the satisfaction of customer objectives [Muller, 2011a]. As such, these objectives are qualities that are continually monitored throughout the design process as shown in figure 3.6.

Design Methodology for Mechatronic Systems [Gausemeier, 2007]; develops a tool and framework to specify a product model. From this model, product instances can then be instantiated. This method spends much attention on the architecture of the system. It provides a number of standard views on a system that interact together to generate the product models: Requirements (**requirement**), environment (**requirement**), system of objectives (**requirement**), application scenarios (**aspect**), functions (**function**), active structure (**entity + parameter**), shape (**entity + parameter**) and behavior (**function + parameter**). Each of these views has a specialized language. Other aspects, besides those in the pre-defined views, cannot be added.

Quality Function Deployment (QFD)[CIRI, 1994] (Inter-)relates technical requirements (**requirement**) to the customer requirements (**requirement**), and gives those relations a metric or benchmark (**parameter**) to manage that quality (**aspect**) of the system under design. It also shows the ownership of the quality (**view**) – the department or people who must control that part of the system quality. The House of Quality (figure 3.7) is used as a tool to make a model of all these relations. The QFD is important to view design trade-offs when requirements are contradicting (such as in a car with large enclosed space for comfort, and small profile against air resistance).

DSM, the design structure matrix [DSMWeb, 2009]: A matrix with on its axis (**decompositions** of components (**entity**), people (**entity**), activities (**function**) or parameters

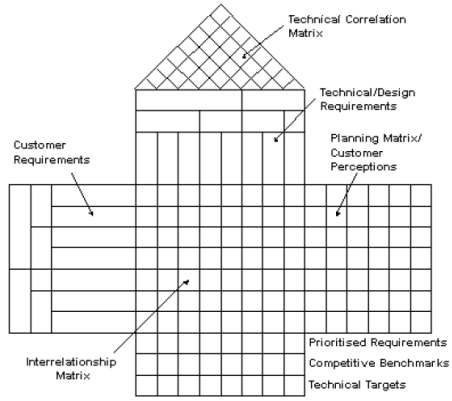


Figure 3.7: House of Quality tool for Quality Function Deployment [CIRI, 1994].

(**parameter**). Inside the matrix the (weighed) relations are given (**mapping** + **parameter**) between the axis elements in a certain domain (**aspect**).

An interesting option to bridge the gap between the DSM – with its system structure and interfacing relations – and the QFD – with its required quality and performance model – could be the FunKey method of [Bonnema, 2008]. FunKey focuses on the allocation and budgeting (**parameter**) of functions (**function**) and key product drivers (**requirement**) on (sub) systems (**entity**) as a way to architect a system from a customer centric perspective.

Architecture frameworks such as the Department of Defence Architecture Framework (DoDAF) [Department of Defence, 2012] are available to do architecting on an organizational scale, and throughout the development process. The DoDAF is a loosely coupled set of 8 viewpoints (**aspect**), each with their own model types (**view**), a set of 44 different model types in total. Many of these models are focused on relating activities (**function**), resources (**domainentity**), roles(**aspect**) and agents(**entity + view**) across the organization(**mapping**) and through time (**functionrelation**). The physical system description (**entity**), or rather its interface description (**entityrelation**) is just one model out of 44. A meta model is available to exchange the information across the viewpoints. For this activity a shared lexicon and shared semantics are defined in the DoDAF Meta-Model (DM2).

3.3.1 Required Capabilities for Modeling System Architecture

Software tools such as the IBM Rational System Architect [IBM, 2012b] can be used to develop some of the models discussed in this architecture section. However, such a tool might not be able to integrate the data from the various models into an underlying interconnecting model such as is desired according to section 3.1. But more importantly, most existing tools do not allow for the mixing of viewpoints in a single view: there are functional views, requirement views, views to map functions to requirements, etcetera, but there are no views

that mix all given information types. This mixing is not necessarily impossible. Therefore, the architecture modeling framework will use a minimal set of concepts to make a model of system architecture. These modeling capabilities we can summarize as:

- Make a model of the abstract system functionality and composition.
- Make a model of the stakeholder goals, requirements and constraints for their design tasks and the models they use.
- Make a model of the design aspects that cut across the design disciplines, and the models and requirements they have.
- Make a model of the abstraction of design goals, requirements, and constraints.

3.4 Design Knowledge Modeling

As reuse is a main concern in this thesis, it follows that reviewing methods for capturing design information for reuse is essential. Roughly speaking, we need two distinct options, capturing reusable pieces of design specification, and capturing the knowledge needed to construct the design specification. Much of the design knowledge of an organization is contained either in a tacit form in the heads of the individual engineers, or in a proprietary format in the design tools and design documentation they use. However, often engineers will define scripts or toolboxes of templates in their design tools to facilitate the construction of models. This building of reusable design primitives is often a 'grassroots movement' – 'Excel engineering' is a term mentioned a couple of times during the ACCSGMS project. However, tool development to help the engineers is sometimes supported by management directly, in what [Muller, 2011b] calls a Shared Assets Creation Process. Such a process can be useful if the design processes are of such uniformity that it pays back the investment of a tool development department (e.g. in baggage handlers §5.4, in printers 5.6 or in lithography systems §5.5, see also section on lessons of a software world §3.6). The benefits of a shared assets creation process can then be large (also from [Muller, 2011b]):

- Reduced time to market by building on shared components.
- Reduced cost per function by building every function only once.
- Improved quality, reliability, and predictability by maturing realization of the product definition.
- Easier diversity management by modularity.
- Increased uniformity and less learning because employees only have to understand one base system (meta model).
- Larger purchasing power by economy of scale.
- Means to consolidate knowledge.
- Increase added value generation by not spending time reinventing existing functionality.
- Enables parallel developments of multiple products.
- 'Free' feature propagation from product-to-product or project-to-project.

In order to make a model of knowledge, it has to be made explicit. An interesting method to make knowledge explicit in relation to the system's architecture is described

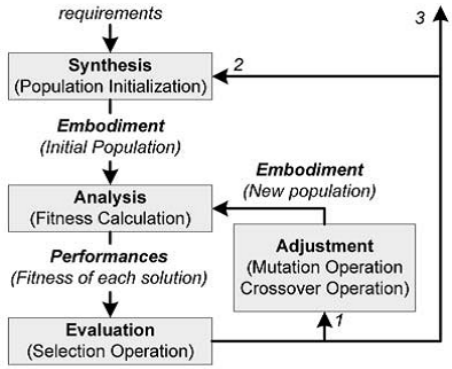


Figure 3.8: The automated design process in a smart synthesis tool according to [Jauregui-Becker et al., 2008]. Note that the (customer) requirements are the input of the process.

in [Borches Juzgado, 2010]. Borches uses A3 forms that circulate among the stakeholders to capture and condense design knowledge of important system aspects. This leads to a common understanding of the aspect as well as providing a common platform for the systems development. Modeling concepts he uses are: context of the system aspect (**aspect**), functional (**function**) and physical decomposition (**entity**), concerns (**aspect**) and requirements (**requirement**) of the stakeholders involved, and how these concepts are interrelated (**mapping**), design strategies and issues (**out of scope**), and the parametric definition and quantification of design decisions and constraints of the aspect (**parameter**).

Once the design knowledge is explicit, design knowledge languages often seem to use the parametric paradigm [Suh, 2001]: Make a design space of parameters, and fill in their values according to a set of algorithms. All value sets that fall within a set of requirements are solutions. In the paper of [Jauregui-Becker et al., 2008] we see how to formulate a multi domain design problem into something a computer can solve (See figure 3.8). We describe this language as it captures both the means of structuring a design space as well as the solution algorithm and solutions. The language has the concepts: “Objective functions (**requirement**): evaluate the overall performance of the design. Element - which are a class description of a design artifact (**entity**). Embodiment parameters: is considered as the subset of descriptions of an element upon which instances are created to generate design solutions (**entity + parameter**). Scenario parameters - attributed to elements in the natural world and considered in measuring a design artifact’s ability to accomplish its function (**requirement + parameter**). Performance parameters: are those descriptions used to express and assess the artifacts behavior, and are calculated using analysis relations (**function/aspect + parameter**). Confinement constraint parameters: determine the range in which a description can be instantiated (**range attribute**). Analysis relations: use known theories, for instance the laws of physics or economics, to make a model of the interaction of the design artifact with its environment and to predict its behavior (**mapping, formula**). Topology relations: define the configuration between embodiment and scenario elements (**composition**)."

Other methods, such as the one described in [Wielinga et al., 1995] have more or less the same concepts to describe a design task: "The problem can be formulated in terms of a set of formal requirements and constraints (**requirement**). The solution can be formulated in terms of a set of design elements (**entity**), with a set of relations over the elements so that design structures or assemblies can be modeled (**mapping, composition**), and a set of parameters (**parameter**) that have values assigned to them (**value attribute**). There exists a theory that allows deriving information about the relation between the solution, requirements and constraints – the Domain Theory (**out of scope**). Design choices that constrain the space of design options, but which are not part of the domain theory, can be chosen freely (**parameter**)."

FBS (Function, behavior, state) modeling by Yoshikawa and Tomiyama [Reich, 1995] focuses on a knowledge capture of processes rather than the structural systems that execute the processes [Alvarez Cabrera et al., 2009]. As such, it provides an interesting way to make a model of mechatronic products, where the functions the system performs are often more complex (e.g. software intensive) than the system's physical artifact. A characteristic of FBS is that it is focused towards qualitative behavior simulation through computational means. This simulation can then – through a Qualitative Process Abduction System [Ishii et al., 1993] – suggests physical features that can achieve that behavior. The modeling concepts in this language are: Function: An abstracted, state-free, behavior description (**function**). Entity: An entity represents an atomic physical object (**entity**). Relation: A relation represents a relationship among entities to denote a static structure (**composition**). Attribute: An attribute (**parameter**) is a concept attached to an entity and takes a value to indicate the state of the entity (**value attribute**). Physical phenomenon: A physical phenomenon designates physical laws or rules that govern or trigger behaviors (**mapping**). Physical law: A physical law represents a relationship among attributes (**formula**). State: current values of attributes in a set of related entities (**entity + entity relation + composition + parameter.value**). Behavior: a sequence of state transitions (**function + function relation + requirement + parameter**). Aspect – denotes sets of all entities, attributes, relations, physical phenomena and time of the current interest (**aspect**).

MOKA is a specification, derived from UML specifically for Knowledge Based Engineering [Brimble and Sellini, 2000]. It specifies a design in the standard views of Structure (**entity**), Function (**function**), Behavior (**function + function relation + requirement + parameter**), Technology (**domainentity**) and Representation (**view + architecture modeling tool**). Using these views, models of system instances can be created automatically.

3.4.1 Required Capabilities to Model Design Knowledge

It is important to recognize that digital product development tools contain a lot of design knowledge. A tool can read a model, reason about it, and then execute the model, provide analysis of the model, make model extensions to this model, or generate another model from the model. In other words, the tool speaks the same modeling language as the model, but the model is just a static representation of the knowledge, while the tools have the semantics and the knowledge to do something with the model. We want to use the knowledge stored

in these tools in our framework. For this to work, the reusable knowledge must be stored in knowledge bases, that speak the same language as the model. This requires the following capabilities:

- Make a model of the external models as parametric abstractions, where the parameters are shared by the stakeholders.
- Make a model of the parametric definition of the system that should attain these goals.
- Enable knowledge bases to interact with the integrated model, for model modification, generation or transformation.
- Align domain specific primitives to generic functional goals or design aspects to facilitate generation of external models.

3.5 Design Workflow Modeling

Apart from the architectural concerns and the knowledge capturing, an important third dimension is present in design processes, namely the people and organization that actually make the design. As stated in the problem definition chapter 2, many issues stem from the interactions between human stakeholders in the design process. Many tools are available to manage these interactions and the design process in general.

Tools – or rather tool suits and frameworks – such as IBM Rational Jazz [IBM, 2012a] or SAP [SAP, 2012] are built specifically for collaborative design. They make sure the stakeholders (**user attribute**) have access to the most up-to-date models (**domainentity**) pertaining to their design activities (**aspect, view**), as well as giving them project timelines (**date attribute**) and requirements (**requirements**) and rationales (**documentation attribute**) for these models. However, the models in these tools are mostly modeled as a piece of data without any representation of its input, output or relation to a system architecture or workflow. This is somewhat mitigated in the Extended Enterprise Modeling Language [Krogstie, 2008], which provides a language to make a model of processes (**function**), data (**entity**), resources (**domainentity**) and goals (**requirements**), and focuses specifically on the ownership (user attribute) of the information.

As can be read in [Mengoni et al., 2010]: “most of the Product Data and Product Lifecycle Management systems (PDM/PLM), Engineering Data Management systems (EDM) and electronic Bill Of Processes systems (eBOP) ... are strongly document-oriented, have a structured and not much customizable data model and suffer from inter-enterprise integration difficulties". They then define a framework where design decisions of design activities are evaluated before starting a new design activity. A JECA (Justified-Event (**parameter**) – Condition (**requirement**) – Action (**function**)) engine can evaluate the design decisions, to maneuver the process through various solution spaces. Such an engine falls out of the scope of this thesis, however, the architecture language must be usable as a data language upon which to use such an engine.

Within the aforementioned DoDAF [Department of Defence, 2012], some of the 44 views are specialized in matching processes, resources and stakeholders together. In the Opera-

tional Viewpoint nine models are specialized in information such as exchanged resources (**domainentity**) and their input/output (**design task + designtaskrelation**), roles and hierarchy in organization (**view**), maps to system capabilities (**function + requirements + aspects + design tasks**), and event trace models that specify triggers to certain process results (**aspect + requirement**). Furthermore, in the Project Viewpoint three models are specialized in information such as time lines, milestones, and maps to system capabilities.

3.5.1 Required Capabilities to Model Design Workflow

Project management information such as time lines and task scheduling will be out of scope of this thesis, as they do not directly relate to the issues in the problem definition. What we do want to make a model of are the design tasks and their context – who made it, what was the input, what were the results, to what architectural concern does it pertain. As such, the desired models will be a snapshot of the design information as it was available in that point in time. The model will change over time, as more information becomes apparent, and more of the design specification is built, which necessitates a form of versioning. This versioning as a required capability was concluded at the end of the research period, and so remained an unresolved issue. Versioning of the integrated shared models can probably best be done on a basis of important design decisions, as argued in [Moneva et al., 2009]. At the moment of writing this thesis, cooperation with the Embedded Systems Innovation by TNO (ESI) is underway to look into this idea, however, it remains out of scope of this thesis.

What we do want to capture is the design tasks, and their input and output, thus forming a workflow that can also be used to define ownership of design information – capture design tasks to show where a certain decision has come from. For example, a 3D CAD model of a cube is defined by its rib length. Where does this value for the length come from? Who determined it and when? What requirements caused it to be that value? We can summarize the required modeling capabilities as:

- Make a model of the information exchange between the design tasks/stakeholders.
- Make a model of the stakeholder goals, requirements and constraints for their design tasks and the models they use.
- Make a model of the information shared and transferred between stakeholders.
- Make a model of the workflow of the design process.

3.6 Lessons from a Software World

Reuse of design knowledge is often a good way to improve time to market, and system performance while reducing cost, as we saw in the design knowledge modeling section 3.4 and in a section of the problem definition 2.2. An effective means of knowledge reuse is to put the knowledge into models and model generators,and guide design specification creation in a process of Model Based Systems Engineering. The discipline of software development has given us important lessons on how to make models and manage design processes. This is a consequence of more and more digital product development (DPD) on one hand, and more

embedded software in systems on the other hand. DPD (CAD, CAM, ERP, PDM) allows for the digital construction, testing and prototyping of a large part of the design specification. Most DPD tools require some skill in programming to be used effectively: think of making a Matlab or Ansys model, the graphical interface will only be used by beginners, advanced users will make scripts, and set up reusable toolboxes and algorithms (knowledge bases).

When expanding into more serious design tool development – such as the shared assets creation process in [Muller, 2011b] – advances in software development practices provide us with the standards to make our own tooling: Domain Specific Languages (DSL), Unified Modeling Language (UML [Weilkiens, 2007]), Systems Modeling Language (SysML [Weilkiens, 2007]), Web Ontology Language (OWL [W3C, 2012a]), Extensible Markup Language (XML) [W3C, 2012b], etcetera...

An important 'philosophy' of thinking about both software and hardware systems is the object oriented (OO) analysis and design paradigm [McLaughlin et al., 2006], which is supported by modeling languages such as UML (see below). Object orientation roughly divides the system in structural objects (**entity**), and behaviors in the form of executing methods of the object (**function**). The 'state' of an object at any given time is defined by the values of its attributes (**parameter**).

Both structure and behavior are defined in a class, that instantiates objects and builds references (**mapping + dependency + composition**) between objects through their methods. In a more mathematical sense, the objects have attributes that define sets of information, and the methods define (directed) graphs of information. Exactly how the attributes change values (or states) is defined inside the methods (**knowledge bases**). The methods are thus also used to construct the relations between objects. For example: object1.setParent(object2) will relate object1 and object2 into a hierarchical set. As such, the object oriented paradigm provides us all the necessary requirements to build (engineering) information structures according to the list of requirements in the MBSE section 3.1.

There are a number of 'standard' ways of defining how the objects will interact. These ways can be defined in the Unified Modeling Language, as described in a systems engineering context in [Weilkiens, 2007]. UML has a number of pre-defined method and attribute archetypes that occur in many (software) systems. Relations between objects can be typed as either aggregation (books are inside a library (**mapping**)), composition (walls are part of library (**composition**)), association (visitor can be member of library, and library has members (**mapping**)), or dependency (a member needs a library to get a book (**functionrelation**)), and classes can have a generalization (Thriller and Scifi are types of Book) and abstraction (Book implements methods of DatabaseItem) relations.

Furthermore, the relation between methods that are not of the aforementioned standard types can be modeled through an interface definition (**mapping**), where ports show which attributes of objects share information (these special attributes are called parameters (**parameter**)). This way, information not relevant, or forbidden to the world outside the object can be 'encapsulated' within an object.

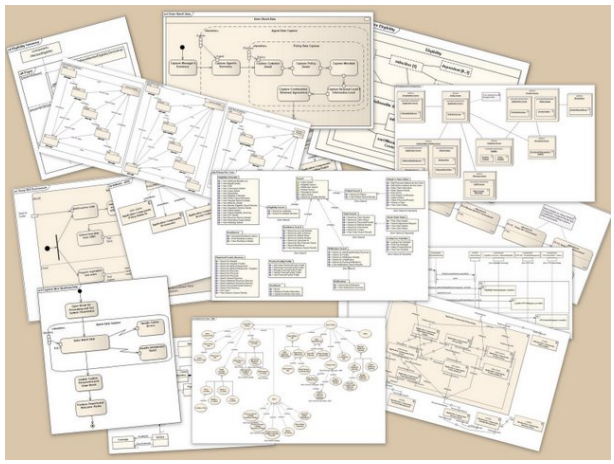


Figure 3.9: A collage of UML diagrams

Good object oriented (OO) models only exchange information through methods that manage the ports in the interface. These relations ensure there are quite a number of things that can be modeled with the UML. To facilitate model construction, UML provides standard diagrams (**view**) (see figure 3.9). Class diagrams and composite structure diagrams can be used to specify which objects can be made and what ports they have and how they are structured and connected (**entity + parameter + entityrelation**). Activity diagrams make a model of the behavior of a system (**function + parameter + functionrelation**).

A good practice is the matching of activities to methods inside a class. Thus one can define which ports are connected to which methods and where important decisions and prerequisites (triggers (**requirements**)) are taken that define the direction of the flow of information. Inversely, the state diagram can model how the value of parameters trigger certain methods. Finally, a sequence diagram can represent how various objects execute and terminate methods over time by exchanging messages (which means executing a method in another remote object, or executing a method locally when listening to the state of another object results in a certain trigger).

A further specialization of the object oriented paradigm is the Systems Modeling Language (SysML, [Sysml.org, 2012]) that is being developed to specifically address the activity of model based systems engineering (MBSE). In SysML some concepts are changed in relation to UML or OO: Classes are called Blocks, the parametrics of the model get their own diagram, and activities are modeled to closely resemble functional decomposition and functional flow instead of execution sequences of methods encapsulated in objects.

Indeed the Activity becomes a class type in itself, apart from the Block. This breaks with the original OO paradigm of encapsulating behavior as methods in objects, as [Weilkiens, 2007] says: “The strict separation of structure (block) and behavior (activity) is not object oriented. A particular characteristic of an object is the unification of structure and behavior

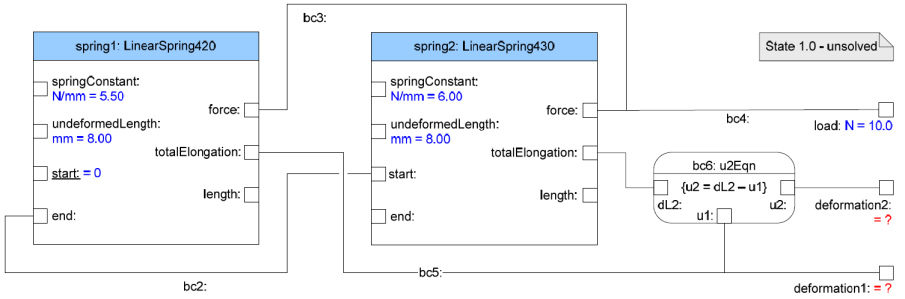


Figure 3.10: An example of a SysML model from [Peak et al., 2007a]. Two instances of a spring class are connected to an instance of a formula, the ports are the parameters that need to be calculated.

in the form of attributes and operations (executing methods). This corresponds to objects in the real world". In the (conceptual) design world this separation of functionality and function performers is commonplace, as allocating such functions to certain components is a large part of the conceptual design process (see also 3.3). However, separating the behavior from the structure at the SysML language level seems to make it difficult to integrate activities, or executed system functions, into system components. A newly introduced 'Allocation' relation can be used to match a block parameter with an activity parameter, but this is only represented as an annotation in a diagram, or in a separate 'allocate activity partition' which is not part of the SysML standard.

Furthermore, 'allocate' is an implementation of the UML 'abstraction' relation, which seems a semantic mismatch to this author. Why this separation from the OO paradigm is chosen is not clear to the author, although [Weilkiens, 2007] mentions this has been done as a way to incorporate the popular Functional Flow Block Diagrams (FFBD) modeling into the SysML standard. Some important distinctions between SysML and the Architecture Modeling Language of this thesis are further evaluated in the discussion chapter 6.

Still, the SysML standard grows in popularity and with good reason. It enables us to make a model of systems at various levels of detail, in an abstraction that people from all kinds of disciplines understand, and provides interesting ways of defining and incorporation of reusable tooling. The Multi Representation Architecture (MRA) from [Peak et al., 2007b] is an interesting example of how SysML can support MBSE.

The MRA shows the possibility of combining design tools, design primitives and design descriptions into a single model generating and model analysis framework. It works with a basic primitive, the composable object (as defined by [Paredis et al., 2001]) that is specialized into other dedicated modeling concepts such as the Analysable Product Model, the Context Based Analysis Model, the Solution Method Models, and the Analysis Building Block. This shows the power of the parametric definition of design problems, and the way such definitions can be connected to solvers, analyzers and system descriptions through their port definitions (see figure 3.10).

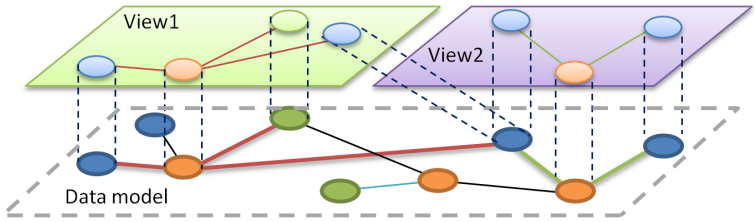


Figure 3.11: A model where the view is the encapsulating object. Views represent pieces of an underlying shared data model, which integrates the information, and can help to keep it consistent.

The aforementioned Domain Specific Language and Extensible Markup Language will be discussed in the Architecture Model as a Meta Model section 4.3, as these were implemented for the architecture modeling framework.

3.6.1 Leveraging Software Development Methods

The object oriented paradigm of the software development world will be an important basis in the development of an Architecture Modeling Framework. And, while some basic lessons have been taken from the UML and SysML, the Architecture Modeling Language (AML) will go in another direction. Basically, the referencing mechanism of OO will be utilized in that we use it to define information networks as were defined in section 3.1. However, we will not define standard views such as in the UML and SysML. Instead, we will introduce the concept of generic, stakeholder centric, views, and allow any and all concept and relation types to be drawn in these views (see figure 3.11).

Also, the attributes of the AML concept types (equivalent to block, activity, etcetera) will be standardized to perform information management functions (who, what, when, where, why, in what way, by what means). Which means all parameterization – that is encapsulated in blocks or objects in the UML and SysML will be externalized: the idea of 'encapsulation' is abandoned, as it will stand in the way of mapping information to multiple 'capsules' or across levels of granularity (see the architecture modeling language section 4.1). Indeed, the idea of the Composable Object is replaced with composition relations (parent/child) between certain types of objects. Because the architecture model will still be machine readable, possibilities like defined in the Multi Representation Architecture are still available, which will be demonstrated in the case studies 5. To summarize, a framework will need the following capabilities:

- A consistent integrated model in which to track the 'who, what, when, where, why, in what way, by what means'
- A integrated model with stakeholder-specific views to maintain overview and context
- A integrated model that facilitates construction of the design specification
- Enable knowledge bases to interact with the integrated model, for model modification, generation or transformation.

- Align domain specific primitives to generic functional goals or design aspects to facilitate generation of external models.

3.7 Perceived Gaps in Current Approaches

As we have read in this chapter, literature and industrial practice provide us with many of the tools and methods to solve the individual issues as stated in the problem definition 2. However, a total solution is not available, and, according to sources such as [Muller, 2011b] and [Torry-Smith et al., 2011], such a solution is not even possible due to the amount of data such a 'Model of Everything' should capture, and the amount of semantics that the solution should 'know' (how to make a model of all knowledge of all different disciplines and execute and calculate all different models). The Architecture Modeling Framework in the next chapter will need to provide a work-around for the 'Model Of Everything' dilemma.

A specific model type is often used to support a single design method, which is not surprising, as one can then optimize the concepts, symbols, syntax and representation of that model to the method one wants to implement (such as the house of quality for quality function deployment in figure 3.7). However, from this state of the art chapter we can deduce that a lot of methods and modeling languages have a great overlap in the use of concepts – printed in **bold** throughout this chapter – while their modeling methods are mostly distinct.

This signifies the first gap in current approaches. There is no unifying modeling language to exchange information across disciplines (sections 3.1 and 3.3), across design process phases (sections 3.2 and 3.5) and across levels of granularity (sections 3.3, 3.4, and 3.6). This means we lack the means to make a model of the design aspects, especially the ones that cut across the design disciplines. We lack the means to model how information from the mono disciplinary design tasks is handed over to subsequent design tasks. And we lack the means to provide a big picture of how mono disciplinary design information is connected to the system wide concerns such as functionality, requirements and decomposition. In the next chapter we will therefore choose a set of concepts that recur in a lot of methods. From these concepts, we construct an Architecture Modeling Language (section 4.1), that can be used to perform (some of the) methods described in this chapter, as will be demonstrated in the case studies 5.

This also implies that we need to specify multiple views per model, as putting in too many symbols in one view will make it unreadable. Therefore we combine the concepts types into views based on stakeholder concerns, instead of combining all partial stakeholder concerns in a view for each concept type. The views enable the modeling of stakeholder ownership of information (section 3.5) and design resources – such as important design decisions or design task results or links to architectural concerns – such as requirements or system functionality (section 3.3).

To ensure that the views will not become isolated models, as they are now in the various models discussed throughout this chapter, the views must be a representation of an

underlying consistent and integrated data model, with relations to and from concepts in different views. This makes sure that hand over of important design information between stakeholders is formalized, and that information is stored according to the rules stipulated in section 3.1. Such a shared model can then be used in two roles, the meta model 4.3 for the abstraction of other design information, and the reference model 4.2 for the navigation, review and exchange of design information.

Effectively, by not specifying the appropriate design or engineering method (or viewpoint) for each part of the model (or view), we are separating the design method from the model and the modeling language. This means knowledge bases can be developed for a partial model, while the rest of the model will be ignored by this knowledge base. In most other methods from this chapter that would be silly; why make a model of things not used in the method? However, as the model can be used by multiple methods, this becomes a strength: automating and reusing knowledge when possible, and keeping the model 'tacit' and flexible when appropriate. To make this possible, we can take stock of how the world of software development has tackled design issues (section 3.6) and formulate how software tooling can be plugged into a design process. Such tooling can contain the knowledge to read and write the models and interact with (external) models as will be demonstrated in section 4.4.

> **Note**: As an example of a partially formalized model, think of a LaTeX 'model' that generates this thesis: Most of the symbols in that model trigger no reaction from the 'knowledge bases' – the packages – of the LaTeX compiler. however, when it finds a '\' then it knows a command will be given in the next string of symbols, for example\textbf{bold} will make a word **bold**. This system makes it possible for a large community of people to build and 'plug in' packages on this foundational syntax: \textbf{} makes text bold,\printindex{} constructs and prints the index of a document, \textbackslash prints a backslash, etcetera...

In this section we reviewed the perceived gaps in multi disciplinary design processes as they were concluded from this state of the art chapter. These gaps will be bridged with the architecture modeling framework (AMF) of the next chapter. However, to provide a more neutral review of the efficacy of the AMF, in the discussion in chapter 6, the AMF will be reviewed in light of the gaps, as perceived by these literary sources such as [Muller, 2011b], [Reichwein and Paredis, 2011] and [Torry-Smith et al., 2011].

4. *Architecture Modeling Framework*

The architecture modeling framework is the main deliverable of the ACCGSMS project. In this chapter, we will explore this framework in the context of consistency, integration, and reuse in multi disciplinary design processes. The chapter follows the main constituents of the framework: the architecture modeling language, the architecture models and the architecture modeling tooling.

This chapter will describe the architecture modeling framework (AMF). This framework has proven to be an important tool in the ACCGSMS project, as it allowed the deployment of experiments and case studies and feed back their results to improvements in the framework. As such, the framework evolved from a multi domain system modeler of the Baggage handler case in §5.4 into a graphical editor for model based systems engineering and architecting. As new options and requirements became apparent through more (industrial) cooperation and experimentation, the framework was updated, while keeping the underlying class structure as 'lean' as possible. The result is a framework that can be split into three distinct components:

1. *The architecture modeling language* (§4.1). The language started out as a set of classes to define parametric design problems on one hand, and the Function Behavior State set of classes to make a model of behavior on the other hand (see the thesis of [Alvarez Cabrera, 2011] for more on behavior modeling in architecture modeling language). Currently, the language is a synergy of concepts as also described in the state of the art chapter (3). This means the language definition is not static, and is not meant to be static. As insight progresses, more classes, and relation types can be added, without disturbing the functioning of the other two framework components (models and tools). Indeed, as this thesis is written, future additions to the language are being discussed (see chapter 8).

2. *The architecture model* (§4.3 and §4.2). The framework provides a tool that enables us to make architecture models. The architecture models are written in the architecture modeling language, and are represented in a diagram format to make them accessible for human interaction. An architecture model can be used in multi disciplinary design as both a meta model (S4.3) and as a reference model (§4.2).

3. *The architecture modeling tooling* (§4.4). Although the models of the AM tool are represented in diagrams, other representations are possible, as the language and its representation are defined separately. This allows for the direct manipulation and con-

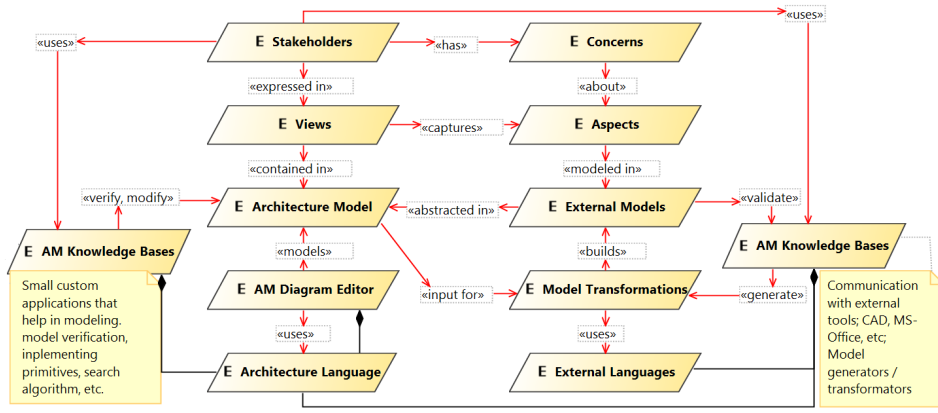


Figure 4.1: Basic use case layout of the intended framework, with the stakeholder on top.

struction of the architecture models by software tools. The most important of these tools is the architecture modeling diagram editor. With the editor human users can construct the AMs. The editor also contains tools to keep the models consistent and enable reuse and referencing of segments of architecture models defined elsewhere. In the case studies (see chapter 5) case specific tools – knowledge bases – have been built to interact with the models, for model generation and transformation, and for knowledge based engineering algorithms.

4.0.1 Architecture Framework Use Case

This section will sketch a use case scenario for the framework to introduce the reader to the content of this chapter. In figure 4.1 we see the framework in its context. The stakeholder – who can be an engineer, designer, manager, researcher, or indeed anyone who uses the AMF – is central. The stakeholder has certain concerns about something (in this thesis we limit these concerns to system development in multi disciplinary design). These concerns are about certain aspects of a system or its design process, for example the mechanical design, or the production cost, or the performance in an untested environment. These concerns can then be modeled with the architecture model diagram editor (§4.4) in a view in the architecture model (AM). In such a view, the aspect can be connected to other design information from other views. The stakeholder can now use the view to communicate with other stakeholders; They can for example abstract and conceptualize the system's architecture, discuss the trade offs of important design decisions, and define ownership over design information.

As the AM has no predefined semantics, it cannot be executed by itself to perform an analysis or synthesis to give the stakeholder an answer to the concern. Instead, the stakeholder makes an external model in a tool that is specialized in such aspects (such as a 3D ge-

ometry modeler, spreadsheet software or multi physics simulator) and references that model in the AM (§4.2). Certain parametric properties of the external model can then be abstracted into the AM to signify the input (design decisions) and output (metrics), for that model, and to connect the parts in the model to certain generic system properties in the AM, such as the topology of components, or the functional composition and sequence (§4.3). The result is an AM that integrates the view of the stakeholder with other views into a shared model. When information was already defined in other views, the stakeholder can reference that information into the new view to keep the shared model consistent and integrated.

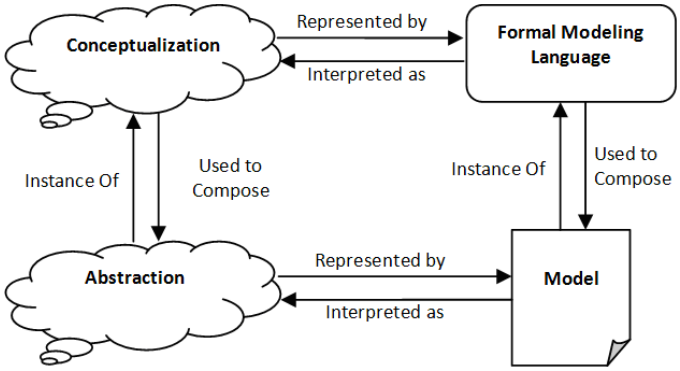
Basically, the AM view is just a diagram – only the basic functionality of the AM Diagram Editor is available, such as search, new instance, delete, filter, save, load, zoom, layout, user name, date of instantiation, unique identifiers, uri of external model, etcetera... The semantics of the model beyond that defined in the AM language (AML, §4.1) are understood only by the human interpreters of the model. For example, a human will understand the 'radius' of a 'wheel', where the basic AMF will not. However, we can envision situations where we want to automate certain design tasks that depend on the information in the AM. Therefore we need to put the semantics of the model into a knowledge base – a software tool dedicated to certain tasks (§4.4). The knowledge base can then interact with the AM, either to do something within the AM, or with both the AM and external models and tools.

Within the AM we can envision knowledge bases that do verification ("is the requirement met?") or perform information management functions ("make a new view, referencing all the entities of the AM"). For external communication, we can envision knowledge bases that know both the AM language and the language of the external tool. This enables the generation and transformation of models to and from the AMs, either by manipulating the models directly, or by using the application's programming interface (API) to utilize the tool's functionality. Expanding this, we can see the possibility of defining design process patterns: a sequence of design tasks, cutting across multiple model types and tools, with design information from various disciplines, all executed automatically (see the case studies §5.3).

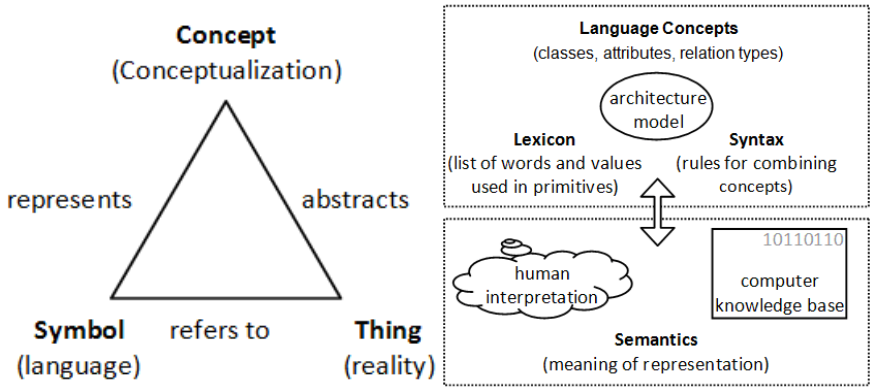
4.1 Architecture Modeling Language

The architecture modeling language (AML) is the language of the AMF. With this language, we make a model of the shared design information in a multi disciplinary design process. This section will expand on the notion of language to explain the 'degrees of freedom' in defining such a language, and defend the chosen formalization of the language.

A language is used to communicate conceptualizations of things in reality through symbols (See figure 4.2b). Examples of symbols are words (text or vocal), gestures, and figures. Humans use a broad set of symbols to communicate, and are likely able to understand each other when they share the set of symbols. For example, two English speaking people can exchange information about soccer because they both use the English language, and know what soccer is. In such a natural language, both parties share the symbols and syntax or grammar of the language, and have a more or less similar (but fuzzy) shared conceptualiza-



(a)



(b) (c)

Figure 4.2: Functionality and properties of a language. (a) The formalization and interpretation of tacit concepts in a model, according to [Guizzardi, 2007] (b) Ullmann's Triangle: The relations between a concept in someones head, the symbol with which it is represented and the object in reality from [Guizzardi, 2007](c) Splitting syntax and lexicon from the semantics of a language.

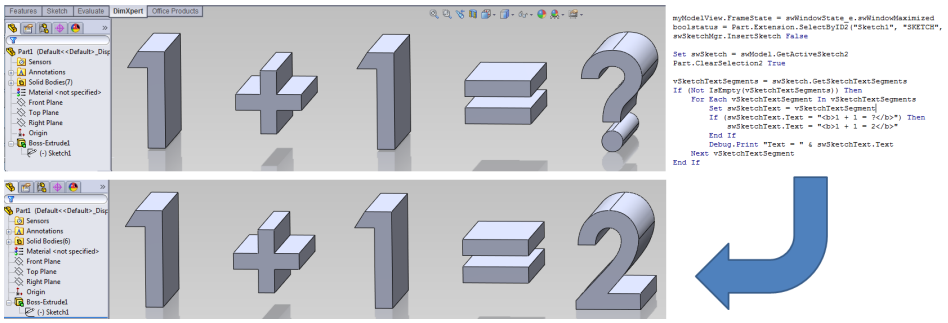


Figure 4.3: (top) A 3D geometry model can represent a mathematical problem, but the CAD software does not have the semantics to understand these symbols as a mathematical problem. (bottom) Software often allows for the addition of new semantics to the tool functionality, in this case by adding a macro algorithm to 'solve' the equation (right).

tion of the thing in reality: "do you see 'blue' the way I see 'blue'?" "What does 'function' mean?" This lack in a shared conceptualization – informality – is not often a problem, as the human brain will just work around or ignore this problem or fill in the blanks, or communicate with the other to converge to a shared conceptualization. However, when putting information in a model, in other words, representing the conceptualization formally as in figure 4.2a, this can become a problem. This lack of a shared conceptualization is often based on (a) the mismatch between the formal modeling language and the natural language of the human modeler (e.g. the modeler has no experience with the modeling tool), and (b) the difference in semantics between those describable in the formal language, and the semantics in the head of the modeler (e.g. using a 3D geometry modeling tool to describe a mathematical model, as in figure 4.3, is not convenient. As a less extreme example, think of how often a powerpoint presentation is made to describe some system architecture; it takes a lot of time to make the presentation, as powerpoint is not specialized in such representations). A modeling language thus has four degrees of freedom, which will be explained in the next sub sections: The semantics (§4.1.1), the concepts (§4.1.2), the syntax (subsection 4.1.3) and the symbols (§4.1.4).

> **Note**: An often used term in this context is the ontology, which seems to have a bit of a difficult definition. In [Guizzardi, 2007] we find the definition of, among others, relations between ontology, artifacts, languages and conceptualizations. As it was described by Aristotle, Ontology is the study of the generic traits of every mode of being; by Husserl two thousand years later as the formal aspects of objects irrespective of their particular nature. In AI and other more modern interpretation it shifts to "The representation of conceptualizations in a concrete artifact is a (foundational) ontology". Because of the complex definitions and various interpretations of the word ontology, the term is not used in this thesis, although it would be appropriate.

4.1.1 Semantics

Semantics is defined as the meaning of (a syntactically correct set of) symbols [Uschold, 2001]. In developing a language, we must therefore decide how much semantics to put into

the formalization, and how much we leave to human interpretation (see figure 4.2c). Picture a designer using a blank mindmap, visio diagram or a model such as the one in the top of figure 4.3 to formalize and structure his or her tacit ideas. The resulting model is an explicit representation that can be interpreted by other people because they share the semantics (meaning) of the used lexicon (words) and syntax (grammar) in the model. However, these representations are not explicit enough for a computer to read. The reason is that a software tool only has a formalized lexicon and semantics for the main purpose of the software, such as modeling 3D geometry. However, the benefit of software is that one can program semantics into it. As a simple example, a macro can be written for the 3D CAD modeler that can solve the particular problem of figure 4.3. In the state of the art chapter 3 we already saw that putting all information into a single model – the 'model of everything' – is unlikely to be possible, and indeed undesirable. Therefore an acceptable level of 'pre defined' semantics need to be chosen, which will give meaning to a syntactically correct set of symbols (in grammars), which represent concepts (or ontologies).

[Uschold, 2001] states a 'scale' of formalizing semantics upon which we can position the architecture modeling language(AML).

- *Implicit semantics* – Shared human consensus. The kind of semantics people learn over a lifetime, as it is expressed in natural language. The implicit consensus among stakeholders can be a problem, as they will not know if what the other thinks corresponds with what they think. Stakeholders therefore need to communicate to learn each other a shared conceptualization of the design information. The AML must be able to provide 'anchors' on which to base such discussions, for example in using a functional model in a discussion of how a system actually works, or in using a parametric model to discuss design trade-offs. In other words, the language must facilitate (implicit) communication between stakeholders to find answers to the who, why, what type of questions (see figure 4.4).
- *Informal (explicit) semantics* – Semantics notated in symbols in text, diagrams or other models, where the lexicon is in natural language. Such as exchanged in meetings and emails, and in the top of figure 4.3. Humans will be able to understand the semantics of the explicit information, software will not. These informal semantics are an important degree of freedom in modeling design information, as modeling all design information in a formal way will lead to the impossible 'Model of Everything'. In the AML, the informal semantics are symbolized by attributing names and types with words from natural language, such as `"AMFunction.name = 'make coffee'"` or `"AMRequirement.type = 'manufacturability'"`. At this level, information about architectural concerns and other important design information can be modeled, so it can be exchanged over larger communication distances, or stored for future reference.
- *Formal semantics (for humans)* – Semantics coded into software, used at runtime. Such as in digital product development tools and dedicated knowledge bases as in the example of figure 4.3. Such semantics need to be captured to be able to reuse design knowledge. We want to use the expressiveness of the natural language in an informal model when using the language for new ideas or idea exploration or human to human communication, but use the machine readability of a formal language when automating routine design tasks (The task is routine if we already know how to solve

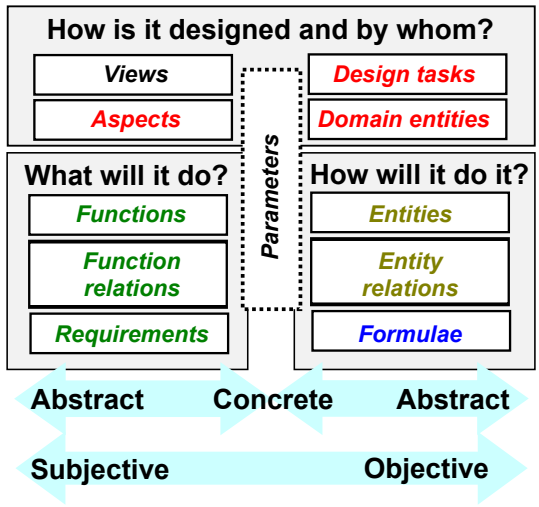


Figure 4.4: The language concepts projected on the Who Why What questions.

it). This can be useful if a routine task is, for example, calculation intensive, boring or repetitive. When cooperating with multiple stakeholders it can be useful to minimize the jargon, and merge the lexicon where possible, to define common ground and common understanding. The formal and informal semantics can be 'symbolized' in a single model, when the formal representation is provided with some 'lexical markers' that allow the software containing the semantics to recognize the formal part of the model (for example: an AMEntity has a 'circle' symbol as its type attribute to mark it as a circle for a circle knowledge base). Also, the software needs to know the syntax of the language to discern the relations between symbols. This syntax is discussed in subsection 4.1.3. Defining formal semantics allows for the plugging in of knowledge bases into the AM framework, while keeping the models largely informal. Some semantics are provided that apply to all information in the architecture models: the diagram editor, saving, loading and manipulating xml files to store the architecture models, search algorithms, etcetera, while other knowledge bases only cover a certain design automation purpose, such as in the case studies 5.

- *Formal semantics (for machines)* – Semantics processed and used at runtime. Such as in the KIEF framework described in [Yoshioka et al., 2004], where a knowledge engine can infer system behaviors from meta models using a qualitative process abduction system [Ishii et al., 1993]. Or in Watson, the jeopardy-playing (and winning) supercomputer designed by IBM [Guizzo, 2011]. In software generally, an executable file (.exe) is a formal model executable by the operating system of a computer, and an operating system is a formal model executable by the hardware of the computer. The AM framework has not been designed to define such executable models, as this implies specifying the semantics of all possible semantics – although it is possible to use an architecture model as an intermediate form for an executable simulation with dedicated knowledge bases (see paper path case study 5.6).

Table 4.1: Concept taxonomy definition in the Architecture Modeling Language

Level	Example	Explanation
AML class	AMFunction	language concept type
class	Connect	specialized class of concept
'type' attribute	Glue Together	'type' attribute value in AMFunction object
instance	Glue A to B	'name' attribute value in AMFunction object

4.1.2 CONCEPTS

Concepts are the units of information that are represented in a model through symbols. The concepts are organized in a lexicon (vocabulary) and a taxonomy [Pidcock, 2003]. An organized lexicon is the set of concepts (words) that carry semantic meaning in a language. The taxonomy determines the 'type-of' relations between the concepts in the lexicon.

How big a lexicon must be to describe all design information in a multi disciplinary design process is impossible to say, but it will probably be a significant portion of the natural language of the stakeholders in question. Defining the semantics of all these words in a formal language will be therefore a huge task. This is why, in the AML, we have chosen not to prescribe the concept words directly. However, without defining any concepts the language cannot be called formal, and it would be hard to specify a syntax, as all concepts could be related to all others following the same syntax.

The answer was found in specifying the first level of taxonomy as the concept types of the language, see concepts definition §4.1.5. This means that any word used in the AM must be the value of the name attribute of an instance of one of the 14 concept types in the language (`AMFunction.name = 'rotate'`). These 14 concept types were formulated as their need became apparent in the progress of the ACCGSMS project, and, while the concepts in the language were not chosen as a result of an extensive meta analysis of literature, there is a large overlap in the concept types in literature and the AML, as can be seen in the state of the art chapter 3. The small set of concept types allows us to specify the syntax for the relations between them. This, in effect, becomes the semantics of the AML, as the AM framework 'knows' what to do with the concepts and their interrelations.

Two other mechanisms are in place to further specialize the taxonomy of specific AM's, see table 4.1. First, as the concepts are defined in software classes in the AM framework, it remains a possibility to specialize the basic concept classes into other types. One can think of specializing the AMFunction class into the functional basis of [Hirtz et al., 2002], which has a four tier taxonomy (for example 'Filter' typeof 'Extract' typeof 'Separate' typeof 'Branch'). Such a specialization can be used to perform more semantically specific design tasks, like using it in a Theory of Inventive Problem Solving (TRIZ) algorithm for computational synthesis models as demonstrated in [Chulvi and Vidal, 2006]. Secondly, any concept in the AM language has a 'AMBasicObject.type' attribute that can be used to classify information. While this attribute is just a string of symbols, it can be used to give a namespace to design information in the AMs. In the AGCCSMS this attribute was also used to namespace the various domain specific model references, to make them recognizable to knowledge bases

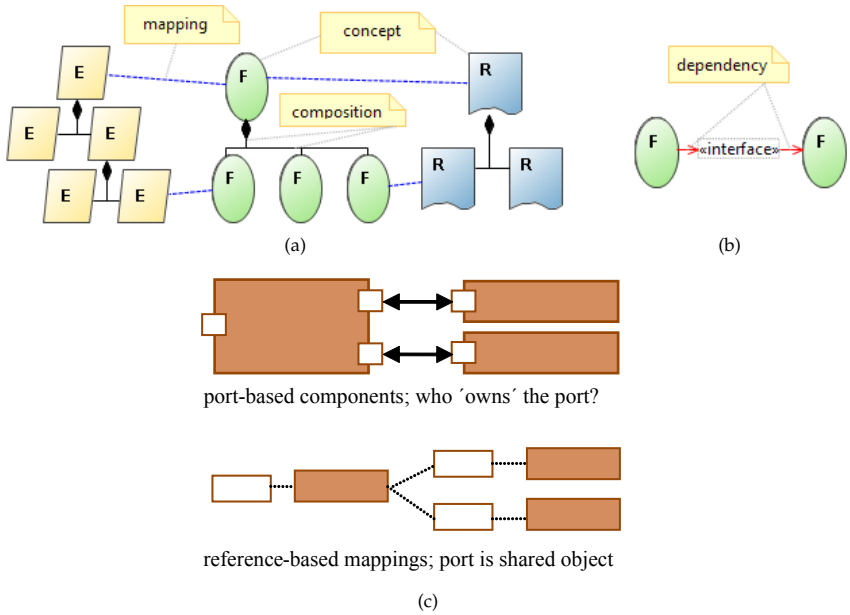


Figure 4.5: Syntax types. (a) Decompositions to describe sets, and mappings to relate between concept types. (b) Dependency or flow between concepts, define an interface. (c) Externalize port to put emphasis on ownership of interface.

which know the semantics of the namespace. As an example, the word 'SolidWorksKB.Part' was used as a 'type' value to mark solidworks parts references in an AM. The knowledge base can then open solidworks and review those parts.

4.1.3 Syntax

The syntax of the AML is the allowable ways the concepts can be ordered into sets and graphs: What if we see the information used and generated in a design process as a set of interconnecting objects? Where information flows from one object to another, where objects have different types, and objects can have multiple levels of granularity. These objects can be systems, or system components, or models or parts of models, or design processes or individual design tasks, or organizations or individual people, etcetera. This section argues that, at an abstract level, there are only three types of relations that we need to make a model of the objects and the information flowing between the objects: the set, the directed graph, and the undirected graph. Or, in AML: the (de-)composition, the dependency and the mapping. Another type of relation, where some objects create or type other objects, the class/object and generalization / specialization, is not to be part of the AML at the moment,

but could be added in the future. Instead, type-of relations will be captured in namespacing of a 'type' attribute as discussed in §4.1.2.

Given this abstraction, we want to define the design process information in one central model, the architecture model. We do not want to insert all the semantic information content or transformation mechanisms though, as this would require a modeling language that can make a model of everything in the world, and such a modeling language is not realistic. Specifying *that* information flows instead of *what* information flows is a valuable representation. With such a model, tooling can be built to ensure consistent information transfer, integrate information flow to and from objects in various design disciplines, and provide instructions for knowledge bases to generate new objects or alter existing ones. The concepts definition 4.1.5 and meta model §4.3 and reference model sections §4.2 discuss this in more detail. As previously mentioned, there are three types of relation – the composition, the mapping and the dependency:

Composition – The composition relation defines sets (figure 4.5(a)). There are four types of sets in the AML: A functional decomposition and a structural decomposition describe the system. From a design process point of view, we must decompose a large multi disciplinary, multi stakeholder process into processes that can be handled by individuals, the design tasks. The design tasks relate to a certain aspect of the design, and the whole design should be a composition of its various aspects. Finally, the aspects have been modeled in external digital product development tools. These models can be abstracted and represented into compositions of domain entities.

Mapping – The mapping relation defines undirected graphs(figure 4.5(a)). There are many types of undirected graphs in the architecture language. They are used to define a relation between two concepts of different types. An example is the function–requirement–entity mapping, which states that a function is performed by an entity within the limits posed by the requirement.

Dependency – The dependency relation defines directed graphs (figure 4.5(b)). There are three types of directed graphs in the AML: The directed graphs between functions define behavior – processes in a sequence, or causalities between functions. The directed graphs between entities define the material, information or energy flow between structural objects (parts, software objects). The directed graphs between design tasks define the steps in a design process, and the hand-over between them. An important aspect of the dependency relations is that they are not 'encapsulated' by ports inside a concept. instead, the dependency relations become concept types of their own (see figure 4.5(c)). This is done because the dependencies signify the interfaces between concepts. The ownership of the interface is very important, as this interface often is where design information is transferred between different design disciplines, different stakeholders (engineers) within a discipline (department), or levels of granularity (detail model from meta model).

The syntax of the AML enables navigation of the relations in the architecture model by knowledge bases. This enables knowledge bases to search for information, or to create, check, and/or change (properties of) connecting concepts.

4.1.4 SYMBOLS

Another language specification option is the usage of symbols. What representation is the most convenient for the communication of information in that specific case? We separate the concepts from the symbols that represent them. This way, we can define the most appropriate form of representation for the information at a given time for a given user.

In most methods from the literature review, information is expressed in matrices. Often, a matrix is used to put one or two concept types on the axes and make a model of a relational aspect between the concepts in the cells that cross the rows and columns. For example, a 2D matrix form can be used to evaluate a single relation type between two concept types, such as in the design structure matrix or the house of quality (see §3.3). However, to be able to overview a set of concepts with various types of relations, a multi dimensional matrix would be needed, which is difficult to represent. Furthermore, the matrix would display empty cells where no relation is available, which means the representation can become very sparse. As the AML has 30 different relation/concept combinations, representing the architecture models in a 30 dimensional matrix would be inconvenient.

For the representation and construction by humans, we have chosen a diagram format. The entity–relationship modeling method has been around since at least 1976 [Pin-shan, 1976] and is a popular form of representation, for example also chosen in UML and SysML diagrams. The entity relation diagram has a number of important advantages:

- Concept types and relation types can be color, size and shape coded, allowing for a greater discernibility of the information than flat text.
- Only the entities and relations in the model are represented, no empty cells as in matrix representation. the diagram can be layed out in such a way that the readability is improved, for example to put composition relations between concepts in a hierarchical tree shape, or mappings from concepts to a shared parameter in a star shape.
- Filtering of pieces of the diagram for various stakeholders can improve readability.

However, the diagram symbols will often take up more space than other representations. This can make large diagrams unreadable. By specifying sub diagrams, for example on various levels of hierarchy or granularity, this can be mitigated.

Pen and Paper. A pen and paper method is probably the most efficient way to transfer knowledge, as there are no pre defined symbols. Use figures, text, diagrams, anything goes. Such a representation is especially useful in explaining difficult information, such as system architecture, to someone (or to reach consensus) as is the topic of the thesis of [Borches Juzgado, 2010]. In this A3 method, we see how convenient freedom of expression is in communication between stakeholders. The AML can be represented in such a pen and paper method, and it would be an interesting idea to combine the Borches method with the AML. However, for this thesis, this will be out of scope as it does not help consistency, integration, and reuse of design information directly.

In order to make it possible for computers to use the AML, a representation needs to be chosen for data storage of AMs. The Extensible Markup Language (xml [W3C, 2012b]) is

used as a data language for the AML. The xml files are readable by humans and software alike. For future development of the AM framework, a representation in a database format would help in storage, multi-user access and versioning of the AMs, however this remains out of scope of this thesis.

4.1.5 Concepts Definition

From the literature research of chapter 3 a set of often used modeling concepts was extracted. These concepts are needed to capture the design process information. The language consists of generic attributes, concept type definitions and relations definitions. Together, these form the modeling concept types of the Architecture Modeling Language. The modeling concept types in the language are defined in the language appendix E to make it easier to use as a reference. In this appendix we see that the language has a small set of concept types and a predefined number of relation types between these concept types. These are summarized in table 4.2.

4.2 Architecture Model as a Reference Model

In multi disciplinary design processes, designers do not make systems, they make models that are used to make the system. In essence, these models contain design information to help the modeler understand something about a system. In the architecture modeling framework we want to have a better understanding of which models there currently are, why they are used, what their results or assumptions are, who made them, when they were or are to be made, by what means the models were made, etcetera. In short, we want to have a reference model for the information in a multi disciplinary design process. Wikipedia, by lack of a better reference, defines some aspects of a reference model, which we will paraphrase as:

- *Abstract*: "The things described by a reference model are abstract representation of things." Details that need not be shared between stakeholders do not have to be part of the reference model. The model does not contain the semantics of the thing in reality.
- *Entities and Relationships*: "A reference model contains both entities and relationships. A list of entities, by itself, is not sufficient to describe a reference model." The types of entities and relationships are the concept types of the architecture modeling language.
- *Within an environment*: "A reference model does not attempt to describe 'all things'. A reference model is used to clarify 'things within an environment' or a problem space. To be useful, a reference model should include a clear description of the problem that it solves, and the concerns of the stakeholders who need to see the problem get solved."
- *Technology Agnostic*: A reference model does not contain the semantics of a particular technology, and it does not prescribe how a problem must be transformed to a solution. This is the role of the meta model.

To further specify the reference model role of the architecture model, we follow the design definition 2.3, where we define three tiers of reference (see figure 4.6):

Table 4.2: A summary of the concept types and relations in the Architecture Modeling Language

	BO	AM	V	P	F	R	E	Fx	A	DE	DT	FR	ER	DR	SF
BO		inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	
BO		con	con	map											
AM			inh												
V			com												
P				dep											
F					com	map	map		map			dep			map
R						com			map			map			
E							com		map				dep		
Fx		cross references											map		
A									com	map	map				
DE										com	map			map	
DT														dep	

Design Information Types:

BO	BasicObject
AM	ArchitectureModel
V	View
P	Parameter
F	Function
R	Requirement
E	Entity
Fx	Formula
A	Aspect
DE	DomainEntity
DT	DesignTask

Flow Interfaces:

FR	FunctionRelation
DR	DesignTaskRelation
ER	EntityRelation

Relation Types:

com	(de-)composition
con	containment (parent model)
inh	inheritance
dep	dependency (flow,sequence)
map	mapping (reference)
SF	StartFunction

* note the parameter can be mapped to basic object, therefore parameters can be mapped to all concept types.

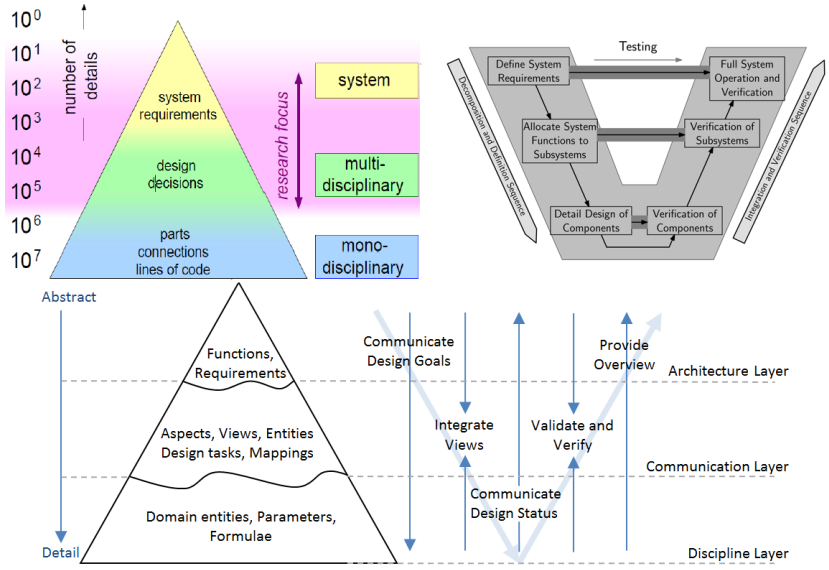


Figure 4.6: The architecture model as a reference model. Combining the Muller pyramid [Muller, 2011b] with the Vee model [Forsberg et al., 2005]

1. *Architecture Layer*: "Design is intended to accomplish goals, in a particular environment, satisfying a set of requirements" In the architecture layer we define how a system is supposed to work – the functions are performed by entities within requirements.

2. *Communication Layer*: "to create a design, in an environment (where the designer operates)" The communication layer formulates the views of the stakeholders. Within the views, information from the architectural concerns are connected to stakeholder resources such as models. The views are partial representations of a shared, integrated datamodel.

3. *Discipline Layer*: "Specification of an object, manifested by an agent, using a set of primitive components subject to constraints" In the domain layer we find the parameterized modeling resources – design tasks, models and tools – of the various stakeholders. These resources can originate from multiple design disciplines.

In the following subsections we will see how these layers can help in ensuring consistency, and facilitating integration. Enabling reuse is a separate section (§4.3), as this will be achieved by using the AM as a meta model. This meta model is something different from the reference model; the meta model is the central model from which external (aspect) models are derived, or which specifies the metrics for evaluating external models. Note, the layers described below are not specified as constructs of the AM, they are just explanations or guidelines on how to use the AM.

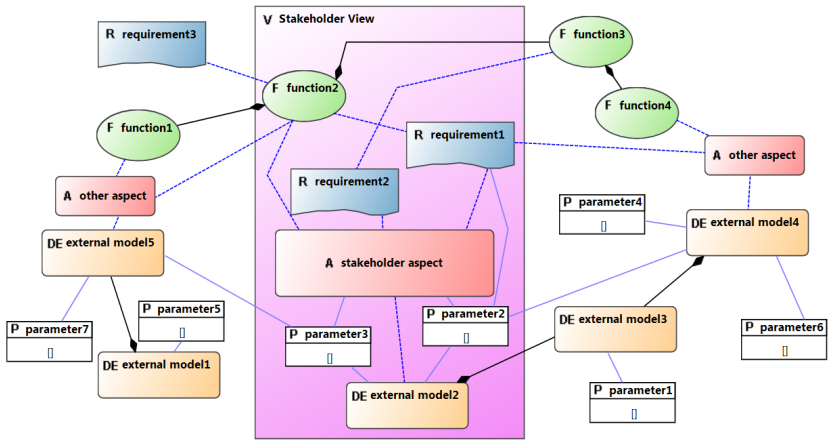


Figure 4.7: A view as a subset of information from a shared data model. In the view the information is presented for the stakeholder's concern.

4.2.1 Consistency in Reference models

The architecture layer helps consistency of a design process in a number of ways. The architecture layer is used to model the goals of the design process: design a system that performs as the customer wants and doing it in a way that satisfies the other stakeholders (such as making a profit, or comply with government regulations). The goals of the design process will be captured in requirements (design for x, system performance, technology constraints), while a functional model defines the basic behavior of the system. This architecture part of the AM thus provides overview to the stakeholders of the common goals and system architecture, and is needed to create a common understanding of the system under development. As the design process progresses, the work of the individual stakeholders can be continually verified against the requirements in this layer, helping to keep the design specification consistent.

The stakeholders can use the architecture layer as a metric for their work, either to find the prerequisites for their discipline specific design tasks (input), or to measure their design task results against the global requirements, thus verifying their work (output). In order to do that, the architecture layer information must be decomposed into views for the various stakeholders, and mapped onto abstracted discipline specific information. This is done in the communication layer. In these views, the aspect the stakeholder is interested in is modeled, see figure 4.7. Those aspects can pertain to anything related to the development of the system. For example, 'what is the cost of the component?', or 'will this structure break under the maximum required load case?', or 'can we produce this part with injection molding?' To answer these type of questions, they need to reference information from the architecture layer and the discipline layer.

As the information from the architecture layer is shared by, and common to, all stakeholders, and the information from the discipline layer – needed to model the aspect – can be made by a number of different stakeholders, it is clear that the communication layer will be the layer where important conflicts come to light and decisions need to be made. The communication layer thus forms an important consistency checking mechanism. If the information in your view is traceable to someone else, you can find this person and start a discussion. Think of defining ownership of shared information, or providing overview in trade off discussions. The design decision can then be modeled in the views of both stakeholders.

The discipline layer contains the references to external information. The architecture models on their own cannot say anything about their own consistency, as the architecture modeling language does not contain the semantics (design knowledge) to solve any particular problem. Therefore other models need to be employed. Think of a model that calculates the cost of a component, a finite element model that calculates dynamic stress response to a load case, or a injection mold geometry generator. Those external models are abstracted into domain entities and referenced in the AM. The results of such analyses can be written to parameters mapped to these domain entities, and connected to aspects and requirements to define which metric the value has. As such the AM can be used as a reference model onto which to verify the design. Modeling the domain entities, parameters and values and performing the analyses can be done by hand, however, by employing the AM as a meta model, some such information transfer can be automated (see §4.3 and paper path case study §5.6).

4.2.2 Integration in Referencemodels

What do stakeholders want to know from each other and from the system under design or analysis? This question is essential to the AM as a reference model. The AM as a reference model must integrate the various information from and to the various stakeholders. The architecture layer models the information all stakeholders have in common – the basic system functionality and the requirements that define the goals of the global design process and the metrics for the various aspects of the design. The architecture layer can also contain a structural decomposition of the system, defined with entities. However, it is important to recognize that in multi disciplinary design processes, there is no such thing as *the* system, or *the* system decomposition. The decomposition will grow in granularity throughout the design process, and the various design disciplines and stakeholders will have various ideas of how the system will be decomposed. For example, a software model of a system has need of a different decomposition of a system in comparison to an manufacturing assembly decomposition. Furthermore, various stakeholders will have various ideas of what is essential and what is detail, what is a constraint to a problem (input) and what is a solution to a problem (output), etcetera.

In order to mitigate these different views on what is important, the architecture model has a communication layer, in which the stakeholders can construct a view in which to define what they think is important for their design concerns. This stakeholder view can formalize

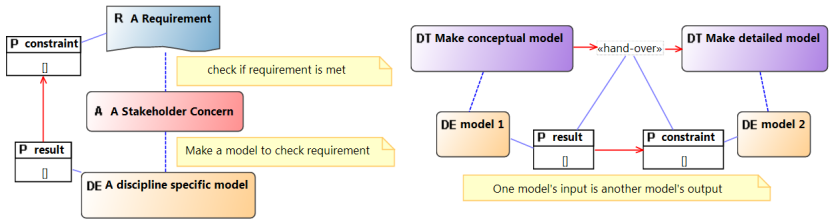


Figure 4.8: Vertical integration expresses accountability. Horizontal integration of information expresses causality.

the concerns about the various design aspects. We can deduce that for a certain size of an organization formalization must be enacted to be able to exchange information effectively, as the communication distance between people becomes too big to communicate face to face. The views can help in this issue, as they can be used to make a model of information in a shared and integrated data model. Stakeholders can reference each other's information, thus enabling sharing and integration. This integration happens 'vertically' between architectural concerns and discipline specific information, and 'horizontally' between cross disciplinary information. In figure 4.8 this is expressed. In vertical integration, the 'results' or metrics of a model should comply with the requirements of the architecture layer for the model to be valid, or in reverse, the result of a model should provide verification of the requirement, if the requirement is not met, something must be done. This allows for the modeling of cross cutting aspects, aspects that must be developed or measured using information form various discipline specific resources. One can also make a model of the design flow in terms of shared parameters, where the interfaces between stakeholder aspects can be made explicit.

Reference models at the disciplinary layer are used to provide a 'content map' to connected models and their properties (see figure 4.10). When a property of a model is important to someone besides the stakeholder who made it, it could be useful to abstract it into the AM. This allows for the horizontal integration of models: A model's results can be the end result of one design task, but could be the input constraints of another (see figure 4.8), it is a matter of perspective: the result and constraint should be integrated or merged into one parameter, with the causality of the design task to describe ownership and direction of the value. This is the task of integration at the disciplinary level. This type of integration allows for the modeling of workflows, which can be automated by using the AM as a meta model 4.3.

4.3 *Architecture Model as a Meta Model*

An important aspect of the architecture model is that it needs to help in constructing the design specification. This means the architecture model must take the role of a 'meta model'. In [Pidcock, 2003] we find a number of related definitions that are also supported by sources

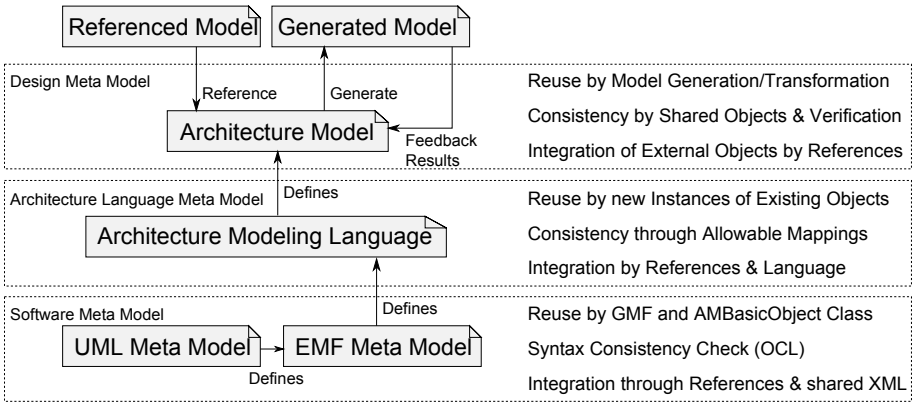


Figure 4.9: A meta model models other models.

as the Object Management Group [OMG, 2011] and other software and system engineering oriented standardization efforts. Of these the definition of a meta model is as follows: "A meta-model is an explicit model of the constructs and rules needed to build specific models within a domain of interest." Examples of meta models are models in Unified and Systems Modeling Language (UML, SysML), which in turn have a Meta-Object Facility (MOF) meta model. A more engineering and design related definition we can find in [Takeda et al., 1990]: "The central description of the design object is regarded as metamodel in GDT (general design theory) because it is used during the design process as a central model from which aspect models are derived." The difference between these definitions is a matter of language transformation. The first definition uses meta models to define the way a model is constructed. As such, someone can make a UML model to define the UML meta model. The second definition uses meta models to describe a (design) object and group its attributes so that it can be analyzed or can be extended by synthesizing new objects and attributes. In this thesis we will need both meta models to come up with an architecture modeling language. in figure 4.9 we see the different layers of the architecture modeling framework meta models.

The reason for including the 'software' meta model in this thesis is that a number of functions relating to reuse, integration and consistency are diverted to the software model, as these functions are not specific to engineering applications. As can be seen in figure 4.9, we have an architecture model as meta model, an architecture language meta model and the software meta model. These meta-models are built on top of each other, thus adding functionality – from a generic software model to a specific architecture modeling framework. This means the starting point of the software meta model is an important 'design choice' for the framework, where the aim is to reuse as much functionality as possible from freely available software.

In this thesis, the software meta model is the Eclipse Modeling Framework and its ECore models [EMF, 2011]. This framework has been chosen because of the availability

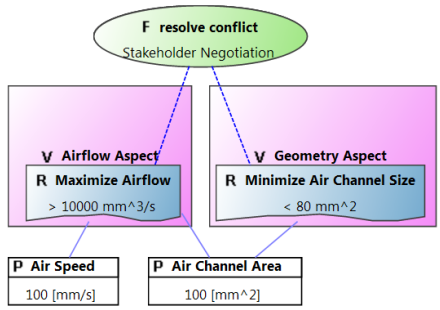


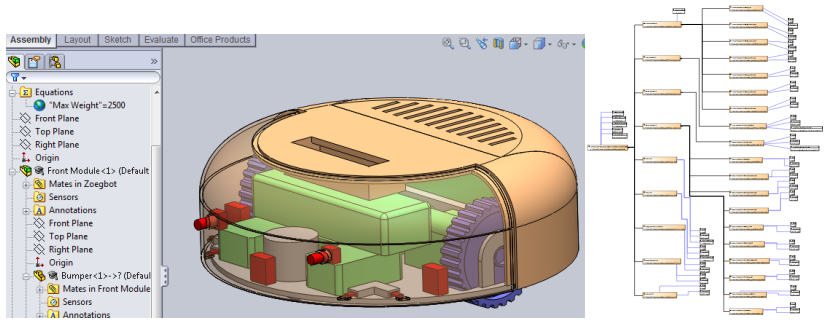
Figure 4.10: A shared mapping can result in a conflicting design decision, but can also localize the involved stakeholders, so they can engage in a trade-off negotiation.

of the Eclipse Graphical Modeling Framework [GMF, 2011], which uses the EMF and ECore as meta models to generate a graphical diagram editor (see §4.3.3). Another option could have been to use a UML meta model [OMG, 2011] to build a Windows Presentation Foundation (WPF) diagram editor [WPF, 2011], which works along the same line. The important thing to note is that the software development environment choice is not constraining or predefining the resulting architecture modeling framework. The following subsections explain the application of consistency, integration and reuse per meta-level.

4.3.1 Consistency in Metamodels

A consistency guard at the architecture model (AM) level is the method of mapping and reusing concepts in multiple diagrams. The language makes sure that a concept is uniquely defined and is then referenced to the various views where it is needed (through hyperlinking the graphical symbol of the concept). This ensures that changes in that concept will promulgate to all views where it is referenced, without the need for manually changing and exchanging documentation. Think for example of a value of an important design decision. If someone would change the value as a result of an analysis on a model, this value could have repercussions on other design decisions. When an end user would open a view to review design changes, the value of the concept is changed in that view also. When someone would detect a design conflict as a result of such a change, the stakeholders who have 'ownership' can find each other, through the shared information, to engage in a trade-off negotiation. As shown in the example of figure 4.10 the two stakeholders can decide to lower the surface area of the air channel, or they can increase the air speed.

The interpretation of what is consistent and what is not at the AM level is based on the semantics of what the AM is describing. As such it is impossible to make a generic consistency checking tool for all AM's, as this would result in a model of everything (also see the section reuse §2.7.3). However. because of the AML, case-specific consistency checking or verification tools can be programmed at this level to automatically check the model if some pre-defined information is available (see workshop findings in appendix C).



(a) (b)

Figure 4.11: In (a) we see a CAD model of a robotic vacuum cleaner. In (b) this model is abstracted in an AM diagram.

When we consider automation of model generation, we can use the architecture model as a meta model to ensure consistency between generated models. As we capture shared input and output between models, we can make sure that the model generators use the most actual data from the design process. This was demonstrated in the paper path case study 5.6.

At the architecture modeling language (AML) level, we guard the consistency of the architecture model by specifying allowable relations. These relations are attributed to the various concepts according to the language definition in §4.1.5. This means the end user cannot make models that do not comply with the syntax definition of the language, which keeps the architecture models syntactically consistent. For example, Functions can have parents and children, but Parameters cannot.

At the software level, we can guard the consistency of the architecture model by specifying rules for model construction (the syntax). For this purpose, the Object Constraint Language (OCL) has been used [OMG, 2011]. These rules comprise the semantics that check the syntax of the architecture model. Using these rules we prevent cyclical decompositions (parent has child has parent), warn against certain attributes being undefined (concept has no name, or type, Parameters have no unit specified), or, warn against 'orphaned' concepts – concepts that have no relations to other concepts and are therefore likely legacy concepts to be removed from the model.

4.3.2 Integration in Metamodels

The architecture models need to integrate information of various external sources in the design process. To do this, abstractions of these sources are modeled in various concepts (The domain entity for model abstractions, the design task for abstracting design tasks, the URI and documentation attributes in all concepts for localization of external data.) This abstraction is needed to facilitate communication with external tooling. In 4.11 we see such an abstraction. As we do not need all information from the 3D CAD model, we can cap-

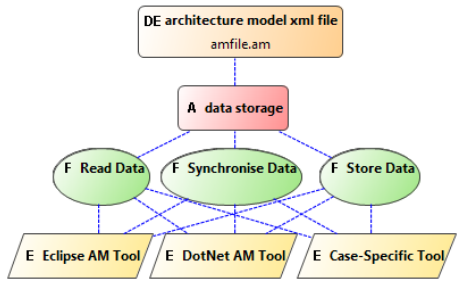


Figure 4.12: Various tools can implement the Architecture Modeling Language to exchange information through the shared architecture model.

ture the meta information (in terms of the design meta model definition) needed for cross-disciplinary aspects of the robot in the architecture model standard. Note: the AM in figure 4.11 was used in the workshop discussed in appendix C to provide context for the participants.

In the architecture modeling language meta model, we use the EMF property of hyperlinking to integrate information from different views into a single unique data model. This is demonstrated in figure 4.10, where the surface area of the air channel is referenced by two views. The parameter storing the information is defined uniquely in the AM and represented in the two views by a diagram symbol. This makes sure that stakeholders will not independently modify their 'copy' of that piece of information, thereby introducing an inconsistency in the model.

However, the most important 'integrating' aspect of the AML in terms of meta modeling is the definition of the language itself. The language uses concepts – such as function and design task – that can be used to define system properties and design process properties at a level of abstraction (meta-level) that can be interpreted by users in many design domains. As such, the AML provides a platform for integrating information from the various stakeholders involved with the design process. This assumption is worked out in the case study chapter 5.

In the software meta model, we want to facilitate integration by (1) providing an exchange format for information, on which design support tooling can be built and (2) a meta model format to specify how models can be generated. The meta models of EMF and UML already have tooling that provide these functions. As an exchange format, the emf models, and thus the architecture models are captured in Extensible Markup Language (xml) files that are defined at the software meta-level [W3C, 2012b], or in other words, the AM's data format is based on the xml meta model. XML is a way of structuring data that is widely used in the software world, and is thus supported by virtually all software development tools, which have built-in parsers to read-write the xml files. As a result, we can build our architecture modeling language on different platforms and for different applications, while using the same data from the xml files (see figure 4.12).

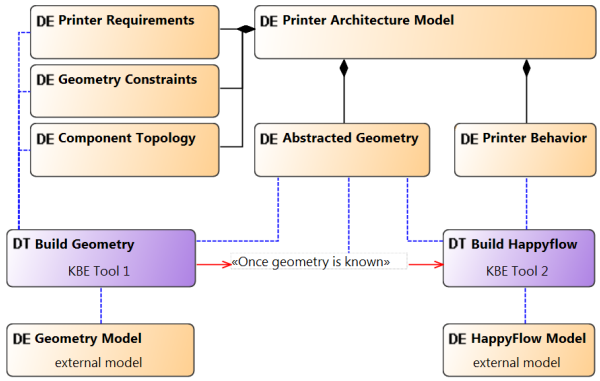


Figure 4.13: Paper path case meta model flow. The lexicon and syntax of design information is captured in architecture models and external models, while the semantics are captured in knowledge based tools that perform design tasks.

The EMF also allows us to cross-reference information across multiple xml files. As such, we can define information in one model and reference it elsewhere, while to the user it looks like the information is stored in a single location. This can integrate multiple AM's, allowing for the specialization of AM's for different applications or for modularization if a single AM becomes too big.

4.3.3 Reuse in Metamodels

Reuse in meta models works by abstracting generic properties to a higher level, so one can build generic modeling functionality that can be used to build a range of specific models. This abstraction was applied in all levels of the architecture modeling framework. The software level reuse enables us to implement a lot of functions needed to work with the architecture models, and is therefore in scope of this thesis.

The 'highest' level of meta model for reuse comes with the architecture models themselves. We can use the AM as a meta model in terms of the design meta model described in the general design theory of [Takeda et al., 1990]. To restate this definition: "The central description of the design object is regarded as metamodel in general design theory because it is used during the design process as a central model from which aspect models are derived."

In all case studies and experiments (see chapter 5), the AM was used as an input for generating other models, or the output of other models was used to alter the AM. This becomes possible through the separation of semantics syntax and lexicon as stated in §4.1. The AM will contain the syntax and lexicon of certain design information, whereas the semantics of the design information are contained in a software tool that can read/write the AM and read/write models of an external type. This allows for the reuse of 'design knowledge' to

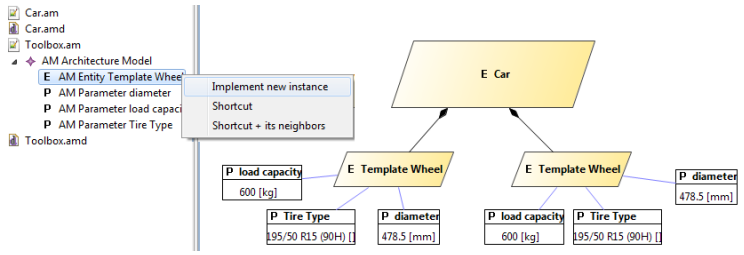
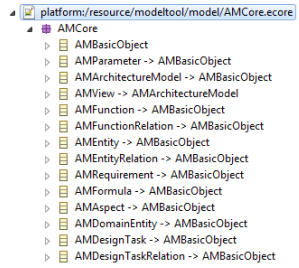


Figure 4.14: The architecture modeling language as a meta model. Reuse existing definitions to quickly construct new models

(1) synthesise a range of different embodiments (configurations) of a design specification. or (2) perform analysis on certain aspects of a design specification. The paper path case study demonstrates this (see §5.6 and figure 4.13. The lexicon and syntax of certain requirements and the basic system topology are defined in an AM. The knowledge of making a geometric model is stored in a knowledge-based engineering (KBE) tool, which generates the geometry, which is abstracted, and written back to the AM. After that, another KBE tool will read the lexicon and syntax of the geometry and a system behavior of the printer from the AM to generate an analysis model (a happyflow model to see how paper flows through the machine).

At the architecture modeling language level, the Eclipse design environment allows for the development of software packages that can be 'plugged in' to other packages. As such, we can add non-generic functionality to the framework as layers on top of the generic functionality. Examples of non-generic functionality are model generators, model verification, and new instantiations of existing concepts. New instantiation of existing objects means we can make a copy of a part of an existing AM, give it new identifiers and user and date information, and implement that into a new AM. As such, the lexicon and syntax of existing design descriptions can be reused to make new design descriptions. This means we can use an AM as a meta model for AM's. As an example of a practical design application, think of defining a toolbox or templates AM, which stores often-used definitions, that can be copied into new models (see figure 4.14,or the Baggage Handler case study 5.4).

As a software meta model itself, the EMF model was used to make a model of the architecture modeling language and then used to actually generate the Architecture Modeling Tool itself by applying the Graphical Modeling Framework [GMF, 2011]. This form of reuse enabled us to alter the definition of the architecture modeling language when new concepts or mappings were needed, and regenerate the AM tool, so we could test it with our industrial partners. This saved time, as we did not have to manually build that part of the tool ourselves, and also kept the tool and the AML consistent. Because of the basic properties of the EMF meta model, we did not have to alter anything in the functionality at that meta level – think of serializing the data model, visualising connectors between concepts, diagram layout options, save-copy-delete methods, model navigation tools, etcetera – we can just reuse it.



(a)

```
Search('searchpattern', 'resultslist', 'alreadycheckedlist')
  am I in the 'alreadycheckedlist'?
    if so, skip me and go back.

  do i comply with the 'searchpattern'?
    if so, put me in the 'resultslist'.

  add me to the 'alreadycheckedlist'.

  what is my class?            //reflection
    what attributes do i have?  //reflection
      is this attribute a list of references to other objects?
        if so, access these referenced objects to search further.
  go back.
```

(b)

Figure 4.15: In (a) we see how the AMBasicObject class is reused in other class definitions. (b) shows a search algorithm based on reflection in the AMBasicClass that can be reused by the other classes.

Another important aspect of reuse in the AM framework is the definition of the AMBasicObject Class, used in the Eclipse implementation as well as in the c# implementation. This AMBasicObject class is an abstract class from which the other concept classes (AMFunction, AMEntity, AMRequirement) are derived. What this means is that we can reuse functionality on the AMBasicClass level and apply it to any class derived from it. This is done mainly with the implementation of reflection (introduced by [Smith, 1982], but now implemented in many popular programming languages), where a software object can reflect upon itself to, for example, see what kind of attributes it has. This is immensely useful to implement functionality that has to work across concept types, such as search algorithms or xml serialization, as we only have to build those methods once, at the AMBasicObject level, and do not have to think about it in other class definitions (see figure 4.15). These methods of reuse enable us to integrate and manipulate information from multiple external sources (see case studies 5), and helps in constructing and navigating architecture models (§4.4).

4.4 *Architecture Modeling Tooling*

After the Architecture Modeling Language (AML), and Architecture Models (AM), the third component of the Architecture Modeling Framework(AMF) is the Architecture Modeling Tooling (AMT). As we saw in the previous sections, the AMs can be used for numerous tasks in the multi disciplinary design process. However, the AM must be made with something. Although the AML allows for use as a pen and paper method, the consistency, integration and reuse goals will only be achievable when employing a software-based editor. As the symbols subsection §4.1.4 of the language section explained, the AML is expressed in diagrams for human readability and xml for software readability. As such, this section is divided in the tool for human usage, and tools for automation (knowledge bases). This is represented in figure 4.16: Human users will make diagrams with the diagram editor, while knowledge bases will work directly with the xml data file. The serialization and representation of the AM are both basic functionality of the AMF.

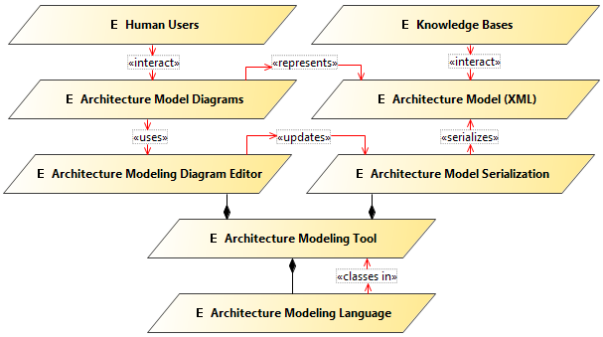


Figure 4.16: Human users use the diagram editor to manipulate architecture models, knowledge bases can manipulate the xml file of the architecture model directly.

4.4.1 The Architecture Modeling Tool

During the ACCGSMS project, an editor was developed to enable human users to make AMs, the architecture modeling tool, or AMTool. In appendix F a set of screen shots can be found of this tool. The architecture language is implemented in the Eclipse Graphical Modeling Framework [GMF, 2011]. This open source project enables the definition of Domain Specific Languages, and implements this language in a graphical editor. The editor allows for the definition of the diagrams that are shown throughout this thesis, in a way that is very similar to other tools such as Visio. The benefit is that the language definition constrains the syntax of the models, so that only allowable relations can be made, and consistency can be assured by containing all views and their content in an interrelated model, the AM. When one concept is changed, it changes in all views, thus allowing evaluation and updating of information throughout the design process and in all connected aspects.

This makes it possible to make the views stakeholder-centered: Instead of pre defining views for functions, requirement tracking, domain models, etcetera, and then let stakeholders find their way around them, we define views for the stakeholders' interests, and let them represent the functions, requirements, etc. from the flat model that they need. The hierarchies are still there, and a specific view for functional decomposition or requirement decomposition is still convenient, but these views have no special status per definition.

The AMTool should be developed in the context of multiple users working on a shared model, through some client/server mechanism. However, the ACCGSMS project focuses more on the academics, the modeling language and capabilities, than on developing a professional tool. Therefore the tool 'as it should be' is discussed in the outlook and recommendation chapter 8. The tool does have the same modeling capabilities as an eventual professional tool though, and multi user modeling was tested: In a workshop (appendix C) by using a hotseat – two or more users using the same computer to make a model of a diagram. And in the case studies by locally extending AMs and exchanging them over email between the project partners.

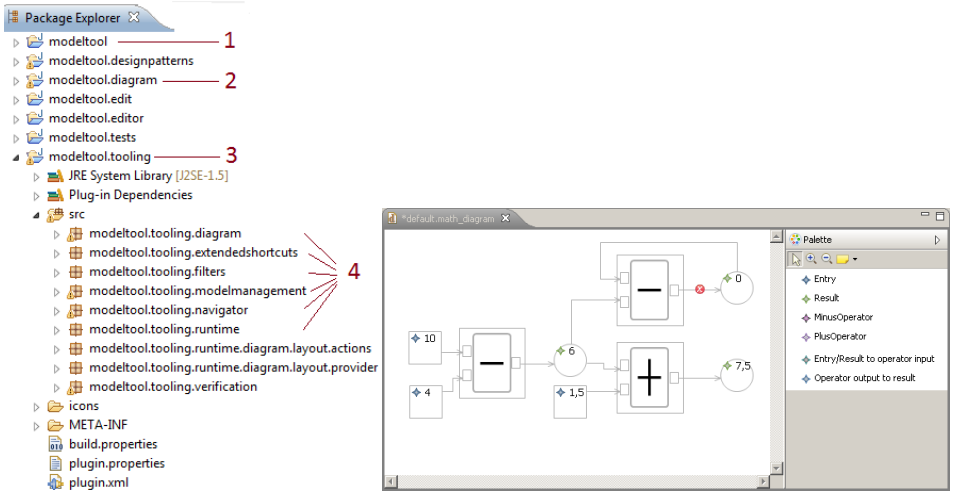
4.4.2 Knowledge Bases

In the AMF, knowledge bases are applied whenever AMs need to be more than human readable. Knowledge bases are plugins or tools that contain the semantics to solve a certain problem. In all case studies, there was a need for automatic transformation or generation of design information. One option was making a kind of programming language with which to make executable models. The language should be able to abstract a specific problem into something that can be solved generically. However, it quickly became apparent that such a strategy would not work, as – if it were possible – someone else would have done it already. The reasons for this impossibility are already stated in the problem definition and literature chapters as the 'model of everything'. What was a viable option was to make an semantics-free exchange format for information, which could be read through multiple means – the AML. During the ACCGSMS project the AML is symbolized in five ways: As a set of EMF/GMF classes in Eclipse, as an xml format for serialization and exchange, as a diagram format for human interaction, as a set of classes in a knowledge based engineering tool, and as a set of classes for Microsoft Visual Studio C#. All five formats use the xml format to transform from one into another.

The Eclipse framework allows for layering of tooling on top of each other, starting with a language definition, to a graphical editor, to tooling for consistency, querying, serialization, exporting/importing, AM reuse – as discussed in the previous section. This is depicted in figure 4.17(a), where the Eclipse environment is shown. The AMTool thus contains the knowledge bases needed to do the generic modeling tasks. The plug-in mechanism can be used to make the models 'executable', which means that the diagrams can represent algorithms (see figure 4.17(b)). The AMF does not have such executability originally, therefore it was added in the form of external knowledge base plugins – tools that contain the semantics to solve a certain problem (manipulating the models with explicitly programmed design knowledge).

Often, the Eclipse environment was not the best or easiest place to develop knowledge bases. Therefore, project partners developed their knowledge bases in other environments, and made them exchange information through the xml format (see figures 4.12 and 4.19). An example of an external format is shown in figure 4.18, where a c# version is depicted. For a workshop (appendix C), material needed to be made to show the exchangeability of information through the AML. However, the Microsoft Dot net framework was far better suited to interact with the Solidworks CAD tool and Microsoft Office tools than Eclipse was. Therefore the c# version of the AM language, developed in the litography case study (§5.5), was used to interact with those tools and import and export it to the shared xml format.

There are other examples of tool integration through application of knowledge bases and the AML, where the author of this thesis was not 'in the lead': A knowledge base to transform an AM into a Finite State Machine executable model, see [Yuan, 2011] and figure 6.1. Knowledge bases to make a model of state tree structures and perform hybrid systems verification [Alvarez Cabrera, 2011]. Application of the knowledge based engineering frameworks[Foeken et al., 2010], and transformation to Matlab [Alvarez Cabrera et al., 2011] to simulate a robot.



(a) (b)

Figure 4.17: 4.17(a) The Eclipse environment allows us to plug in new functionality. (1) the definition of the architecture modeling language, and the serialization to xml. (2) the diagram editor. (3) additional functionality, grouped under a single namespace. (4) Various packages to make modeling easier 4.17(b) External example, possibility of using a knowledge base to execute models within the Eclipse GMF environment. From a tutorial by [Brazeau, 2011]

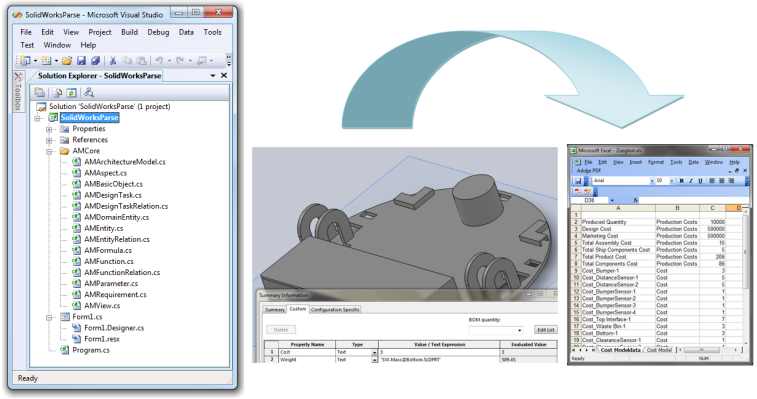


Figure 4.18: For some applications, Microsoft visual studio was used instead of Eclipse. For that purpose, a parser/writer of the architecture modeling language needed to be built. This model was used as an example in a workshop (see appendix C).

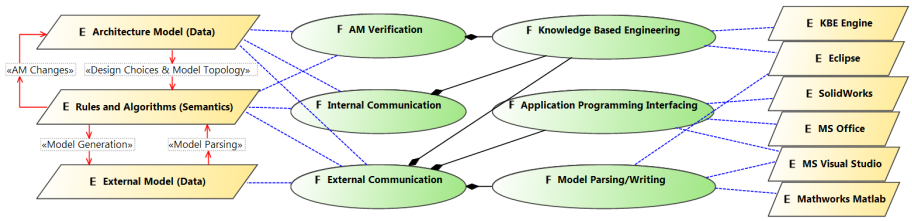


Figure 4.19: A number of different tools were used in the ACCGSMS project, all of which needed to exchange information through the Architecture Modeling Framework.

4.5 Architecture Modeling Framework Conclusion

The Architecture Modeling Language is an amalgamation of proven modeling concepts (chapter 3), practical industrial research (chapter 5), and a lot of iterative programming in various languages. The – preliminary – result is a practical tool that can be used to model much of the information in multi disciplinary design processes normally left to the tacit understanding of the people involved.

The resulting framework is, however, not the only answer to the research questions posed in chapter 2. The iterative way the case studies, interviews and meetings were conducted led to the AM framework in its current form. A different approach would probably have led to a different implementation. This will be expanded upon in the conclusion (7)

For example, in hindsight, choosing the Eclipse Modeling Framework as a basis for the tool was not the right choice. It was too cumbersome and too hard to customize. Also, an important part of the design process, capturing, archiving, and defending design decisions was not considered as a major part of the AM language. These topics will be addressed in future research (chapter 8), as time did not permit to follow up on it within the research period.

Concluding the chosen implementation of the AM framework, we can say that knowledge bases allow us to reuse design knowledge in a consistent way. Information can be exchanged in the shared data format of the AML across tools and disciplines, thus integrating the design process and the design information. Furthermore, by exchanging information through the knowledge base transformations, consistency of these transformations is ensured. This places the AM framework as a theoretical solution to the issues discussed in chapter 2. Through a series of industrial case studies, the efficacy of the framework in practice is demonstrated in chapter 5.

5. Case Studies

The case studies conducted throughout the AGCCSMS project were both the input and the testing environment for the development of the architecture modeling framework. This chapter will discuss the case studies, and how they pertain to the improvement of consistency, integration and reuse in multi disciplinary design. The validity of the research and the framework will be discussed afterwards.

5.1 Cases Introduction

Multiple case studies were conducted throughout the research project. These case studies served a double role. First, to provide insight into the real world issues and practices of large multi disciplinary design processes, and second, to provide a testing, implementation and verification environment for the academic ideas. This chapter describes the case studies from two perspectives:

- The case studies from a research process perspective: Industry-as-a-lab (§5.2), the concept of design process patterns (§5.3), and research validation (next chapter, §6.4).
- The case studies from a descriptive perspective (§5.4 to §5.6): Finding relevant industrial issues, finding solvable design problems, and developing tooling to mitigate these issues.

In figure 5.1 the research project is shown in this context. This thesis collates the results of this project. The issues and practices were refined into what is stated in the problem definition of chapter 2 and the literature of chapter 3. The ideas were developed into a concrete architecture modeling (AM) framework, as stated in chapter 4. The placement of this case study chapter is thus not chronologically, and its case studies are not conducted using the fully progressed AM framework. Rather, this chapter tells how and why the framework was developed to tackle company-specific issues with methods that can be applied generally. In the conclusion §7.4 we will discuss if the research about the framework is validated, and if the framework is perceived as a viable answer to the problems of chapter 2 by independent observers.

Three main case studies comprise the practical side of the conducted research. Who participated in these case studies and how and why they were conducted is the subject of the next section, §5.2 Industry as a lab. The case studies themselves are discussed in detail in §5.4, §5.5 and §5.6. Other, smaller, testing projects were undertaken throughout the

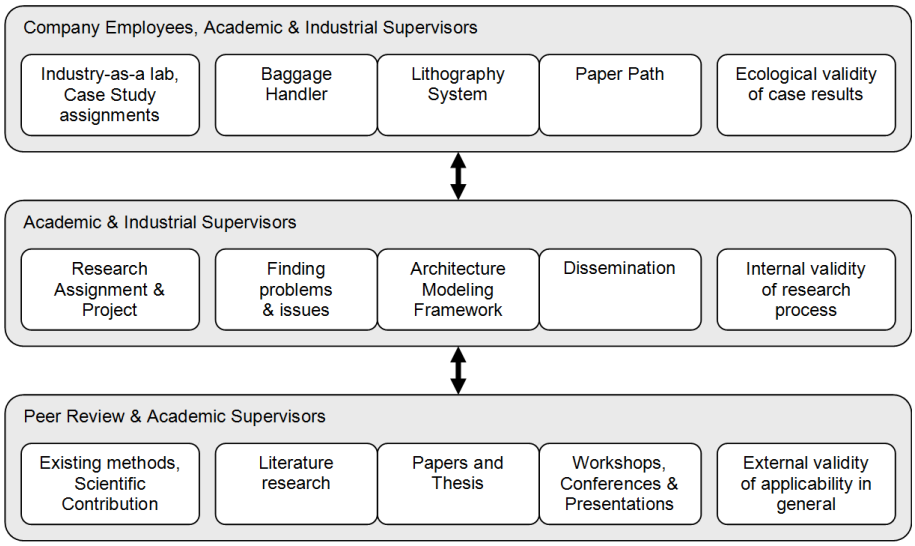


Figure 5.1: The research project in its context. Industry as a lab to find and solve relevant real world issues. The academic world to provide tested methods and review the relevancy of the project's research in a broader sense.

research period. However, as these were not reviewed by non academic partners, these are not included in the thesis.

The three companies represent an interesting cross section of high tech industry, and thus may offer a benchmark to the applicability of the conducted research to a broader set of companies. See '§2.3.1 Defining the Design Process' for the used definitions.

Case study 1 (§5.4). Vanderlande Industries manufactures baggage handling systems. These systems are one-off products, as all systems are unique. However, Vanderlande focuses on a design process with a high reusability. This means that they constantly search

Table 5.1: A qualitative comparison between the companies involved in the case studies. Relative goals between the companies are given in the second row. (goals not relative to their individual competitors)

	Vanderlande	**ASML**	**Océ**
Production size	one-off	small series	series
Design reuse	highly standardized design primitives	moderately standardized design primitives	low standardized design primitives
Artifact reuse	highly standardized product artifacts	highly unique product artifacts	moderately standardized product artifacts
Relative importance of drivers:			
Cost	min. cost	less important	min. cost
Time to market	min. t-t-m	less important	less important
Performance	less important	max. performance	max. performance

for reusable design primitives in multiple design domains. The case study focused on the connectivity of the design primitives across the design domains. We found a way to map all primitives to abstract functional primitives that can subsequently be instantiated to a complete structural design specification. Additionally, the network of newly defined multi disciplinary primitives can be evaluated in various ways, for example to find unused primitives or missing 'ingredients'. The case study led to the insight that automation across design domains was feasible, and to the basic framework layout of a function centered system description, a meta/reference model, design primitives and knowledge bases to generate design specifications (system models).

Case study 2 (§5.5). ASML is one of the world's leading providers of lithography systems for the semiconductor industry, manufacturing complex machines that are critical to the production of integrated circuits or microchips. These machines combine many interdependent feedback mechanisms for the correct positioning and exposure of the wafers that are to contain the circuits or microchips. ASML focuses on their ability to print layers of incredibly small patterns (with the smallest possible 'Critical Dimension') at high speed (productivity) exactly on top of each other (overlay accuracy). The case study focused on the development of (mechatronic) software control of these systems. This software is generated from a number of input models (electronics specification and their connectivity, mechatronic constraints and behavior model, mechanical properties). The infrastructure on which this software is generated is reviewed, and an attempt was made to find reusable design primitives. The case study led to important insights in the human communication factor of the architecture framework, mainly by defining the need for insight in the system architecture, also modeling aspects and views.

Case study 3 (§5.6). Océ is a global leader in digital document management and delivery technology, offering software solutions, digital printers, copiers, plotters and scanners. Océ-Technologies develops the print engines for these various systems. Within these print engines, the paper path – the way paper is transported and treated throughout the engine – is an important architectural unit. The design of the paper path determines to a large extent the positioning of the image on the paper and the throughput of the machine as a whole. The case study focused on bringing design primitives, design rules, and requirements together in a single meta model. From this meta model, one can quickly generate a conceptual print engine layout, as well as the input for a behavioral analysis of the paper path. This case study thus brought together all the aspects of the architecture modeling framework.

5.2 *Industry as a Lab*

This section discusses the chosen process of finding out what issues are relevant to the industrial practice, and how the resulting research was conducted and evaluated. This is summarized in figure 5.1.

The case studies were conducted at Dutch companies that joined the research program 'IOP/IPCR0602 – Automatic Generation of Control Code Software for Mechatronic Systems' (AGCCSMS). Where IOP/IPCR means Innovation Oriented Research Program / Integral

Product Creation and Realization of the Dutch ministry of economic affairs, agriculture and innovation. This program name already informs us on why certain companies wanted to join this program. They develop mechatronic systems for which the companies run a multi disciplinary design process, comprising dozens to hundreds of people.

This means that the opinion and cooperation of the people from these companies is very important to this research. Employing help from industrial partners in a research process is called 'industry-as-laboratory' [Potts, 1993]. The industrial partners had a group of 'spokespersons' who came together with the academic partners (the researchers and their supervisors) in a biannual meeting, the user committee (UC) meeting. This user committee can be seen as the supervisory focus group of the research project (groups of people (10-15) brought together in a guided discussion about a topic they are knowledgeable in [Babbie, 1997]). The user committee confirms certain issues common to the overall industry, on which basis the academic project partners arranged case studies at the individual companies (see middle of figure 5.1).

Within the case studies, we used a 'panel study' [Babbie, 1997], where we interviewed the same people throughout the case study. These panels comprise experts from various fields, but mostly system architects and engineers (Top of figure 5.1). From the panel, a single person was appointed as a permanent liaison between company and academics, to provide data and explanation for the case studies. The case studies were reviewed by these panels to control the applicability of the research to the company specific situations. For validation, the opinions of these people about the case study were compared with a focus group (UC meeting) of people not engaged with the case study. A more quantitative account of these meetings was given in the problem definition, in the section 2.4 Sources for Analysis of Consistency, Integration, and Reuse. The notes and minutes of these meetings, as well as the intermediate reports, have not been included in this thesis.

The programming and literature research of the case studies were conducted at the university offices. When new questions arose, more data was needed or (intermediate) results were presentable, the companies were visited. After a progress presentation and a tool demonstration, a review through qualitative interviewing of the company spokesperson(s) followed [Babbie, 1997]: "An interaction between the interviewer and a respondent in which the interviewer has a general plan of inquiry but not a specific set of questions that must be asked in particular words and in a particular order. Ideally, the respondent does most of the talking." Subsequently, the interview conclusions were used to iterate toward a new version of tooling and the case independent architecture modeling framework.

The results of the research by the author were presented for peer review in various shared papers. In these papers, the other authors focus on the mechatronic design applications of the Architecture Modeling Framework, whereas I focus on the consistency, integration and reuse of design processes, and the definition of the framework itself [Alvarez Cabrera et al., 2008, 2010, 2011, 2009; Foeken and Tooren, 2009; Woestenenk et al., 2011, 2010]. Other project papers where the author did not contribute [Tekin et al., 2009; Alirezaei and van den Boom, T. J. J. Babuska, 2012; Foeken et al., 2008; Foeken and Voskuijl, 2010; Foeken et al., 2010], and a dissertation by [Alvarez Cabrera, 2011] complement the practicality of the framework in software control design. Also, to get a response to the validity of the main re-

Table 5.2: Case Study Workflow

Case Step	Research Goal
Initial meetings	
Global goal of project	Validation
Global issues of company	Verification
Matching the company personnel to the PhD's	
Second round	
Discuss issues	Issues
Find achievable goals	Framework
Find publishable goals/research goals	
Model workflow current situation	
Identify hand-over	Integration
Identify non-value adding models	Integration
Identify sources of inconsistency	Consistency
Model design process pattern current situation	
How is design specification constructed from initial information?	(Knowledge) Reuse
Cut unnecessary model transformations	Consistency
Combine design decisions (constraints, requirements) to input of process	Integration
Combine known system properties to input of process (e.g. functional model, topology)	Integration
Combine design specifications to end of process (domain models)	Integration
Take human out of loop where possible	Consistency
Model design process pattern new situation	
Model remaining data and I/O in shared format	Integration
Model knowledge in software knowledge base	(Knowledge) Reuse
Experiment with company data	Framework
Upgrade framework with new results	Framework
Publish paper	

search deliverable, the architecture modeling framework, three workshops were conducted (appendix C). To disseminate the framework outside academic and user committee circles, various (poster) presentations were given, and a programmer was hired to professionalize the framework software (see bottom of figure 5.1).

5.3 Finding Design Process Patterns

The case studies started with an initial meeting at the company. In this meeting the academic partners were introduced to a broad group of people from the company. The academics then gave a presentation on the global goals of the AGCCSMS project. Then a company spokesperson presented the global issues of the company. After the presentations, a discussion between all people present resulted in a matching of company personnel to the researchers. Those groups would further work out a possible case study scenario. In subsequent visits, the PhD students and the company employees would work out the specific issues at a company, and iterate towards achievable goals that were interesting for the company, and fitted into the research objectives of the researchers. The case studies all had comparable formats, as expressed in table 5.2.

To express what we tried to find in the case studies, we will introduce the term **design process pattern**: A sequence of design tasks, cutting across multiple model types and tools, with design information from various disciplines. These patterns are not immediately apparent, as they are distributed over the work of multiple people – often even in different departments – and can also be distributed over a long period of time. The following case-based explanation gives a more comprehensive definition:

The goal of a design process is to come up with a design specification (again, see '2.3.1 Defining the Design Process' for the used definitions). In the first phase of the case study we look at what design tasks and stakeholders are involved in constructing the design specification, and how they are communicating with each other. This is what we call the workflow of the current situation (top of figure 5.2). This means we want to identify the hand-over between the stakeholders; what models do they make and how and when do they hand them over. What information is contained in these models? This leads to an analysis of non-value added models. Are there workflow steps where the design information is only transformed, and no information is added? How is the hand-over information compiled? Is it done by hand? If yes, this can lead to inconsistencies in the design process. The design tasks involved are the flow of the design process pattern.

The next step is formalizing the design knowledge needed for the design process pattern: The design knowledge needed to calculate or construct that part of the design specification. This can be for example knowledge of how the topology of the parts should be constructed, or how certain behavioral specifications need to be analyzed. This knowledge is often split over multiple design tasks, performed by multiple stakeholders, from multiple disciplines. Therefore we pull all the individual actions into one workflow model. Now we can see if certain tasks are not contributing to the final design specification. For example, in the paper path case of section 5.6, there was someone who made a pdf document from a CAD drawing, that was sent to someone else who measured the printed pdf document, and scaled those results to put them into an excel spreadsheet, which was subsequently sent to a third person, who derived timing information from the excel spreadsheet. In that case study, those transformations were replaced by a single topological model and a single derived timing model, written in the same language. The basic calculations for the transformations in this case were quite easily generalizable and automatable, and were programmed in a knowledge base that can operate on the design process pattern. In general: When possible, simple or repetitive design tasks can be captured in software – the knowledge bases. This will leave the human stakeholder free to spend time on more creative or exploratory design tasks, and will make the automated design tasks more consistent.

When it is known how a specification should be built, the decisions leading up to that specification can be abstracted from the workflow model. Often these decisions are constraints and results from the individual design tasks. These decisions are modeled as configuration options for the design process pattern, ideally in a central meta model (bottom of figure 5.2). The meta model can then provide a step by step assembly instruction for the design specification. When these decisions and constraints are used as the input of the design process pattern, we can use them as design parameters to generate various solutions for the design specification.

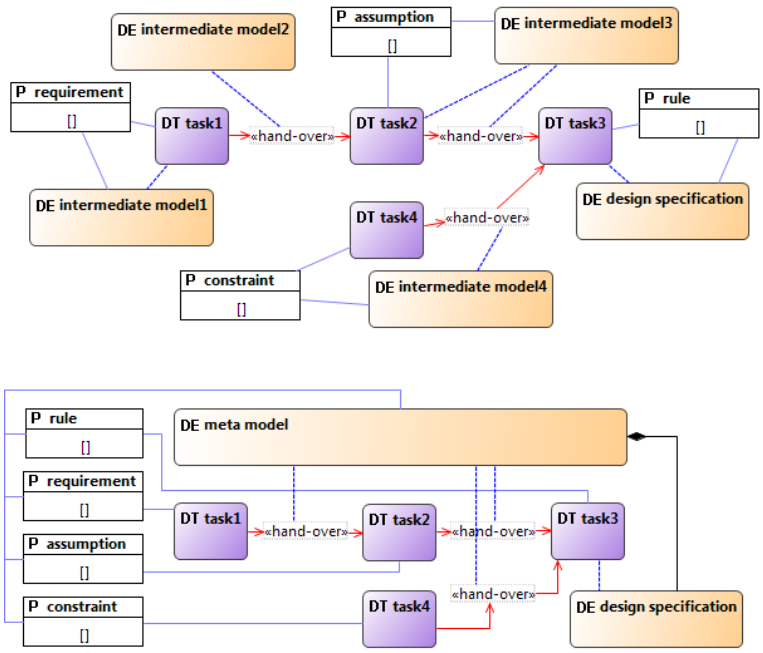


Figure 5.2: Modeling of a design process pattern, a sequence of design tasks, cutting across multiple model types and tools, with design information from various disciplines. Top: Original workflow, with different models for the different design tasks, often by different people in different disciplines. The flow of information is difficult to track. Bottom: A central meta model replaces the intermediate models. The design tasks use this single model to generate the design specification, which is also part of the meta model. The rules, requirements/constraints and assumptions are isolated to the front of the process as much as possible to make generating multiple design specifications easier. Note that the actual design process does not change, but because the pattern is found, it can be formalised.

Together with the knowledge bases, the design process pattern can then be redefined as a sequence of design tasks, cutting across multiple model types and tools, with design information from various disciplines, all executed automatically.

5.4 Industrial Case: Structural Model of Baggage Handler

Vanderlande Industries manufactures transportation systems. Within their product portfolio, we focused on baggage handling systems. According to the Vanderlande website: "Vanderlande Industries designs, builds and services leading baggage handling systems for airports of all sizes. These belt, tub and/or track solutions combine operational effectiveness, short connection times and high conveyability together with effective integral control of your baggage operation. Based on proven technology, in-depth business knowledge and industry best-practices, we deliver the highest availability, reliability and lowest costs per bag."

Figure 5.3: The end part of a baggage handler.

The specification of a baggage handler comprises a largely automated design process, consisting of assembling standardized components from multiple design domains. These domains deliver design specifications such as electronic layout, bill of material, embedded software, pricing, safety, simulation, and visualization. Each of these domains has its own library of design primitives and knowledge – a set of rules of when to add certain components to the design specification. In the case study, a multi-domain library reference model was implemented, with which system models can be created by configuring standard, re-usable, design primitives directly from a functional model instead of evaluating a rule-base as in the original situation. Automated consistency checking of the design and validation of the design libraries and the resulting system model were additional results. In appendix G, we see the end result: a new tool that can help in making abstract system models using transformed Vanderlande libraries.

This case study contains the following steps:

- Formulating company issues as perceived by company spokespersons (§5.4.1)
- Analyzing the current workflow for generating baggage handler topology (§5.4.2)
- Finding bottlenecks, and eliminating unnecessary model transformations in a new model generation process. Applying automation of design knowledge to automate this process. Providing a framework of tools and editors to help this process (§5.4.3).
- Testing the method with company data(§5.4.4)
- Upgrading the case independent architecture modeling framework with the results of this case(§5.4.5)
- The effect of this case study on consistency, integration and reuse, the topic of this thesis. (§5.4.6)

5.4.1 Company issues and relations to Consistency, Integration and Reuse

Vanderlande can automatically generate a large part of the design specification. This focus on automatic generation has shifted effort away from making the design specification directly towards designing and maintaining reusable design primitives and model generation. This transition is a continuous process that is largely conducted by people in the various design disciplines. Note this is a good example of what was described as a shared assets creation process by [Muller, 2011b]. Inside these disciplines, libraries of design primitives are continuously filled with more and more reusable design information. As a result, inconsistencies and non-value adding information transformations in the hand over between design tasks are few. However, these libraries do not have a fixed format shared between the disciplines, and there is no tooling to quickly review the content or connectivity in and among the libraries. There is an excellent design knowledge reuse system, but the consistency and the integration of design libraries can be improved. This was the focus of our case study. The generation of the design specification originally takes five subsequent steps. Therefore we reduced the number of steps to two, which reduces the amount of information transformations, which makes the process faster and easier to integrate and manage.

As the design process at the company is already highly formalized and automated, this case study reused the existing design libraries and workflow to a large extent. However, this 'automated design' case gave the research project interesting insights into emerging problems of such automation. The shift from working on a design specification to working on design primitives knowledge bases (libraries) introduced a new set of consistency, integration and reuse issues (these issues were abstracted into the general issues of this thesis in appendix B). According to company discussions, these are:

- *Lacking overview in existing design primitive libraries*. The libraries are originally defined as xml files. Basically the libraries are a list of rules for constructing the design specification. A person can only parse through this file by reading it, there is no graphical representation that can help visualize certain information, and there are no filters or search methods to make navigation easier. This poses a potential consistency problem.
- *Difficult to manage libraries, interrelations are hard to identify*. This is an integration issue. The libraries, or knowledge bases, have different owners, and have been historically built in different modeling languages. As a result, the relations between components on a system level only becomes apparent after construction of the system's design specification.
- *Hard to find propagated design changes when changing design primitive in knowledge bases*. This consistency issue is a consequence of not being able to detect interrelations between design libraries. Deleting or changing a component could create legacy artifacts in the libraries.
- *No consistent way of modeling new 'abstract objects' if they share libraries*. Someone could add a component to one library, without any corresponding component in another library. As an example: add a motor electrical scheme without a motor PLC software control. As a result, the generated motor model would not be controlled by anything. There is no integration checking method in place to check this automatically.

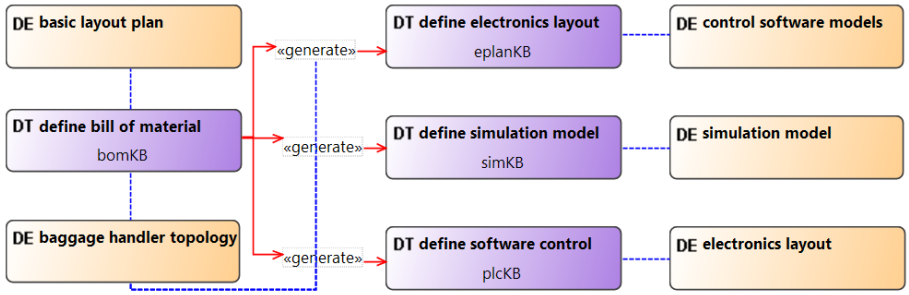


Figure 5.4: Original Vanderlande framework for generating the design specification. From a simplified 2D plant layout, the bill of material is generated, which is used to generate the other domain specific models. The gaps in the figure show where various checkpoints, switches and scanners are located.

- *Five intermediate steps in filling the product definition*. Every design process step extends the baggage handler topology file – the design specification. However, these steps are sequential, and need to evaluate a lot of design rules at every step to 'calculate' the appropriate design object from its primitives. Not all rules are needed however, as many of them lead to the same design object for any possible configuration, which means we can skip evaluating the rule and implement the primitive directly.

5.4.2 Model workflow current situation

The workflow of compiling the system specification was already largely automated at the company – up to 80 percent of the design specification is generated as domain specific models (see figure 5.4). The workflow starts with a basic layout containing the geometric constraints of the system, so it can be fitted in a specific airport. This layout is used to generate the bill of material of the various mechanical components, and used as a basic topology of the system. From the functionality of the components, the need for electronics, software and simulation is deduced (among other things) and added to discipline specific meta models. From these meta models, the actual domain specific engineering models are constructed (e.g. PLC software control code) in a separate process that is not reviewed in this case.

5.4.3 Model design process pattern new situation

The method chosen to mitigate the issues is the construction of a new primitives library, by solving all the different rule combinations of the original domain specific libraries. In this library, we now have a set of multi domain design primitives that can directly prescribe the instantiation of design objects in the various domains. The instantiation of these multi domain design primitives is then prescribed in a single meta model – the baggage handler architecture model – instead of solving design rules at the mono domain level as in the original situation. On this meta model, we can implement various additional tools to manage consistency, integration and reuse, resulting in a new framework for building design spec-

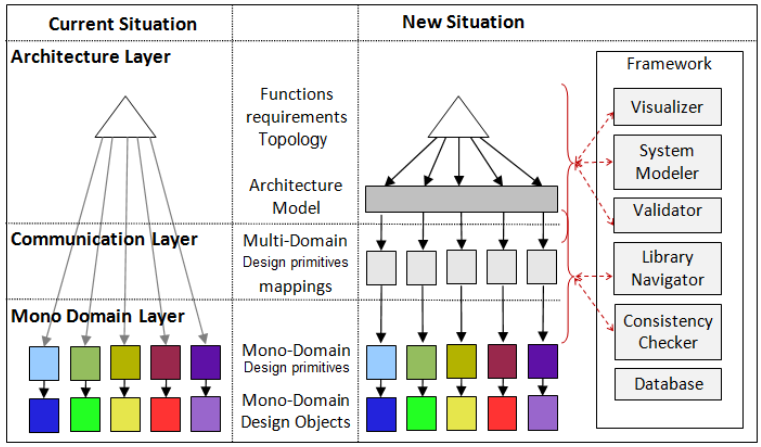


Figure 5.5: The new data structure allows for addition of pluggable tools, such as visualization or a system modeling editor. These tools can act on an intermediate architecture model that specifies how the domain specific models need to be constructed.

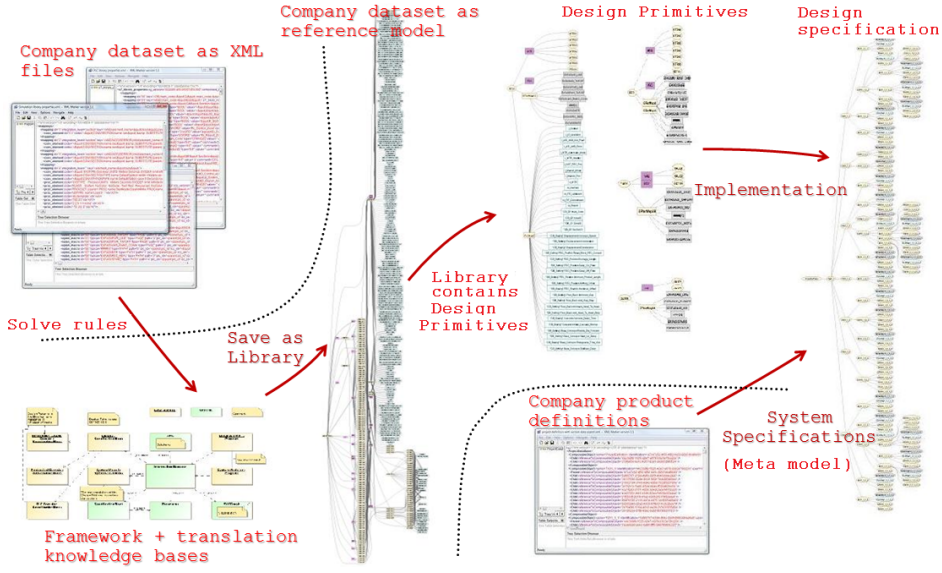


Figure 5.6: Case Overview. The company data was mined and transformed to an integrated library. This library contains a set of multi disciplinary design primitives. These can be implemented into a system's design specification using the original company input file. This input file specifies the functionality and geometry constraints of the system.

ifications as well as managing design primitives. All this tooling and the new library were written in a single language to facilitate exchange and transformation of information. The old and new situation are represented in figure 5.5. The whole process of transforming the original libraries into a new one is depicted in figure 5.6. The result is a new 'tree' of design primitives. The objects in a functional model can be put into this tree to select the appropriate multi disciplinary components, while the sequence of the functions determines the topology of the design specification.

As a result of this approach, we can tackle the company issues stated in 5.4:

- Overview in design primitive libraries was improved: A generic library and framework were introduced, in which relations between and within libraries were made explicit and 'navigable'. Functionality such as searching and copy/pasting/deleting were made possible. This network was visualized by a new 'visualization knowledge base' based on Graphviz [Bilgin et al., 2011].
- Managing libraries and their interrelations was improved: The design specification is solely determined by the rules in the knowledge base, the relations can be made apparent beforehand. All relations were made explicit by a tool programmed in C# and serialized in a new XML library, using a single modeling language for all domains. Design primitives are now unique, making sure no duplicate references can exist. Information management can be done by addition, alteration or removal of nodes in the library tree. Missing implementations in the library would be apparent immediately when there are no references to 'child' primitives in a certain domain.
- Find propagated design changes when changing design primitive in knowledge bases: Deleting or changing a primitive can create legacy artifacts in the knowledge bases. Deleting or changing consistently can be achieved by giving the possible configuration context of any design primitive. As the new library stores this context, it enables searching for parents and children of the changed primitive, thus affected components have been be identified (see figure G.2 for an example).
- A consistent way of modeling new 'abstract objects' if they share libraries: This issue was not addressed in the case study for time reasons, but an extension can be envisioned that informs the user if a primitive misses 'ingredients' from certain domains.
- Reduce number of steps for defining the design specification: By solving all design rules in all the original libraries beforehand, we only need to parse the original input file. The calculation of the end result then takes one step. This was implemented by aligning the design primitives with functional goals of the various parts of the system. The function is passed through the primitives library, adding all relevant primitives to the design specification. This design specification is at the same time configured by implementing a 'length' parameter to dimension the whole system. The process is recursive, and stops when there are no more levels of detail to add from the library.

5.4.4 Experiment with company data

In appendix G, a demonstration of the delivered tool is given. To test the new library and the surrounding framework of tools, we implemented a company input file on our framework.

Figure 5.7: Output model of the design specification configurator. Holes in the model are holes in the function sequence or holes in the library. Both signify traceable inconsistencies in the new design process.

If the framework works properly, the generated system's design specification should be the same as what the companies existing tooling would generate. This was indeed the result of our test, and therefore verified the new tooling works. The components were generated with the correct functionality and also the system layout was correct (plotted in 3D CAD in figure 5.7). Apart from working correctly, we also tested some improvements resulting from the use of the new multidisciplinary library and tooling.

The new multidisciplinary library replaces the original company xml files (company dataset as reference model in figure 5.6). This model can be represented in multiple ways, providing overview of libraries and product definitions. As all libraries are now written in the same language, information management is applied on the reference model in a runtime environment (e.g. Databases, visualisation, consistency in figure 5.5). Furthermore, because the model is now loaded in memory, it reduces calculation time with respect to sequential XML libraries. By recursively selecting needed design primitives instead of evaluating knowledge rules further reduces calculation time.

We introduced tooling to reflect on the new multidisciplinary library. This allows for knowledge bases that check the consistency of the design before it is implemented, such as the design assistant we developed (appendix G). Childless objects (or stubs, or leafs) are now traceable in the primitives tree of the multidisciplinary library. These can be used as elementary design primitives for the construction of new multi domain design primitives. Such as a transport-and-check conveyor with electric motors, scanners and PLC software control. In reverse, the existing multi domain design patterns can be checked for consistency – do they have primitives from all the necessary domains? For example, one could check if all motor driven configurations have software control primitives.

In the original rule based situation, it could occur that a rule was doubly implemented, resulting in two configuration types that would instantiate the same primitives. This can happen if the stakeholders find it easier to add a new item to the knowledge base than to first check the whole knowledge base on duplicates. In the new situation, such a con-

struction leads to one function implementing double the amount of primitives. The error becomes immediately apparent through applying a graph representation (figure G.2 in the appendix).

The new network based design primitive tree can be parsed and scanned from every point in this tree. This is a starting point in determining causality in design changes. Top down this means finding the software code, cost calculation, simulation model primitives that need to be updated as a multi domain design primitive is changed. Bottom up this means that one can find the affected multi domain design primitives from changes in a domain specific primitive (see figure G.2 in the appendix). Note that this method does not say anything about *how* the change will work out.

As the design primitive network can be parsed as a whole, it provides other metrics on the design portfolio of the company. What is the possible solution space of the library (all possible configurations)? How many elements are there (for standardization or diversification)? What are the decision path lengths of the multi domain design primitives: Need they be cut up into smaller configurations, or can they be combined into larger primitives (modules)?

The new framework of tools and the new library were submitted to the companies spokesperson for evaluation. The conclusion of the case study, according to Vanderlande contact person Peter Kamps, is that *"the demonstrated case helped Vanderlande formulate ideas to: Analyse the library doing impact analysis browsing the library network. Find mismatches between the libraries (situations covered in one but not in all libraries). Find mismatches between a given project definition and the common library (situations in the project, not covered in the library)."* Currently , Vanderlande is working on an integrated library, that covers all design domains in a single design step. An information manager is now implemented that keeps track of things such as the multi user information flows, versioning and serialization.

5.4.5 Upgrade framework with new results

The issues derived in the initial company discussions were mitigated by introduction of a design framework that would later in the research project grow into the architecture modeling framework. The case study allowed us to construct and test the rudimentary architecture modeling framework. Of the architecture modeling language only the Parameter and BasicObject class were developed (see E). At the time, the BasicObjects were related as parent/child sets to act as 'composable objects', thus enabling the definition of sets of information. The Parameters were related in a causal sequence, thus enabling the definition of flows of information. Separating the semantics from the structure of the information enabled us to build generic tooling to work in conjunction with the case specific information network (see the right side of figure 5.5). This generic tooling came back in evolved form in a C# version of the architecture modeling framework later in the research period. The same classes of abstract BasicObject and Parameters seemed to work in situations very different from the baggage handling case, which indicated that we were on the right track in defining generally usable language concepts.

It also became apparent that it is useful to separate the information from its representation, as multiple users existed for the information. This lead to a tiered framework as shown in figure 5.5. Together with the conclusion of using functions to organize the mono domain design primitives into multi domain design primitives, this lead to the definition of an architecture model, where the connection between architectural goals of the organization or system and discipline specific information was made, thus closing the design gap between abstract information and implementations (see figure 3.4).

5.4.6 Results concerning Consistency, Integration and Reuse

To summarize this case in terms of the research: Integrate existing design libraries into both a meta model for reusable multi disciplinary design information, and a reference model for consistent information management and communication. In table 5.3 the global thesis issues are transposed on the results of this case study.

Table 5.3: Baggage Handler Issues and Resolutions, see appendix B

Index	Issue	Resolution
	Improve Consistency	
1	Big Picture	Information from all design domains was collated in a single new multidisciplinary library, representable as a graph.
4	Prevent Ambiguity	Duplicates or missing implementations became apparent by consistency checking the new library.
8	Provide Traceability	The new library can be navigated and queried to find specific information. Consistency between mono disciplinary design libraries can be accounted for.
13	Automate Hand-over Generation	Information is stored in a single library, making the construction of the design specification a single step, thus eliminating handover from the design process.
14	Do Not Duplicate Information	Information is stored centrally and uniquely in a new multidisciplinary library. Consistency checkers are in place to detect duplicate or missing information.
	Improve Integration	
16	Shared Data Model	Information is stored centrally and uniquely in a new multidisciplinary library.
19	Multi-Composition	Certain design primitives have multiple 'parents', thus minimizing the number of unique components in the system. The primitive needs only be edited in one location.
20	Multi-Mapping	A functional flow and decomposition was used to construct a design specification with multiple discipline specific design primitives.
23	Simple Shared Data Language	The new multidisciplinary library is constructed with two language concept types, the BasicObject and the Parameter.
24	Abstract and Reference Model Content	Domain specific information was referenced to a central meta model, that enables model generation.
25	Support Parametric Definition	The topology from the baggage handler can be instantiated with only one length parameter per component (see figure 5.7).
26	Support Parametric Integration	All domain specific information is abstracted in the same language, integrating the domains.
27	Mechatronics Design	The baggage handler is a mechatronic system.

Continued on next page

Table 5.3 – continued from previous page

Index	Issue	Resolution
28	Align to Functions	The functional decomposition and flow was used as a backbone for the new meta models.
	Improve Reusability	
9	Meta Modeling	The new meta model can instantiate much of the design specification.
31	Model Multi-Domain Information	The new meta model combines design information from four disciplines.
32	Defining Reusable Design Primitives	New multi disciplinary design primitives were defined based on component functionality.
35	Impact Analysis of Reusable Design Primitives	In case of design changes, affected primitives can be found. How the changes propagate is not knowable.
36	Lean Models	The new model is 'leaner' by a single language, removing handover, and unique referencing of design information.
38	Platform Development and Support	The construction of the baggage handler design specification is a product platform: a huge variety of baggage handlers can be instantiated by implementing a meta model with the multidisciplinary library design primitives.
39	Freedom vs. Speed vs. Consistency	The new meta model makes construction of the design specification slightly more consistent (lower risk) and faster. The design freedom is helped in consistently adding new or changing existing design primitives in the multidisciplinary library.

5.5 Industrial Case: Software Generation of a Lithography System

ASML is one of the world's leading providers of lithography systems for the semiconductor industry, manufacturing complex machines that are critical to the production of integrated circuits or microchips. Headquartered in Veldhoven, the Netherlands, ASML designs, develops, integrates, markets and services these advanced systems, which continue to help customers – the major chipmakers – reduce the size and increase the functionality of microchips, and consumer electronic equipment. A lithography system is filled with servo control groups, consisting of actuators, sensors and controllers, which all need to be controlled by embedded software. In order to reduce the variety in the software development, a standard design pattern for servo control groups is desirable. This should improve communication between different design stages as well as improving the implementation of cross cutting concerns through all aspects of the system.

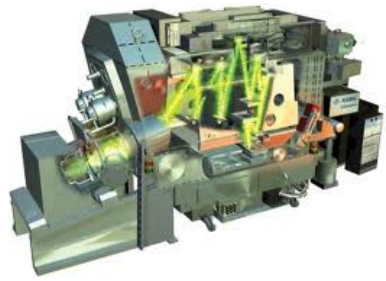


Figure 5.8: ASML extreme ultraviolet lithography concept.

In terms of this research, the case study focused on collecting design information and classifying it in order to define an integrated and consistent information model. The goal was constructing the design specification from this information model by reusing specific design knowledge. The case led to various new language concept types for the architecture modeling language, defined issues in scalability and representation, and led to a new framework implementation in C# instead of the Eclipse graphical modeling framework. This case study was only partially successful, as will be explained in subsection 5.5.5. We will explore the following steps:

- Formulating company issues as perceived by company spokespersons (§5.5.1)
- Analyzing the current workflow for generating embedded software (§5.5.2, §5.5.3).
- Finding bottlenecks, and eliminating unnecessary model transformations in a new model generation process. Applying automation of design knowledge to automate this process. Providing a framework of tools and editors to help this process (§5.5.5).
- Testing the method with company data(§5.5.5)
- Upgrading the case independent architecture modeling framework with the results of this case(§5.5.6)
- The effect of this case study on consistency, integration and reuse, the topic of this thesis. (§5.5.7)

5.5.1 Company issues and relations to Consistency, Integration and Reuse

ASML builds the most advanced lithography systems in the world. Their market share has been expanding continuously, which demonstrates that their machines are the best product for their customers. While this is good news for ASML, it also leads to some serious

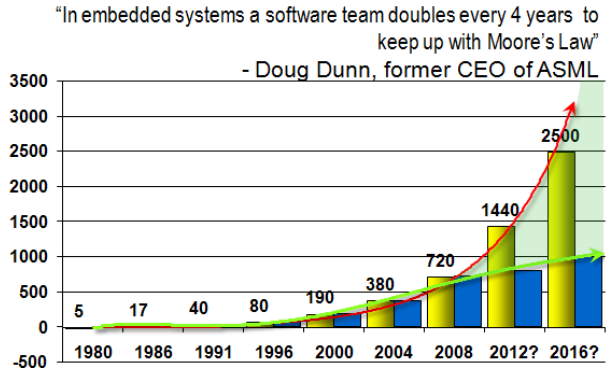


Figure 5.9: Following the trend of software engineers over the last 20 years, we see an unattainable exponential growth [Embedded Systems Instituut, 2002].

issues. Mainly, developing more and more advanced machines and receiving an ever increasing market share necessitates a fast growing number of employees (figure 5.9). In this case study, the reasons for this explosive growth in employees are identified, and the main issues that arose as a result of this growth. This was done by repeatedly interviewing employees, recording, filtering and collating the ensuing discussions, and reporting them back to the employees (§5.2). The issues were abstracted into the general issues of this thesis in appendix B.

Issue: A single model for ASML systems

The ASML Design process contains lots of information, shared between hundreds of engineers, making it difficult to model. Integration issues arise matching multiple sub systems, disciplines, departments, projects, teams that need to work together while there is no common view of the system or the design information. Consistency issues concern the informal and variable way of design specification, and documents covering only parts of the designs. Reuse issues concern communication in terms of implementation instead of thinking in generic terms: Insufficient abstractions from motion control, temperature control, fluid control, gas control, machine damage control towards generic process control.

A better understanding of the generic system architecture should make modeling it possible. Formalization of architecture in a model means you can communicate and share it. This means the domain engineers get a better idea of what they have to do, systems engineers get a set of metrics and a language to manage the design process, and management gets an overview of the issues and progress of the design process in terms of Key Performance Indices (eg. system precision, throughput speed). With such a model in place, it could be possible to identify abstract components that need similar design processes to develop. If this is the case, it can lead to reusable design primitives, design patterns, automated model transformation, validation and verification models and tools, etcetera.

Issue: Bridging the (logical) Design Gap

The design process is currently not supported by an integrated model that captures the design process: the transformation between the architectural purpose of a device and its mono disciplinary design specifications. A single engineer will have a good idea of what his/her design task is, but probably has less overview of how and what his/her design decisions will effect other parts of the design process. The reason is models tend to capture the results of design decisions, not why, when, by whom or how they were taken. As a result, recurrence of the original design problems will not be recognized, and the company can make the same mistake twice, or will lead to duplicate engineering work. Finding and using design process patterns (§5.3) can make the design process faster and more consistent. A clear transformation sequence between design tasks will result in less errors in specification during the design process, by eliminating unnecessary, error prone, untimely transformation of information and hand over steps (see §2.2).

Issue: Better Separation of Domains/Concerns/Aspects

On a device level, and on an architectural level, all components and functionality in a lithography system are integrated, otherwise the machine would not work. In the design process they are separated (discretized) in design tasks for departments and individual engineers. Separating the design tasks is not always straightforward, because of complexities such as cross cutting concerns, interfacing to multiple domains, interrelated behaviors, etcetera:

Domains are the disciplines involved in developing an entity: Mechatronics, Software, Embedded Software, Electronics, Mechanics, and Management. These domains tend to have their own models and tools. With these models complete – the design specification – you can actually manufacture a system. Aspects are concepts of a system that cut across the domains: Expected transmission delay, Cost, power profile, Configuration management aspects, Diagnostics Aspect, Calibration Aspect, Real time performance. Concerns are attributes that cut across entities: Throughput, Overlay, Imaging, Safety, Startup / Calibration / Shutdown / Error Behavior, etcetera. In the ideal case, you can budget concerns to aspects of the entities or functionality, and budget aspects to domain attributes. However, this is not straightforward, as ASML currently has no 'language' to model this type of information.

Issue: Reusability

While ASML systems rely on a lot of creative and innovative technology, there are still partial design processes that need to be iterated often, or have a very similar characteristics at an abstract level. Then it becomes useful to look at ways of formalizing and automating these design tasks in order to speed up the design process and improve the design process consistency. A good example is the introduction of process control, which could be a generalization of: Motion Control, Temperature control, Fluid control, Gas control, Machine damage control. While the physics, sensors and actuators involved are different, the required mathematics and embedded software to control these systems, and the way the signals need to

be routed through the hardware are very similar. Also, currently these process controls are modeled in a variety of languages, which does not help in finding similarities. Why should hundreds of engineers separately come up with similar solutions for similar problems?

Issues to case goals

The case goal for ASML is to get an insight in the aspects involved in the design of a control system: *"Goal is to come to a well defined and complete system design methodology based on both a use case driven top down approach for system decomposition and a bottom up implementation technology driven approach for uniform and comprehensible system documentation, harvesting the advantages of both approaches and bringing these together in an integrated methodology covering complete systems, to accelerate system development within the ASML organization."* As case material, the design data from the Wafer Positioning Short Stroke software design was available. By looking at the development of this sub system, we can generalize what should happen when developing other control systems.

We used an existing ASML framework to formulate possible improvements §5.5.3. At an abstract level, translating the issues of section 5.5.1 to this framework should help in putting ideas in a context. This is done in §5.5.4. The dataset itself was also analyzed, of which the results are stated in section 5.5.5. Additional functionality and layers will be described to set up a context that allows for the definition of reusable design process pattern for Control Systems.

5.5.2 Model workflow current situation

In figure 5.10 the approximate current ASML workflow to generate input for embedded software generation is represented. This is approximate, as the workflow might change from system to system, and the organization grows in the meantime. Also, it is no single person's responsibility to manage this whole process, as it is done by three different design disciplines. The figure is made unreadable on purpose for intellectual properties reasons, however, we can see that a lot of steps are needed. Every step has its own language, design primitives and design tooling. The question is, are all these steps necessary? or can we combine some of them? Is it possible to express all steps in a single language? What should such a language look like? And what design primitives are really necessary? can they be generalized into more basic objects that can be used for multiple disciplines? Finally, some of the design tasks result in manual transfer of data, and others have a rapidly changing set of design primitives. This can be a source of inconsistency, as people downstream will not automatically be informed over the validity of their data.

To evaluate a more concrete situation of the workflow of figure 5.10, a dataset was evaluated. This dataset contains the files needed to develop the wafer stage positioning control software. The dual wafer stage positioning consists of a chuck, which is positioned by a Short Stroke, a Long Stroke and a Balance Mass (Figure 5.11).

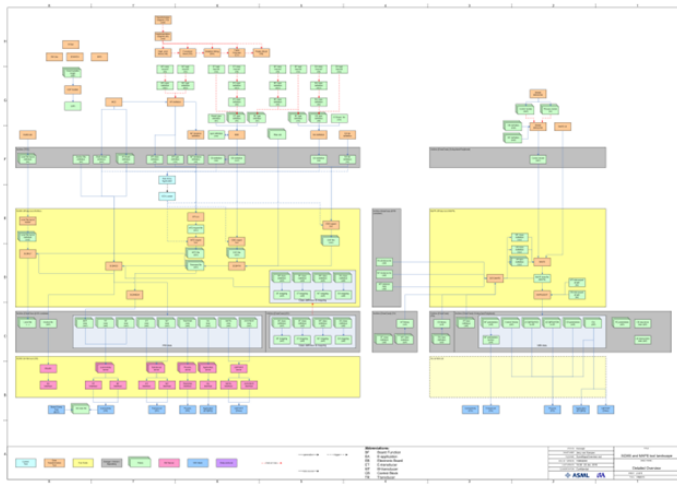


Figure 5.10: The model transformations needed to provide the input for software generation in the Lithography System case. Regions are different departments, shapes are design tasks. Figure is blurred for intellectual property reasons.

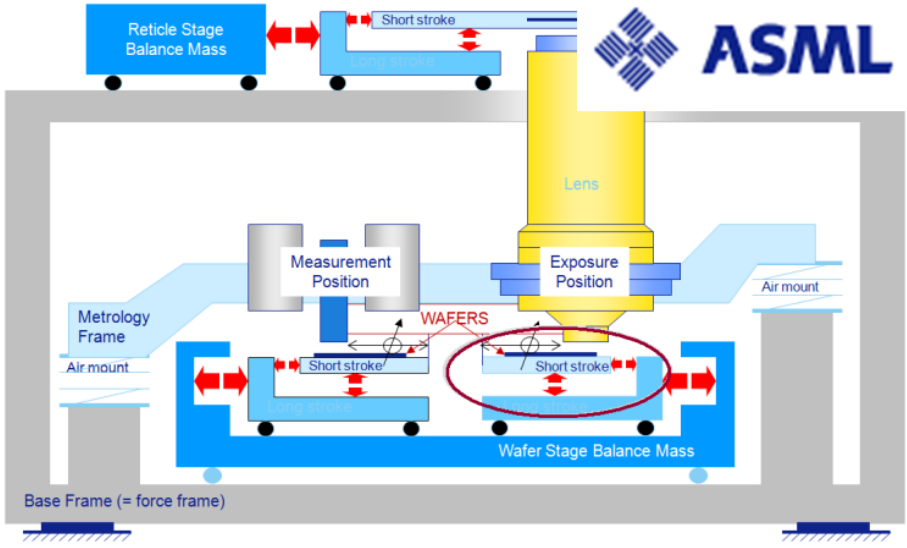


Figure 5.11: A schematic drawing of the actuated components of a lithography system. The wafer positioning components were considered in this case study.

5.5.3 Workflow analysis

To help organize and formalize the design process, ASML is developing the Controller Architecture Reference Model (CARM) framework: This CARM describes the components needed for control of a process in an ASML machine. CARM is meant to be used in a multidisciplinary environment covering the software, electrical and mechanical disciplines. According to the ASML spokesperson, the goal of CARM is to create a model that:

- Has a recognizable structure thereby easing maintenance and transfer of the model to other persons.
- Offers a view on the system that can be used as a means to let the different disciplines communicate amongst each other.
- Minimize variation points in order to create a model that has a long lifetime.

In figure 5.13, we see the CARM (expanded with points added by this case study). These components are represented in four layers of abstraction: (Desired) system behavior in terms of the controller functions in a software **application layer**; Control algorithms for processing sensor input and calculations for output to actuators in the **process layer**; Physical components and their properties such as sensors, actuators, loads and processors in the **transducer layer**; Physical, electronical and logical connections between components and the rest of the system in the **connectivity layer**. A CARM is constructed with a set of tools called the *CARM facilities*. CARM phases 0 to 4 provide for a stepwise transition through the design process, keeping the overall integration in view at any time (See figure 5.13 for these CARM phases expanded with the case study additions).

In the mono-disciplinary domains, design objects are often reused. We can data-mine these reusable objects to find a mechanism to implement them from a CARM level higher. This can be done by extracting design decisions into a knowledge base. However, the design decisions that have led to that specific object tend not to be captured, which lands us back at the CARM2 reusability remarks. Individually, the engineering domains have tools and frameworks to facilitate the various design aspects of the control system design:

- CARM facilities for embedded software. To be able to develop the embedded software control, an architecture has been developed on the embedded motion control platform. The servo controllers are designed using a newly developed modular motion control system, called the 'CARM facilities'. Each controller is a control network, built from a library of control primitives.
- MAPS for behavior modeling. The MAPS tool is able to configure a servo group in process context. It can model the behavior of a control system. As such, the MAPS configuration tool contains a formalization of part of the control system design process.
- Stages Configuration Management System (SCMS) for configuration of electronics. The SCMS is used to model the electronics connections between all the hardware. However, this is done for the wafer stage as a whole, which can make it difficult to find back a single control system.

- Software. In the software department, the inverses of MAPS and SCMS provide a bottom up integration of the SCMS and MAPS worlds. This makes it possible to implement the right software at the right place.

Similarities of mechatronic dominant (sub) systems, such as control systems involved in process control, temperature control, fluid control, gas control, machine damage control, etcetera, can be exploited to define design process patterns and standardized architecture patterns. For these systems, the domain languages are available, and they already have well defined structures; the design patterns are well known by the people involved. Also, their implementations and configuration tools are available and their functional performance is as required. The result is that this case study has a dataset on which to experiment.

Formalization of engineering data from the various domains should be researched. The successful implementation of reusable design process patterns hinges on a number of additional demands. For a whole sub-system, too many implementation details need to be managed; a proper layering of abstractions could hopefully reduce the level of detail. Finding causal dependencies can automate implementation in multiple levels. Apart from models for implementation, we can at the same time also attach models for the analysis of related aspects, such as 'overall timing performance and predictiveness', to the design pattern. Object oriented concepts such as specialization and encapsulation make it possible to define robust patterns, of which parts can be changed according to paradigm shifts or supplanting of other control algorithms, as was demonstrated in the Baggage Handler case study in section 5.4.

5.5.4 Model design process pattern new situation

We used the CARM as a framework to formulate possible improvements. At an abstract level, translating the issues of section 5.5.1 to this framework helps in putting ideas in a context. This is done in the next section. The dataset itself was also analyzed, of which the results are stated in section 5.5.5. Additional functionality and layers will be described to set up a context that allows for the definition of reusable design process pattern for Control Systems.

With these models, we can envision a framework as seen in figure 5.15 that introduces a common data language, information structure, and some modeling tools. Within such a framework it would be possible to formulate a template data structure for a Control System, considering its main design aspects. This central structure can than replace many of the existing steps in figure 5.10, and act as a meta model to instantiate domain specific models.

Issue: A single model for ASML systems

To come up with a single central architecture model for lithography systems, it is necessary to capture correlations between the system aspects and its architectural concerns. In figure 5.12 we see a concept for such a model as it was developed for the case study. It combines the idea of an architecture model, meta model and reusable design objects. The architecture

model acts as a reference model for the goals and concerns of the design process. A meta model of a lithography system informs how various design objects must be connected, and transformed to domain specific engineering models and software. A communication framework supports these flows in providing a single language and the automatic transformations from and to various external models.

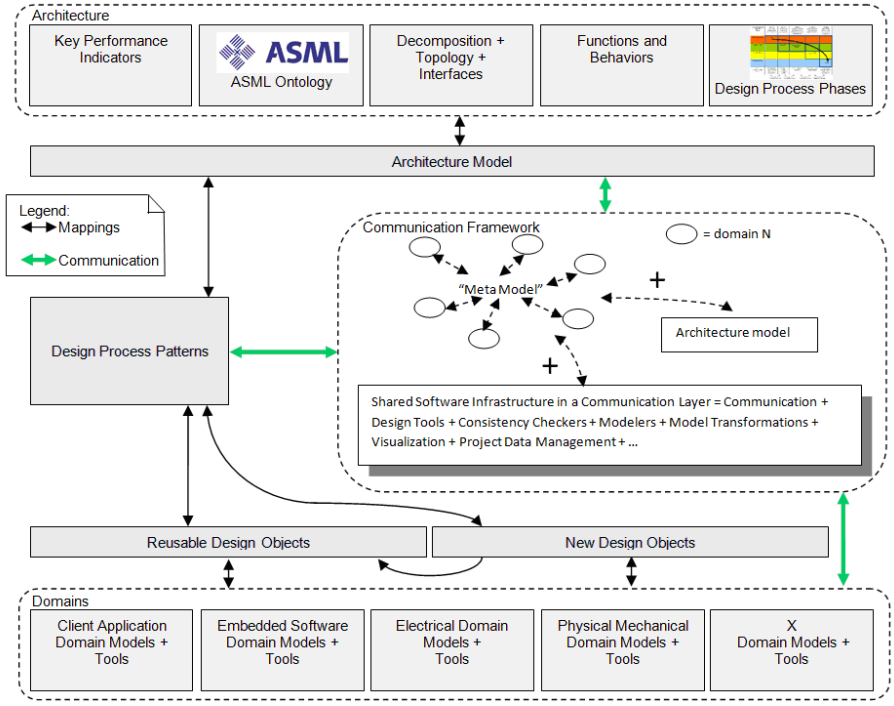


Figure 5.12: How can we bridge the gap between architectural concerns and domain engineering models?

The need for an architecture model comes from the growth in personnel and system complexity at ASML. Companies often copy their system architectures from the previous version of a system, but do not record it in a consistent way. Therefore, the architectures exist only in the mind of a few individuals. Over time the architecture grows under influence of added system functionality. Parallel to this, more people are needed to design the system and its behavior, and the number of dependencies and other relations among the system's constituents grow exponentially. Eventually, the architecture becomes too big to fit in a single designer's brain. In other words, the architecture becomes more complex. Because the architecture was never captured in a model, and there are no tools to capture the architecture in the first place, the design process becomes very difficult to manage. As a result, people need to spend much time in meetings to convey what they want or mean. An architecture model should therefore be foremost a communication tool: 1. try to get people to formalize what they are doing. 2. Synchronize these results between people and put it in a model.

3. Align the model with a reference architecture that is more or less robust over multiple products.

The architecture model can also capture some global metrics of a system and its development: Management gets informed in terms of key performance indicators. When systems do not function within these indicators, they want to know how much time and money it costs to avail the problem. However between the definition of a key performance indicator and the actual system behavior, the indicator gets lost through all design decisions, which transform it into hardware and software specifications, and eventually machines and code. When a problem occurs, it is often found only at the machine level. It is difficult to relate that specific problem to the actual design choices taken from the original indicator.

At the moment, the communication from several domains to the software domain is automated at ASML. This development is likely to be expanded to incorporate more tools, domains and directions (the CARM facilities). As the information exchange is already being developed, it is possible to add information management functionality to this exchange. Think of checking for allowable names of parameters through dictionaries, checking ranges of parameter values, checking if versions of referenced information are up-to-date, navigating the meta model to find causal relations, visualizing the connectivity for review and management purposes, etcetera. Most importantly for the exchange of information is the development of a single language that can be used as an ASML exchange format to transform domain knowledge and data.

Issue: Bridging the (logical) Design Gap

From the issues section 5.5.1 we deduce that a lot of things need to be taken into account in developing a control system for lithography systems. In figure 5.13 we see a design process pattern developed for this case study. The design process runs over various CARM phases, and at each phase, more concrete design specifications are delivered. The figure was compiled after taking into account the CARM framework and facilities as discussed in section 5.5.2.

The figure shows the layers of the CARM framework, and extends them with the temporal CARM phases. In the intersections of the layers and the phases, we can define deliverables of the design process up to that phase. The large arrow shows the global design process direction from an idea to design specification. Additionally, we see a new Communication Layer. This layer consists of a language to abstract information in the other layers (meta model), and a mechanism to make relations between the layers (reference model) as well as specify how information must be transformed (knowledge based engineering).

We start with a required function for the control system, such as 'position a wafer'. certain ideas will be formed as to how to perform that function within the global requirements of the whole lithography system. These ideas will be combined with the lessons learned of previous systems into an architecture. This will broadly describe the processes needed to be controlled and measured by the control system. The architecture model must show

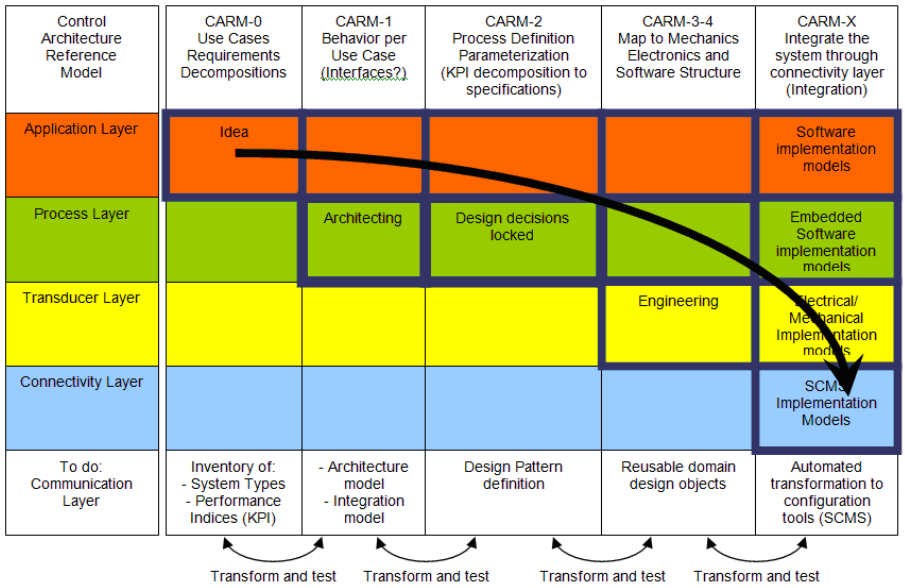


Figure 5.13: CARM Reference Layers combined with CARM Phases (I to IV) to show an ideal design process. This case study will mostly be focusing on the "communication layer". This layer should support formalizations for design information and tools for modeling and transformation.

in advance how the later engineering work must be evaluated in the context of the larger system.

At this level, we can specify what kind of behavior we want of our system. After the desired behavior is known, the system needs to be defined in terms of a set of parameters. For example, wafer diameter, desired mass of chuck, position of sensors, desired actuator precision and speed. This is where most of the design decisions will be taken, and it needs to be known how to measure the performance of the later system in terms of the key performance indicators. Of course later engineering developments will make changes in the decisions necessary, but for this figure we assume a straight line to the design specification.

The engineering phase needs to translate the abstract system behavior, structure and the requirements to engineering artifacts. They will find out through designing, engineering and modeling what physical (and digital) artifacts will together provide the desired behavior. The end result will be a design specification of the control system: a set of specifications for all artifacts of the system and a set of test results to ensure the behavior is correct. This translation of parametric design decisions to engineering models can be supported by design patterns and reusable design primitives.

It could also help to divide the design per design aspect (controllability, throughput), complementing the division per component per design discipline (chuck geometry, cable routing). Aspect oriented design leads to iterative development of the aspect towards some key performance indicator enabling verification of the key performance indicator through-

out the design process. By taking the key performance indicator measuring into account from the start of the design process, aspect oriented design could lead to a better quality of designs (less fixes, faster integration), which takes less testing on expensive prototypes, which reduces costs, and has a faster lead time. Aspects models can be constructed using the abstracted information from the communication layer.

Issue: Better Separation of Domains/Concerns/Aspects

As was discussed in the issues §5.5.1, people in the various design disciplines involved in developing control systems have different engineering backgrounds and tools. They do not speak the same language, as it were. The diversity between development of different control systems is also large (temperature control, fluid control, gas control, machine damage control) while these developments have much in common on a 'meta level'. To make design and engineering possible, the design is decomposed into more or less discrete design objects and tasks for hundreds of engineers. These objects, mostly represented in (partial) models, need to have a clear relation to architectural goals and requirements, so model results can be integrally evaluated when they are done. Design objects at ASML tend to be 'viewed' in domains, disciplines, aspects or concerns.

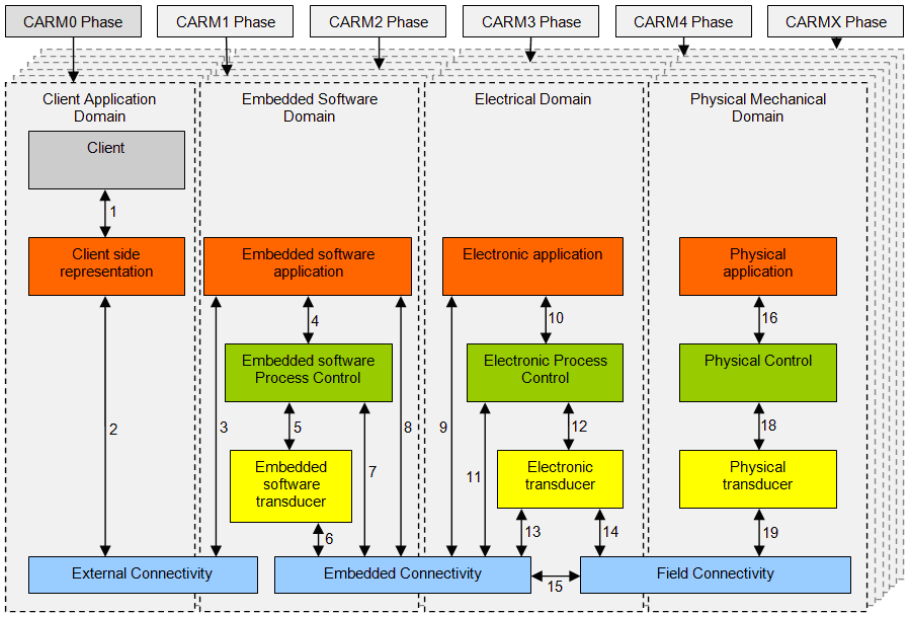


Figure 5.14: Interactions between various generic entities in the various domains. This can be a base template for the development of design patterns of Control System and reusable design objects for Servo Groups. Besides the design objects themselves, the interactions can be formalized too (arrows 1-19).

In figure 5.14 we see a conceptual generic control system design pattern, developed for this case study. It combines an abstraction of a control system and the hand overs of

the design domains in one model. This is an extension of the available CARM material provided by the company. In the gray columns, the engineering domains are represented that have ownership over that part of the control system design. In the rows, we see the abstraction level. On top we have the client application that tells the system what to do and when – coordinating with other subsystems. The second 'application layer' translates these instructions into system functions. The third 'process layer' contains the logic needed to perform the function – basically a mathematical operation on a physics model (eg. Matlab Simulink model). The fourth 'transducer layer' describes the specification of the physical components of the control system, such as CPU capacity, sensor resolution, amplifier delay or actuator i/o data. Finally, the fifth 'connectivity layer', which basically describes where the signals, electrons and photons go – from bus to bus, through cables, and from physical phenomena to and from sensor and actuator signals.

The arrows between the various blocks in figure 5.14 show where signals need to be processed from system goals to moving electrons around. For each such step, transformations need to be made – inside the system, but also in terms of design tools and models. Also, every block will have an engineering team responsible for developing it. So the arrows also represent important design interfaces to other teams.

The figure can also be seen in a temporal sense. From the basic architecture in this simple figure, the engineering process will fill in the design objects during various CARM phases, going from abstract diagrams to breadboards of electronics, thereby extending the complete design description. Each such a phase will need an integration and evaluation of the overall control system to check if the control system development is still on track. The phases were described in more detail in the previous section.

Issue: Reusability

At the mono-domain level (mechanics, electronics), people tend to dislike thinking in generalizations. While the software code for a temperature control loop is practically the same as a motion control loop, the specialists in the domains tend to deliver very heterogeneous models to the software department to develop the control software. Can we define a generalization/specification combination that both the specialist and software people can use?

Considering the amount of software-controlled units, and the number of parameters per unit (around 10000 x 10 servo groups as found in the dataset), it is clear that the software department wants to make generating this software as uniform as possible. This will reduce the number of parsers, visualizers, editors and model transformations. Therefore, efforts are undertaken to automate the hand-off between electronics/software and mechatronics/software departments. This is done by template interfaces in the hardware to software interface. If the department follows these templates, software can be generated automatically.

There is still a long way to go to implement the hand-off for all flows to the software department. Because other departments do not see the need for such a formalized hand-off, they are not that willing to participate in the project. However, software now builds

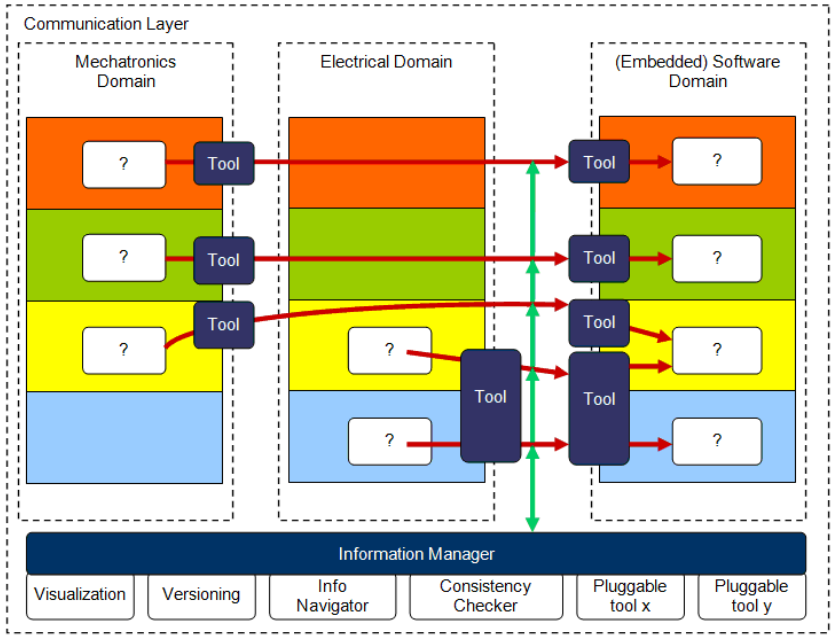


Figure 5.15: Information exchange (red arrows) between the domains, for automated model generation, using transformation tools. These transformation tools use an ASML language to capture the information from the domains. We can analyze these information flows (green arrows) in a system independent information manager framework. The tools can be integrated, and the language can be unified.

custom tools for these departments so they can generate their models more easily, and see the benefits of working in the new way.

In figure 5.15 we see the desired tool chain over the various design domains as conceptualized during the case study (in respect to the existing situation depicted in figure 5.10). In every layer and domain of the design process, models and specifications are made with specialized design tools. The figure shows a communication layer surrounding these other layers. The main proposition is to transform the information communicated between domains and layers into a shared modeling language. As such, there needs to be a tool for every possible deliverable to other departments. This seems much work, but ASML already has a lot of such tools. However, the ASML tools do not generate data in the same language. Every red arrow in the figure has a dedicated tool. A single language should also make it easier to recognize possible reuse of design information across the design domains and between the various control system developments (fluid control, wafer positioning). The development of a single language was the main goal of the next section.

5.5.5 Experiment with company data

This case study tried to define a design process pattern for a control system. A design process pattern formalizes a reusable design process rather than the design object under consideration. As such, the design process pattern should combine information about the design process, the architectural concerns and the models of the patterned system as a template for model based engineering. The question was what abstractions to choose, and the way of collecting, processing, representing and transforming data.

The design process pattern of the current situation follows figure 5.10, which shows the problem. Reusable primitives are made all over the place, shifting the benefits of reuse to more and more tooling maintenance and problem specific instead of generic solutions. The basic 'layout', however remains unchanged, combine a behavioral prescription (MAPS, §5.5.2) with the specification of physical components (SCMS and mechanical properties such as component inertia and geometry §5.5.2) to generate control software.

Roughly four options were available to make the data more insightful:

- A fully dedicated ASML language: Class definitions and tooling defined specifically for ASML
- A generic design language as specified in ACCGSMS project (eventually the architecture modeling language in this thesis). Which attempts to define a standard architecture modeling language, with a standard architecture model editor. This can be expanded with specific knowledge bases and model generators for case studies.
- Generic language (SysML/UML, stored in XML): Rewrite everything in more generic formats. No specification for the usage of the data.
- Graphic overlay on existing data: Use existing data formats, but provide some information management tooling to make sense of it.

Given the size, complexity and possibilities for reusability of data in the ASML design process, the first option was tried out, which has all the benefits of the underlying options, although from a ACCGSMS project point of view was less desirable in terms of genericity of the architecture modeling framework. The approach is represented in figure 5.16.

A big challenge for ASML is to develop a design language, which will help guide the product development. As every discipline works in their own format of data, it is very hard to manage deliverables, ownership and status of the design process. The current situation is a big 'silo' of data, which is hard to reverse engineer. This was already found out by Bart Theelen and others of the Embedded Systems Innovation by TNO (ESI) in a separate research project, the Davinci-Wings-project [Alberts et al., 2011]. ESI already tried to set up a data structure, but that was of too low an abstraction for this case study. According to the ESI project at the time, the data seems to be a set of non-coherent, configuration-specific, implicit knowledge and data roughly characterized in:

- 57 servo groups (sensor-controller-actuator combinations)
- 3937 functional blocks (functions of the system as directed by software)
- 11334 functional dependencies

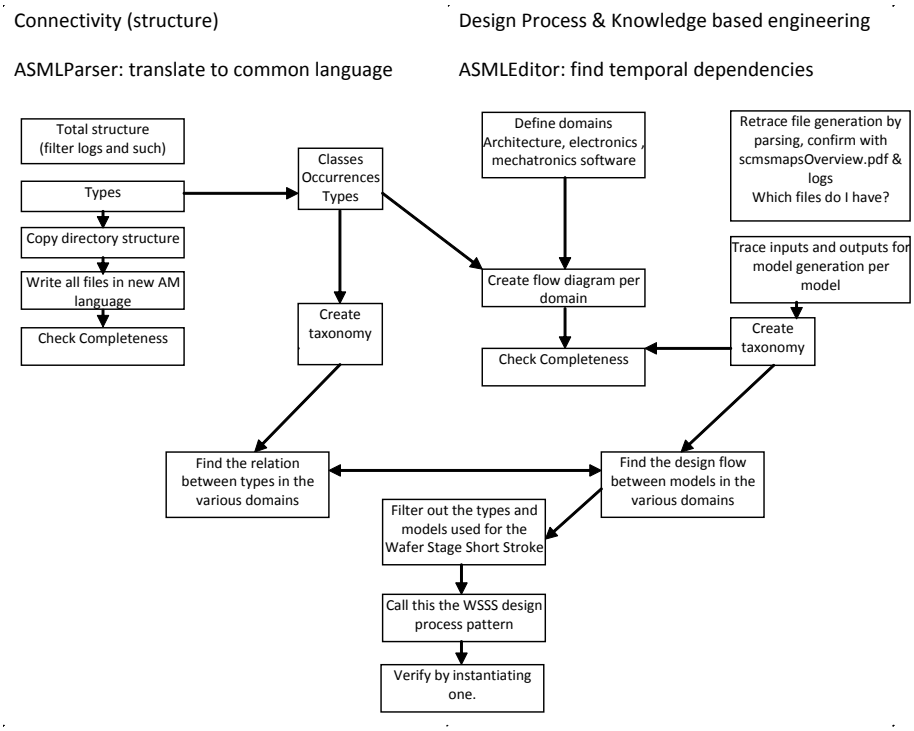


Figure 5.16: The approach chosen to make sense of the huge dataset in the Lithography System case study. This approach broke down at the first steps as a result of the variety in the data – just too much time would be needed.

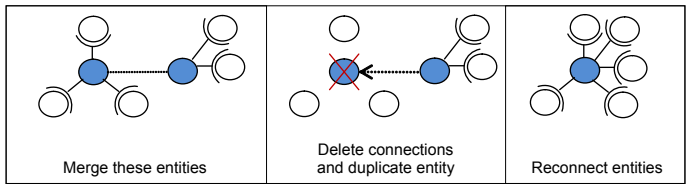


Figure 5.17: Cutting out duplicate design information to make a single integrated model.

- 250 transducers (physical, digital or logical components, transforming signals)
- 2500 transducer dependencies

In an effort to make sense of the workflow in figure 5.10, a dataset was dissected. The data consisted of the raw data needed to define the input for automatic software generation of the many actuated components of the ASML system. This construction of a design process pattern was attempted following the process in figure 5.16. By making a set of document parsers for the various data formats, and figuring out the flow between design tasks, a clear picture in design primitives should have emerged. After that, data has been integrated into a single datafile in a single data language (as schematically depicted in figure 5.17). Using this generic language, other concerns of the design process, such as depicted in figure 5.14, was mapped onto the transformed dataset. Unfortunately, parsing the dataset is where the case study bogged down:

This dataset is just too big. For example, something called the 'WSSS ddf' is about 5000 'attributedef' and 'const' parameters dispersed over 25 files (what the file does or what it represents is not discussed here). In total there are 17 file types with different formats and languages, covering 1899 files – a huge variety. Also within such files the variety is very large. For example, in parsing MAPS files we find 110619 design parameters. Those parameters have various formats, but are structured the same everywhere in about 8 types (based on the number of attributes in the statements). Almost all 110619 statements are unique, however, the types can be filtered. When filtering history parameters, numbers and dates, 4727 unique statements are left, mostly the values of the design parameters that configure the software for the lithography system.

Unfortunately, this parsing of data would take more time than the research project allows. The case study was discontinued with a tool that only partially transforms the ASML dataset into a single language (figure 5.18). However, the process of making sense of lots of data did lead to the refinement of insights of §5.5.4, as will be discussed in the next section. This means that, although the case study did not succeed in making sense of the data or to make a generic data format, the results do suggest that a concerted effort by ASML could lead to a smaller set of design primitives, simplifying the design process. Developing a shared data language to perform these analyses will be an improvement in any case.

5.5.6 Upgrade framework with new results

The issues stated by ASML were not resolved in this case study, mainly due to a lack of time. However, for this thesis, and for the AGCCSMS project in general, the case study gave a number of interesting insights, and important advancements. First of all, the issues were not related to technical problems of how to generate control code for mechatronic systems. The issues were related to how to collate the information from around the organization into something that can be used as input for code generation. This distinction is important, as it shifts the attention from modeling a system to modeling an organization's information flow. Where in the previous case study (the baggage handler of §5.4.2) this flow was well known, in this case study it was not.

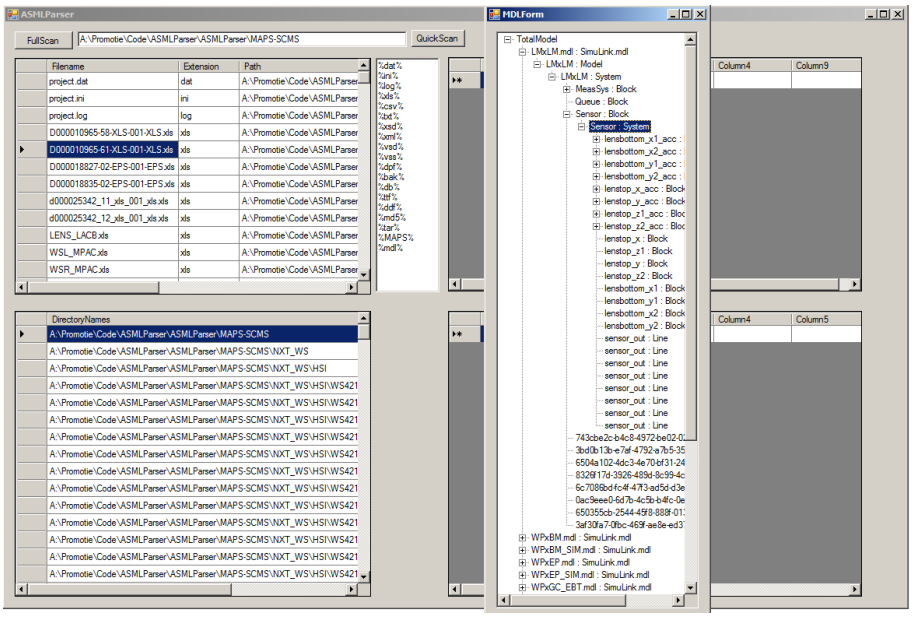


Figure 5.18: A tool was built in this case study to read all ASML datafiles at once, and try to make some connections among the files. Apart from some basic hierarchy, not much more was learned.

The main modeling capabilities of the framework thus needed to be extended. Where at the start of the case study, we had a framework that had a language of two and a half primitives (the composable object, the `Parameter` and the `Function` attribute), we now needed more information of a more 'causal' nature. This led to the introduction of (1) the `DesignTask` to separate information flow in the design process from the system specification, (2) the `Aspect` to define interrelated sets (compositions, hierarchies) of information that cuts across each other, and (3) `DomainEntities` to model hierarchies in a specific domain or discipline, so it can be separated from the more organization-wide architectural hierarchies. (4) Also, more importance was given to the boundary conditions of the design process, in the form of the `Requirement` concept.

The framework was extended with these new concepts as specializations of a new `BasicObject` class. Where all the generic information functionality was implemented on this basic class, such as serialization, representation, and the basic attributes as described in the language definition in appendix E. Also, because the size of the dataset incapacitated the existing Eclipse version of the framework, a C# version of the framework software was made to work with the data, which was then partially represented in the Eclipse tool. This gave insight in using the architecture modeling language as a platform independent exchange language between tools, which was later extended to Matlab and a knowledge based engineering tool.

5.5.7 Results concerning Consistency, Integration and Reuse

To summarize this case in terms of this thesis: Develop a shared architectural model for an integrated approach to generating control software, defining the goals and stakeholders of the design process. Integrate discipline specific information into a shared model using a common data language to keep information transformation consistent. Define a more generic 'process control' level of design primitives that can be reused in multiple aspects (optical, thermal, mechanical control) of the design process. In table 5.4 the global thesis issues are transposed on the results of this case study.

Table 5.4: Lithography System Issues and Resolutions, see appendix B

Index	Issue	Resolution
	Improve Consistency	
1	Big Picture	The introduction of an architecture model should help ASML in providing a big picture of the design process, its goals, and its resulting design specification.
2	Overview of the Design Information Flow	Design tasks can be modeled to separate the 'process of making a design description' from the 'design description'.
3	Provide Design Information in a Context	An architecture model can capture relations between design specification oriented design information and more organization wide architectural concerns and metrics.
8	Provide Traceability	Ownership of design information was attributed to all information concepts as a way to provide traceability through standard 'id', 'user' , 'date', 'status' and 'documentation' attributes.
9	Ownership of Shared Design information	See 8.
10	Consensus Making	A more stakeholder centric way of viewing information was introduced, where different design aspects or stakeholder concerns can be made explicit.
13	Automate Hand-over	A shared data language enables knowledge bases to transform information.
	Improve Integration	
15	Communication Distance	Provide an abstract language to communicate and make models across disciplines.
16	Shared Data Model	Provide an abstract language to communicate and make models between disciplines and design tools.
17	Model and Communicate Architectures	Make a model of stakeholder specific design aspects and concerns on the architecture in a shared language.
18	Multi-View System Architecture	Make a model of stakeholder specific design aspects and concerns on the architecture in a shared language.
19	Multi-Composition	Allow for discipline specific system information through DomainEntities.
20	Multi-Mapping	Allow for the discipline specific information to be mapped to that of other disciplines in a shared data model.
22	Design Process Tasks	Make a model of the tasks, deliverables, and flow of the design process with design task concepts, in the context of the design aspects.
23	Shared Data Language	The architecture modeling language was expanded.
24	Abstract and Reference Model Content	The data from 110619 pieces of design information in 1899 files was abstracted into two concept types and one attribute. This method can be repeated to come up with a ASML specific language.

Continued on next page

Table 5.4 – continued from previous page

Index	Issue	Resolution
25	Support Parametric Definition	Instantiating a control system from a parametric model seems to be possible, but was not implemented.
27	Mechatronics Design	Lithography systems are mechatronic systems.
	Improve Reusability	
29	Meta Modeling	The data from 110619 pieces of design information in 1899 files can be transformed into a much easier meta model.
30	Reference Modeling	Many of the design aspects in a lithography system are highly interwoven. A reference model can help map them.
31	Model Multi Domain Information	The information needed to generate control software was collected from three different domains, mechanics, electronics and mechatronics.
32	Defining Reusable Design Primitives	The control software is generated with reusable primitives, which need to be simplified and generalized, as there are currently too many. A generic set of primitives can potentially cover different control groups, such as motion, temperature and fluid.
33	Defining Reusable Design knowledge	A knowledge base can be envisioned that leads the user in instantiating reusable primitives across various configuration options. Also, reusable knowledge can be used to keep track of the adherence to key performance indicators.
36	Lean Models	110619 pieces of design information in 1899 files in dozens of transformation steps seems to be too much. The case study indicates the likelihood of 'slimming down' this process.
38	Platform Development and Support	The current situation is a result of numerous independent product platforms assisting in small pieces of the software generation process, even within design disciplines. These platforms need to be transformed into one shared platform, possibly founded on the CARM framework.
39	Freedom vs. Speed vs. Consistency	Currently it is easier to add new design primitives for specific situations than to find out if an existing one is available. This causes the whole software generation process downstream to become more bloated and potentially inconsistent. a better architecture and framework could help in mitigating this issue.

5.6 Industrial Case: Paper Path Design of Printer Copier

Océ focuses on production of engines for high quality printing. Generation of timing schedules for state machine control of a print engine is done by experts from multiple domains. This design task takes much time and needs to be updated for every change in the printer's design. According to this case study, often, information exchange, design knowledge and model generation can be automated. This reduces design time and increases consistency by removing the manually written intermediate documents from the design process.

Figure 5.19: A paper path in a printer copier

Océ delivered the case study where all academic participants of the AGCCSMS project came together for a unified solution. As a consequence, this case study was the most extensive, and contained elements from behavioral and architectural modeling, knowledge based engineering, and systems engineering and architecting. This case study is described from the point of view of the author of this thesis. When the other project researchers – A. A. Alvarez Cabrera and Maarten Foeken – were in the lead, that will be expressed explicitly. The main result of the case study was the architecture modeling framework in the form described in this thesis.

This case study contains the following steps:

- Formulating company issues as perceived by company spokespersons (§5.6.1)
- Analyzing the current workflow of the Océ paper path design (§5.6.2)
- Modeling the design information into a design process pattern (§5.6.3) by Alvarez Cabrera.
- Finding bottlenecks, and eliminating unnecessary model transformations in a new design process pattern (§5.6.4).
- Applying automation of design knowledge to automate the design pattern (§5.6.4) by M. Foeken.
- Testing the method with company data(§5.6.5)
- Upgrading the case independent architecture modeling framework with the results of this case(§5.6.6)
- The effect of this case study on consistency, integration and reuse, the topic of this thesis. (§5.6.7)

5.6.1 Company issues and relations to Consistency, Integration and Reuse

According to several meetings with Océ personnel, the following issues played at that moment:

- Keeping the product information consistent during concurrent development in teams of different disciplines. This is a problem of the temporal (sequence of design process tasks) and physical (other buildings) separation of the teams, as well as a difference in design tools manual formats of hand over documentation).
- Supporting the development of platforms. There is a desire to quickly design variant configurations of products.
- Managing changes of the information over time, and how adaptations in the design specification influence the system performance. Make the design tasks refer to specific requirements to make them 'measurable'.
- Providing the information for domain specific and multi disciplinary simulations, to give fast feedback on the design and provide domain specific visualizations.
- Capturing design decisions in the product information.
- Sharing product information among team members in an intuitive way.
- Providing the information to synthesize software and hardware models.
- Estimating the project progress based on the completeness and quality of the product structure.
- The structure of the information model should support the development processes over the different phases.
- Reporting the information automatically into readable documents.

5.6.2 MODEL WORKFLOW CURRENT SITUATION

Textual descriptions currently are the basic exchange format of design information in the design of a paper path, sometimes accompanied by excerpts of domain specific models. This develops into ambiguous descriptions because of the shortcomings of textual information (takes time to make, can be copied wrong, etc.). It also increases the required effort to extract the shared data, because the reader accessing the information in domain specific-models must first learn many particularities of such models and domains. Three engineers from three different disciplines work together in such a manner to provide a simulation model for a conceptual paper path, see figure 5.20. This process takes around three weeks, and is repeated as long as the paper path, topology and geometric design do not fit together and do not fit the architectural goals of Océ in terms of ten key performance indicators (such as throughput). In this manner about ten concepts are evaluated, greatly extending the lead time of the development project. This workflow was used to formulate a design process pattern (in the next section) on which experimental tooling was developed by M. Foeken to reduce the lead time.

5.6.3 MODEL DESIGN PROCESS PATTERN CURRENT SITUATION

In the original design process under study, design information is mostly mapped to a component that belongs to a physical decomposition, describing the structure for assembling the product (Bill of Material). Components in this structure are used in every document and discussion. This view of the system is then 'polluted' by directly associating to these com-

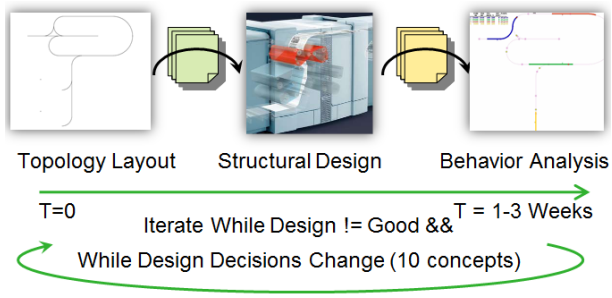


Figure 5.20: Workflow of three weeks due to manual documentation and separation of engineers in knowledge, space and time.

ponents properties that can just partially relate to them. On the other hand, the other design domains have to deal with these structural object definitions while they barely correspond to their interests and models.

According to project partner [Alvarez Cabrera, 2011], to improve communication by reducing ambiguity, the involved developers at the company can be made more aware of the existence of the different information object types and their use. In particular, they could recognize the importance of functions, the system's entities, and the design aspect requirements in the design specification. The functions are used as a means to introduce the purpose of designs (to Print A4), and the entities to refer to the product itself (Paper Heater). Requirements can be formulated by any group of developers to specify more precisely how certain concerns can be verified (Check Minimum Throughput speed).

In the case study, an intermediate step has been taken to use an architecture model to index design documentation using the existing Product Lifecycle Management (PLM) data from the company. This hierarchical composition of entities is used as the description of the system structure. Additional decompositions are added for the functionality and other aspects (e.g., safety, stability) corresponding to the views of other groups of stakeholders. We interrelated these views in a single architecture (reference) model, described in §5.6.4.

The files of the geometry model in the assembly decomposition are represented by a structure tree representation in the PLM. This structure has been taken over in the architecture model as a decomposition of entities. For the other discipline specific decompositions, each element can be related to one or more documents that detail the design specification, employing the architecture model as a reference model (method in §4.2). This approach does not imply replacing any of the existing documentation, but provides a means of indexing documentation from different perspectives. Moreover, it provides the recognition of the existence of other views on the system and their relation to the system components.

Alvarez Cabrera made this reference model, in which functions and requirements, respectively, contain qualitative and quantitative data on the paper path behavior such as the required functionality for the paper path during a single side printing operation, and the mapping identifies the function performing components in the structure. Also, the phenom-

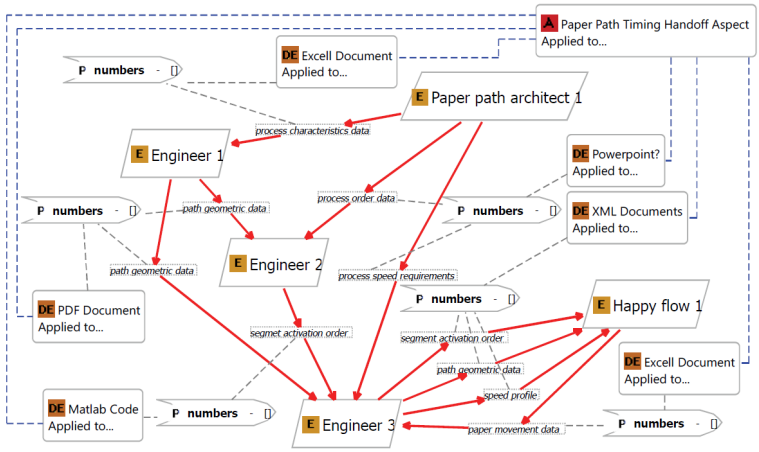


Figure 5.21: The information flow between the engineers in the paper path development. The model is made before the Design Task concept type was introduced.

ena that correspond to setting the rotation speed of a pinch is represented by two simple formulas relating a control value and the corresponding rotational speed (see figure 6.1 for a comparable model). The aim of these formulas, functions, entities and requirements is to make a clear behavior representation – through its focus on the syntax (that is, rules of construction) and context (i.e., limiting requirements and interfaces) of the model rather than on the semantics (or the meaning of the individual symbols or words). Thus, the meaning of each node in the model is specified mainly through its relations with surrounding nodes rather than only by the node itself. Together the groups of nodes form a description of the context, and the different groups in the model obtain complete descriptions of design information.

Many aspects of a print engine need the models from engineers working in multiple domains to be developed. The design aspect handled in this case study is the paper flow through the engine. Simplified, 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). In the current Océ workflow, this domain information is not gathered concurrently, but is defined in subsequent design tasks as per figures 5.20 and 5.21. At each design task, these models are made by domain specialists – who use their own domain specific languages and tools – based on constraints in a textual document (Microsoft Word, Visio or Excel mostly) supplied by the previous design task. After their models are verified against the input constraints, the model information is summarized by hand in a textual document and handed over to the next design task.

Specifically, a conceptual two dimensional CAD drawing is transformed into an input file by manually indicating which parts of the drawing belong to the paper path, where the

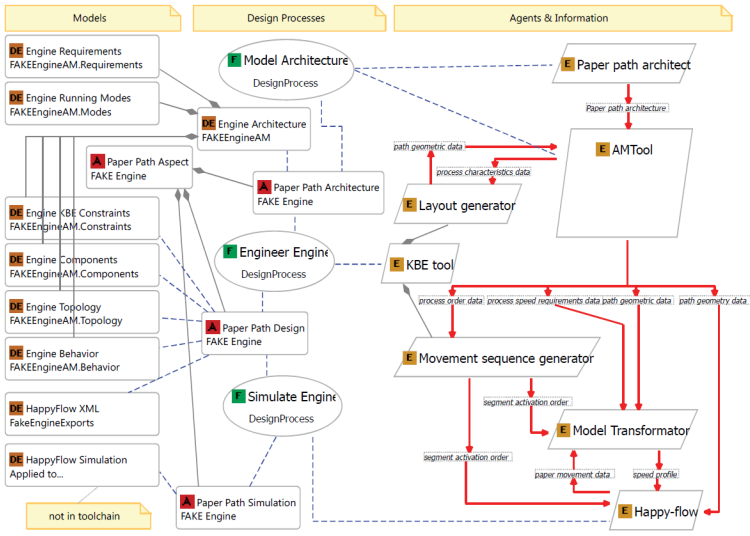


Figure 5.22: Separation of models design process and the designers in a new design process pattern for design automation. This figure was made before the Design Task concept type was introduced

sensors and pinches are located, and where the various path segments start and end. 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 transformation. Furthermore, these steps have to be performed after every significant design change. One of the desirable characteristics of a model of the architecture is that it should enable relating discipline specific data to the common model in order to share that data. Furthermore, it should provide groups of objects to represent the intended specification and the actual components of the system. Both these characteristics can be used to support, and possibly automate, the design process pattern at hand, as we shall see in the next section.

5.6.4 Model design process pattern new situation

To improve the multidisciplinary communication and integration, we introduced an architecture model (AM) that captures an abstract version of the paper path design aspect described above. We recognized 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 information flow view needed to specify the aspect is shown in figure 5.22. By making an AM of the aspect, we can 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 (as shown in figure 5.21). Because the aspect information is kept in a single, project based model, there is no duplicate information, no transcription error and an

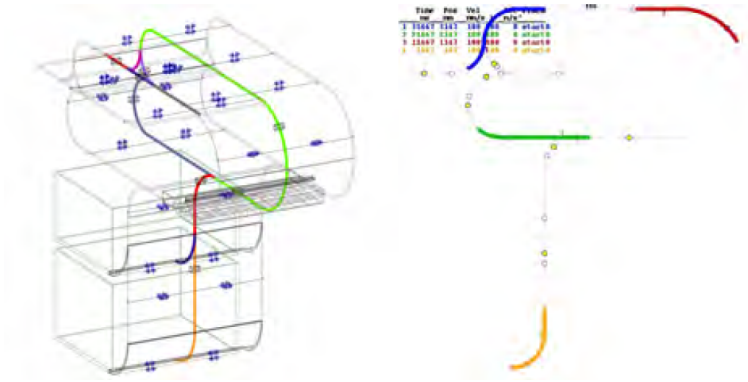


Figure 5.23: The geometric model an behavior model generated by the design process pattern and the connected knowledge based engineering tooling. Figure is censored for intellectual property reasons.

always up to date version. The approach reduces the amount of data needed to be shared considerably; no more extra manual documents, and no more multiple versions of these documents. While this aspect architecture model says nothing about the validation of the values or the knowledge inside the domain models, it structures the design process in such a way that (automatic) validation becomes a possibility, as will be discussed below.

The amount of information (number of nodes and relations) inside the (engineering) models can grow very quickly. Therefore it is advised to only model information that is exchanged amongst designers and tools. Even then, the information content of the model can be too large to be evaluated by a person alone. Therefore a mechanism was introduced to limit the information that a certain stakeholder sees based on his interests, the View concept type. 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, automated tools can automate parts of the design process of the paper path.

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 that are able to process the model data can directly use the model, allowing automated model transformations. This was explained in detail in §4.3: using the architecture model as a meta model. In this case study, the paper path design process pattern follows the transformation of data from various design disciplines to create a simulation model used to analyze the timing performance of the paper path in a printer. The deliverable of the pattern is the input for the Océ timing performance analysis tool, a Matlab application developed in-house named Happy Flow (see figure 5.23). This model 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.

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. Also, 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 not only collect and share data from existing domain-specific model, but to generate new models based on the desired function specification and the actual components of the system.

To this end, the use of knowledge-based engineering (KBE) methods was employed by project partner M. Foeken (see also [Foeken et al., 2010]). As the design rules of the involved disciplines are known, a software application incorporating these rules was created, which performs the task of engineering (composing, sizing, positioning) the system defined in the product architecture. This prototype KBE application was used for:

- Generating 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.
- Determining 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.
- Partitioning the paper path in segments, following the segment definition specific for the Happy Flow simulation tool.
- Determine segment ordering for various operation modes, which are defined using function sequences and component mappings in the AM.
- Export 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 automatically 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 5.23 shows the generated threedimensional geometry, together with the 2D paper path split up in segments. These discipline-specific segments both cross and split within modules, such that it is not possible to determine a fixed M to N mapping. The exported AM objects can be transformed into the Happy Flow specific format from within the AM tool using a plugin developed for the case study. This plugin relies on the 'knowledge' attribute – available in every architecture modeling language concept – to collect the necessary data. A snapshot of the resulting animation is given on the right-hand side of figure 5.23, with the colored lines indicating the position of the sheets within the paper path.

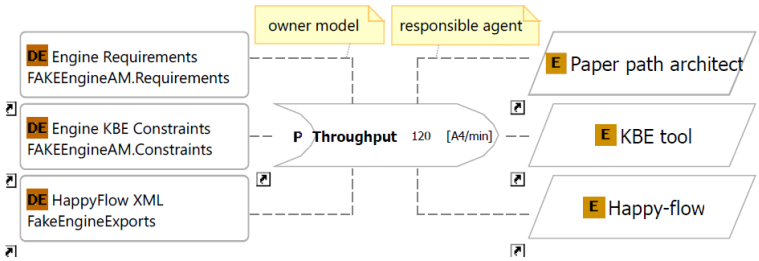


Figure 5.24: A single parameter can be shared by multiple models (left) and Tools and stakeholders (right).

5.6.5 Experiment with company data

Integration through parameterization was applied to the whole design process pattern. Because, originally, there is no clear ownership of shared parameters, there can be different values at a given point in time and ambiguity about constraints. Manually handing documentation over is prone to consistency errors coming from manual transcription of values. The manual hand-over documentation is not versioned properly; at any stage an engineer might use obsolete values. In the case study, we identified a set of driving parameters, which are used as the input for the design process pattern, and centrally stored in the AM. When such a shared parameter value (for example printer throughput) is changed, any engineer or tool can update their models 'downstream' to the new value, keeping the whole aspect concurrently verified. An example is given in figure 5.24. The owner of the parameter in this case is the engine requirements model. The parameter is stored there uniquely, while the other models reference it by shortcut (see §4.2). The paper path architect has ownership over the parameter value, so the 'KBE tool' and the 'Happy Flow tool' may not change its value automatically. Such parameter values become the variables that form the boundaries of the solution space of the paper path generation.

The architecture model graphical editor was introduced as a new means to keep track of the various types of design information. This provided a graphical overview of the design information and information flow. The architecture model (AM) that was a result of our case study was not present as a mental map among the engineers at that moment. Therefore, the problems stated above were difficult to see. Also, there was no information on the context of the design (for example: Which aspect or functionality of the system relates to the exchanged information?) Furthermore, the step by step generation and interpretation of manual documentation slows down the development process, while not adding any information to the design specification.

Now, with the AM, the stakeholder can observe/query the context of each object and take the relevant information without creating a new document to describe the context again. On the other hand, as the AM language only has a couple of concepts with limited semantics, making each description is more laborious and often less concise than when written in natural language. This forces the user to think harder about how the system actually works in the concepts of the AM language. This can be beneficial, as it encourages the user to make

a model complete (for example: no missing steps in function dependencies, no unlinked requirements/entities, defining key parameters). Another point regarding the usefulness of the AM is that, when creating a design description using the AM, it is easy to distinguish when consensus has not been achieved by the mismatch between stakeholder views, and the involved parties can iterate and refine the model until an agreement is achieved. Again, the model facilitates this by promoting sound descriptions of the context describing the interactions of the object under discussion.

It must be noted that, though the separation of the information into the concept types of the architecture modeling language makes clearer descriptions, the stakeholders involved in the tests required many attempts to effectively perform such separation, especially between the functionality and entities of the paper path system. This situation is seen more often when the model is used to describe 'low level' subsystems where functionality and implementation structure are decomposed in a very modular fashion (a paper heater heats paper). It can be concluded that though the ability to separate functionality from function performing entities is conceptually simple, some practice is needed to learn how to do it.

The Océ case also saw the introduction of the View concept type. The size of a shared AM can increase very quickly by addressing stakeholder interests due to the amount of information flowing in a complex design process. The AM figures in this chapter often do not even state the underlying parametric information exchanged between the models and tools. This is a result from our approach to separate the data from its representation. The AM is a 'flat' model, which means it is a list of elements with their references to each other as attributes, as reviewed in appendix E. As such it is very easy for a computer to handle. Stakeholders can select elements from the AM to construct their own views on the architecture. Manipulating the elements in such a view will update them through the AM data into the other views, which are contained in the same AM (Although referencing external architecture models is also possible through the same mechanism). The AM diagrams of the AM editor are a representation of objects referenced to a certain view in a human-readable form (See also §4.4 for the end result). As such, the figures in this section show filtered 'views' on the larger underlying architecture model, with elements selected to make a point clear to the readers.

While View is another concept type of – and thus owned by – the AM language, the diagrams of the models can refer to elements in other parts of the architecture model using shortcuts. This is another mechanism to keep the AM size manageable. Because multiple views can reference one element of data, the views can be used as for negotiation between stakeholders. In Figure 5.24 we see that the 'throughput' parameter is used by various stakeholders. If the Happy Flow simulation stakeholder would conclude that the throughput value is unattainable, he or she could find the references to other stakeholders through the attributes of the 'throughput' parameter. He can then start a trade-off meeting with those stakeholders, to see if the value may be changed. This mechanism can thus be a starting point for impact analysis of design changes. In the case study we facilitated views to 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 pro-

cesses) and structural (components, decompositions) views on the system, as this relates to the daily work of the engineers.

Knowledge based engineering (KBE) was introduced by M. Foeken to work with the AMs to make them 'executable'; the KBE tools 'know' how to work with the information in the AM. As the KBE application was implemented in a common 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 figure 5.23).
- 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 can be automatically added. For example, add a heater between the paper input and the print pinch of the machine to make the image stick to the paper.
- 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 required for a certain domain specific model and transform it into a model specific 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. The happy flow model generated in this case study is an example of such 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 can be marked with a KBE specific value in its 'knowledge' attribute.

This case study relied on a specific commercial KBE modeling platform. However 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 engineering trends showing an increasing attention to formalized knowledge retention in the form of custom software tools to support or automate routine design and engineering tasks, the capability to capture the decisions and result of creative tasks, such as developing the product architecture, in computer readable models is of great importance (as was discussed in the problem definition of chapter 2).

Finally, Guy Stoot, the Océ systems architect working most closely with the academic partners is now using the Architecture modeling tool to design and develop his Yoga studio. This indicates a possible usage of the AM framework well beyond the design of print engines.

5.6.6 Upgrade framework with new results

The architecture modeling framework got most of its current functionality as a result from this case study. Almost all concept types of the AM language were introduced in the case as a result of combining systems architecting and engineering concept types with behavioral modeling from the Function Behavior State (Structure) types and Knowledge Based Engineering concepts such as object orientation and parameterization.

The architecture modeling tool was also deployed, as a means to construct, serialize, visualize and otherwise use the AMs. Especially the View concept type was an idea that evolved out of the hands-on approach of building the tool. As this tool is independent of the language, it can be seen as one of the extensions of the framework. Another extension is the integration of a KBE tool into the tool chain of the design process pattern without any of the academic partners having both the source code for the editor tool and the KBE tool. As a conclusion of all the upgrades of the framework, the reader is best referred to reading chapter 4.

To reflect, we shall revisit the original company issues from §5.6.1, and describe how the case study has approached a solution:

- *Keeping the product information consistent during concurrent development in teams of different disciplines.* For the case study, a partial development process of the paper path was captured in a design process pattern. The paradigm of the design process pattern has become one of the new concepts resulting from this research (§5.3).
- *Supporting the development of platforms.* By employing model based systems engineering (MBSE), this can be facilitated. However, the tooling had to be built first, and cover design information from multiple disciplines. The print engine customer requirements, the engine's printing modes, the engine's paper path topology and geometry, and the engine's dynamic behavior were made configurable. The Aspect concept was introduced to the AM language to group such information (§E).
- *Managing changes of the information over time, and how adaptations in the design specification influence the system performance. Make the design tasks refer to specific requirements to make them 'measurable'.* This was done by automatically generating and evaluating the design description of the print engine and its paper path behavior, using a Knowledge Based Engineering (KBE) tool by M. Foeken. This demonstrated the extensibility of the AM with external KBE tooling.
- *Providing the information for domain specific and multi disciplinary simulations, to give fast feedback on the design and provide domain specific visualizations.* This was tested by making a design process pattern of the printer paper path. For general overview of the printer design and architecture, the architecture modeling tool was introduced (§4.4).
- *Capturing design decisions in the product information.* In the case study such design decisions for the paper path were collected and used as configuration options in a MBSE tool chain. The parameter class of the architecture modeling language was used to capture the design decisions. Modeling design decisions as parameters at the input of the design process pattern was a prerequisite of using the AM as a Meta Model for model generation (§4.3).

- *Sharing product information among team members in an intuitive way.* A multi view editor was introduced that allowed stakeholders to make their own aspect models within the (integrated) context of the larger architecture model (§4.4). In the case study, the different design tasks of the tool chain got their own view.
- Providing the information to synthesize software and hardware models. The paper path design process pattern of the case covers all information en synthesizing knowledge to build a 'happy flow' model, which is used to generate software automatically with existing Océ tooling.
- *Estimating the project progress based on the completeness and quality of the product structure.* An architecture model was introduced that captures the product structure as well as the requirements. This enables an estimation of project progress and completeness and quality.
- *The structure of the information model should support the development processes over the different phases.* In the case study, various views were introduced for various stages in the design process pattern. The View concept has become an essential part of the AM language (§4.2).
- *Reporting the information automatically into readable documents.* With predefined reports it should be possible to generate the currently used documents. This was not implemented. However, the architecture model can be used as a meta model for design documentation in the same way as it is used to generate engineering models, using specialized knowledge bases. Also, the architecture model can be used as a reference model to various other, discipline specific, information sources (§4.2).

5.6.7 Results concerning Consistency, Integration and Reuse

To summarize this case in terms of this thesis: Define a shared architectural model for an integrated approach to design a conceptual print engine structure and behavior model. Base the architecture model on the views and concerns that the various stakeholders in the design process have. Integrate discipline specific information into a shared model using a common data language, and exchange that information with automated Knowledge Based Engineering tools to keep information transformation consistent. In table 5.5 the global thesis issues are transposed on the results of this case study.

Table 5.5: Paper Path Issues and Resolutions, see appendix B

Index	Issue	Resolution
	Improve Consistency	
1	Big Picture	An architecture model was introduced to model divergent stakeholder concerns in views that are integrated in a singe model. These views can be interrelated to provide a big picture of the overall design information,
2	Overview of the Design Information Flow	The Views are causally related through 'dependency' relations. Later the design information flow was made concrete with the Design Task concept type.
3	Provide Design Information in a Context	See point 1.

Continued on next page

Table 5.5 – continued from previous page

Index	Issue	Resolution
4	Prevent Ambiguity	The shared model enables the representation of a single piece of information in various views, instead of copying that piece, therefore preventing ambiguity of that information. Also the architecture model itself can be used to create a shared understanding between stakeholders, thus preventing ambiguity.
5	Check the Requirements	Requirements can be connected to various design aspects to begin a review process (can be automated as in this case's generating a happy flow model)
7	Modeling Design Decisions	Design decisions were captured in parameters, and evaluated automatically by a knowledge based engineering tool.
9	Ownership of Shared Design information	The ownership of design parameters for multiple engineers in a workflow was determined and formalized in a shared architecture model.
13	Automate Hand-over Generation	The hand-over of design information between multiple engineers and tools was formalized in an architecture model and automated in a knowledge based engineering tool.
14	Do Not Duplicate Information	The shared information between between multiple engineers and tools was formalized in a single architecture model, thereby storing the information uniquely.
	Improve Integration	
15	Communication Distance	Communication distance of the design process pattern was reduced from three engineers in three disciplines over three weeks to a single model and an automated tool that performs the same task in several seconds.
17	Model and Communicate System Architectures	The architecture of a paper path was captured by repeatedly communicating with the industrial partners and updating the architecture model and design knowledge.
18	Multi-View System Architecture	Multiple views were needed to provide an overview for this case study.
21	Capture Design Decision	Design decisions were modeled and put in the first stage of the design pattern, in order to generate behavior and geometry models from them.
24	Abstract and Reference Model Content	The architecture model was used as a reference model for generating a happy flow model by an external knowledge base.
25	Support Parametric Definition	The knowledge base worked with parametric information from the architecture model.
26	Support Parametric Integration	The work of multiple engineers was formalized in a single integrated architecture model.
27	Mechatronics Design	The paper path of a print engine is a mechatronic system. The happy flow model that was the deliverable of the design process pattern is the input for embedded software generation.
28	Align to Functions	The functions along the paper path and the functionality of various printer running modes (print A4, duplex) were used as the input to generate system geometry and topology, and system behavior models.
	Improve Reusability	
29	Meta Modeling	The architecture model was used as a meta model for the paper path design process pattern.
31	Model Multi-Domain Information	The work of three engineers of three design disciplines was captured, formalized and automated.
32	Defining Reusable Design Primitives	The design primitives were the segments of the paper path, the functions along this paper path, and the different running modes as behavior primitives.

Continued on next page

Table 5.5 – continued from previous page

Index	Issue	Resolution
33	Defining Reusable Design knowledge	A knowledge based engineering tool was implemented to configure the print engine topology and behavior from the primitives using a set of constraints (design decisions).
34	Optimization	By automating the way to generate the happy flow model, the design process became much shorter, enabling fast feedback on design concepts, thus enabling faster iteration to a more optimal solution.
37	Use Existing Tools	The happy flow model was the only formal model, and was thus used as an end point for the case study. The other tools were not engineering tools, but documentation tools (Adobe Reader, Microsoft Excel) and that documentation could easily be replaced with a central shared architecture model.
38	Platform Development and Support	The design process pattern is not print engine specific, and the design primitives and knowledge bases can be extended. Together, this provides the means to quickly configure various system concepts on a single platform.
39	Freedom vs. Speed vs. Consistency	The speed and consistency were increased by having a single design document, and formalized rules for model configuration and transformation. The freedom remained about the same, as the tooling can be extended. Also, the freedom of having a fast tool can translate into trying more design options.

6. Discussion

This discussion chapter will reflect upon the Architecture Modeling Framework. How can it help in the issues facing multi disciplinary design? We will look at user experience and external literature sources, and compare the Architecture Modeling Language to other languages to find this out.

Up to now, the thesis has formulated an answer to some major issues in multi disciplinary design. From stating these issues based on literature and interviewing in chapter 2, to finding available methods and tools to mitigate the issues in chapter 3, to the author's addition to these methods and tools – the Architecture Modeling (AM) Framework, to the testing and development of the framework in an industry–as–a–lab setting. In this chapter we will discuss the efficacy of the proposed framework, given three perspectives.

- The experience of the Architecture Modeling Framework users. Discusses the usability of the tool, by an industrial partner, by the focusgroup in a workshop, and by the author's experience, both in the case studies as in other applications. This will deduce if the framework is a workable tool for users.
- The AM framework as an answer to multi disciplinary design issues perceived by third parties. Discusses issues formulated in literature, and how the framework can mitigate these issues. This will deduce if the framework is a useful tool for tackling multi disciplinary design problems.
- A comparison between the Architecture Modeling Language and Object Oriented modeling languages SysML and UML. Only SysML/UML will be compared to the AM language, as this way of modeling is popular, and is notably dissimilar from the AM language at a fundamental level, while aiming at overlapping goals.

Section 6.4 will reflect upon using industry-as-a-lab research method as a valid means of conducting research. In the conclusion, chapter 7, we will discuss the usefulness of the architecture modeling framework in improving consistency, integration and reuse – the topic of this thesis.

6.1 User Experience with the AM Framework

The Architecture modeling framework was used extensively by the research project team to implement the various case studies (chapter 5). The framework provided a tool to perform analyses and modeling techniques that could not be delivered by other existing tools. For

the project team, the AM framework, thus was a great success. Of course, the observations of the project team do not represent am unbiased view on the tool we made ourselves. Therefore, other users were reviewed. The first basis for an unbiased opinion was the workshop discussed in appendix C. This workshop provided the opinion of experts in various fields of high tech engineering, from managers to engineers, to architects.

The general opinion was that an AM framework type of tool can be useful in an industrial setting. Abstract modeling, in a formal way, in a single model, can help develop an integrated model between stakeholders. This has a possible benefit over disparate models in MS office products and other specialized modeling tools. The concepts of the AM language were considered a novel combination, providing a wide range of modeling capabilities. The addition of parametric information was perceived as a powerful way to express system drivers.

Limitations were perceived to be the lack of adding content such as figures and text, instantiation of a meta model into system models (as in object oriented UML), and the development of parallel scenarios based on the same AM model. The tool itself was to considered too much of a prototype to be used in a live engineering environment.

Note that two other workshops were given, but in the first, no prototype framework was present yet, and in the second, only one participant used the framework. The findings of these workshops are therefore not added to this thesis.

The second independent opinion comes from Guy Stoot, who used the tool in his job as a systems architect at Oce. A short interview was conducted a year after the research project ended to review his experiences, see appendix D. He tested the framework extensively, and concluded that he still uses the AM language. The AM tooling is not used anymore, as this was not considered good enough for a production environment. Instead, he implemented the AM language in MS Visio.

These opinions express an overall positive evaluation of the AM framework. It provides for industrial needs not filled by other tooling. However, the prototype tooling is not considered good enough for a production environment. Some of the main shortcomings will be discussed as future research in chapter 8.

6.2 The AM Framework as an Answer to Multi Disciplinary Design Issues

The literature review in this chapter spans a number of important areas of research related to Model Based Systems Engineering and other design activities needed in multi disciplinary design. From the aforementioned tools and methods, we have deduced a number of recurring concepts that we can use in an architecture modeling framework. Also, some requirements for improving consistency, integration and reuse were mapped to the various literature topics, and, where possible, related to existing methods.

What is interesting for this thesis, however, are not the tools and methods that already exist, but those that are still lacking – at least as far as they relate to the hypothesis: The consistency, integration and reusability of multi-disciplinary design processes can be improved by facilitating communication between stakeholders, modeling architectural concerns and other important design information, and facilitating capturing and reusing design knowledge across disciplines. In this section, we look in more detail to three sources who express concern in the way state of the art modeling techniques have shortcomings. This extends the author's own observations from the state of the art gaps (3.7, with external sources. First, [Muller, 2011b], who describes a set of gaps in the integration of abstract models of systems and their design processes. Second, [Torry-Smith et al., 2011], who describe the state of the art of issues in multi disciplinary design. Third, [Reichwein and Paredis, 2011] discusses the practical issues in applying Model Based Systems Engineering and Architecting. For these papers, a short analysis will be given how the AM framework may assuage these shortcomings. These three sources thus

Gaps in System Architecting

In [Muller, 2011b] we find figure 3.4, which shows important gaps that we want to make a model of in an architecture modeling framework, as it relates to the communication between stakeholders:

The multi disciplinary gap is the gap between product specification and detailed design decisions. This gap originates from the growth of organizations needed to develop complex products. As people specialize to perform the various design tasks, they will need their own tools, languages and education. This gap we attempt to close by a shared integrated model that models architectural concerns and other important design information from the multiple design domains and disciplines. The AM framework is especially suited for this purpose. It abstracts a way from engineering specific data, but only up to a point: it can still be referenced and modeled by the various concepts of the AM language. The design decisions can be expressed in parametric data, also modeled in the AM language (§4.3).

The context gap is the gap between stakeholders and product specification. Stakeholders can be people within the company, but can also be the customer, or legislative bodies who want different things from the product, such as profitability, safety, renewability or manufacturability (see also the problem definition 2). The AM provides stakeholders specific views on a shared and integrated model. The Views concept provided in the AM language enables any stakeholder to model his or her views, and integrate it with the views and the concepts used by other stakeholders. As the other concepts of the language describe the functionality, embodiment and design aspects and models of the product specification (design specification), we are able to close this context gap (§4.2).

The marketing gap is the gap between the detailed outside world with billions of individuals and our abstracted understanding in terms of stakeholders, concerns, and needs. The concerns, and needs, of the stakeholders can be modeled in AMs in terms of design aspects and requirements. However, the validity of these concerns for the 'billions of indi-

viduals' – as they are modeled by only a few persons within the design process – is out of scope of the thesis (see §2.9 for reasons).

From this short analysis, we can observe that the AM framework is able to bridge important gaps mentioned by Muller. The AM language is especially tailored to model the interconnectedness between abstract architectural concerns and context, the systems engineering of the design specification and design aspects, and the detailed domain specific parametric information.

Challenges in Mechatronic Design

In [Torry-Smith et al., 2011], see table 6.1, we find a meta analysis of various methods and tools used in mechatronics design and the perceived gaps and issues addressed in them or left open. Thus, this paper supplies us with some 'soft' metrics for the missing functionality in existing tools. We can compare the AM framework to these points to see if the framework can mitigate some of these points.

Table 6.1: Challenges in (mechatronic) design, adapted from [Torry-Smith et al., 2011].

Product	
A	Lack of a common understanding of the overall system design
B	Difficulty in assessing consequences of selecting between two alternatives
C	Lack of a common language to represent a concept
D	Modeling and controlling multiple relations in the product concept
E	Being in control of the multiple functional states of the product
F	Transfer of models and information between domains (expert groups)
Activity	
G	Synchronizing development activities
Mindset	
H	Different tradition within the domains for how to conduct creative sessions
I	Reluctant to interact with engineers from other disciplines
J	Different mental models of the system, task and design related phenomena
Competence	
K	Lack of common language to discuss freely at creative meetings
L	Education within disciplines do not call for integration in professional life
M	The nature of design is different
Organizational aspects	
N	Product complexity affects the organization complexity
O	Knowledge transfer between domains is inadequate
Other Aspects	
P	Lack of a broadly accepted methodology
Q	Mechatronic ownership is lacking
R	System engineers are lacking detailed information of the system
S	Complexity as a generic problem

The first set of points in table 6.1 concern the issues in modeling the product or system from multiple perspectives (6.1a, 6.1c, 6.1d). In AM models, various stakeholders can share

their views on the design from their own perspective or domain, while still keeping the information connected §4.2. The AM Language §4.1 can relate most concept types to each other, enabling multi aspected representations.

Applying a single method throughout the design process is also supported by the AM framework (points 6.1b, 6.1c 6.1g 6.1p 6.1k 6.1r). An initial AM can be a conceptual model, that can then be expanded upon, and finally used as a meta model for the generation of more detailed design specifications §4.3. The semantics of technical details and behavior are not prescribed by the AM language definition – only the basic compositions, mappings, sequences, flows and parametrics can be defined, after which an expert or a knowledge base can actually conceptualize the model with symbols that have semantic meaning. In other words, the AML can make a model of an entity and name it a 'wheel' but the AML does not 'know' what a 'wheel' is. That this method works to represent all kinds of system concepts is demonstrated in the case studies chapter 5. The AM language allows to model what aspects of the design need to be analyzed against what requirements. Two AMs can then be compared for performance, allowing selection between two design alternatives. How to conduct such an assessment is out of scope of this thesis.

Transfer of models and information between domains (6.1e, 6.1f, 6.1g) is facilitated by abstracting the models to a parametric definition inside an architecture model (AM) and defining model transformation knowledge bases that use the AML as a shared language. See the baggage handler 5.4 and paper path 5.6 case studies. The information hand-over and goals and design tasks can be modeled in the AM. This makes it possible to at least define what design activities need to be synchronized. The development activities themselves fall out of scope of the thesis. This mechanism can also be used for more domain specific modeling. Demonstrations of this in the form of (embedded) control architectures we find in partner researchers [Alvarez Cabrera, 2011] and further tried out in a master of science thesis [Yuan, 2011]. In these works, the authors explore how to model system behaviours based on multiple functional states of the system. An example of such an AM model is shown in figure 6.1.

The previous points all focus on 'what' can be modeled. However, [Torry-Smith et al., 2011] also focus on the more cultural and organizational points of system design (6.1h to 6.1s). These are interesting points, that are often not discussed in proposed methods. As the AM framework was developed in an industry as a lab setting §5.2, many of these points were considered as part of the research project. The AM language is meant to abstract domain specific knowledge to a shareable level. The AM represents the information in views per domain, or per any other design aspect stakeholders deem necessary.

[Torry-Smith et al., 2011] states there are no tools that can assess several properties from various domains in a single model, because there is no tool or method that can cover everything, collaborators should be willing to work with ill defined models, and 'fill in' their knowledge with discussions, meetings and workshops in natural language. The AM can be used as a brainstorming / mind mapping tool to quickly conceptualize ideas and facilitate interaction of various stakeholders, or used to capture decisions at meetings (as was demonstrated in a workshop in appendix C). Mental models inside the stakeholder's head are not formalized in a model per definition, however the AML provides generic concept types,

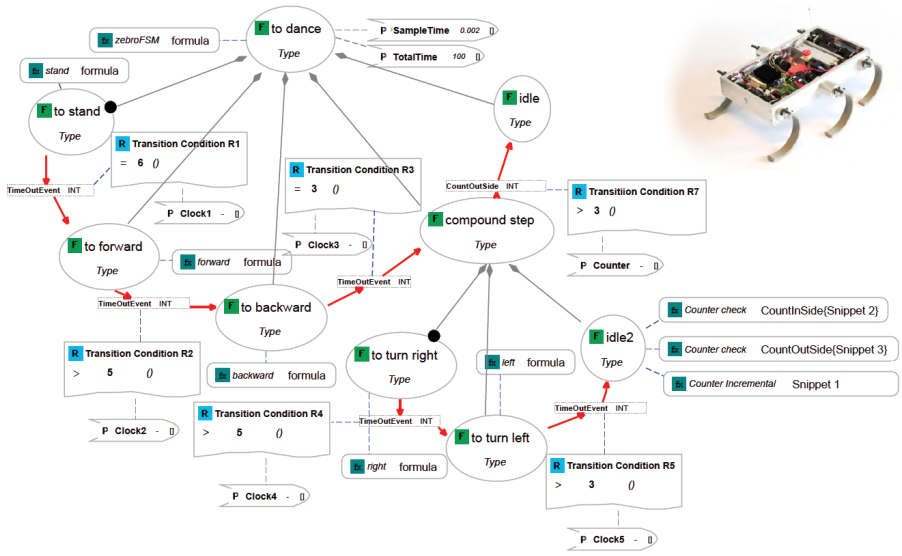


Figure 6.1: Using the Architecture Modeling Language to bridge the gap between an architectural description and a state transition behavior model [Yuan, 2011].

such as aspects, functions and design tasks, that could be used to transform such a mental model to a more formal model, which can then be communicated to others. The amount of viewpoints or goals of views are not specified in the AML, and need not be in any prescribed semantic or symbolic format, which offers enough freedom for such tasks. The case studies demonstrate the value of the AM framework in making more formal engineering models (chapter 5). However it should always be considered the AM language and AM will add extra overhead to the tasks of engineers, if the AM is not used by them directly (as a meta model 4.3 or a reference model 4.2). So the usage of AMs will not always be worth the effort of making them.

System engineers are often lacking detailed information of the system, according to [Torry-Smith et al., 2011]. This is partially a matter of supply and demand of information. A system engineer needs to be able to find the supplier or owner of the information, who can then supply the information in the appropriate format (information pull). The alternative (information push), having a repository of all details of the system, filled by information suppliers whenever they come up with new information, seems unattainable. In the AM language, the ownership of information is automatically stored on every object in the model. When more details are needed, the owner can be found and asked for clarification. When the details become more important than details, they can be conceptualized in the AM, and even formalized in a knowledge base that will perform some task automatically, as can be seen in the paper path case study 5.6.

Ownership of the overall mechatronic, or integrated, design is often lacking because of the existence of the classical mechanical, electrical and software departments. This seems a matter of organization of the design process and the people in it. As such, it is out of scope of the thesis. The ownership will be different for different products. However, especially for mechatronic systems, a more behavior-based or service-based decomposition of a system seems appropriate in contrast to the often seen structure decomposition approach (Bill of Material). Such a behavior/service view is historically more a software domain, however, as can be deduced from the literature in chapter 3, it is now developing into a systems architecting and engineering discipline of its own. (systems engineering has been around for years, only the number of organizations that need to do systems engineering to manage complexity grows). Functional views are usable as a 'skeleton' model for behaviour on which to integrate other views. In the case studies, the functional decomposition is used as a backbone for multi disciplinary model generation in 5.4, and for the system behavior, for which the multi disciplinary models are generated with a functional flow as an input 5.6.

[Torry-Smith et al., 2011] further note that after decades of research and applications, there still is no single common language to use in the integration of design and engineering disciplines. Probably, efforts in generic modeling (such as SysML for systems and Modelica for physics) will be commonplace eventually, as growing system complexity will necessitate a more synergistic approach to system design. As more complex systems are developed, so the organization complexity grows, which is mainly a result of the communication between the various stakeholders and disciplines involved in the design process (see the problem definition 2). The AM language is a synthesis of many existing modeling concepts (see appendix E), combining a focus on the design process as well as the system under design, and thus can further the goal of a single language. On the other hand, it can also be viewed as an additional language, blocking the goal of a single language. In the future, education could drive the change in mindset away from discrete mechanical, electrical and software disciplines and towards systems disciplines, such as mechatronic systems.

Architecture Frameworks and Languages for MBSE

A third source of gaps in current practice and literature is a review paper by [Reichwein and Paredis, 2011]. In this paper, the authors discuss the current state of the art in model based systems engineering and architecting (MBSE). It focuses more in the capabilities and depth of MBSE modeling techniques.

"MBSE should focus on improving decision-making during the system design process by ensuring the consistency between heterogeneous models . Models can differ in abstraction level and in domain focus. One approach is considered the 'central' model approach and consists of a system model spanning across multiple discipline-specific views and ensuring consistency among them. [Reichwein and Paredis, 2011]". The AM is to be used as a reference model for decision capturing and making throughout the design process as both a meta model (§4.3) and a reference model (§4.2). Parametric abstraction of the (input and output of) external models enables the control of the exchanged information between design tasks. Metrics can be added in the form of requirements to test the input and out-

put values of executed models. Note that the internal consistency of those heterogeneous external models is out of scope of the thesis – the AML has no pre-described (set of) ontologies for any external conceptualizations (§4.1). However, the case studies in chapter 5 have shown that the architecture modeling framework can be used to capture semantics used in a design process in knowledge bases, and the ontologies of the reusable modeling primitives in the concepts of the AML. The paper path case 5.6 is a good example of a piece of multi disciplinary design process that was captured and automated in this way.

"A different challenge in MBSE consists of efficiently creating a detailed system architecture based on a high-level system description, possibly only containing a set of requirements. This generative process can be performed through model-driven development. This technique consists of capturing design knowledge in model transformations which can then be invoked on initially abstract models to automatically create more detailed models or artifacts such as software code or documentation, thereby ensuring consistency throughout the design process. [Reichwein and Paredis, 2011]" The thesis of project partner [Alvarez Cabrera, 2011] discusses such a mechanism with the AM framework in the case of control architecture generation. Furthermore, in the baggage handler case 5.4 we specify functional requirements and a simple topology to generate the model input for four different domains. The paper path case 5.6 shows that a set of about 15 technological constraints and some decisions on the topology and behavior of a print engine are enough to generate much of the systems geometry and behavior specification. [Reichwein and Paredis, 2011] Specifically mentions the model transformation from UML to software code as a problem. This seems to be a question of modeling classes in UML, while the designer is more interested in software instances and activities. This mismatch in modeling activities and objects on one hand, and classes on the other, is further discussed in §3.6 and §6.3.

"Before starting to model a system, designers need to consider the cost and benefits of modeling in order to decide which system aspects need to be modeled and at what level of detail. Designers are faced with a similar dilemma in system space exploration by having to choose between the creation and evaluation of many low-fidelity architecture models or only few architecture models with high-fidelity. [Reichwein and Paredis, 2011]" The mentioned concepts can be modeled in the AM, however, the exploration of the appropriate level of detail for multidisciplinary design processes is out of scope of the thesis. One can envision that as more integrated models are developed, the knowledge of the common architecture of the systems is increased (an observation also made by Océ architects, who mentioned that their print engines are ultimately governed by a set of only about ten parameters, dubbed the diamonds). As the knowledge of the common architecture is increased, more stable models can be developed.

"Another challenge in MBSE consists of not only modeling single system architectures but a family of system architectures.The idea is to easily reuse knowledge about an application domain for the creation of similar system architectures related to similar business needs. In general, a reference architecture, also called a base model, is used to represent the system aspects which are common to all system variants. However, even a family of architectures has to evolve over time according to new business needs. Changes may then quickly turn an initially cohesive set of separate base and variation models into an unmanageable set

of disjoint models similar to the problems faced in aspect-oriented techniques. [Reichwein and Paredis, 2011]" Product families have not been considered directly in this thesis. And it would be interesting to know how the evolvability of such AMs could be ensured. However, in the AM framework, this is facilitated by providing a single architecture modeling language, and a single meta model that can capture the different views needed to describe the architecture. Variations in configuration to describe architecture families are not considered directly in the AML, however, the baggage handling case study in 5.4 does consider generation of products from a number of different configuration options.

6.3 The AM Framework Compared to SysML/UML

UML and SysML are two popular modeling languages. They are described in this thesis in section §3.6. In this discussion, we will examine the differences between the Architecture Modeling Language and UML/SysML. No comparison will be given in the efficacy of the two languages in the practice multi disciplinary design, as not enough experimental data is gathered for such a comparison. This will be an interesting point for future research. As was seen in §3.6, many of the language concepts seem to overlap between SysML and AM language, so this section will discuss the main differences. For a comparison to other languages and modeling tools, the user is referred to chapter 3 State of the Art.

UML/SysML have specific views for specific model content (class structure, process diagram, instance diagram), whereas the AM language models all types of information in all views concurrently. The AM therefore focuses on the interconnectedness of the information concepts. AM language can visually couple a flow of a process to a structural component, for example, or model the parametric values of a requirement, and the connection of those to certain design aspects. This gives users the tool to model their information in a single view.

UML/SysML base their models on the definition of a system class structure, which can then be instantiated into an instance of that system (which does not work very well in a UML/SysML editor such as Magicdraw [No Magic Inc, 2013]). AM language does not use instantiation, it models the instance of a system and its connectivity to other design information directly.

UML/SysML often uses encapsulation to specify ownership or containment of objects, parameters or attributes within a class. In AM language, we chose to externalize this information. First, because those objects/parameters can contain important design decisions or information in their own right, and need to be and modeled, owned, and managed individually, not as a part of a 'parent' object. Second, because interfaces between objects can be so networked that no single encapsulation would be appropriate – multiple stakeholders would have multiple preferred encapsulations (for example, a 'throughput aspect' view would encapsulate objects cutting across a 'component decomposition' view.

UML/SysML model processes as object's methods in runtime (Sequence diagram), and see processes as a thing a single component does (again encapsulated). In AM language, the

order and composition of system functions and processes run across multiple components or systems, and can therefore not always be encapsulated directly by a single object. This is the main difference: AM language making views based on the multi mapping of relations between things in the system and the system behavior (or the design process) is contradictory to the basic object oriented paradigms of UML/SysML.

Whether one approach or the other is better is not clear. It is different. For software development, the instantiation mechanism of classes to objects may be more appropriate, but for systems engineering it may be not. The choice for a specific modeling language is less important than its goal: communicating the goal and status of the design process, and reducing complexity resulting from working with a large number of stakeholders. So the language or tool cannot solve anything by itself, it is a facilitator for communication, which is a task of the organization. However, as was seen in the conclusion of §3.7, the AM language specifically facilitates bridging certain gaps between the single paradigm approach of functions vs requirements vs component decomposition vs (design) process flows, combined in interconnected views – making architecture models suitable as a backbone of the design process.

6.4 Validity of the Framework

To find a valid answer to the issues stated in the problem definition 2, a process of industry-as-a-lab was employed to test and validate the framework development. This approach was chosen important for academic reasons. To do a classical experiment, multiple companies should design a complex system with and without the framework, during which we could gather some metrics to compare the results. Apart from the obvious high cost of the experiment, the resulting metrics would not be comparable (reproducible) between companies, as they have different original design processes. However, Potts [Potts, 1993] states the importance of including the real world in the research as opposed to the controllable experimental setting – industry-as-a-lab – as expressed in the following points:

- *"Greater reliance on empirical definition of problems. It is no longer sufficient to rely on your intuition or anecdotes as a practical justification for research. Empirical observation of real projects becomes a legitimate focus of research in its own right."* This problem definition was distilled out of recurring interviews with industrial partners and academics, and literature.
- *"Emphasis on real case studies. Real system development projects become the main vehicle for conducting research. Not merely a way to affirm the value of the research afterwards."* The development of the framework was done in an iterative process of industrial case studies, scientific paper production, academic discussions and prototype building.
- *"Greater emphasis on contextual issues. Interactions among technical and nontechnical factors become part of the subject matter of the research, not an unwelcome complication for technology transfer."* The interaction between the industrial partners and the researchers in the 'AGCCSMS' project actually led to a number of important insights that have to a great extent shaped the end results as stated in this thesis.

For the validity of the research, we will look to definitions of Shadish, Cook and Campbell [Shadish et al., 2002]. They define the types of case studies conducted in the AGCCSMS project as quasi experimental, as a causal relation between variables and parameters in a case study cannot be determined due to the nature of the experiment. This means the focus group and panel study must say if assumptions about consistency, integration and reuse are correct, as there is no objective way of determining this otherwise. This leads to a different set of metrics than in a natural experiment, where causality *is* determined. We have attempted to achieve validity in three areas defined by [Shadish et al., 2002]: ecological, external and internal validity.

Ecological validity According to [Brewer, 2000], for a research study to possess ecological validity, the methods, materials and setting of the study must approximate the real-life situation that is under investigation. This has been taken into account by applying the case studies in the industrial setting, involving the focus group as a means of testing the ecological validation. It further refers to the generalizability of the treatment/condition outcomes to other companies and organizations. This was ensured by picking multiple companies for the case studies. These have in common that they produce complex systems, with a lots of embedded software, through a multi disciplinary design process. The systems themselves were highly divergent, further expanding the ecological validity. Furthermore, a workshop (appendix C) confirmed that people outside the research process recognized the issues (appendix B), and found merit in the architecture modeling framework as a solution. Concluding: the research and the resulting framework certainly are ecologically valid.

External validity [Shadish et al., 2002] refers to the generalizability of the treatment / condition outcomes. Only partial design processes were considered in the case studies. However, these parts were chosen on the premise that the company had difficulty in that particular partial design process. By improving the hard part, we can assume that other parts can be improved in a similar manner when this would be desirable. This does not mean that a tool such as the architecture modeling framework should always be applied §6.5. We can assume that the research is externally valid, as we have successfully applied the methods we were aiming at in all case studies we undertook.

Internal validity The external validity of this research can be seen as a self fulfilling prophecy: find a problem you can solve, and solve it, and say your method works. This is an issue of internal validity [Shadish et al., 2002], which refers specifically to whether an experimental treatment/condition makes a difference or not, and whether there is sufficient evidence to support the claim. In this thesis, the consistency, integration and reuse in the case study were improved by applying the architecture modeling framework *only* when this is confirmed by the focus group. A follow up study after the validity of the – now more mature – framework should be conducted to say anything conclusive about if the framework will improve consistency, integration and reuse, and in which situations. There exists a set of metrics to test the internal validity, given in [Shadish et al., 2002]. Examples of concepts that are to be taken under consideration are 'experimenter bias', where the experimenter affects the outcome of the case study, 'history' where conducting a case study a second time will give other results, or 'maturation', where the subjects 'learn' the methods of the case study and thus affect measurement of the progressing experiment.

6.5 Concluding Remarks

From the topics discussed in this chapter, we can conclude that the AM framework can help to mitigate many issues in multi disciplinary design. Reflecting upon the case studies, and the user experience, we can say that the AM framework provides a new tool to architects and (system) engineers – the framework 'works' to facilitate multi disciplinary design.

However, this does not mean that the AM framework should be applied in all situations, or that it is the only tool these stakeholders need in their toolbox. In the scope (§2.9), it was mentioned that not all things should be formalized, and that those situations are not part of the research. This mainly considers situations of informal communication, such as in brainstorming for ideation, or explaining a problem in a face to face meeting. What we saw in the workshops, and in day-to-day practice, however, is that people do use the AM tool for ideation, as the AM language does provide a clear set of concepts for this purpose. The boundary for when to use the framework, and when not, is thus not easy to define. In §2.5 such a boundary could be when the information needs to be conferred to other people in a later stage in the design process, where there is a high risk of losing that information. Doing something wrong at such a stage will result in high rework costs later on.

Finally, the AM framework is not intended to replace CAD tools, or other software packages. It aims instead on forming a 'glue' for the models made in these tools. For example, adding 'richer' information in the form of figures or text was considered in §4.1.4, but not implemented as an integral part of the AM framework. One can make references to external design documents by using the `uri` property of all AM language concepts (appendix E.1). The case studies show that this mechanism can be very powerful, even to be used to automate design processes (chapter 5).

7. Conclusion

In this conclusion, we will discuss the findings of the research done for this thesis. First, to see if consistency, integration and reuse in multi disciplinary design processes can be improved by using the Architecture Modeling Framework. Then, novelties of this framework are discussed. The third section contains generic conclusions about modeling architectures and systems. We end with remarks on the validity of the chosen research approach.

This thesis approaches issues in modern multidisciplinary design from a broad perspective. Where many researchers try to isolate a subset of issues in order to be able to develop novel methods or theories within well defined theoretical borders and practical methods, in my research I reasoned the other way around: By finding issues that are normally too all–encompassing to qualify for academic research and by finding gaps and overlaps between existing theories and methods. This approach resulted in the attempt to integrate many modeling concepts and methods into a single framework, the Architecture Modeling (AM) Framework. This framework was constructed with two main functions in mind (see introduction 1).

First, to provide a communication mechanism – for people from various disciplines – in a common, machine-readable, **Architecture Modeling Language** that can describe both the system architecture and information flows in the design process. The language contains a limited set of concepts to capture architectural concerns, views, and the domain-specific abstractions and decompositions, and their interrelations. Every concept has properties to capture external resource information and ownership information.

Second, to provide automation mechanisms for this common language in an extendable **Architecture Modeling Framework** of design tools to support the design process: Such as the Architecture Modeling Editor: used to graphically construct, use, serialize, and automate architecture models. Automation of design tasks is facilitated with add-ins for this editor, or can be connected to an Architecture Model directly, using the machine readable language.

The **Architecture Models** themselves define multiple views and mappings, from the function and system decomposition and requirements to the various aspects and domain. They can model the design flow in terms of shared parameters, the interfaces between stakeholder aspects, and their relation to requirements and functionality, and define the required input/output parameters of design tasks and external models, and connect these to the architectural context.

In this conclusion, I will defend the chosen approach and the results of the research in a number of sections:

- Problem definition revisited (§7.1): I hypothesized that consistency, integration and reuse are the main drivers for improvements in multi disciplinary design. Did we find supporting evidence for the hypothesis?
- Gaps in state of the art (§7.2): Following literature research, some gaps in modern approaches were highlighted. Has the research in this thesis contributed to the state-of-the-art in the related academic fields?
- The AM framework Format (§7.3): The AM framework and its modeling language were the result of the chosen case studies and researched literature. What can we generalize about the framework?
- Chosen research method and validity (§7.4): Industry-as-a-lab was applied as the main research method. Was this the right method? and what can we say about the validity of the research?

7.1 Problem Definition Revisited

In the problem definition (chapter 2) we expanded upon the issues in multi disciplinary design. These issues are tabulated in appendix B for reference. From analysis of these issues followed the hypothesis: *The consistency, integration and reusability of multi-disciplinary design processes can be improved by facilitating communication between stakeholders, modeling architectural concerns and other important design information, and facilitating capturing and reusing design knowledge across disciplines.* The three topics of consistency, integration and reuse were defined as the main drivers that improve multi disciplinary design. For these three topics, metrics were defined, which we will analyze here.

Consistency

In the hypothesis (§2.8), I deduced from the problem definition that "Consistency is improved with a multi disciplinary model in which to track the 'who, what, when, where, why, in what way, by what means' design information". I conclude that consistency is improved with models written in the AM language (appendix E). This language is able to capture all these types of design information:

The *Who*, by assigning ownership to every concept in the model. The *What* by modeling `Functions`, `Entities`, `Requirements`, `Aspects`, etc, in concrete objects and hierarchies thereof. The *When* by modeling the (temporal) dependencies between `Functions` and between `Design Tasks` and the flow between `Entities`. The *Where* by various mappings between `Functions`, `Requirements`, `Domain Entities` etc... The *Why* by specifying the `Requirements` and `Aspects` of the design process. *In What Way* by modeling the `Parametric` information of the design process. *By What Means* by modeling `Design Tasks` and `Domain Entities` that prescribe the domain specific models that need to be developed and analyzed.

Together, the concepts of the AM language are able to describe a wide range of design information in a single model. This information is normally distributed over wide ranges of models, design documents and in 'tacit' understanding in various participants of the design process. By making all this information explicit, and centrally available, consistency of the design process can be more easily be guarded. In the case studies in §5.4 and §5.6 we saw this in practice.

Integration

In the hypothesis, I deduced from the problem definition that "Integration is improved with a multi disciplinary model that maintains overview and context between the views containing stakeholder (user) specific design information". To make these models, I have chosen for a combination of a meta model (§4.3) and a reference model (§4.2) called the Architecture Model (AM). These models can capture a wide range of information, not only from the design specification (definition in §2.3), but also of the design process itself. This design information is modeled in a set of pre–defined concepts of the AM language, which are (among other reasons) chosen for being discipline–independent.

While this means that design information from various disciplines can be integrated in a single model, another crucial concept was introduced to actually partition the model. This concept, the View, enables stakeholders to focus on their part of the design information without being encumbered by the whole model. This is not paradoxical, as the objects in the views are still part of the integrated model – the view just establishes a subset of the model. As the views do not own the objects, they can share objects between them – the 'interwovenness' of relations and objects can be expressed. The View concept is not a novelty from this thesis – it is for example also adopted in the IEEE architecture standard [Hilliard, 2007]. However, it is novel that views in the AM are not specialized, such as in sequence and class diagrams in UML. This enables the user to combine all types of modeling objects in any view. In the case studies in §5.4, §5.5 and §5.6, we saw how Architecture Models integrate design information from various design disciplines.

Reuse

In the hypothesis, I deduced from the problem definition that "Reuse is improved with a multi disciplinary model that facilitates (partial) construction of the design specification by applying knowledge bases". The Architecture Models are implemented as such models, thereby supporting this deduction. Because of their function as a meta model (§4.3), a computer is able to parse the model and perform operations on its content. And because of their function as a reference model, they can be connected to external models and tools 4.2. These functions together allow for the building of toolchains upon or in the AM framework, such as the development of Knowledge Bases (§4.4) that capture design knowledge in software. These knowledge bases thus reuse knowledge for specific sets of design tasks.

The AM can reference other AM's, thereby referencing (reusing) existing information instead of copying it. However, the structure of the AM's allows for copying partial AM's, using them as reusable building blocks to expand AM's or to instantiate new ones. The AM editor has such a method, which makes every AM a 'toolbox' for constructing other AM's.

Together, we can both reuse the structure of the AM and the design knowledge of engineers. In the case studies in §5.4, and§5.6 we saw how Architecture Models are constructed from reusable building blocks by automated knowledge bases. These knowledge bases were also able to transform the data in the AM to external engineering models and back.

Can we Confirm the Hypothesis

Throughout the thesis, the emphasis was on ways to improve the way multi disciplinary design is conducted. The research for the problem definition 2 has highlighted a large set of issues, collated in appendix B. The resolution of these issues in practical situations were handled point for point in the tables 5.4.6, 5.5.7 and 5.6.7 at the end of the case studies. This provides a closed loop between the issues, the chosen methods (AM framework) in the case studies, and the evaluation in terms of consistency, integration and reuse.

Reiterating all these points here will not add much information, so the reader is deferred to the mentioned sections and appendix. What we can conclude from these points is that the hypothesis can be confirmed. We have seen that we can build an integrated model of important concerns and design information that can be used for communication between stakeholders. We have seen that this model can be used in interaction with software tooling to provide reusability of design knowledge as well as design artifacts. And we have seen that this model can provide a consistent 'data platform' for the communication between the people, software tools and connected models.

Improvement of multi disciplinary design processes in general can probably not be defended because of a lack of independent study. However, I *can* claim that the design processes researched in the case studies have been improved. In the Baggage handler case study 5.4 – By providing management of the interconnectivity of the design library to manage library consistency and reducing calculation time for synthesizing system models. In the Software Generation of a Lithography System case 5.5 by demonstrating that much of the data circulating in the design process can be formatted in the same language, which simplifies management of these data flows and development and maintenance of software tooling. And in the Paper Path case 5.6, which shows that disparate design information can be modeled concisely in a single model, using a single tool. It also showed that, with some effort, a design process normally taking up to three weeks can be automated to be done in seconds. This can make the AM a central repository of design information for many aspects of design, with the added benefit of simplifying design documentation.

7.2 Contributions to State of the Art

The current state of the art of multi disciplinary modeling tools and methods was analyzed in chapter 3. This chapter was concluded with a number of gaps found to impede an encompassing method for consistency, integration and reuse in multi diciplinary design. These gaps mostly stem from a narrow application area for the individual methods under review. As we saw in the case studies (chapter5) the industry–as–a–lab approach chosen in this research works the other way around: Find a set of issues as observed by the company, and develop an approach as broad and generic as possible. The resulting Architecture Modeling (AM) Framework thus contributes to the state-of-the-art in various areas, of which this section focuses on novelties.

First, many of the modeling concepts in the AM language are similar to often–used concepts occurring in other methods (e.g. `Function`, `Parameter`, `Requirement`). The AM language casts a wider net than these individual methods by prescribing the syntax between these often–used concepts. As such, AM language can be used to apply many of the discussed methods directly, and create synergy between these methods as a bonus. As an example, think of a Function Behavior State model coupled with an CAFCR model. Where the FBS part states behaviors of a system, and the CAFCR part states the customer wishes in the form of `Requirements`, tunable `Parameters` and the `Entities` that execute these behaviors (see the paper path case study §5.6). Also, more thought is given to the design process. Most methods focus on modeling systems and their behavior. While the AM language can be used for this, we also added concepts to define the design process itself.

Another contribution is the separation of automated design knowledge, model data, and model representation. This is a paradigm often used in software engineering, but is not often applied in design and (system)engineering models, where the models are constructed to facilitate a specific engineering method (e.g. CAD/CAM, Matlab). The AM language barely contains prescribed mathematical/algorithmic methods. This has been a choice to prevent the 'Model of Everything' dilemma (§3.7). The only mathematical constructs prescribed are sets and (un)directed graphs. This makes it possible for software to manipulate any AM model in terms of navigation, serialization, representation and editing, but does not prescribe the application of the model. However, knowledge can be easily automated in separate knowledge bases to extend the applicability of the AM's to automate specific design processes, as we have seen in the case studies 5. The result is an application area from informal diagrams like mind maps, to automated design specification generation such as in the baggage handler case study (§5.4), and any point, and mix of points, in between.

As a final contribution to the state–of–the–art, the `View` concepts of the AM language are a representation of an underlying consistent and integrated data model, with relations to and from concepts in different views. This enables not only stakeholder/user specific views to exchange information across disciplines, across design process phases and across levels of granularity, but also the formulation of subsets of the model where automated knowledge bases can work upon. In contrast to many existing methods – where a view often has a specialized goal (Class or Sequence diagram §3.6, IDEF0 to 14 §3.3) – an AM `View` can contain references to all other concept types.

7.3 The AM Framework Format

The AM framework resulting from my research is not the only answer to improving consistency, integration and reuse. This was concluded at the end of the AM framework chapter §4.5). However, the basic function of the AM framework – providing a communication mechanism – remains, even if different concepts would have been defined for the AM language. What is important is having discussions within organizations to come to a shared understanding of what the goals are and how to represent the system. For this you need to develop a shared language, the means to record what you have said in this language, and the means to use these records in attaining the goal of the organization.

Whereas any natural language is usable for this purpose, a specialized language is better for a number of reasons. In discussing in natural language, individuals will always have a different semantic meaning attached to the used lexicon. This is due to the way an individual grows up, is educated and experiences the world, and even the context in which the lexicon is used – the individual's semantics are 'formed' by communication with others. Scaling down the lexicon to only a few words should help the formation (learning) of a shared semantic meaning to the lexicon, therefore resulting in a more formal language for the exchange of information. Furthermore, computers are very bad in 'understanding' the semantics of lexicon. The smaller the lexicon, the easier it is to develop software to handle it. The same holds for the syntax used. By limiting the allowable relation types between the various concepts in the lexicon, we force the user to model in a rigid way. However, the resulting models should be easier to explain or interpret by other people (and software tools), compared to natural languages.

The usage of natural lexicon is still possible in the values of the various attributes of the concepts (e.g. `Entity` name is *"Car"*, `Parameter` unit is *"kg"*), as these values have no semantic meaning to the basic framework. This is intended to give a large degree of freedom to the user in making models. However, custom software (knowledge bases) can be written that actually has some semantic understanding of the values inside the attributes, making it possible to extend the framework with automatic design tools (e.g. if `Entity` name is *"Car"* then this object should comprise four `Entities` with name *"Wheel"*). The usage of these values then have to conform to the semantics in the software tool, making the model formal and verifiable. However, this formal representation only extends to partial models, the rest is still free to the more tacit interpretation of the human users. This makes it possible to extend the tooling base of the framework one model parser, model generator or verification algorithm at a time, while leaving the rest of the model 'tacit'.

The amount of concepts and relations to the AM language (defined in appendix E) is thus a limited set. This was a design choice that resulted from generalizing both available modeling concepts (chapter 3) and generalizing concepts needed to accomplish the case studies (chapter 5). This is why `Entity` has made it as a concept, and *"Car"* has not. Different case studies would probably have resulted in a different set of concepts. In the next chapter (8), some interesting additional concepts will be discussed, which were not fully considered during the four year research program.

Whatever the final set of modeling concepts will be, in terms of AM modeling, the basic functionality will remain: By offering the concepts of the AM language (in a toolbox in a AM (diagram) editor), the user is guided in formulating his or her thoughts in terms of the language. This will encourage the user to think of functionality of the system, requirements, applicable design tasks, etcetera, and make these thoughts explicit to both other users and software tooling. Making specific views for the various users, aspects or design domains will keep the AM's manageable, while remaining integrated. Finally, making design tasks explicit will identify workflow between design tasks, thus identifying design process deliverables, and allow for validation and verification feedback and management documentation.

7.4 Chosen Research Method and Validity

Research in the area of systems engineering and architecting is difficult to validate. Whatever method the researcher comes up with, it must be 'tested' against industrial applicability. In this research, I have chosen the Industry–as–a–Lab approach (§5.2) as the basis for defining the Architecture Modeling Framework. This meant an iterative development of the framework, keeping in mind consistency, integration and reuse at all times. The companies associated with the case studies provided the real world issues as well as a 'sounding board' for our ideas and solutions.

In [Muller, 2011c], we find a scientific method of working that – in hindsight – was applied in this research. The paper mentions the basis for such research: *make explicit, substantiate, try to validate*. This approach was applied (and reflected in the sections) to the three case studies mentioned in chapter 5. In this chapter, we also discussed how to extend and integrate the AM framework beyond any of the individual case studies (figure 5.1). The result is a AM framework that is:

- technically feasible, as demonstrated by the prototype framework,
- technically valuable, as demonstrated by the people interviewed during the research (§2.4 and appendix C)
- practically feasible, is demonstrated, in a limited way, by adoption of the framework at OCE. (see §6.1)
- practically valuable, is demonstrated, in a limited way, by adoption of the framework at OCE. (see §6.1)

In section §6.4, the validity of the framework was discussed. Here, I concluded that the chosen case studies are a valid representation of the targeted industrial ecosphere, and that the AM framework does represent a valid solution to many industrial issues. Also, I conclude that new research to the efficacy of the AM framework by independent researchers is needed. The reason for this is that the Industry–as–a–lab approach actually *led to* the framework, instead of objectively evaluating it: *"The researcher is wearing two hats in these research approaches: as systems engineer and as researcher. The attitude of the systems engineer is result oriented and cooperative. The attitude of the researcher is questioning and challenging. Where*

the systems engineer will promote the use of a proposed method or technique, the researcher needs to question its validity." [Muller, 2013].

8. Outlook and Recommendation

There is a lot of potential for further development of the prototype Architecture Modeling (AM) Framework. In this outlook and recommendation chapter, we will formulate ideas for follow up research. The chapter is split in improvements to the framework in its current form (§8.1), a new framework that allows for modeling design decisions and design history (§8.2), implementing a Design Task paradigm for Design Automation (§8.3), and possibilities for disseminating the research to a wider audience of both industrial users and academia (§8.4).

The current form of the AM framework can also be further researched. The framework as it now is, is roughly the end point of a set of subsequent case studies. The framework was only tested in its final form in a workshop and the Océ case study (chapter 5), which may not be the complete scientific basis to make claims about consistency, integration and reuse. Of course this research was executed with these claims in mind, but an independent study would be better to validate them. The author suggests following a social sciences approach as advocated by [Shadish et al., 2002] to determine this validity, as was discussed in §7.4.

8.1 Improvements to the Framework

Already, the groundwork for an improved Architecture Modeling Framework is realized by Clemens Wolters. A switch was made from the Eclipse Graphical Modeling Framework (GMF) to a new Microsoft .Net Framework platform using the Windows Presentation Foundation. This groundwork was necessary to overcome several shortcomings in the Eclipse version. While GMF was a good platform to build an initial academic tool, programmatically unstable behavior related to GMF implementations proved a constant hindrance to customizing the tool. Combined with a steep learning curve, and the reliance on a large number of ever changing open source libraries, this GMF stands in the way of professionalizing the tool. The new .Net version provides a good platform for a new phase of development. These developments focus on two main points: Implement a multi user application to enable work on the diagrams, and synchronize this with a shared database (§8.1.1), and, implement a database environment (§8.1.2).

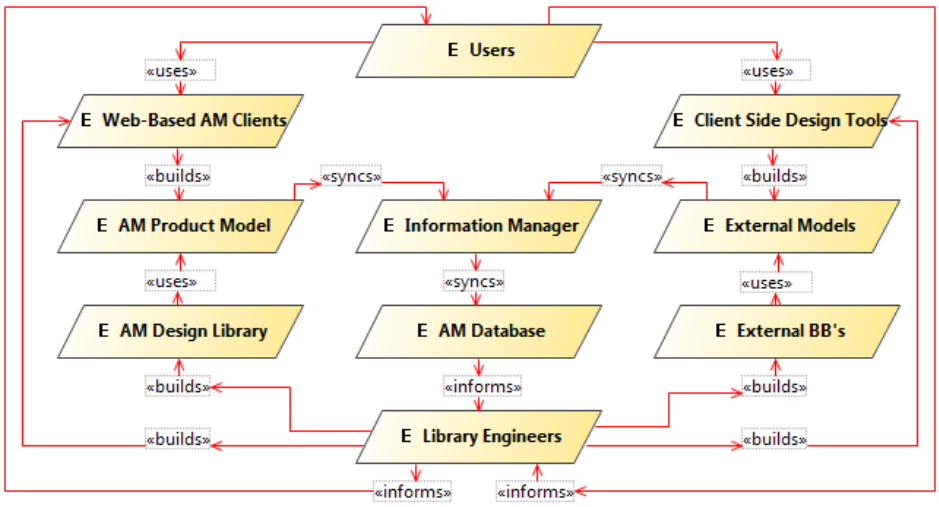


Figure 8.1: Intended toolchain of a follow up Architecture Modeling Framework. The follow up consists of a separation of client editors communicating with a central database. The editors work on one View at a time, while the database stores the shared architecture model. A special role is dedicated to the reuse of design objects in libraries that enable faster modeling and knowledge based engineering tooling.

8.1.1 Multi User Architecture Modeling Framework

The main shortcoming of the current tool is that it is single user. Only one user can work on a model at any time. For a tool that aims at providing communication assistance between various users (stakeholders), this is a major shortcoming. The main expansion of functionality must therefore focus on allowing multiple users to use a centrally managed Architecture Model.

The new .Net tool is still a stand alone editor, but can be expanded to multi user using existing .NET support. In figure 8.1 we see the desired scenario. The user will make models and documents in dedicated tools (e.g. CAD or Microsoft Office). Some metadata of these models, defined beforehand by the users, is translated to the Architecture Modeling (AM) Language to be shared between users from various disciplines. This meta modeling has been discussed conceptually in §4.3 and demonstrated in case studies §5.4 and §5.6. The meta data is serialized to a database by an information management tool.

Users can use the current framework to 'freeform' AMs, like in mindmaps or Visio/ SysML diagrams. As different AM diagrams are written in the same AM language, they can be connected and stored in the same database in a future framework version, allowing the reference model functionality discussed in concept in §4.2 and in practice in case studies §5.4 and §5.6. The result is a single architecture model connecting users to each other and to their models.

To allow for reuse of design information, a library of design patterns (§5.3) can be maintained that can be instantiated as partial AMs on the editor side, and as building blocks for the various domain models at the design tools side. Based on these patterns, custom tooling can be built to execute models, such as building the paper path model discussed in §5.6. This custom tooling can be maintained by a specialized group of library engineers, and is especially interesting for companies with a high degree of reusable components such as discussed in case study §5.4. It is also possible that such tooling is bought or exchanged between companies is an open source platform should be available. The .NET environment has many tools to facilitate building such a framework. This will be expanded on in §8.1.2.

8.1.2 Implement a Database Environment

Many software development tools are available for further development of the AM framework. In this recommendation we will focus on a set of tools developed by Microsoft on the .NET platform. The reason for the focus on Microsoft tools is the great integration, longevity and community support available for .NET, and the fact that most tools are available for free. The author has years of experience in these tools, and it is remarkable how easy it is to adapt and include new functionality in software development.

The .NET framework is available on all Windows platforms, which means tools developed on it can run without many problems on all kinds of pc's without the need for big installations. It includes graphical support for the diagram editor (Windows Presentation Foundation) support for web development (ASP.NET), supports database development and database connectivity (ADO.NET and SQL Server Express), and supports communication to all kinds of third party software also developed on .NET (Microsoft Office, Component Object Model (COM) integrated software such as SolidWorks 3D CAD, XML serialization). Finally, all .NET development can be done using Microsoft Visual Studio Express, a free software development tool supporting many programming languages.

At the start of this chapter, we mentioned the new AM editor developed by Clemens Wolters. He started developing the editor in the Windows Presentation Foundation. However, the serialization of the AM (storing the data on a disk) was out of scope of that project. The next step is to develop the AM database. A database on which multiple client editors can manipulate a shared AM. Microsoft Server Express is a good tool to build such a database. It can easily be connected to the graphical editor. A hint of that is given in appendix H, where such a database/editor class description is given.

For a database to work, some modifications to the AM language need to be made. First, the relations between objects in the AM are now stored inside arrays inside the objects themselves. This is undesirable in a database, as a database field will only contain a single value (or a single relation in this case). The relations must therefore be stored in a dedicated set of tables, corresponding to at least the three relation types: Composition, Mapping, Dependency (see appendix E). These tables will refer to the unique indices of the connected objects.

Table 8.1: The AM database and two AM revisions. Left column shows the database tables, then the columns in these tables, and to the right the values in the rows in the table. Note that this is a very simple representation where the AMs only have one object of ClassA, ClassB and Relation

Table name	Columns	Values		
AM	id	1	2	
	name	am1	am1	
	revision	1	2	
	ClassA ref	1	3	
	ClassB ref	1	2	
	Relation ref	1	2	
ClassA	id	1	2	3
	name	a1	a1	a2
	version	1	2	1
ClassB	id	1	2	
	name	b1	b2	
	version	1	1	
Relation	id	1	2	
	name	r1	r1	
	version	1	2	
	ClassA ref	1	3	
	ClassB ref	1	2	

In table 8.1 we see an explanation of such a database: Revision 1 of the AM (am1–1) contains the objects a1–version1, b1–version1 and r1–version1. The second revision of the AM (am1–2) contains the objects a2–version1, b2–version1 and r1–version2. Given this, the AM in its current form will be a set of lists of objects (corresponding to the AM language concepts) and a set of lists of relations. This versioning of objects introduces an interesting expansion on the AM, as is depicted in figure 8.2. By versioning the objects, a design history model can be stored, which will be useful in a cooperative expansion of the AM framework with the Design Framework of ESI, discussed in the next section (§8.2).

8.2 Architecture Modeling Framework Synergy with Design Framework

During the last stages of the research project, the "Embedded Systems Innovation by TNO, or ESI", appeared as a likely partner for project follow up. They, among many other things, develop a Design Framework to support architecting and systems engineering of complex development processes. According to the ESI [Embedded Systems Instituut, 2013]: *The Design Framework supports the architects work by enabling him to store crucial information of the architecture and the development process in a single tool. External information will remain external to this. The central idea is:*

- *to gain an overview of the design issues, which can be open, closed, or under construction with diverse options*
- *to keep the information about these issues actual at all times, as well as*

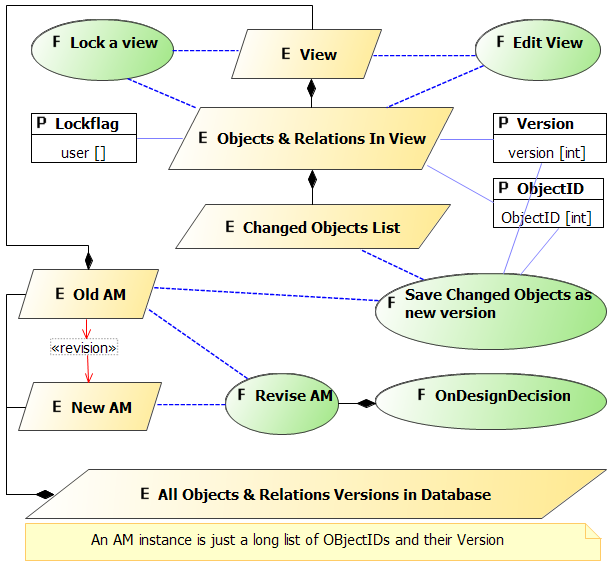


Figure 8.2: Save new object versions – both concepts and relations – for changes on saving a view. Revise an Architecture Model on a Design Decision by saving the list of objects and their versions of the AM at the moment of the design decision. Thus, unchanged objects are not duplicated, and the design history is conserved.

- *to link the information to underlying evidence that is gathered. in the project or elsewhere.*
- *to capture design rationales over the different life-cycle phases*
- *to model multiple views to cover multiple disciplines and system aspects*
- *to model how design problems are related to actual models in this context without having to model all aspects of the system, and*
- *to test how design parameters and their dependencies can be validated to ensure consistent designs.*

The ESI design framework is built upon the Eclipse Graphical Modeling Framework, analogous to the AM editor. As a consequence, it was perceived that an integration of the AM framework and the Design Framework would be possible. After some open discussions, both ESI and the AGCSMS teams concluded that a great synergy could be achieved through an integrated framework. In such an integrated framework, the positive properties of both frameworks can be combined. The AM framework can deliver a the AM language to replace the 'system blocks' of the Design Framework as a much richer descriptive language. The Design Framework design history model and design decision modeling can then model the development and rationale of AMs over time. The integration of the Parameter concept of both AM language and Design Framework is an additional synergy. This could enable parameter dependency analysis and validation of the Design Framework, as well as leverage the Knowledge Based Engineering basis of the AM framework.

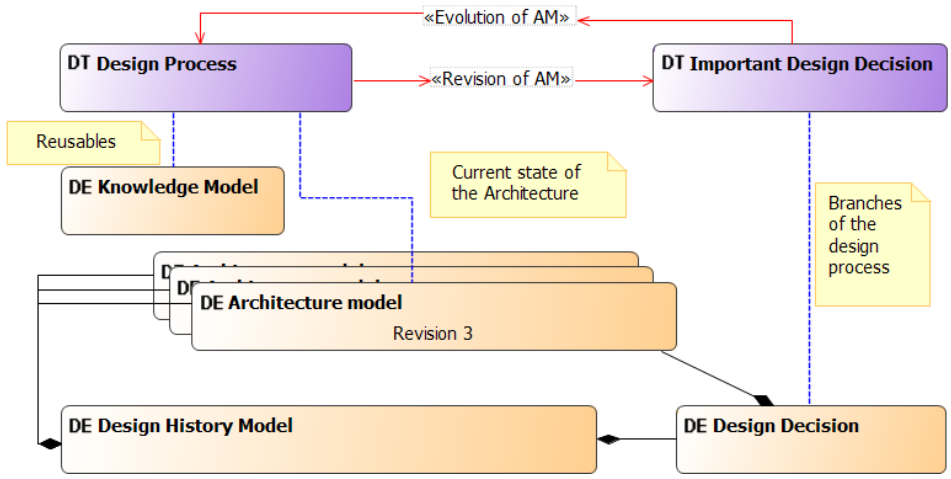


Figure 8.3: Mix of Architecture Models to describe the architecture at a point in time with Design Framework from Embedded Systems Institute that stores design rationale and history. Architecture model instances are stored in a design history model. This mechanism must allow for branches in the history model.

In figure 8.3 we see the integration of multiple AMs over time in a design history model. The AM is revised based on the outcome of important design decisions during the design process (as discussed in §8.1.2). The AM revisions are thus archived. When afterwards the design decisions lead to a dead end in the design process, or when other design approaches need be considered, the older revisions can be restored as the active AM.

In the appendix H, a class description is given that formulates the synergy of the AM framework with the ESI Design Framework. The descriptions are given in three formats, Eclipse EMF for expansion using the current GMF implementation of ESI, Microsoft .NET for extension of the AM framework as described in 8.1, and Microsoft SQL Server Express for serialization to a database for future multi user implementations (8.1.2). Contact the author to get these class descriptions, and support for applying them.

8.3 *Design Task Paradigm for Automation of Design Tasks*

The design task concept was introduced in the AM language as a means to help model design processes §E. Design tasks are often part of design process patterns: A sequence of design tasks, cutting across multiple model types and tools, with design information from various disciplines §5.3. These patterns were found and used in the case studies of chapter 5, but a more generic procedure of using the design tasks is not yet developed. This section will therefore outline a design task paradigm, specifically considering design task automation.

A basic workflow for modeling with (automated) design tasks work as follows (see figure 5.22 for an example): First, set up design domains and aspects in various views. This will represent the basic resources of a company. Then, model the system specific architecture (Functions/Requirements/Entities) of the design object. This will capture the goal of the design process in terms of the system, and the available resources stated in the previous point. With this context in place, the organization can implement generic design primitives for known design Domain Entities and Aspects. This will 'embody' the structure of the system in a design description. Then, connect the design primitives to system specific architecture in the views. This will state the desired functionality and requirements for the various aspects and entities (components). Lastly, a set of predefined automated design tasks can be connected to those parts of the architecture that can be designed automatically (see figure 5.4 for an example). Non-automated design tasks can be connected to the other objects, to declare ownership and formalize them for the people in the organization that execute them.

Executing an (automated) design task will also follow a basic workflow. First, validate the input to the Design Task. Is all information needed for the Design Task available, and are available values within logical limits. Also check the existence and connectivity of external models and tooling. The Design Task can execute now. Start with implementing domain models and/or changes to the Architecture Model. These external models can then be executed or knowledge base rules can be run to generate Design Task output. The output must be verified first. Are the values within required ranges and logical limits. The output, or results from the external models and knowledge bases then needs to be converted to Architecture Modeling Language, in the form stated in the Design Task context. The results can be used to either implement new external models, and/or change the Architecture Model or its values. This whole process can be repeated to iterate towards a set of Requirements values connected to the Design Task. Note that the Paper Path Design case study in §5.6 is an example of this workflow.

Often, a lot of reusable design information is available, as companies will have experience with the product under design from previous versions or comparable product families. In that case, it would be expedient to make a reusable product model parallel to the architecture model. This is a process of Knowledge Based Engineering: Transferring engineering knowledge and tasks to dedicated software tooling §4.4.2. The Design Task concept of the AM language can be a placeholder of such an engineering task, linking the dedicated software tooling (Knowledge Base) to the specific AM it is implemented upon. An example of this approach was demonstrated in the Paper Path Design case study in §5.6. Development of an automated Design Task can be formalized as follows:

- Start with an analysis of workflows common in the company. These have been defined as design process patterns in section (§5.3)
- Create tooling with the support of the designer/engineer in question, create ownership. The tool needs to help a user in a certain design task after all.
- Identify design decisions that are taken in order to configure these process patterns.
- Construct generic parametric design primitives, which can be configured into model instances.

- Construct executable Design Task algorithms that can instantiate and connect the design primitives. These are called Knowledge Bases.

To make the process of automating design tasks easier, design tasks must have a set of predefined generic characteristics. These can be implemented in a Design Task class with the following characteristics:

- Design task handler: Access point of the design task from the diagram, opens a graphic user interface for the input of design decisions (configuration parameters).
- Design task graphical user interface: Shows the input needed for the algorithm, the available context in the existing model, and the projected output (target models, changes in architecture model). Also can include execute button to run external tools.
- Design task filter: Shows the part of the architecture diagram associated with the knowledge base of the design task. Hides the rest.
- Design task consistency checker: Tests the topology of the architecture model on required input/output. Validates the content of the connected information (semantics, units, values, ranges, etc. Verifies existence of external models.)
- Design Task Updater: Runs the diagram without the GUI, when no design decisions have to be taken. Returns error messages when information content is no longer valid or consistent. (out of scope, topology changes, models missing). Make this method exposed to the outside to quickly promulgate design changes throughout the architecture model.

8.4 Dissemination of Research

The Automatic Generation of Control Software for Mechatronic Systems Project was set to finish in June 2012. At this stage, the project had already generated practically applicable output in the form of the AM editor as a support tool for System Architecting and Model-Based Engineering. This tool can be used as a means to improve the design process by maintaining consistency and promoting sharing of information. This information is not part of the actual product that is produced, but is useful as a communication tool while designing. During the finalization of the project, several dissemination (valorization) options were considered to follow up on the project, based on the developed tool.

- Researchers start a company to sell the architecture modeling tool: This has a low chance of survivability, as companies will not trust a startup company, because they are not sure of long term support or availability of tool versions, and there is no portfolio of successful company implementations. Other issues are customer support, client usability issues, versioning, scaling to industrial size problems and integratability in existing design/software environments.
- Open Source: An online community will develop and support the tool framework. There is currently no such community. This means a forum and website needs to be developed to bring such a community together. On top of the framework, commercial company specific packages can be developed once the survivability is ensured.

- Merge method with tool suite of existing commercial software developer: This is the best option for the industrial customers, as they just want to buy a working tool off-the-shelf. However, implementation of other possibilities of the method could be limited. Making an open source framework on which the existing commercial software developers can build tooling would be the best of both worlds.
- Training sessions: To focus on the method instead of the tool, workshops could be given to teach the way complex design problems can be modeled. For these workshops, a workbook would be needed that explains the method, and there should be some case studies to work with during the sessions.
- Researchers start a consultancy company: Expanding on the training sessions, it would be possible to start a consultancy company to show companies how to implement the method. This would lack the actual tooling needed, and could face the same problems as the first point.

Industrial members of the AGCSMS project's user committee have expressed that this tool can fill a need in current mechatronic design practice that is not available commercially. This claim is supported by an experimental implementation at Océ, but the tool could be used by other organizations and academics too. A discussion was held by the user committee about the valorisation or dissemination of this tool or its underlying method, to prevent the tool from 'dying' from lack of interest or maintenance. The results of this discussion are described below.

Several members of the Automatic Generation of Control Software for Mechatronic Systems Project User Committee have expressed a willingness to commit resources to a follow up of the research. The University of Twente and University of Delft want an open source platform with a developing community. Academics could use the generic framework to build problem specific tooling on top of it for other research projects. Industrial case studies could be implemented on top of the framework to limit the amount of effort for researchers in developing their own tools. The universities can organize workshops and symposia to attract more interest, and spread the word to spin-off companies of their university. ESI can also contribute knowledge and resources to extend the framework as discussed in section §8.2. Industrial partners (not given by name) have expressed a willingness to contribute in:

- Development of their toolsuite on top of a architecture modeling framework, once all software development issues have been resolved. They can supply additional case studies, centered on developing a plan for smooth step-by-step deployment of tool functionality in a company, thus ensuring integration with existing tooling for architecture, consistency, model based systems engineering, product lifecycle management, etcetera. Smooth introduction of functionality is also a prerequisite for acceptance by the end-users. If the tool is not developed to fit with the current design practice of a company, or if no real engineering knowledge is captured in tool knowledge bases, the architecture models would become 'abstract monsters', that would not serve anyone.
- Sharing expertise in developing the framework, preferably through a community of partners to share experiences in using the framework. They can provide lessons learned, as some companies have 15 years experience in developing such tooling, and review

How-To manuals (as opposed to academic papers) on making architecture models. Companies will lend their name to the framework / method once more cases have been successfully implemented, and 'spread the word' by featuring the framework developer on their community partnerships websites once some applications have been developed. They will also help sell the idea of the AM framework through attending workshops and consultancy.

- Purchasing the AM framework software once a trusted vendor or open source community is available. Some companies will develop their own framework in-house if no vendor is available, but could supply lessons learned to an open source community. Only companies with existing tool builders and design automation experience will be able to use the framework, so it would be preferable if a network of partner companies is available to share experiences.

Concluding, we see a lot of potential in further development of the AM framework. The ideas are there, there are industrial partners willing to commit effort once a tool reaches practical usability, and in ESI (§8.2), we have a partner for further development beyond the scope of the AM framework.

9. Acknowledgments

I would like to thank a number of people who made it possible for me to complete this thesis. Foremost, Andres Alberto Cabrera, he and I conducted much of the research together. While he focused primarily on the feasibility of using the architecture model to model system behavior and the writing of papers, I focused more on the integration of the design process and the programming. As can be seen by the late delivery of this thesis, I am not a prolific writer. Thanks Andres for all the help and the good times! Maarten Foeken, thanks for helping in implementing the knowledge based engineering approach to the AM framework and the good times in Delft and Montreal. Professor Tomiyama, your leadership of the research project opened a lot of doors, and provided excellent mentoring in becoming a critical thinker and researcher. Your knowledge of the worlds of engineering and academic politics were a great help! Hans Tragter, you have always supported my academic tendencies to 'go off the suggested path' and I have learned a lot from you. My development into a half-software engineer, half-academic with a basis in mechanical engineering has opened a lot of doors for me. Thanks for all the coffee. Maarten Bonnema, your patience with me in writing this thesis was impressive. You learned me how to write this book in an academic way. Were it up to me, it would probably end up in a technical specification with source code and a manual for the AM framework. You always helped me keep the big picture in mind, and formulate a consistent story throughout the thesis. Thanks for all the tutoring over the years!

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.

Bibliography

Aberdeen Group (2007). The Digital Product Development Benchmark Report. Technical Report March, Aberdeen Group. 14, 15, 186, 187, 188, 189, 190, 192

Alberts, W., T. Baan, N. Brouwers, M. Hamilton, and W. Tabingh Suermondt (2011, September). Modelgebaseerd ontwikkelen getoetst aan de praktijk. *ENGINEERINGNET MAGAZINE*, 49–51. `http://www.mainpress.com/nederlands/dossier_automation/pdf/asml.pdf`. 116

Alirezaei, M. and R. van den Boom, T. J. J. Babuska (2012). Max-plus algebra for optimal scheduling of multiple sheets in a printer. In *American Control Conference*, pp. 1973 – 1978. 90

Alvarez Cabrera, A., M. Foeken, O. Tekin, K. Woestenenk, M. Erden, B. De Schutter, M. van Tooren, R. Babuška, F. van Houten, and T. Tomiyama (2010, December). Towards automation of control software: A review of challenges in mechatronic design. *Mechatronics 20*(8), 876–886. `http://linkinghub.elsevier.com/retrieve/pii/S0957415810000838`. 90, 190

Alvarez Cabrera, A. A. (2011). *Architecture-Centric Design: Modeling and Applications to Control Architecture Generation*. VSSD. `http://www.narcis.nl/publication/Language/EN/id/85/RecordID/oai:tudelft.nl:uuid:29fce0ab-5b53-4d59-9688-ad5118e04a5b`. 3, 59, 84, 90, 124, 141, 144

Alvarez Cabrera, A. A., M. S. Erden, M. J. Foeken, and T. Tomiyama (2008). High Level Model Integration for Design of Mechatronic Systems. *Materials Engineering*, 387–392. 90

Alvarez Cabrera, A. A., M. S. Erden, and T. Tomiyama (2009). On the Potential of Function-Behavior-State (FBS) Methodology for the Integration of Modeling Tools. In *Competitive Design-Proceedings of the 19th CIRP Design Conference*, Number March, pp. 30–31. 50, 90

Alvarez Cabrera, A. A., K. Woestenenk, and T. Tomiyama (2011, April). An architecture model to support cooperative design for mechatronic products: A control design case. *Mechatronics 21*(3), 534–547. `http://linkinghub.elsevier.com/retrieve/pii/S0957415811000237`. 84, 90

Anderson, P. (1999). Complexity Theory and Organization Science. *Operations Research 10*(3), 216–232. 22

Babbie, E. (1997). *Practice of Social Research*. Wadsworth Pub. Co. http://books.google.nl/books?id=s__MjwEACAAJ. 90

Badgerland Users Group (2007). The Top 10 Things Engineers Can Do to Get into LEAN PRODUCT DEVELOPMENT. Technical report, Badgerland Users Group. 13

Bahill, A., B. Bentz, and F. Dean (1996). Discovering System Requirements. http://prod.sandia.gov/techlib/access-control.cgi/1996/961620.pdf. 43

Bahill, A. T. and B. Gissing (1998). Re-evaluating Systems Engineering Concepts Using Systems Thinking. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS 28*(4), 516–527. 42, 43

Bilgin, A., D. Caldwell, J. Ellson, E. Gansner, Y. Hu, and S. North (2011). Graphviz. http://www.graphviz.org/. 98

Boehm, B. W. (1988, May). A spiral model of software development and enhancement. *Computer 21*(5), 61–72. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=59. 42

Bonnema, G. M. (2008). *FunKey Architecting - An Integrated Approach to System Architecting Using Functions, Key Drivers and System Budgets*. Ph. D. thesis, University of Twente. http://purl.org/utwente/58868. 47, 191

Borches Juzgado, P. D. (2010, December). *A3 architecture overviews : a tool for effective communication in product evolution*. Ph. D. thesis, University of Twente, Enschede, the Netherlands. http://doc.utwente.nl/75284/. 49, 69, 186, 189, 196

Boucher, M. and D. Houlihan (2008). System design: new product development for mechatronics. *Aberdeen Group 1*(January). http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:System+Design+:+New+Product+Development+for+Mechatronics#0. 9, 11, 13, 27, 186, 187, 189, 191

Brazeau, J.-F. (2011). GMF executable model tutorial. http://gmfsamples.tuxfamily.org/wiki/doku.php. 85

Brewer, M. (2000). Research Design and Issues of Validity. In *Handbook of Research Methods in Social and Personality Psychology*. Cambridge University Press. http://books.google.com/books?hl=en&lr=&id=j7aawGLbtEoC&oi=fnd&pg=PA3. 147

Brimble, R. and F. Sellini (2000). The MOKA Modelling Language. In *EKAW '00 Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*. http://web1.eng.coventry.ac.uk/moka/doc-papers.htm. 50

Brooks, F. P. (1975). *The mythical man-month*. Addison-Wesley. http://userfs.cec.wustl.edu/~cse528/MythicalManMonth.pdf. 17, 18

Chang, a. S.-T., J. S. Shih, and Y. S. Choo (2011, April). Reasons and costs for design change during production. *Journal of Engineering Design 22*(4), 275–289. http://www.tandfonline.com/doi/abs/10.1080/09544820903425218. 25

Chivagomez, R. (2004, September). Repercussions of complex adaptive systems on product design management. *Technovation 24*(9), 707–711. http://linkinghub.elsevier.com/retrieve/pii/S0166497202001554. 21

Chulvi, V. and R. Vidal (2006). TRIZ on Design-oriented Knowledge-based Systems. *CTQ Media*. http://www.triz-journal.com/archives/2009/03/02/. 66

CIRI (1994). Quality Function Deployment. 46, 47

Department of Defence (2012). DoDAF Architecture Framework Version 2.02. http://dodcio.defense.gov/sites/dodaf20/. 47, 51

DSMWeb (2009). Design Structure Matrix. http://www.dsmweb.org/. 46

Embedded Systems Instituut (2002). Intelligente Apparaten, Een visie op embedded systemen in Nederland. Technical report, Embedded Systems Instituut. 104

Embedded Systems Instituut (2013). Design Framework. http://www.esi.nl/symposium/2013/df-workshop.dot. 160

EMF (2011). Eclipse Modeling Framework Project. http://www.eclipse.org/modeling/emf/. 76

Estefan, J. A. (2007). Survey of Model-Based Systems Engineering (MBSE) Methodologies. *INCOSE MBSE Initiative*. http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf. 38, 39, 40

Foeken, M., A. Cabrera, M. Voskuijl, and M. van Tooren (2010). Applying Knowledge-Based Engineering to Control Software Generation. In *Proceedings of the ASME IDETC/CIE*. ASME. http://link.aip.org/link/abstract/ASMECP/v2010/i44113/p1213/s1. 3, 84, 90, 128

Foeken, M. and M. Voskuijl (2010, October). Knowledge-based simulation model generation for control law design applied to a quadrotor UAV. *Mathematical and Computer Modelling of Dynamical Systems 16*(3), 241–256. http://www.tandfonline.com/doi/abs/10.1080/13873954.2010.506745. 90

Foeken, M., M. Voskuijl, and M. V. Tooren (2008). Model Generation for the Verification of Automatically Generated Mechatronic Control Software. *Proceedings of IEEE/ASME MESA 2008*, 275–280. 90

Foeken, M. J. and M. J. L. V. Tooren (2009). Object-Oriented Simulation Model Generation in an Automated Control Software Development Framework. *Competitive Design-Proceedings of the 19th CIRP Design Conference* (March), 30–31. 90

Forsberg, K., H. Mooz, and H. Cotterman (2005). *Visualizing Project Management* (Third ed.). John Wiley & Sons, Inc. http://mmsa.kpi.ua/files/keruvannya-proektom. 38, 72

Friedenthal, S., A. Moore, and R. Steiner (2008). *A Practical Guide to SysML The Systems Modeling Language*. Elsevier. 39

Gausemeier, J. (2007). Design Methodology for Mechatronic Systems. http://wwwhni.uni-paderborn.de/en/pe/research/design-methodology-for-mechatronic-systems-dm/. 46

Gautam, N., R. Chinnam, and N. Singh (2007). Design reuse framework: a perspective for lean development. *International Journal of Product Development 4*(5), 485–507. http://inderscience.metapress.com/index/0461431142612783.pdf. 31

GMF (2011). Graphical Modeling Project. http://www.eclipse.org/modeling/gmp/. 77, 81, 83

Guizzardi, G. (2007). On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. In *Proceeding of the 2007 conference on Databases and Information Systems IV: Selected Papers from the Seventh International Baltic Conference DB&IS'2006*, pp. 18–39. IOS Press. http://portal.acm.org/citation.cfm?id=1565425. 62, 63

Guizzo, E. (2011). IBM's Watson Jeopardy Computer Shuts Down Humans in Final Game. *spectrum.ieee.org*. http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/ibm-watson-jeopardy-computer-shuts-down-humans. 65

Hilliard, R. (2007). All About IEEE Std 1471. http://www.iso-architecture.org/ieee-1471/docs/all-about-ieee-1471.pdf. 45, 151, 188

Hirtz, J., R. B. Stone, D. A. Mcadams, S. Szykman, and K. L. Wood (2002). A Functional Basis for Engineering Design : Reconciling and Evolving Previous Efforts. *NIST Technical Note 1447*. 66

Hoffmann, H.-p. (2008). A SysML Based Systems Engineering Process. Technical report, Telelogic. http://download-na.telelogic.com/download/ugcagenda/Mon_HarmonySEASysMLBasedSystemsEngineering_PeterHoffman.pdf. 38

Hoover, S. (1991, October). Models and abstractions in design. *Design Studies 12*(4), 237–245. http://linkinghub.elsevier.com/retrieve/pii/0142694X9190039Y. 22

Hoult, D., C. Meador, J. Deyst, and M. Dennis (1996). Cost awareness in design: the role of data commonality. *COST ENGINEERING 38*, 13–16. http://papers.sae.org/960008. 188

IBM (2012a). IBM Rational Jazz. http://www-01.ibm.com/software/rational/jazz/. 51

IBM (2012b). Rational System Architect. http://www-01.ibm.com/software/awdtools/systemarchitect/. 39, 47

INCOSE (2006). Systems engineering vision 2020, version 2.0. *Seattle: International Council on Systems Engineering 1*(September). http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:SYSTEMS+ENGINEERING+VISION+2020#3. 20, 37, 38, 189

Ishii, M., T. Tomiyama, and H. Yoshikawa (1993). A Synthetic Reasoning Method for Conceptual Design. In *Proceedings of the IFIP TC5/WG5.3 Conference on Towards World Class Manufacturing*. 50, 65

Jackson, C. (2006). The Mechatronics System Design Benchmark Report - Coordinating Engineering Disciplines. Technical Report August, Aberdeen Group. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:The+Mechatronics+System+Design+Benchmark+Report#0. 185, 188, 189, 190, 191, 192, 193

Jauregui-Becker, J., W. Wits, and F. Houten (2008). Reducing design complexity of multidisciplinary domain integrated products: a case study. *Manufacturing Systems and Technologies for the New Frontier*, 149–154. http://www.springerlink.com/index/hp778r3254u23275.pdf. 49

KBSI (2011). Knowledge Based Systems. http://www.kbsi.com/. 45

Krogstie, J. (2008). Using EEML for Combined Goal and Process Oriented Modeling : A Case Study. In *Exploring Modelling Methods for Systems Analysis and Design*. 51

Lutters, E. (2001). *Manufacturing Integration based on Information Management*. Ph. D. thesis, University of Twente. 40

MacLeamy, P. (2004). Macleamy Curve. http://www.buildinggreen.com/articleimages/1711/MacLeamy_Curve.gif. 12

Manson, S. (2001). Simplifying complexity: a review of complexity theory. *Geoforum 32*(3). http://www.sciencedirect.com/science/article/pii/S001671850000035X. 20, 21

McConnell, G. R. (2000). Emergence : Still a Challenge for the Systematic. In *INCOSE International Symposium*. 187

McLaughlin, B., G. Pollice, and D. West (2006). *Head First - Object Oriented Analysis & Design*. O'Reilly Media. http://books.google.com/books?hl=en&lr=&id=-QpmamSKl_EC&oi=fnd&pg=PR9&dq=Head+First+-+Object+Oriented+Analysis+%26+Design&ots=nd3R1TX5oU&sig=fAyHG4nEwIdBQ0pmtWx5K9DQGSs. 53

Melching, M. (2012). *System Design Communications Tool*. Ph. D. thesis, University of Twente. 196

Mengoni, M., S. Graziosi, M. Mandolini, and M. Peruzzini (2010, October). A knowledge-based workflow to dynamically manage human interaction in extended enterprise. *International Journal on Interactive Design and Manufacturing (IJIDeM) 5*(1), 1–15. http://www.springerlink.com/index/10.1007/s12008-010-0103-7. 51

Meredith, J. R. and S. J. Mantel (2008). *Project Management: A Managerial Approach*. John Wiley & Sons. http://books.google.co.in/books?id=qKS0GAAACAAJ. 10

Moneva, H., R. Hamberg, T. Punter, and J. Vissers (2009). Putting Chaos under Control: on how Modeling should Support Design. *esi.nl*. http://www.esi.nl/projects/multiform/0211.pdf. 52

Mosterman, P., J. Ghidella, and J. Friedman (2005). Model-based design for system integration. In *Second CDEN International Conference on Design Education, Innovation, and Practice, Kananaskis, Alberta, Canada*. http://msdl.cs.mcgill.ca/people/mosterman/papers/cden05/fp.pdf. 17

Mulbury Six Sigma (2011). Design for Six Sigma. http://www.designsixsigma.com/whatis_dfss.htm. 13

Muller, G. (2008). When and What to Standardize. *Focus*. http://www.gaudisite.nl/INCOSE2008_Muller_standardization.pdf. 193

Muller, G. (2011a). CAFCR: A multi-view method for embedded systems architecting; balancing genericity and specificity. *Buskerud University College*. http:/ThesisBook.pdf. 42, 46

Muller, G. (2011b). *System architecting*. http://www.gaudisite.nl/. http://www.gaudisite.nl/SystemArchitectureBook.pdf. 44, 48, 53, 57, 58, 72, 95, 139

Muller, G. (2011c). Systems Engineering Research Validation. http://www.gaudisite.nl/SEresearchvalidationpaper.pdf. 155

Muller, G. (2013). Systems Engineering Research Methods. In *Conference on Systems Engineering Research*, Volume 00. http://www.gaudisite.nl/cser2013_Muller_SEresearchmethods.pdf. 156

Mylopoulos, J. (2004). Structured Analysis and Design Technique. http://www.cs.toronto.edu/~jm/2507S/Notes04/SADT.pdf. 45

No Magic Inc (2013). MagicDraw. http://www.nomagic.com/products/magicdraw.html. 145

Noor, M. J. (2007). *A comprehensive approach to complex system product development : operations management tools applied to automotive design*. Ph. D. thesis, Massachusetts Institute of Technology. http://hdl.handle.net/1721.1/39885. 13

Nuseibeh, B. (2000). Requirements engineering: a roadmap. *on the Future of Software Engineering*. http://dl.acm.org/citation.cfm?id=336523. 16, 42, 186, 187, 189, 192, 193

OMG (2011). Object Management Group. http://www.omg.org/. 76, 77, 78

Pahl, G., W. Beitz, and K. Wallace (1996). *Engineering design: a systematic approach*. Springer. http://books.google.nl/books?id=8fuhesYeJmkC. 42

Paredis, C., a. Diaz-Calderon, R. Sinha, and P. Khosla (2001, July). Composable Models for Simulation-Based Design. *Engineering With Computers 17*(2), 112–128. http://www.springerlink.com/index/10.1007/PL00007197. 55

Peak, R., R. Burkhart, S. Friedenthal, M. Wilson, M. Bajaj, and I. Kim (2007a). Simulation-based design using SysML part 1: a parametrics primer. In *INCOSE intl. symposium, San Diego*, pp. 1–20. `http://eislab.gatech.edu/pubs/conferences/2007-incose-is-1-peak-primer/2007-incose-is-1-peak-primer.pdf`. 55

Peak, R., R. Burkhart, S. Friedenthal, M. Wilson, M. Bajaj, and I. Kim (2007b). Simulation-based design using SysML part 2: celebrating diversity by example. In *INCOSE intl. symposium, San Diego*, pp. 1–22. `http://eislab.gatech.edu/pubs/conferences/2007-incose-is-2-peak-diversity/2007-incose-is-2-peak-diversity.pdf`. 30, 55

Pidcock, W. (2003). What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta-model? `http://infogrid.org/wiki/Reference/PidcockArticle`. 66, 75

Pin-shan, P. (1976). The Entity-Relationship Model-Toward a Unified View of Data. *ACM Transactions on Database Systems 1*(1), 9–36. 69

Potts, C. (1993). Software-engineering research revisited. *Software, IEEE 10*(5), 19–28. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=232392`. 90, 146

Proud, S. and M. Wetzer (2003). The Product Lifecycle Management Opportunity. *mThink website*. `http://mthink.com/revenue/content/product-lifecycle-management-opportunity`. 12

Ralph, P. and Y. Wand (2008). A teleological process theory of software development. *Proceedings of JAIS Theory Development Workshop 8*. `http://sprouts.aisnet.org/8-23`. 16

Ralph, P. and Y. Wand (2009). A Proposal for a Formal Definition of the Design Concept. *Design Requirements Engineering: A Ten-Year Perspective 14*, 103–136. `http://filer.case.edu/nxb41/templates/Wand_DRW.pdf`. 16

Reich, Y. (1995). Engineering Design A Critical Review of General Design Theory Yoram Reich. *Research in Engineering Design 7*, 1–18. 50

Reichwein, A. and C. J. J. Paredis (2011). Overview of Architecture Frameworks and Modeling Languages for Model-Based Systems Engineering. *IDETC/CIE 2011*, 1–9. `http://srl2.gatech.edu/btw/files/ASMEIDETC2011-reichwein-paredisv16.pdf`. 58, 139, 143, 144, 145, 185, 186, 189, 190, 191, 193

SAP (2012). SAP. `http://www.sap.com`. 51

Scherer, R. and P. Katranuschkov (2006). From Data to Model Consistency in Shared Engineering Environments. *Intelligent Computing in Engineering and Architecture 1*(c), 615–626. `http://www.springerlink.com/index/U533572K5M15V342.pdf`. 193

Schöner, H. (2004). Automotive mechatronics. *Control engineering practice 12*(11), 1343–1351. `http://www.sciencedirect.com/science/article/pii/S0967066103002314`. 185, 188, 189, 191

Shadish, W. R., T. D. Cook, and D. T. Campbell (2002). *Experimental and quasi-experimental designs for generalized causal inference*. Houghton Mifflin. http://books.google.nl/books?id=o7jaAAAAMAAJ. 147, 157

Smith, B. C. (1982). *Procedural reflection in programming languages*. Ph. D. thesis, Massachusetts Institute of Technology. http://publications.csail.mit.edu/lcs/specpub.php?id=840. 82

Suh, N. (1998). Axiomatic design theory for systems. *Research in Engineering Design 10*(4), 189–209. http://www.springerlink.com/index/XBTG4A8GAVR9LRFY.pdf. 40

Suh, N. P. (2001). *Axiomatic design: advances and applications*. The Oxford series on advanced manufacturing. Oxford University Press. http://books.google.nl/books?id=EBtg_1M1lnEC. 49

Sysml.org (2012). SysML Open Source Specification Project. http://www.sysml.org/. 54

Takeda, H., P. Veerkamp, and H. Yoshikawa (1990). Modeling design process. *AI magazine 11*(4), 37. http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/855. 76, 80

Tekin, O. A., T. Tomiyama, and B. D. Schutter (2009). Toward a flexible control design framework to automatically generate control code for mechatronic systems. *Proceedings of 2009 American Control Conference*, 4933–4938. 90

Tomiyama, T., M. van Tooren, R. Babuska, and F. van Houten (2008). Automatic Generation of Control Software for Mechatronics Systems; Proposal for IOP-IPCR 2nd Tender. Technical report. 1

Torry-Smith, J., S. Achiche, N. Mortensen, A. Qamar, J. Wikander, and C. During (2011). Mechatronic Design - Still A Considerable Challenge. *Proceedings of the ASME IDETC*. http://www2.md.kth.se/~ahsanq/files/DETC2011_48306.pdf. 57, 58, 139, 140, 141, 142, 143, 185, 186, 187, 188, 189, 190, 191

Ullman, D. G. (2003). *The mechanical design process*. McGraw-Hill series in mechanical engineering. McGraw-Hill. http://books.google.com/books?id=iF_8XPVmrOEC. 10, 12, 42

Unger, D. and S. Eppinger (2011, October). Improving product development process design: a method for managing information flows, risks, and iterations. *Journal of Engineering Design 22*(10), 689–699. http://www.tandfonline.com/doi/abs/10.1080/09544828.2010.524886. 42

Unger, D. W. and S. D. Eppinger (2006). Improving product development processes to manage development risk. *MIT Sloan Research Paper No. 4568-06 1*(January). http://papers.ssrn.com/sol3/papers.cfm?abstract_id=876618. 42

Uschold, M. (2001). Where are the Semantics in the Semantic Web? *AI Magazine* (June 2001). 63, 64

Vitech Corporation (2012). Vitech Corporation Website. http://www.vitechcorp.com/. 39

W3C (2012a). OWL Web Ontology Language. http://www.w3.org/TR/owl-features/. 53

W3C (2012b). XML Technology. http://www.w3.org/standards/xml/. 53, 69, 79

Walton, M. (1999). Strategies for lean product development. *The Lean Aerospace Initiative Working Paper Series*, 1–91. http://dspace.mit.edu/handle/1721.1/7519. 13, 192

Wang, L., W. Shen, H. Xie, J. Neelamkavil, and A. Pardasani (2002). Collaborative conceptual design: state of the art and future trends. *Computer-Aided Design 34*(13), 981–996. http://www.sciencedirect.com/science/article/pii/S0010448501001579. 185, 186, 187, 188, 189, 190, 191, 192, 193

Wasiak, J., B. Hicks, L. Newnes, A. Dong, and L. Burrow (2009, August). Understanding engineering email: the development of a taxonomy for identifying and classifying engineering work. *Research in Engineering Design 21*(1), 43–64. http://www.springerlink.com/index/10.1007/s00163-009-0075-4. 185, 186, 187

Weilkiens, T. (2007). *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann. http://books.google.com/books?hl=en&lr=&id=LKC9F5gQt3AC&oi=fnd&pg=PR9&dq=Systems+Engineering+with+SysML/UML&ots=gcgEb9iDT-&sig=YopVqEPI-qU56vtLwMzAJ6_gr1s. 53, 54, 55

Wielinga, B., J. M. Akkermans, and A. T. Schreiber (1995). A Formal Analysis of Parametric Design Problem Solving. In *Proceedings of the 9th Banff Knowledge Acquisition Workshop*, Number D. 50

Wikipedia (2012). Brooks's Law. http://en.wikipedia.org/wiki/Brooks's_law. 19

Woestenenk, K., G. M. Bonnema, A. A. Alvarez Cabrera, and T. Tomiyama (2011). Capturing Design Process Information in Complex Product Development. *Proceedings of the ASME IDETC-CIE*, 1–10. 90

Woestenenk, K., H. Tragter, G. M. Bonnema, A. A. Alvares Cabrera, and T. Tomiyama (2010). Multi Domain Design: Integration and Reuse. In *Proceedings of IDETC/CIE 2010*, pp. 1–10. ASME. http://link.aip.org/link/abstract/ASMECP/v2010/i44137/p519/s1. 90

WPF (2011). Windows Presentation Foundation. http://msdn.microsoft.com/en-us/library/ms754130.aspx. 77

Yoshioka, M., Y. Umeda, H. Takeda, Y. Shimomura, Y. Nomaguchi, and T. Tomiyama (2004, April). Physical concept ontology for the knowledge intensive engineering framework. *Advanced Engineering Informatics 18*(2), 95–113. http://linkinghub.elsevier.com/retrieve/pii/S1474034604000175. 65

Yuan, S. (2011). *Architecture Modeling for autonomous robotic missions*. Master of science thesis, University of Delft. 84, 141, 142

A. Deliverables

IOP-IPCR 0602 Automatic Generation of Control Software for Mechatronics Systems
Output Lists

Project Leader: T. Tomiyama (TU Delft)
Researchers: A.A. Alvarez Cabrera, M.J. Foeken, O.A. Tekin, K. Woestenenk, M. Alirezael

A.1 Tooling

K. Woestenenk, A. A. Alvarez Cabrera. The Architecture Modeling Framework. Download available upon request at k.woestenenk@gmail.com. Also available in c# prototype.

A.2 Dissertations

Krijn Woestenenk, 2014, *Consistency, Integration, and Reuse in Multi-Disciplinary Design Processes. Through an Architecture Modeling Framework*. PhD thesis from University of Twente. ISBN: 978-90-365-3614-1, DOI: 10.3990/1.9789036536141. Defended on 6 March 2014.

A. A. Alvarez Cabrera, 2011, *Architecture-Centric Design: Modeling and Applications to Control Architecture Generation*. PhD thesis from Delft University of Technology. ISBN: 978-90-6562-281-5. Defended on 12 October 2011.

A.3 Journal Papers

A.A. Alvarez Cabrera, M.J. Foeken, O.A. Tekin, **K. Woestenenk**, M.S. Erden, B. De Schutter, M.J.L. van Tooren, R. Babuska, F.J.A.M. van Houten, and T. Tomiyama: *Towards Automation of Control Software: A Review of Challenges in Mechatronic Design*, Mechatronics, Vol. 20, No. 8, (2010), pp. 876-886, ISSN: 0957-4158, DOI: 10.1016/ j.mechatronics.2010.05.003.

M.J. Foeken and M. Voskuijl: *Knowledge-Based Simulation Model Generation for Control Law Design Applied to a Quadrotor UAV*, Mathematical and Computer Modeling of Dynamical Systems - Special Issue on Object-Oriented Modeling, Vol. 16, No. 3, (2010), pp. 241-256, ISSN: 1387-3954, DOI: 10.1080/ 13873954.2010.506745.

A.A. Alvarez Cabrera, **K. Woestenenk**, and T. Tomiyama: *An Architecture Model to Support Cooperative Design for Mechatronic Products: A Control Design Case*, Mechatronics, Vol. 21, No. 3, (2011), pp. 534-547, ISSN: 0957-4158, DOI: 10.1016/ j.mechatronics.2011.01.009.

M.J. Foeken, A.A. Alvarez Cabrera, M. Voskuijl, M.J.L. van Tooren: *Enabling Control Software Generation by Using Mechatronics Modelling Primitives*, Advanced Engineering Informatics, Special Issue on Knowledge-based Engineering, Vol. 26, No. 2, (2012), pp. 196-206, ISSN: 1474-0346, DOI: 10.1016/ j.aei.2012.02.005.

A. A. Alvarez Cabrera, G.A.D. Lopes, and T. Tomiyama: *An Architecture-Level Specification for Automated Supervisory Controller Design*, Research in Engineering Design, ISSN: 0934-9839 (print), 1435-6066 (online), under review (material available in the dissertation of A.A. Alvarez Cabrera).

A. A. Alvarez Cabrera and T. Tomiyama: *Architecture-Level Representation and Analysis of Regulatory Controller Configuration for Complex Mechatronic Systems*, Research in Engineering Design, ISSN: 0934-9839 (print), 1435-6066 (online), under review (material available in the dissertation of A.A. Alvarez Cabrera).

T. Tomiyama, T.J. van Beek, A.A. Alvarez Cabrera, H. Komoto, V. D'Amelio: *Making function modeling practically usable through reasoning*, Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM), Special Issue on Functional Descriptions, P. Vermaas and C. Eckert (eds.), ISSN: 0890-0604 (print), 1469-1760 (online), under review.

A.4 Conference Papers

A.A. Álvarez Cabrera, M.S. Erden, M.J. Foeken, and T. Tomiyama: *High Level Model Integration for Design of Mechatronic Systems*, in Proceedings of 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (IEEE/ASME MESA 2008), October 12-15, 2008, Beijing Friendship Hotel, Beijing, China, IEEE Intelligent Transportation Systems Society, ISBN 978-1-4244-2368-2, (2008), pp. 387-392, (CD-ROM).

M.K. Chmarra, A.A. Álvarez Cabrera, T. van Beek, V. D'Amelio, M.S. Erden, and T. Tomiyama: *Revisiting the Divide and Conquer Strategy to Deal with Complexity in Product Design*, in Proceedings of 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (IEEE/ASME MESA 2008), October 12-15, 2008, Beijing Friendship Hotel, Beijing, China, IEEE Intelligent Transportation Systems Society, ISBN 978-1-4244-2368-2, (2008), pp. 393-398, (CD-ROM).

M. Foeken, M. Voskuijl, A.A. Alvarez Cabrera, and M. Van Tooren: *Model Generation for the Verification of Automatically Generated Mechatronic Control Software*, in Proceedings of 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (IEEE/ASME MESA 2008), October 12-15, 2008, Beijing Friendship Hotel, Beijing, China, IEEE Intelligent Transportation Systems Society, ISBN 978-1-4244-2368-2, (2008), pp.275-280, (CD-ROM).

M.J. Foeken, M. Voskuijl: *Automatic Simulation Model Generation for Control Law Design Applied to a Quadrotor UAV*, Special Session on Object Oriented Modelling and Simulation, the 6th MathMod Conference, 11-13 February 2009, Vienna, Austria.

Tekin, R. Babuska, T. Tomiyama, and B. De Schutter: *Toward a Flexible Control Design Framework to Automatically Generate Control Code for Mechatronic Systems*, in the Proceedings of 2009 American Control Conference (ACC '09), June 10-12, 2009, St. Louis, MO., ISSN: 0743-1619, ISBN: 978-1-4244-4523-3, pp. 4933-4938 (2009). DOI: 10.1109/ ACC.2009.5160184 (Awarded Best presentation award).

A.A. Alvarez Cabrera, M. S. Erden, and T. Tomiyama: *On the Potential of Function-Behavior-State (FBS) Methodology for the Integration of Modeling Tools*, in R. Roy and E. Shehab (eds.): Competitive Design-Proceedings of the 19th CIRP Design Conference, 30-31 March 2009, Cranfield University, Cranfield, Bedford MK43 0AL, UK, Cranfield University Press, ISBN 978-0-9557436-4-1, (2009), pp. 412-419.

M.J. Foeken, M.J.L. van Tooren: *Object-Oriented Simulation Model Generation in an Automated Control Software Development Framework*, in R. Roy and E. Shehab (eds.): Competitive Design-Proceedings of the 19th CIRP Design Conference, 30-31 March 2009, Cranfield University, Cranfield, Bedford MK43 0AL, UK, Cranfield University Press, ISBN 978-0-9557436-4-1, (2009), pp. 443-450.

K. Woestenenk, A.A. Alvarez Cabrera, H. Tragter, T. Tomiyama, and G.M. Bonnema: *Multi Domain Design: Integration and Reuse*, in the Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2010), (The 22nd International Conference on Design Theory and Methodology (DTM)), 15-18 August 2010, Montreal, Quebec, Canada, Paper No. DETC2010-28640, ASME, New York, NY, USA, ISBN 978-0-7918-3881-5, (2010), DVD-ROM, 10 pages.

M. Foeken, A.A. Alvarez Cabrera, M. Voskuijl, and M. van Tooren: *Applying Knowledge-Based Engineering To Control Software Generation*, in the Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2010), (The 30th Computers and Information in Engineering Conference (CIE)), 15-18 August 2010, Montreal, Quebec, Canada, Paper No. DETC2010-28833, ASME, New York, NY, USA, ISBN 978-0-7918-3881-5, (2010), DVD-ROM, 8 pages.

A.A. Alvarez Cabrera, M.J. Foeken, **K. Woestenenk**, G. Stoot, and T. Tomiyama: *Modeling and Using Product Architectures in Industrial Mechatronic Product Development: Experiments and Observations*, in the Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (DETC 2011), (The 23rd International Conference on Design Theory and Methodology (DTM)), 28-31 August 2011, Washington, DC, Paper No. DETC2011-47148, ASME, New York, NY, USA, (2011), DVD-ROM Order No. I872DV, 10 pages.

K. Woestenenk, A.A. Alvarez Cabrera, G.M. Bonnema, and T. Tomiyama: *Capturing Design Process Information in Complex Product Development*, in the Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (DETC 2011), (The 31st Computers and Information in Engineering Conference (CIE)), 28-31 August 2011, Washington, DC, Paper No. DETC2011-48105, ASME, New York, NY, USA, (2011), DVD-ROM Order No. I872DV, 10 pages.

A.A. Alvarez Cabrera, H. Komoto, and T. Tomiyama: *Supporting Co-Design of Physical and Control Architectures of Mechatronic Systems*, in the Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (DETC 2011), (The 31st Computers and Information in Engineering Conference (CIE)), 28-31 August 2011, Washington, DC, Paper No. DETC2011-48200, ASME, New York, NY, USA, (2011), DVD-ROM Order No. I872DV, 8 pages.

M. Alirezaei, T.J.J. van den Boom, R. Babuska: *Max-Plus Algebra for Optimal Scheduling of Multiple Sheets in a Printer*, in the Proceedings of 2012 American Control Conference (ACC '12), 27-29 June 2012, Montréal, Quebec, Canada, Paper WeC11.6.

T. Tomiyama: *Architecture-Centric Model-Based Product Development*, in R. Scheidl and B. Jakoby (eds.): The Proceedings of the 13th Mechatronics Forum International Conference, Vol. 2/3, 17 - 19 September 2012, Linz, Austria, Austrian Center of Competence in Mechatronics (ACCM), Johannes Kepler University Linz (JKU), Trauner Verlag, Linz, Austria, ISBN: 978-3-99033-045-6, pp. 434-443, (2012), (Invited Keynote).

A.5 Book Chapters

A.A. Alvarez Cabrera, T.J. van Beek, H. Komoto, and T. Tomiyama: *Architecture-Centric Design Approach for Platform Development*, in T.W. Simpson, R.J. Jiao, Z. Siddique, and K. Hölttä-Otto (eds.): Advances in Product Family and Product Platform Design: Methods & Applications, Springer, Berlin, Heidelberg, under review.

A.6 Poster Presentations and Other Publications

A.A. Alvarez Cabrera and M.J. Foeken: *Automatic Generation of Control Software for Mechatronics Systems*, Poster presentation at DCED Symposium, Delft Center for Engineering Design, 12 December 2007, Delft.

Aydin Tekin, R. Babuska, and B. De Schutter: *Automatic control code generation for mechatronic systems*, Book of the Abstracts of the 27th Benelux Meeting on Systems and Control (Heeze, The Netherlands, March 18-20, 2008) (website: http://www.wfw.wtb.tue.nl/benelux2008/) page 58.

M.J. Foeken: *Automatic Generation of Control Software for Mechatronic Systems*, IOP Theme day, May 20, 2008, Rotterdam

A.A. Alvarez Cabrera, **K. Woestenenk**, O.A. Tekin, and M.J. Foeken: Point one poster presentation, 9 March 2009.

A.A. Alvarez Cabrera, **K. Woestenenk**, O.A. Tekin, and M.J. Foeken: First workshop on complex system architecture poster presentation, TU Delft, 22 June 2009.

M.J. Foeken, Designing Mechatronic Systems, article for Leonardo Times, magazine of VSV Leonardo Da Vince, society of Aerospace Engineering students

2nd Complex systems architecting workshop (3 posters presented)

K. Woestenenk, M.J. Foeken, and A.A. Alvarez Cabrera: *Support tool for system architecting and model-based engineering - an industrial case study*, presentation at Bits & Chips 2010 Embedded Systemen, Eindhoven.

A.A. Alvarez Cabrera , **K. Woestenenk**, M. Alirezaei, M.J. Foeken, System architecture workshop: Modeling, integration and use. Workshop, September 2 2011, Delft.

A.A. Alvarez Cabrera , **K. Woestenenk**, M. Alirezaei, M.J. Foeken, System architecture: Modeling, integration, and use with the AM. Presentation at ESI, October 3 2011, Eindhoven.

A.7 Internal Documents

Report for the Philips internship: Contains a detailed description of the activities carried out at the internship in Philips Applied Technologies during March and April of 2008. Authors: Andres Alvarez, and Maarten Foeken.

Reports about meetings with industrial partners: Available at the project shared point. Author: Andres Alvarez.

Chapter on *Knowledge ModelIng with UML* in reader AE4-232 Advanced Design Methodologies, Faculty of Aerospace Engineering, Delft University of Technology. (Mentioned in previous progress report, now published)

Point one poster presentation (09-03-2009) (Andres, Krijn, Aydin, Maarten)

First workshop on complex system architecture poster presentation (22-06-2009).

Vanderlande case study report. Authors: **Krijn Woestenenk** and Andres Alvarez.

A.A. Alvarez Cabrera: Reports about meetings with industrial partners (Period October 2007-January 2008): Available at the project shared point.

M.J. Foeken: *Chapter on Knowledge Modelling with UML* in reader AE4-232 Advanced Design Methodologies, Faculty of Aerospace Engineering, Delft University of Technology, 2008.

B. Issues

B.1 Consistency Issues

1. Big Picture. A 'big picture' of the information shared between stakeholders, and between stakeholder and design process goals can provide stakeholders with the overview they need to make sense of their design tasks. Typical overview questions are: Who supplies my information? Who depends on my information? What am I supposed to do? Within which requirements is my design space? What is the up-to-date version of design information? Where can I retrieve and store design information? (Also stated as important issue in: [Wang et al., 2002], [Torry-Smith et al., 2011], [Jackson, 2006], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

2. Overview of the Design Information Flow. Stakeholders get information from other stakeholders, and in turn provide information to other stakeholders. The information exchange on the interface between stakeholders is often not modeled, but can be the cause of a lot of consistency problems (for example, the unit of numbers is not handed over consistently, so the Mars Climate Orbiter disintegrates in Mars's atmosphere). There often is no overview of this flow on an organizational level. Such a picture provides valuable insights into wasteful or automatable flows. (Also stated as important issue in: [Torry-Smith et al., 2011], and in case studies 5.5 Lithography System, 5.6 Paper Path)

3. Provide Design Information in a Context. Stakeholders need information to perform their design task. However, the persons who provide that information are not always informed on that task. Therefore it can occur that the provider makes a manual document with only the requested information, while the user could make better use of the underlying model. Also, it can occur that important design decisions become 'design rules' in an organization, the rationale behind the decision may not apply anymore, however there is no way of knowing (especially when the decision maker has left the organization).(Also stated as important issue in: [Schöner, 2004],[Wang et al., 2002], [Torry-Smith et al., 2011],[Wasiak et al., 2009], [Reichwein and Paredis, 2011], [Jackson, 2006], and in case studies 5.5 Lithography System, 5.6 Paper Path)

4. Prevent Ambiguity. When design information is not stored centrally and uniquely, it can happen that stakeholders will use data with different versions, units and values, leading to inconsistencies in the design, which in turn can lead to sub-optimal, or even failing systems. Therefore important shared information must be stored unduplicated in a shared model. Different stakeholders can then extract the information from that model or update

the model for all other stakeholders integrally. (Also stated as important issue in: [Wasiak et al., 2009], [Reichwein and Paredis, 2011], and in case studies 5.4 Baggage Handler, 5.6 Paper Path)

5. Check the Requirements. A clear feedback of intermediate design steps to the requirements is often missing. The system's architecture description that captured these requirements is often only used as a conceptual model. This can lead to a deviation from the requirements that will only be noticed at the prototype phase of the design process, when the system is tested. Therefore checking the intermediate design specification against important design aspects needs to be part of the design architecture description, which needs to be followed throughout the design process. (Also stated as important issue in: [Boucher and Houlihan, 2008], [Aberdeen Group, 2007], [Nuseibeh, 2000], and in case study 5.6 Paper Path)

6. Tacit Understanding. The 'mental model' the stakeholders have of the system or of the lexicon they use to explain things to each other is often not in agreement. As a result, people will have difficulty coming to a mutual understanding or agreement, or worse, think that they know what the other thinks, but actually think something else. (Note. The A3 method of [Borches Juzgado, 2010] targets this sense making and consensus making process specifically). (Also stated as important issue in: [Wang et al., 2002])

7. Modeling Design Decisions. As a design process progresses, design decisions will lead to a growth of the design specification. However, the decisions could be based on incorrect assumptions, or the contexts (e.g. technology) of old design decisions can become obsolete. Therefore, it would be useful to have a model of the design decisions, modeling the rationale and the available context of that time, so that information can be reviewed from time to time. (Also stated as important issue in: [Torry-Smith et al., 2011], [Reichwein and Paredis, 2011], and in case study 5.6 Paper Path)

8. Provide Traceability. Allow stakeholders to find the where/who/when/why for important design information. Documentation will often not be updated and distributed for every changing or new design decision. Design decisions taken between two stakeholders at the coffee machine can have influence far beyond the scope of the tasks of the two stakeholders. This information on the decision makers should be stored together with the actual design information. Ideally we would like to have a model that can trace all design parameter values up to the causal design requirements and down to the resulting design specifications, as this will make measuring system performance possible at every stage of the design process (Also stated as important issue in: [Wang et al., 2002], [Torry-Smith et al., 2011], [Wasiak et al., 2009], [Boucher and Houlihan, 2008], and in case studies 5.4 Baggage Handler, 5.5 Lithography System)

9. Ownership of Shared Design information. If information is shared between stakeholders, someone must be responsible for and authoritative of the content. This ownership of information must be clear, so it can be used to engage in (1) finding the rationale behind the information, (2) negotiations to change the content of the information. (Also stated as important issue in: [Wang et al., 2002], [Torry-Smith et al., 2011], [Wasiak et al., 2009], and in case studies 5.5 Lithography System, 5.6 Paper Path)

10. Consensus Making. During a design process, situations will arise where different stakeholders have different ideas of how to proceed. During meetings they will negotiate and compromise trade-offs resulting in a design decision, which is then promulgated to other stakeholders. This consensus making is not properly supported by modeling techniques. (Also stated as important issue in: [Nuseibeh, 2000], and in case study 5.5 Lithography System)

11. Versioning. Whereas there are tools to do versioning on files of models, there is a lack of tools that support versioning the decisions that led to those models or version the content of the models in the context of the other design information. As such, stakeholders cannot see if changes in their model will have influences on models elsewhere.

12. Allow for Emergent Properties. [McConnell, 2000] says: "Systems do not exist except through the fact that someone has termed it a system, a system is there for a purpose, it must be considered within a context." This statement is important in two ways. First, various stakeholders will have various (but equally valid) ideas of what the system is, therefore there can be multiple decompositions and conceptualizations of a system. Second, as a design process evolves, properties of the system will emerge, that may not fit into an original conceptualization of the system, as emergent properties of a system are often properties of the system as a whole. Therefore the often-used reductionist strategy of decomposing a system in parts will not always work. To mitigate this problem, we can do two things: Flexible views should be employed to fit multiple possible decompositions, and cross-decomposition mappings of relations. Aspect objects should be employed to capture non-structural or non-functional or non-requirement system properties. Aspects model cross cutting concerns of the stakeholders. (Also stated as important issue in: [Torry-Smith et al., 2011], [Boucher and Houlihan, 2008], [Nuseibeh, 2000])

13. Automate Hand-over Generation. Stakeholders often summarize their models and design tasks in a design document, which they hand over to another stakeholder. However, the generation of these documents increase the (overhead) work load of those stakeholders while they do not have any benefit of the documentation themselves. This increased work overhead reduces model update and feedback instances and is prone to consistency errors coming from transcription errors. The manual hand-over documentation is not versioned properly; at any subsequent stage a stakeholder could use obsolete or incorrect values. Automating this hand-over through a shared data model limits the risks of these issues. (Also stated as important issue in: [Torry-Smith et al., 2011], [Aberdeen Group, 2007], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

14. Do Not Duplicate Information. So no multiple versions can come into existence, preventing inconsistencies or obsolescence. During a design process a stakeholder could use an obsolete dataset, because the supplying stakeholder already made a new version as a result of new design decisions. Storing design information in a central and unique place can mitigate this issue. (Also stated as important issue in: [Wang et al., 2002], [Wasiak et al., 2009], and in case studies 5.4 Baggage Handler, 5.6 Paper Path)

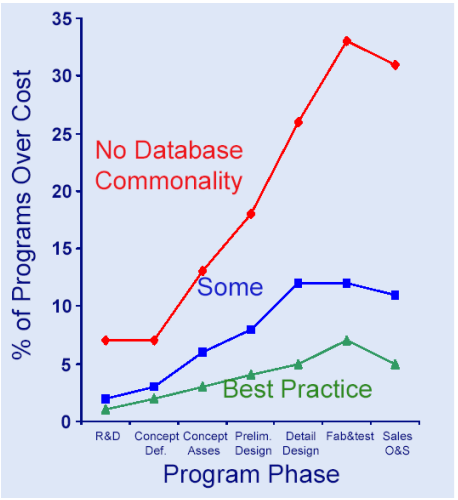


Figure B.1: [Hoult et al., 1996] shows that having a common data language in a design process (Lean Aircraft Initiative) will prevent cost over-run by (1) Easy availability of design information, making decisions based on data from multiple disciplines and (2) Communication of emerging knowledge to enable learning and innovation.

B.2 Integration Issues

15. Communication Distance. Efficiently sharing design information directly among stakeholders within a design process becomes unmanageable when the design team can no longer work side by side (in project stage, design discipline or physical workspace). The developed models and documentation do not efficiently support sharing or transferring information. Generation and interpretation of manual documentation slows down the development process, while not adding any information to the design specification. (Also stated as important issue in: [Torry-Smith et al., 2011], [Aberdeen Group, 2007], [Jackson, 2006], and in case studies 5.5 Lithography System, 5.6 Paper Path)

16. Shared Data Model. A shared data model is needed to exchange information through software tools. This will leverage the functionality of information and communication technology to improve the design process. It facilitates information transformation between different models and different stakeholders, making it easier to (1) ensure the information remains consistent, since it is stored centrally in the shared data model, (2) integrate the work of the various stakeholders (see figure B.1), and, (3) speed up the design process through the reuse of design information. (Also stated as issue in: [Schöner, 2004], [Wang et al., 2002], [Torry-Smith et al., 2011], [Jackson, 2006], and in case studies 5.4 Baggage Handler, 5.5 Lithography System)

17. Modeling and Communicating System Architectures. System architectures are modeled in system architecture descriptions. As such they capture 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 [Hilliard, 2007]. There is a need for an architecture model that can capture more design information (see other issues),

and can be used to capture the current state of the design specification, and thus facilitate communication between multiple stakeholders. As such, the architecture model can be the basis for model based system engineering: the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases [INCOSE, 2006]. (Also stated as issue in: [Schöner, 2004], [Wang et al., 2002], [Torry-Smith et al., 2011], [Boucher and Houlihan, 2008], [Nuseibeh, 2000], [Reichwein and Paredis, 2011], [Jackson, 2006], and in case studies 5.5 Lithography System, 5.6 Paper Path)

18. Multi-View System Architecture. Modern approaches for Systems Engineering or Architecting have a limited set of specialized views (use-case, functional, requirements...) and often use a single structural decomposition (classes, parts, components) as a core. However, combining information from different views in a new view is not possible in many of the available tools. This does not agree with the multi-view way stakeholders think about a system (Can this function be performed by this entity within requirements? Does a certain aspect of the system adhere to the requirements?). Having multiple decompositions is often not supported in the existing tools, however it is a fact in the domain specific models of the stakeholders (Software model, Bill of Material, Assembly model). (Also stated as important in: [Torry-Smith et al., 2011], , and in case studies 5.5 Lithography System, 5.6 Paper Path)

19. Multi-Composition. Allow for the definition of multiple system decompositions, for different phases of the design process, for different aspects of the design, for different stakeholder concerns. (Also stated as important in: [Torry-Smith et al., 2011], [Reichwein and Paredis, 2011], [Jackson, 2006], and in case studies 5.4 Baggage Handler, 5.5 Lithography System)

20. Multi-Mapping. Allow the cross-referencing of different types of information, to provide context for the interrelatedness of design information, make sure of the uniqueness of design information to keep the multiple compositions consistent. (Also stated as important issue in: [Torry-Smith et al., 2011], [Reichwein and Paredis, 2011], [Jackson, 2006], and in case studies 5.4 Baggage Handler, 5.5 Lithography System)

21. Capture Design Decisions. Communication between various domains can be realized through meetings. At these meetings, project overview and progress, the system decomposition and interfacing, work integration, and other design decisions are discussed. Often, the results are documented in text documents and stored in an archive, without anyone reading them again. A living design decision reference model can be maintained, and through a communication framework, can be used to store and keep track of these design decisions, making information accessible for everyone online. Reverse engineering existing design decisions is a laborious but often very insightful process for a company, as Borches explains in [Borches Juzgado, 2010]. (Also stated as important issue in: [Torry-Smith et al., 2011], [Aberdeen Group, 2007], and in case study 5.6 Paper Path)

22. Design Process Tasks. The design process consists of work done by stakeholders in design tasks. These tasks are performed for a certain purpose, given a set of assumptions and facts recorded in models, and will lead to results, often also recorded in models. It would

be helpful to make a model of these design tasks in the context of the models, purpose, and input/output needed to perform them. In the companies associated with the AGCCSMS project, the modeling of design tasks gave important insights into making their design process more 'lean'. (Also stated as important issue in: [Wang et al., 2002], [Torry-Smith et al., 2011], [Aberdeen Group, 2007], and in case study 5.5 Lithography System)

23. Simple Shared Data Language. Define a small set of modeling primitives to support communication independent of stakeholder's background. Graphical representations are of great value for this purpose, as long as the amount of graphics remains limited. The data language must not be used to make a model of all information of the organization, as that will make it complex very fast. Focus on the shared information and important organizational goals for the design process. (Also stated as important issue in: [Torry-Smith et al., 2011], [Jackson, 2006], and in case studies 5.4 Baggage Handler, 5.5 Lithography System)

24. Abstract Model Content and Reference to Models. Enable referencing external, domain specific data and information to a common model, to facilitate sharing of information between stakeholders, and provide context for that information. Do not simply 'copy' all the information of the external model, just use an abstract description of the content, and reference the pieces of the model that are shared by other stakeholders. (Also stated as important issue in: [Wang et al., 2002], [Torry-Smith et al., 2011], [Reichwein and Paredis, 2011], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

25. Support Parametric Definition. In a design process a lot of information is exchanged. We can differentiate between the information content(design decisions, assumptions, model input/output, (system) properties and attributes, etcetera.), the information context (ownership, views, functionality, design aspects, requirements, structure, behavior, version, etcetera.) and the information carriers (conversations, meetings, emails, documents, models, etcetera.). We will define a parameter as a carrier for a unique piece of such information content. With this definition, a design will become a vast network of causally interrelated parameters mapped to various contexts, and a design process will become a process of instantiating, connecting, and valuing parameters. (See case studies 5.4 Baggage Handler, 5.6 Paper Path)

26. Support Parametric Integration. A stakeholder will have a set of parameters in a certain context. Most parameters will be specific to the stakeholder (the domain specific content of a model, such as the layout of a diagram, a set of finite element equations, or a snippet of software code). However the stakeholder will share a number of these parameters with other stakeholders. For example, the constraints to build a model are provided and the model results will be supplied to someone else. If a parameter is shared, its values need to be communicated between stakeholders. Therefore it stands to reason to store the shared parameter in a shared model, thus integrating the parameter mappings and causalities. (See in case studies 5.4 Baggage Handler, 5.6 Paper Path)

27. Mechatronics Design. [Alvarez Cabrera et al., 2010]: Developing mechatronic products requires intensive collaboration between engineers of the mechanical, electronic, control, and software domains in a design team. A central issue is that design decisions cannot be taken from the point of view of a single domain, as often they will have an impact in other

domains. The procedure in traditional sequential design is that the mechanical design has to be 'frozen' before proceeding to the design of control software. This responds to the required preparations for production of the hardware, while for the software still last-minute changes may occur, sometimes to fix problems from the rest of the design. This approach usually does not lead to optimal overall behavior, since it does not properly address the interaction among mechanical, electronic, and control behaviors. Furthermore, it does not reflect the importance of software design, which can have a major impact on overall system design and performance. Development of mechatronic systems therefore requires collaboration among experts from different design domains. The challenges are mostly related to integration of design and analysis tools, and automation of current design practices. Addressing these challenges enables the adoption of a concurrent development approach in which the synergetic effects that characterize mechatronic systems are taken into account during design. The main argument is that in order to deal with software development problems for complex mechatronic systems, there is a need to look at system design practices beyond concurrency, i.e., there is a need to consider the complex interdependencies among subsystems and the designers that develop them. (Also stated as important issue in: [Boucher and Houlihan, 2008], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

28. Align to Functions. Describe the behavior of the system and the important design aspects as they relate to desired functionality, as these functions are discipline independent. Structural parts or components will be function enablers, developed in the traditional design disciplines. (Also stated as important issue in: [Bonnema, 2008], [Schöner, 2004], and in case studies 5.4 Baggage Handler, 5.6 Paper Path)

B.3 Reuse Issues

29. Meta Modeling. Provide a central model that defines what design information is needed to instantiate to or parse from other models. (Also stated as important issue in: [Torry-Smith et al., 2011], [Reichwein and Paredis, 2011], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

30. Reference Modeling. Provide a central model that explains the who, what, where, when, why context of information. Provide abstracted description of design information stored in external models. (Also stated as important issue in: [Wang et al., 2002], [Torry-Smith et al., 2011], and in case studies 5.5 Lithography System)

31. Model Multi-Domain Information. Design specifications are a composition of information from various design disciplines. Because mono-domain models and software tools cannot always communicate with each other directly, design information must first be abstracted to a shared common format in order to be communicated. We can use this mechanism to also provide the context of the models, their input and output, ownership, versions etcetera. This information can be stored in a central meta model and reference model. (Also stated as important issue in: [Wang et al., 2002], [Torry-Smith et al., 2011], [Reichwein and Paredis, 2011], [Jackson, 2006], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

32. Defining Reusable Design Primitives. A mechanism that can keep track of design resources and external models can also be used to store reusable versions of that information. Use this to develop multi-disciplinary toolboxes for model generation and analysis. (Also stated as important issue in: [Wang et al., 2002], [Aberdeen Group, 2007], [Nuseibeh, 2000], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

33. Defining Reusable Design knowledge. Because there are a lot of possible repositories for reusable knowledge, (specialized design tools, macro's in existing tools, design configuration sheets, knowledge based engineering tools, etc...) the storing, language, format, and modeling of design knowledge must not be prescribed. Instead, it must be possible to 'plug in' knowledge bases into a framework that can analyze and synthesize models. (Also stated as important issue in: [Wang et al., 2002], [Aberdeen Group, 2007], [Nuseibeh, 2000], [Jackson, 2006], and in case studies 5.5 Lithography System, 5.6 Paper Path)

34. Optimization. When design knowledge is automated in a software tool, it can be used to quickly iterate a design problem for various inputs. Therefore, one can look to a set of possible design solutions to the requirements and constraints to see which one fits the goals of the design the best. Or one can 'quickly' converge to an allowable solution for a algorithmically complex problem. This enables local optimization of the design – local because the tool can only optimize what that specific tool can model. (Also stated as important issue in: [Aberdeen Group, 2007], and in case study 5.6 Paper Path)

35. Impact Analysis of Reusable Design Primitives. When synthesizing design specifications automatically, the individual design primitives can originate from multiple disciplines. When we consider a company employing a hundred engineers and designers all working with, and expanding libraries of, these primitives, we can see that managing such libraries can become a problem. If someone re-writes a block of control code for a certain motor control, that person perhaps does not know that that software component is also used in another motor-control design object. The libraries of these primitives should be abstracted into a shared model, where the interrelatedness of primitives can be checked, and thus can be maintained consistently. (See case study 5.4 Baggage Handler)

36. Lean Models. Automatic modeling of a design specification using reusable design primitives must go fast: designers tend to dislike waiting on computers, even if a lot of calculations need to be made. (This waiting time is very relative to the expectations of the engineer of the task at hand, waiting two seconds can be long in a structure synthesis of a baggage handling system, while two hours can be fast in a finite element calculation of a turbine engine). This means the reusable primitives should be as 'lean' as possible. The primitives should contain just enough (but no less) information to describe an implementation. Much of the domain-specific model information can then be made available through referencing external information. (Also stated as important issue in: [Walton, 1999], and in case studies 5.4 Baggage Handler, 5.5 Lithography System)

37. Use Existing Tools. Clearly, changes in the design process must be carried by the engineers who work with them. Specialists want to design within their known domain tools and models, not in some new design tool (see also [Jackson, 2006]). Therefore, any changes in the way of working must have a low learning curve and must not replace functionality

of existing tooling. The use of existing tooling is often also cheaper than investing in a new tool suite. (Also stated as important issue in: [Wang et al., 2002], [Jackson, 2006], and in case study 5.6 Paper Path)

38. Platform Development and Support. If a company has a product line that can be designed with a high degree of automation, a dedicated department for developing technology for the analysis and synthesis tools might be needed to automate, maintain and improve the design process, and support communication across the disciplines. These system-independent investments over the long term are often not well 'sold' to higher management. Therefore it is necessary to be able to 'retro-fit' a reusable primitive method on existing modeling systems without too much addition of new tools. [Muller, 2008] describes some trade-offs involved in deciding when to start technological standardization. (Also stated as important issue in: [Wang et al., 2002], [Reichwein and Paredis, 2011], [Jackson, 2006], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

39. Freedom vs. Speed vs. Consistency. Clearly, design freedom, and speed & consistency conflict with each other. Good reusable primitives decrease the time needed to make a model of a system, and a primitive will always be implemented from a library in a predictable and consistent way. However, the contents of the primitive are not easily changed when a situation calls for it – the design freedom is restricted to the various configurations allowed by its parameterization. If a designer cannot use the available design primitives, a new reusable design primitive needs to be developed, which will take more time than a one-off primitive. Therefore there is a trade-off of developing reusable information, and the tooling to support its usage. It may sometimes be too expensive. Furthermore, every technology changes: components get updated, tools are replaced, and new design methods are implemented. Pre-defining the parameterization of a reusable primitive means it cannot be re-defined without risk. Each primitive has been engineered and tested for integration in a certain context. If the primitive is changed, there is no certainty that it can still be integrated into a new system without errors. The primitives must be constructed in such a way that domain specific content of the primitive can be replaced or updated without throwing the entire primitive away. Version control of the primitive must then be applied to ensure consistency. [Scherer and Katranuschkov, 2006] defines a procedure to keep parametrically defined chains of models consistent during changes, however, the actual algorithm and implementation of these types of methods will be out of scope of the thesis. (Also stated as important issue in: [Nuseibeh, 2000], and in case studies 5.4 Baggage Handler, 5.5 Lithography System, 5.6 Paper Path)

Table B.1: Issue index versus Case studies. Which issues concern which case study?

Index	Issue	Baggage Handler	Litho System	Paper Path
	Improve Integration			
1	Big Picture	X	X	X
2	Overview of the Design Information Flow		X	X
3	Provide Design Information in a Context		X	X
4	Prevent Ambiguity	X		X
5	Check the Requirements			X
6	Tacit Understanding			
7	Modeling Design Decisions			X
8	Provide Traceability	X	X	
9	Ownership of Shared Design information		X	X
10	Consensus Making		X	
11	Versioning			
12	Allow for Emergent Properties			
13	Automate Hand-over Generation	X	X	X
14	Do Not Duplicate Information	X		X
	Improve Integration			
15	Communication Distance		X	X
16	Shared Data Model	X	X	
17	Model and Communicate System Architectures		X	X
18	Multi-View System Architecture		X	X
19	Multi-Composition	X	X	
20	Multi-Mapping	X	X	
21	Capture Design Decision			X
22	Design Process Tasks		X	
23	Simple Shared Data Language	X	X	
24	Abstract and Reference Model Content	X	X	X
25	Support Parametric Definition	X		X
26	Support Parametric Integration	X		X
27	Mechatronics Design	X	X	X
28	Align to Functions	X		X
	Improve Reusability			
29	Meta Modeling	X	X	X
30	Reference Modeling		X	
31	Model Multi-Domain Information	X	X	X
32	Defining Reusable Design Primitives	X	X	X
33	Defining Reusable Design knowledge		X	X
34	Optimization			X
35	Impact Analysis of Reusable Design Primitives	X		
36	Lean Models	X	X	
37	Use Existing Tools			X
38	Platform Development and Support	X	X	X
39	Freedom vs. Speed vs. Consistency	X	X	X

C. Workshop on Systems Architecture Modeling

Apart from using the focus group and the expert panels as a means to validate the research (§5.2), a second method was applied. To get greater insight into our work, from a perspective of people not involved with the AGCCSMS project, we conducted three workshops. The first was a general orientation into the field of complex systems modeling, and was not focused on the architecture modeling framework, the third unfortunately only got one participant. The second workshop did see a large response, and will be discussed here.

The *"Workshop on Systems Architecture Modeling, Hands-on experience with the Architecture Modeling Tool"* on 2 September 2011, centered on transferring the main practical research results and has been given to several industrial and academic participants (mainly from Océ, Philips, Embedded Systems Institute). The main purpose of the workshop was to show the use and need of system architecture models (AM), and to demonstrate how such models can be built and used with the help of the proposed AM language and tool. Additionally, the workshop featured a primer on architecture modeling methodology and example use cases covering scenarios ranging from data transfer to supervisory embedded software control generation. At a discussion at the end of the workshop it has been agreed that the activities made the main practical results very amenable to be used by the participants, that the proposed language has good potential to support stakeholder communication and design, that the transformation and generation cases provide good additional motivation to consider the proposal. However, the tool is not developed enough or robust enough for direct industrial implementation but it can support model management of the presented cases at a demonstration level.

Participants background and setup: Academic (6x), R&D management (3x), System architects / engineers (3x), Engineering / ICT (3x) (In total, there were six industrial participants; four from Philips and two from Océ.) The participants were given a workstation and a hotseat control over the architecture modeling editor. After an introduction on various possible use cases and theory of the architecture modeling framework, the participants could make a model of an assigned system. The hotseat control made it possible for three to four people to work together to model a system aspect of an (non existing) robotic vacuum cleaner. This hotseat setup could test the applicability of the AM framework as a facilitator of communication among people from different disciplines. Unfortunately, the time was limited to about 1.5 hours of modeling, which proved a bit short to master the tool and, more importantly, the AM language. After this session, a discussion ensued over the validity of the AM framework as a tool to help systems architecting and engineering. The next subsections show the results of that discussion.

C.0.1 Architecture modeling

One person expressed that in his profession he had not much to do with people of other disciplines. As such, he said they were not very interested in the 'integrating' properties of the architecture model, to connect their work with that of other people. However, he found that being able to model his own work abstractly, and the interfacing to others can be practical. Others replied that they found that the Architecture Modeling (AM) tool does exactly that, as it provides the means to make stakeholder specific views, which still provide overview and connections (interfaces) to other stakeholders. They found that modeling aspects as a combination of requirements, functional, structural and behavioral system descriptions and the modeling of the needed resources for this aspect was something they had not seen before. In their opinion, this provided added value of this AM tool over their normal way of working, where they normally used disparate documents in MS office (Word, Powerpoint, Visio), emails or UML to model this information.

Room for improvement: The lack of apparent interfacing between the design tasks of the stakeholders is being addressed in the version of the tool discussed in this thesis, where a concept called 'DesignTask' can be used to model workflow.

C.0.2 Architecture model versus normal documentation

The participants saw that having an information exchange model can be useful, but are concerned that making and maintaining such a model could take too much time. When asked how they exchange information now, they replied they mostly make text documents. These documents also take a lot of time to make and maintain, and can be hard for (technically oriented) people to follow, overview and discern. As such, these documents often defeat their purpose of informing people, and become extra workload for the designers who make them. Modeling diagrams in for example MS-Visio is often clearer, they found, but it takes a company-wide effort to develop the formalisms and shared vocabulary to use them effectively. When asked what then would be better, the AM or their original text documentation plus diagrams, the participants generally found a synergy of the two would be useful if one would use the AM as an index or root model for other documentation.

Room for improvement: All concepts have a Uniform Resource Identifier attribute to allocate external resources and documentation, which can be used for this purpose. A masters student [Melching, 2012] at the University of Twente has researched ways to connect the AM framework to the A3 architecture overview documents developed by [Borches Juzgado, 2010].

C.0.3 Other means of architecture modeling

A discussion of the used AM formalism versus mindmapping, Microsoft Visio or SysML diagrams ensued. The first two lack a formalism, making it hard to come to an agreement of how to model systems. This is a continuous process of improvement, which helps ar-

chitectural discussions within a company. However, the resulting models cannot be used in engineering directly. The SysML was found to lack the possibility to mix concept types, as it has specific model types for specific concerns (class, parametrics, process). The AM helped to define a number of formal concepts the participants did not think about before, and they saw the benefit of modeling with the chosen concepts: Linking parametric information, adding aspect information, limiting the concept types, separation and subsequent integration of 'who does it? - aspects, views, domain entities', 'what does it do? - requirements, functions, entities', 'how does it do it? - parameters, formulas, dependency, flow relations'. To be forced to use only a small number of concepts could help clarify the resulting models. However, most participants had trouble finding out what the concepts were for, of when to use which concept. With some help, they did figure it out, but a more helpful training document is needed.

Room for improvement: What is missing in the AM language was a 'type of' relation, so it would be possible to instantiate a system model from a meta model. This possibility is being looked into. For now, 'copy' functionality was added, that instantiates a new piece of model from a design primitive. This can be upgraded to more object oriented instantiation.

C.0.4 Architecture model in practice

According to the participants, the workshop made it clear that it is actually possible to model architecture, and gave a number of insights in how to do it, even from a practical standpoint of using it in a company. Diagrams can become cluttered quite fast, which limits the readability. This is a problem that was addressed in a number of ways. The concept types have distinct shapes and colors, the user can make sub-views by hyperlinking existing information, and the tool has filters that can tidy-up diagrams. The automatic inclusion of 'user' information (who made it, who changed it) in all the concepts makes it possible to assign and track 'ownership' of the modeled information. This kind of information is in current practice often omitted, or documented somewhere apart from where that information is useful or accessible. The AM provides a means to connect management concerns (e.g. key performance indices, cost, ownership) down to the practical synthesis and analysis models of the engineers. As such, the model can provide practical metrics to facilitate discussions between management and product development.

Room for improvement: Some participants would like to see the ability to define scenarios. These scenarios could then provide a sort of 'load case' for an architecture model to test concepts. This ideally would require the modeling of class-instance types plus a knowledge based semantic model to provide the instancing. This is out of scope of the AM language, however, the same result can be achieved somewhat. The framework allows for the plugging in of algorithms for model generation. These should be custom made on a case by case basis. This way, a scenario AM can generate a system AM, however, this is not tried yet (although generating external models works, as demonstrated in the other case studies of this chapter).

Introducing a new 'choice' relation type could also help: the 'choice' relation could be a triggered decomposition that specifies a configuration option given a certain requirement or function. Another option would be to just make multiple AM's for multiple scenarios and system concepts with a shared set of performance metrics.

D. Interview Guy Stoot

Email interview, conducted with Guy Stoot, 3rd of september 2013

Guy Stoot is a systems architect for the R&D department of Océ. The interview was a short one, as Guy Stoot was in the process of starting a new career at DAF Trucks N.V.

How did you come into contact with the researchers?

My former departmental head arranged a meeting.

What were the reasons for OCE to participate in the research project?

Find solutions at academic level for the challenges we face in the area of information structuring and using the information for several applications (e.g. code synthesis).

What was your role in the research project?

My role was to provide a real life use case from the industry, to contribute in finding solutions and testing the solutions in practice.

Do you think the AM framework adds value to the tools OCE uses in their design process?

The concepts in the AM Framework really helped us to get a clear understanding of how the information can be structured and which concepts are required in a tool to support this structuring. There is however no tool available yet with these concepts that can be used in a production environment, and therefore no Oce tools are connected to the framework.

If yes, in what way does the framework add practical value?

- Provide the essential overview in the architecture and design of a complex mechatronic product.
- Allow to make changes in this architecture and design without creating overlap in information, which potentially could undo the consistency of information. These changes can be made by several people in the same model.
- Stored parameters can be used in external models.
- Stored parameters can be used to synthesize code.
- The model can be used to find which aspects of the design are affected by a change and even determine the quantitative impact of such a change
- Perform Unit checks.

Do you use the AM framework in your daily work?

Not the tool, but the concepts are certainly used.

Do you have practical examples?

I currently use the concepts to model a complex design, in order to get a good overview which enable is to redesign the object. The layering functionality of MS Visio is used to create 'views'. Of course using the parameters in external models is not possible in this way.

What do you think are the main conceptual and technical shortcomings of the AM framework?

- Conceptual: Managing changes of multiple users in one model.
- Conceptual: Calculate the effect of parameter changes in both directions. Example: In F=m*a change a and calculate F or change F and calculate a.
- Not enough control over automatic reordering of objects in a view.
- Better exploration functionality.

What is needed for the AM framework to be a more acceptable tool in systems engineering?

Robustness of the tool. Simple installation and use of the tool since mostly non-software colleagues should work with it. Entered data should be exportable to make it tool-independent, for the case the tool is not supported any more.

E. Architecture Modeling Language

E.1 Generic Attributes

There is certain design information that needs to be defined regardless of what system is modeled in the Architecture Model(AM). This information can therefore be attributed to every concept type in the architecture language. The information specified is used for basic information management functionality, and will be discussed per attribute:

- `name` – of the piece of information, name of the concept.
- `type` – for namespacing the concept in its taxonomy (e.g. 'Solidworks.Part', 'integer', 'Layout.Segment', etc...).
- `user` – to define ownership of this information. The value is initialized on the name of the owner of the computer that instantiated this concept. The user attribute can be used for traceability of ownership through search algorithms.
- `identification` – a unique (128 bits) identifier for consistency purposes. In case some concept would have the same values for the name, type, etcetera, information must be defined uniquely. The identifier can be used to perform searches and navigation of the information network, for example to exclude already parsed information in a recursive search algorithm.
- `date` – to specify when the information was last changed. The value is initialized on the date and time when this concept was instantiated.
- `status` – the status of the information in relation to the design process (e.g. 'initialized', 'unknown', 'final', 'changeable', 'under review'. etc). The usage of this attribute is not prescribed.
- `version` – to specify the version of this information to keep design decisions consistent over multiple versions of architecture models. The value is initialized as '1.0'. The usage of this attribute is not prescribed.
- `uri` – to provide an Uniform Resource Identifier to an external file or location. Currently used to reference external files in the case studies. Think of the server location of a model for a certain aspect of the system.
- `documentation` – to provide a rationale to this concept. The documentation can be a string of text, but can also be a uri to an external file, for example a word document or powerpoint presentation.
- `knowledge` – to provide a marker for knowledge based engineering tools (e.g. 'SolidWorksKB.Assembly.Mate', 'LayoutKB.Constraint', 'LayoutKB.Segment', etc...). A knowledge base can parse the AM and find all occurrences of concepts that conform with

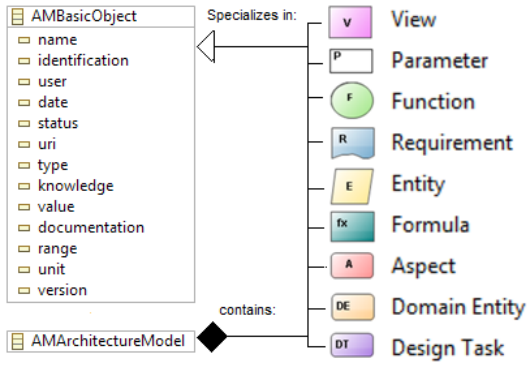


Figure E.1: The concepts used in the Architecture Modeling Language

a given 'knowledge' attribute value. The knowledge base then has the semantics to perform an action on the concept: for example to evaluate or calculate the value, or to evaluate or change the context of the concept by navigating, adding, altering or deleting concepts connected to the concept.

- `value` – current value of the concept, especially practical in parameters, to for example embody a design decision. The 'value' could also reference external data (row1, dataset1, 'D1@circle@feature1'). The usage of this attribute is not prescribed.
- `range` – allowable range of values. Think also of multi dimensional ranges (e.g. '[-1,1][-1,1][-1,1]'). The usage of this attribute is not prescribed.
- `unit` – unit of the 'value' attribute for consistency of information flow (e.g. '[double,double]' or 'm'). The usage of this attribute is not prescribed.

E.2 Concept Definition

The Architecture Modeling Language consists of a minimized number of modeling concept types. As discussed before, these types are a cross section of often used words in systems architecting and engineering, as well as in knowledge based engineering. The user will construct Architecture Models using these concepts as primitives. Every concept will than have a set of standard attributes to specify it, for example by giving it a name and a value.

- `BasicObject` – The standard attributes (of section E.1) are defined in a basic class called BasicObject. From this class the other concepts are specialized. The BasicObject is an abstract information unit used for basic information management functions, needed for all concepts, such as databasing, querying, and referencing external information. This class enables us to use reflection: the software can ask itself what it is, and what it is doing at runtime, which enables us to develop a lot of tooling for only one class, and then reuse it in derived classes (see also the meta model 4.3 section)

- `ArchitectureModel` – The container of the model. All data in the architecture models is stored in a list. This list is manipulated by the editors, modeler and external tools such as knowledge bases. The human interpretable information is presented visually in the other concepts.
- `View` – A view 'packages' part of the architecture model from a stakeholder's point of view. This enables us to keep the information humanly manageable and viewable. A view also is a container of the model, to provide hierarchy in the information, and to specify the ownership of information, but it can also 'shortcut' information that are defined elsewhere. Like the other concepts, the views are contained in an architecture model.
- `Parameter` – The most basic unit of information. The parameters define the possible states of a design specification, embody a design/solution space, specify interfacing data between stakeholders, specify input/output of models, etcetera. The parameters can define information content of all the other concepts, making the other concepts 'semantic containers' for the parameters. The parameters thus hold the important design decisions throughout the design process. They contain a name, value, unit and range, and sometimes a knowledge attribute that tells us which algorithm can evaluate the parameter.
- `Function` – The function describes what a system should do, or does. It follows the basic logic of `entities` perform `functions` within `requirements`. This separation of functions and entities is important to be able to define what a system needs to do, and what a system uses to do it. Functions are partly defined by customers (functional requirements) at the start of the design process, and partly defined by modeling them as they emerge during the embodiment and specification of the design.
- `Requirement` – The requirement is the 'border' of the allowable design space: the design description, and thus the system behavior must comply with the requirements. Requirements can be seen as key performance indicators of a system, as a design process works towards achieving the desired customer functionality within customer requirements. Requirements can also be applicable on other, non functional, aspects of the design (design for x, such as maintainability, manufacturability or cost price).
- `Entity` – The structural units (or artifacts, or objects) of a system. Entities perform the system's functions. Entities can be, for example, software objects, agents, components/parts/assemblies. A system can contain multiple compositions of entities, mostly one for every design discipline or system aspect in an organization (e.g. manufacturing assembly, software UML class diagram, bond-graph physics simulation). However, these entities are used to make a model of the entities that are shared in compositions in multiple domains, the architectural structure ('engine' for cars, 'conveyor' for transport systems, 'paper path' for printers). For entities that are only used in one discipline or design domain, or for one design aspect, the DomainEntity can be used. Entities can be seen as lines in a Bill of Material, whereas Domain Entities reflect more the (content of) CAD documents.
- `Formula` – Formula specify how parameter values change. They are also used to define how a flow of information, energy or material changes in case of an EntityRelation.
- `Aspect` – While the function-entity-requirement combination describes the system's behavior, structure and limits, the aspects add other information related to the design

process. An aspect is a quality or characteristic of a system or of the design process that is of interest to a stakeholder. Design for x, for example, specifies a number of such aspects that will be present in most design processes, such as design for manufacture, cost, reuse, usability, etcetera. Company specific aspects can be for example throughput, real-time performance or modularity. These aspects often cut across multiple design domains (e.g. electrical, mechanical, software), and as such form the modeling glue of the design process across these domains. Aspects can be evaluatable by connecting them to requirements.

- `DomainEntity` – A mono-disciplinary entity defined in a certain design domain. These concepts are used to define (fragments of) models with which the stakeholders work. That is different from the abstract entity, as the abstract entity is a shared view of the system. DomainEntities are the pieces of information a stakeholder needs to make a model of his/her/its aspect. Examples are blocks in a simulink model, features / parts / assemblies in a CAD drafting tool, excel sheets, a bill of material entry, a PLC code snippet. The domain entities can thus be used to make a model of the detail level information. Hand-over information (e.g. documentation in excel sheets) between design tasks can also be modeled with these DomainEntities.
- `DesignTask` – A concept used to make a model of a design task. For example, 'calculate the parameter values for a certain aspect given certain requirements' or 'make a 3D CAD model'. The DesignTask could implement algorithms pertaining to the connected information in the form of automated knowledge based engineering tools connected to the architecture framework.
- `FunctionRelation` – The interface definition between functions. (see relations below)
- `EntityRelation` – The interface definition between entities. (see relations below)
- `DesignTaskRelation` – The interface definition between design tasks. (see relations below)

E.3 Relation Definition

The Architecture Modeling Framework is used to integrate information from various stakeholders, disciplines, and design stages. This is made possible by defining relations between the concepts discussed above. The allowable relations are used to construct the information model of the design process, the Architecture Model. There are three types of relations, namely, the composition relation ('parent/child', 'has'), the dependency relation ('input/output', 'sequence') and the mapping (between different concept types). Other types of relations such as inheritance are still being considered and are future work. In this section we discuss the relations per concept according to figure E.2.

- `BasicObject` – This concept is used to provide the basic container for a parametric design, as it contains mappings to the parameters. As the BasicObject is specialized in the other concepts, all other concepts can map parameters as well.
- `ArchitectureModel` – not shown in the figure. The architecture model is the file that contains the whole architecture model. It can reference to concepts in other architecture models to build a large model of models. The architecture model is a flat list of

	BO	AM	V	P	F	R	E	Fx	A	DE	DT	FR	ER	DR	SF
BO		inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	
BO		con	con	map											
AM			inh												
V			com												
P				dep											
F					com	map	map		map			dep			map
R						com			map			map			
E							com		map				dep		
Fx		cross references											map		
A									com	map	map				
DE										com	map			map	
DT														dep	

Figure E.2: The allowable relations in the Architecture Modeling Language

concepts, which enables fast parsing and manipulation of the file by automatic tooling, but is not very human readable. Therefore the architecture models have a diagram representation, split in multiple views.

- `View` – The views are composable to build a hierarchy of the design according to the concerns of the stakeholder. The views can either contain other concepts directly (assuming ownership over that information), or they can contain shortcuts to these concepts. Through shortcuts the user can reuse concepts defined elsewhere, and keep shared information consistent (See section 4.3.1).
- `Parameter` – The parameter is mappable to all other concepts. As such, the parameters form a network of important values in a multi disciplinary design process. This network acts as the glue of the design information, as parameter networks can cross all domains and views. In figure 4.4, we saw how the parameters act as the glue between the important questions in a design process. In figure E.3, we see the different roles of the parameter to parameterize the design information: On an architecting level, the parameters give requirements a value, and state which variables measure the system performance. If the performance parameters' values fall within the requirements parameters' values, the design is verified. The parameters are mapped onto the various entities and functions of the system to provide overview for an architectural definition and evaluation. On the other side, the performance parameters are mapped to the system aspects that evaluate or define the system performance. The design tasks then define how to go from the architectural input of the design process to the output. The system engineering stage maps parameters to models that contain the engineering work – building a design specification that, when analyzed, performs within the requirements. The other concept types act as a 'semantic container' for the inherently domain independent parameter. For example, in a car, 'the radius' could clearly be a physical attribute of a wheel, however it could at the same time be an attribute of a speed calculation given a rotational speed of an axle, while at the same time defin-

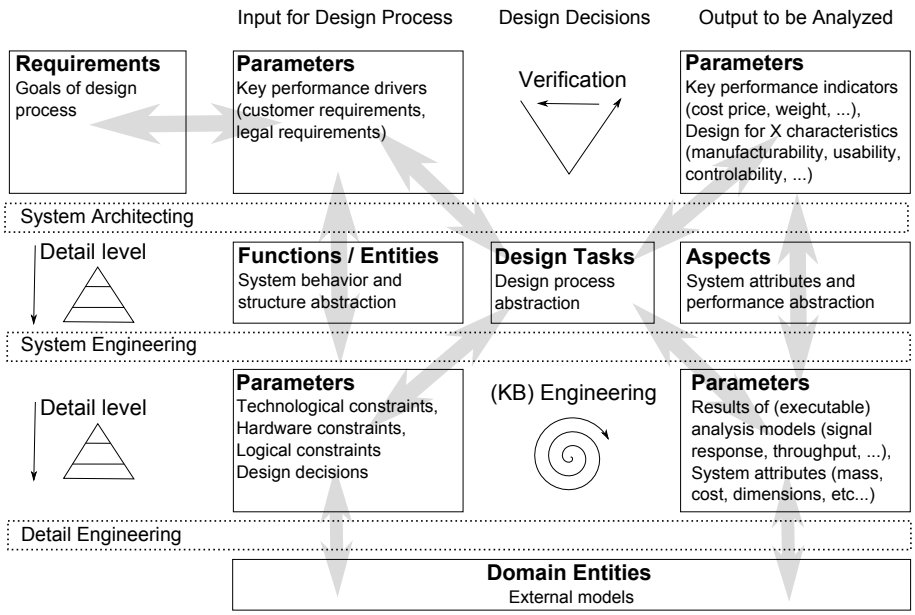


Figure E.3: Parameters capture important values in all stages of the design process.

ing the tire type needed for a bill of material. It can also be placed in a dependency chain of parameters, where the input dependencies provide the mapping to calculate the current parameter value, and the output dependencies provide the mapping to the parameter values that can be calculated from the current parameter values. No semantic knowledge is added in the parameter definition that tells us how the calculation is made, as we consider that makes the model too big. External tooling, in the form of knowledge based engineering software, can however be built on top of the framework to do such calculations – the 'knowledge' attribute of the BasicObject can then be used to refer to that specific knowledge base.

- `Function` – The functional decomposition is the basis for many architecting activities, which is why we added the composition relation to functions. However, from a system behavior point of view, a function can also be seen as a process when it is linked to a sequential (time based) input and or output. Therefore we introduced a dependency mapping to FunctionRelations, to build a state-based model. A further mapping to functions can then suggest a flow through function decompositions. While these dependencies make a model of the abstract behavior, we also need mappings to the entities that perform the functions, and mappings to the requirements that constrain the functions. As such, we can map different entities to perform the function, and map entities can perform multiple functions. We also map the aspects to the functions instead of to the entities, as the aspects are often technology independent design concerns, and the entities are the embodiment of the technologies used (see aspect).

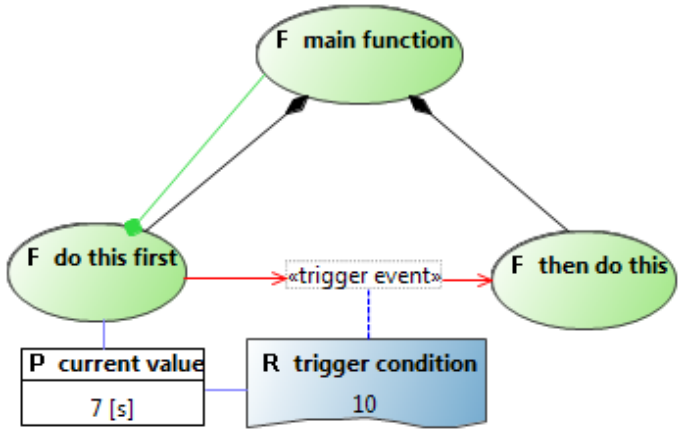


Figure E.4: A combination of Functions and FunctionRelations can make a model of a process.

- `Requirement` – The requirements can be decomposed and then mapped to the different functions of the system, in order to make a model of the customer desired performance indicators of the system ('mow grass' limited by 'max 50 Watt'). They can also be mapped to the FunctionRelations to make a model of important process constraints (such as triggers in behavior, see figure E.4). Requirements can also be mapped to other, non functional, design aspects (keep the cost below 50 dollar).
- `Entity` – The entities can be decomposed into the various abstract system components. These decompositions can then be mapped to the functions to make a model of which entities perform the function ('mow grass' with a 'grassmower'). Entities can also have dependency relations such as 'power supply' gives stuff to 'motor' or 'layout design' will lead to 'component design'.
- `FunctionRelation` – a class that is used in defining the interface between functions. This is used to make a model of temporal phenomena such as workflow, processes or activities. The FunctionRelation can be constrained by a requirement, to make a model of state transitions or allowable ranges of values (see figure E.4).
- `EntityRelation` – a class that is used to link entities. With these, one can make a model of non-temporal information flow between entities. For example materials, energy or information.
- `DesignTaskRelation` – The interface definition between design tasks. In a workflow setting the EntityRelation can also make a model of the documentation or information an entity needs by mapping it to a DomainEntity.
- `Formula` – As it can be mapped to EntityRelations, it is used to make a model of transformations inside the flow between entities.
- `Aspect` – Aspects can be decomposed into smaller sections, as to better map them to the functional decomposition. Aspects can be used to check cross cutting requirements: A case study example (from chapter 5) of a print engine is the maintainability

aspect, which has a child aspect called mechanical stability. This stability makes sure, that when the engine is opened for maintenance, it does not tip over and falls on the toes of the maintenance person (or breaks the machine). The aspect needs information cutting across the design domains, as it maps to the parametric information of the geometry domain (the guide rail length and the position of the frame) and the physical domain (masses of the extended engine vis-à-vis the rest of the engine). By defining these mappings to the pertaining DomainEntities containing these Parameters, the aspect can be checked or verified throughout the design process. Mappings to DesignTasks could be made to automate this checking procedure.

- `DomainEntity` – these can be decomposed to mimic the decompositions in the various design domains. For example, Assembly has parts has components has features, or Simulink has SystemBlocks has Blocks has Ports. The information needed in or from these models can then be represented by mapping parameters, which can read or write to the domain models using a model transformation tool. This can be automated by implementing a DesignTask that does this for a certain context.
- `DesignTask` – These can be mapped to domain models for extracting or implementing information in external models automatically, or to aspects to construct or retrieve information across domains automatically.

F. Tool Examples

In figure F.1 the various user interfaces are depicted:

1. *Current view on the AM* – The diagram of the stakeholder's view on the design information.
2. *Project tree* – Navigational aid to keep track of multiple AMs.
3. *Current AM project, toolbox of templates, and Car AM* – Example of two AMs working together. The Toolbox AM contains templates to build the Car AM, see point 7. From this project view, manage exchange of design models and data, by modeling the interconnectedness of models.
4. *Content of AM for navigation* – The navigator can access the AM in an ordered list representation, to easily make references across views or AMs.
5. *Concept types for AM construction* – The Palette shows the available concept types, the AML, with which to construct the AM.
6. *Currently selected concept and its attribute values* – The contents of a concept are only partially represented, often only the relations of the concept to others, and its name. Most attribute values can be accessed through a properties view in the bottom of the screen.
7. *Context menu for reuse of concepts* – Information can be reused or referenced across AMs. 'New instance' makes a unique copy like instantiating an object from a class, ' shortcuts' put in references to the original concept, thus linking AMs or views together while keeping the underlying data concept unique and consistent. Reusing information in various views supports consistency of the design process.
8. *Diagram layout, filter and format options* – Aids for representing the diagram: scaling concept shapes, changing connector shapes, ordering, filtering and laying out concept shapes.
9. *Custom filter for AM* – Filters the selected shapes, to make a view only partially visible, when a user does not want to make a whole new view.
10. *Make a shortcut to adjacent nodes* – A search algorithm will add all referenced concepts from the selected concept to the diagram. This can be used as a navigational tool to sift through the data of the AM not represented in the view. For example, 'represent all concepts related to the currently selected requirement' could be used to construct a performance aspect view.
11. *Export current view as a new AM* – Can be used when a user wants to split off a new project or AM from a view – for example as a test environment without changing the rest of the 'official' AM, for new projects, or for new versions or configurations.

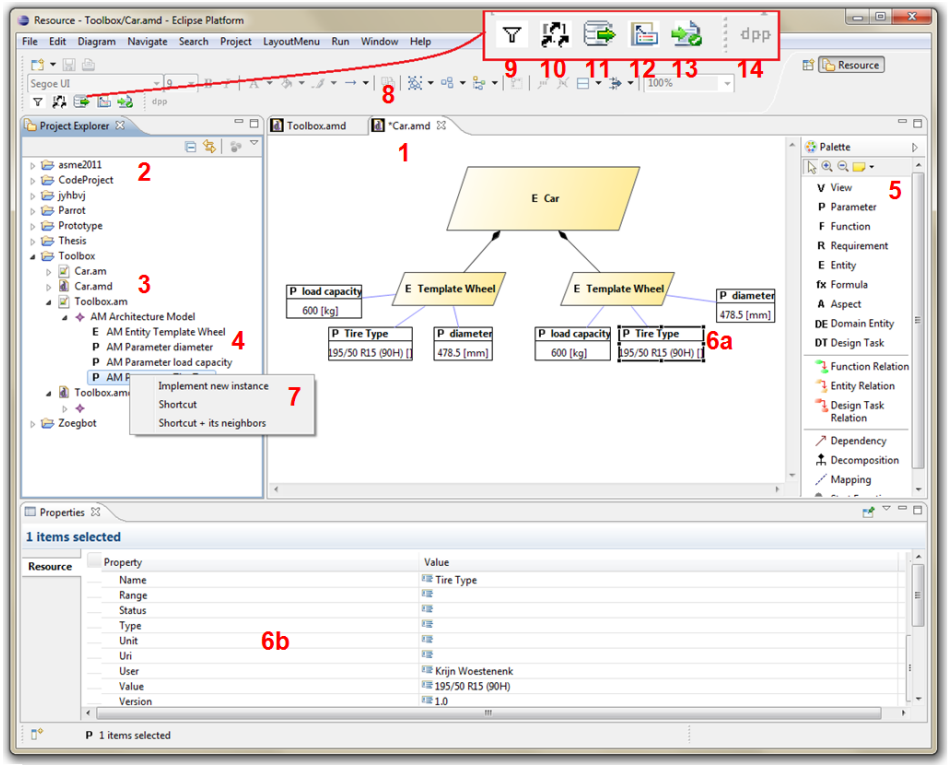


Figure F.1: The architecture model (AM) diagram editor. (1) Current view on the AM. (2) Project tree. (3) Current AM project, toolbox of templates, and Car AM. (4) Content of AM for navigation. (5) Concept types for AM construction. (6) Currently selected concept and its attribute values. (7) Context menu for reuse of concepts. (8) Diagram layout, filter and format options. (9) Custom filter for AM. (10) Make a shortcut to adjacent nodes. (11) Export current view as a new AM. (12) Add subscription to view. (13) Verify current view. (14) Run design process patterns if contained in this view.

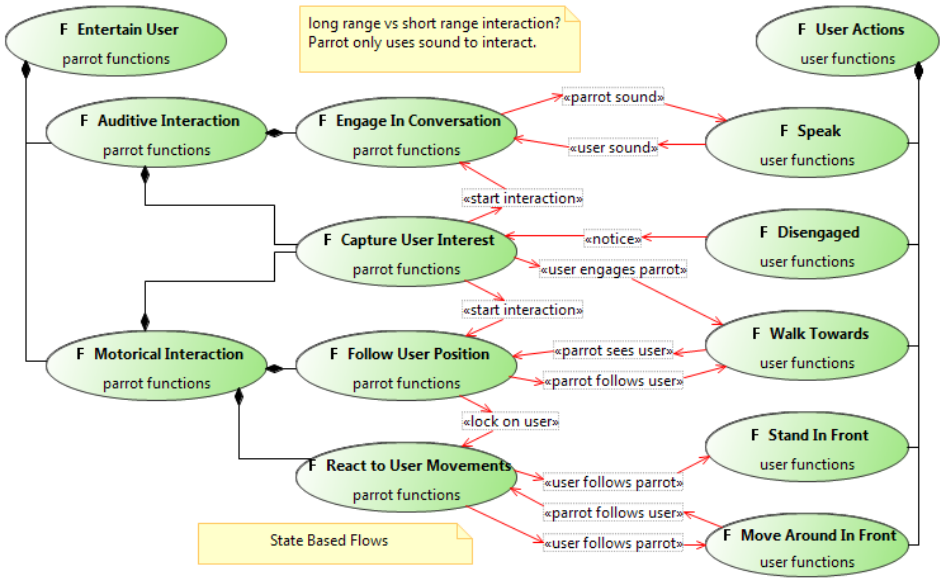


Figure F.2: Function modeling*: Functions of system, functions of user (scenario) Mix use case and process flow*

12. *Add subscription to view* – Can be used to define a search pattern through the AM. All concepts corresponding to the search will be added to the current view, and new additions to the AM will appear when the search is re-run. This way, stakeholders can keep track of certain information without the need for other people to send them updates. This is depicted in appendix figure F.10.
13. *Verify current view* – On pressing this button, the AMF will search the current view for 'knowledge' attributes to see if their values correspond with an available set of knowledge bases. When such a knowledge base exists, it should contain an algorithm to verify the part of the AM that corresponds with the knowledge contained in the base. See appendix figure F.9 for an example.
14. *Design Process Pattern* – On pressing this button, the AMF will search the current view for 'knowledge' attributes to see if their values correspond with an available set of knowledge bases. When such a knowledge base exists, it should contain an algorithm to partially automate the design process. This we call a design process pattern. xxxxx

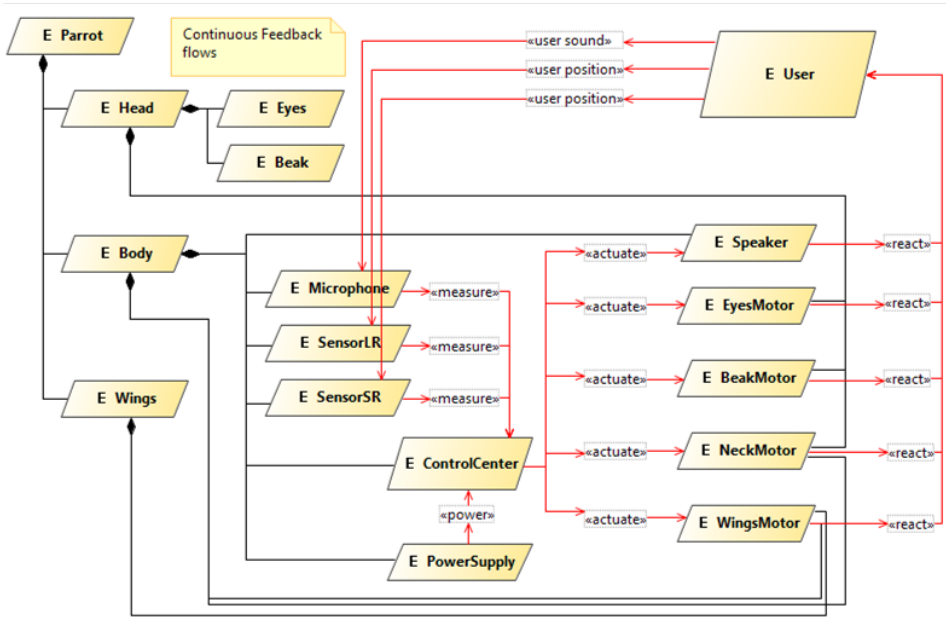


Figure F.3: System structure*: Decompose system, Define flow of energy/signal/material*

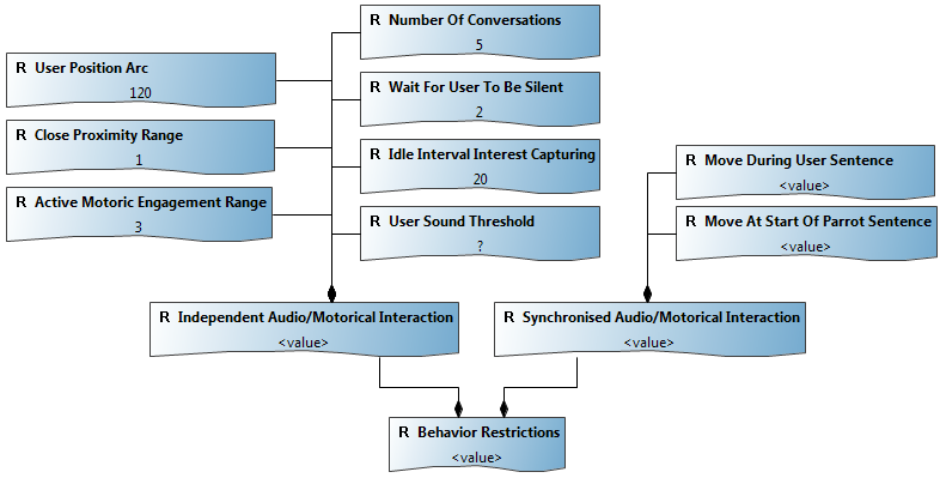


Figure F.4: Requirements capturing*: Requirements constrain solution space of system. Use as input AND validation of parameterized model.*

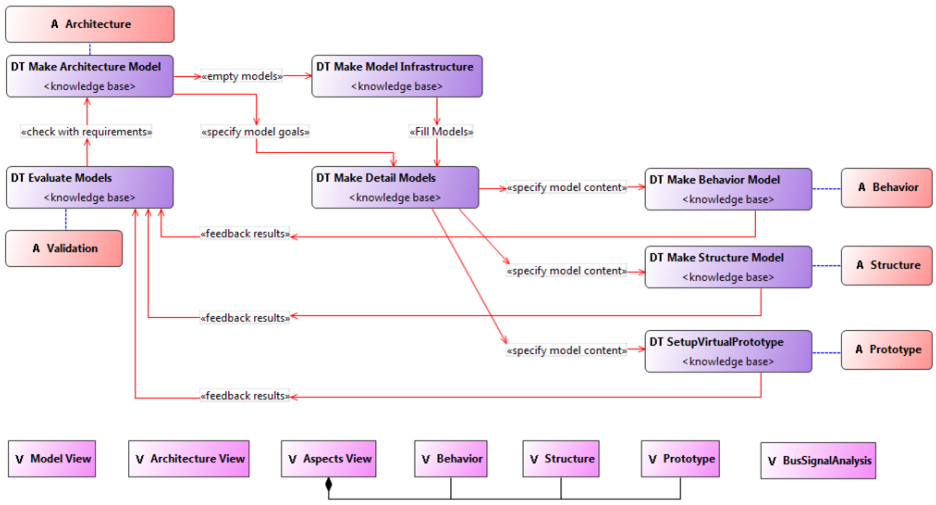


Figure F.5: Workflow modeling*: take workflow into consideration. Links system architecture to product (development) architecture.*

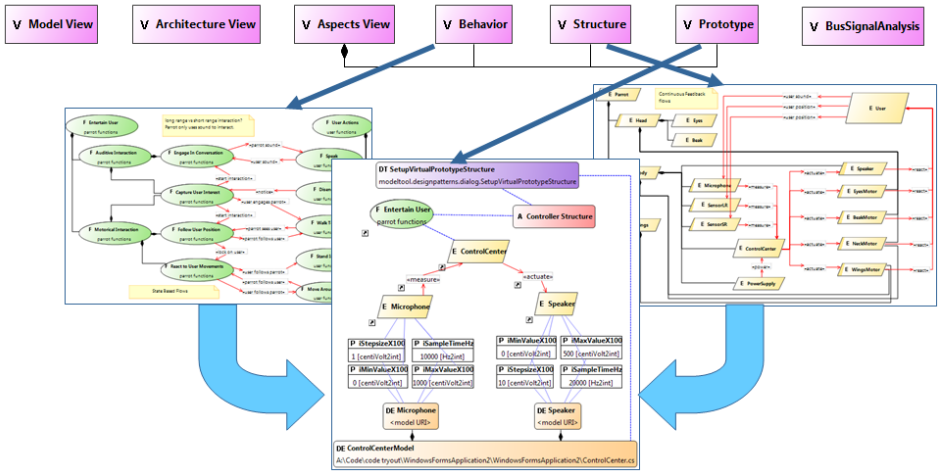


Figure F.6: View modeling*: Define data in different views, or mix them: Integration*

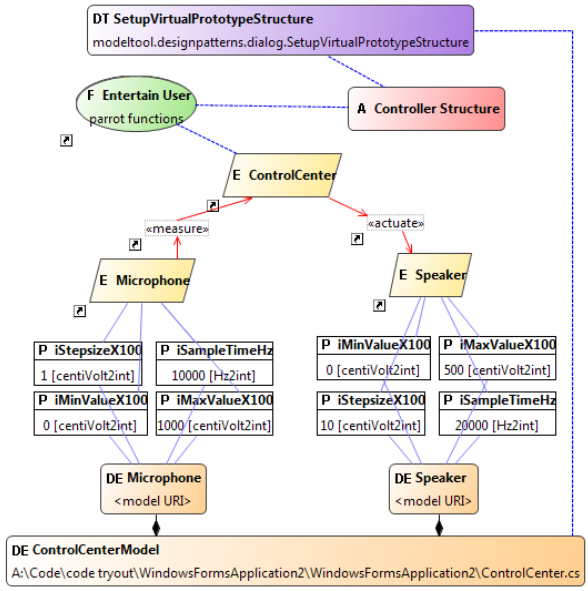


Figure F.7: Parameterizing the system*: Design tasks from workflow (who?), Function / Aspect (why?), Entities; Function Performers (what?), Design Parameters (which?), Models describing this system (how?).*

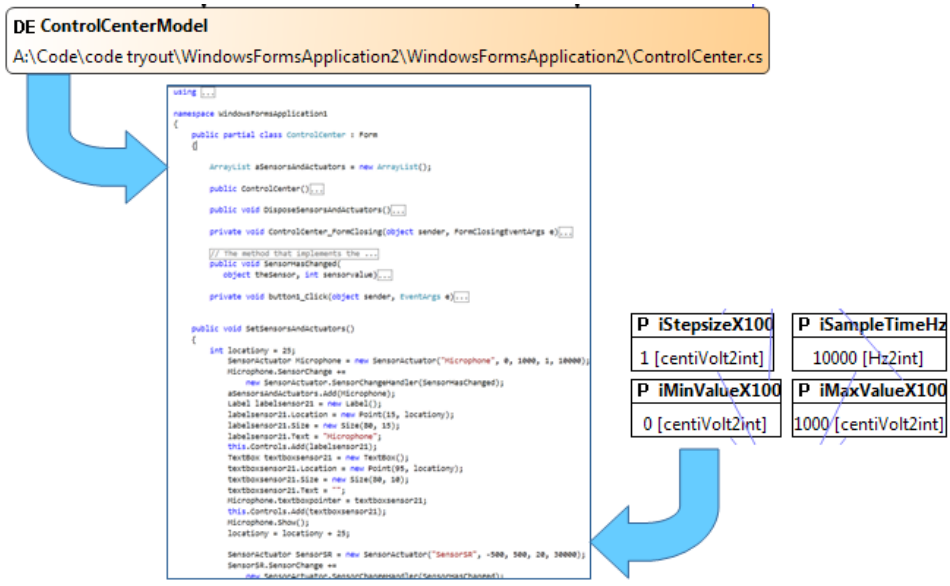


Figure F.8: Active documentation*: AM is an active document throughout development. Link external documents to objects in AM. Match parameters of AM to those in external model. Optional: Automate model transformation.*

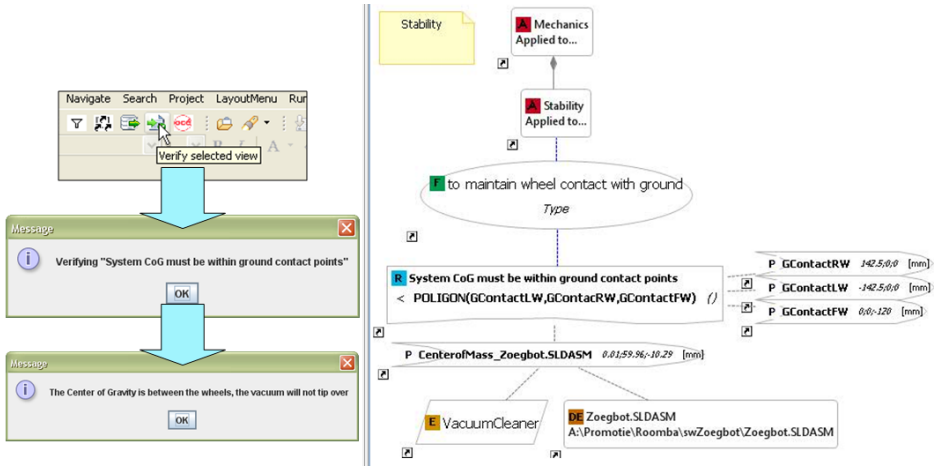
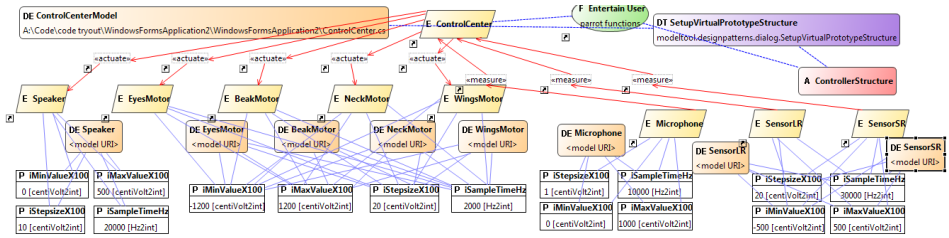
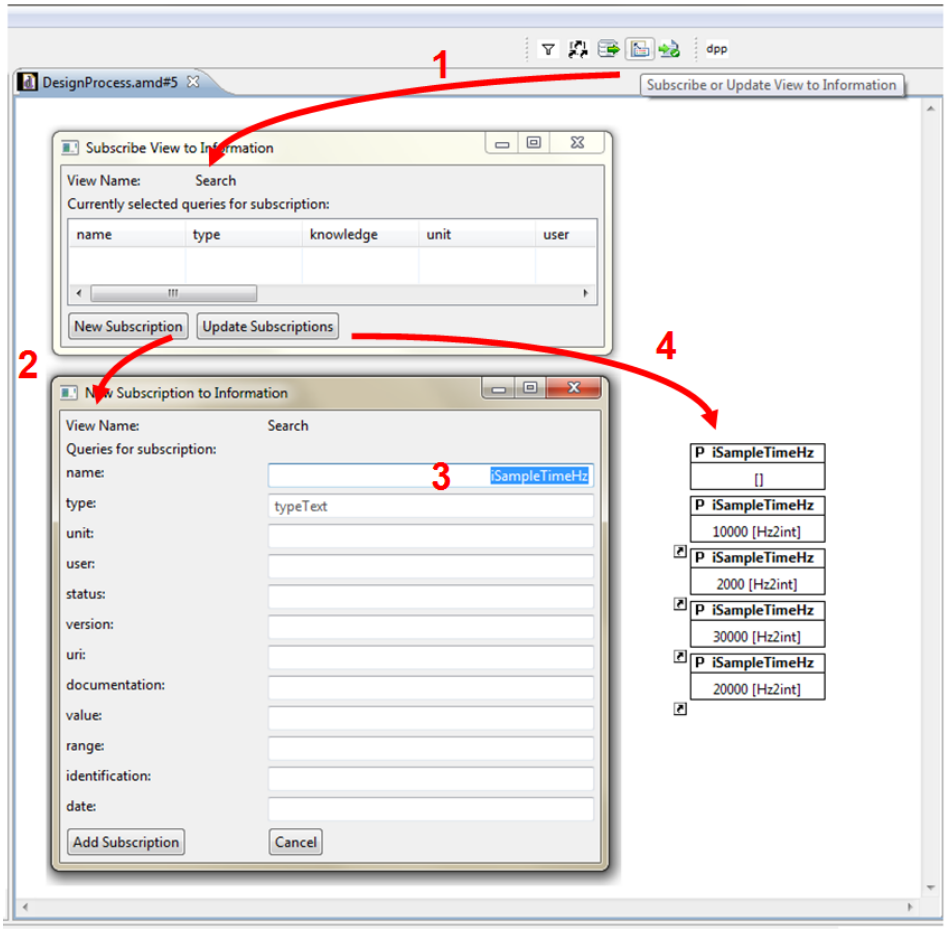


Figure F.9: Verification*: Knowledge bases can be developed to perform specific verification tasks. In this case, the user is informed of the system is still mechanically stable.*



(a)



(b)

Figure F.10: Subscription Patterns*: A user can 'subscribe' to certain information in his or her view. In (a) we see a lot of information, but the user is only interested in the 'iSampleTimeHz'. A subscription (b) can search the architecture model and add shortcuts to that information in a (new) view.*

G. Case Deliverables Baggage Handler

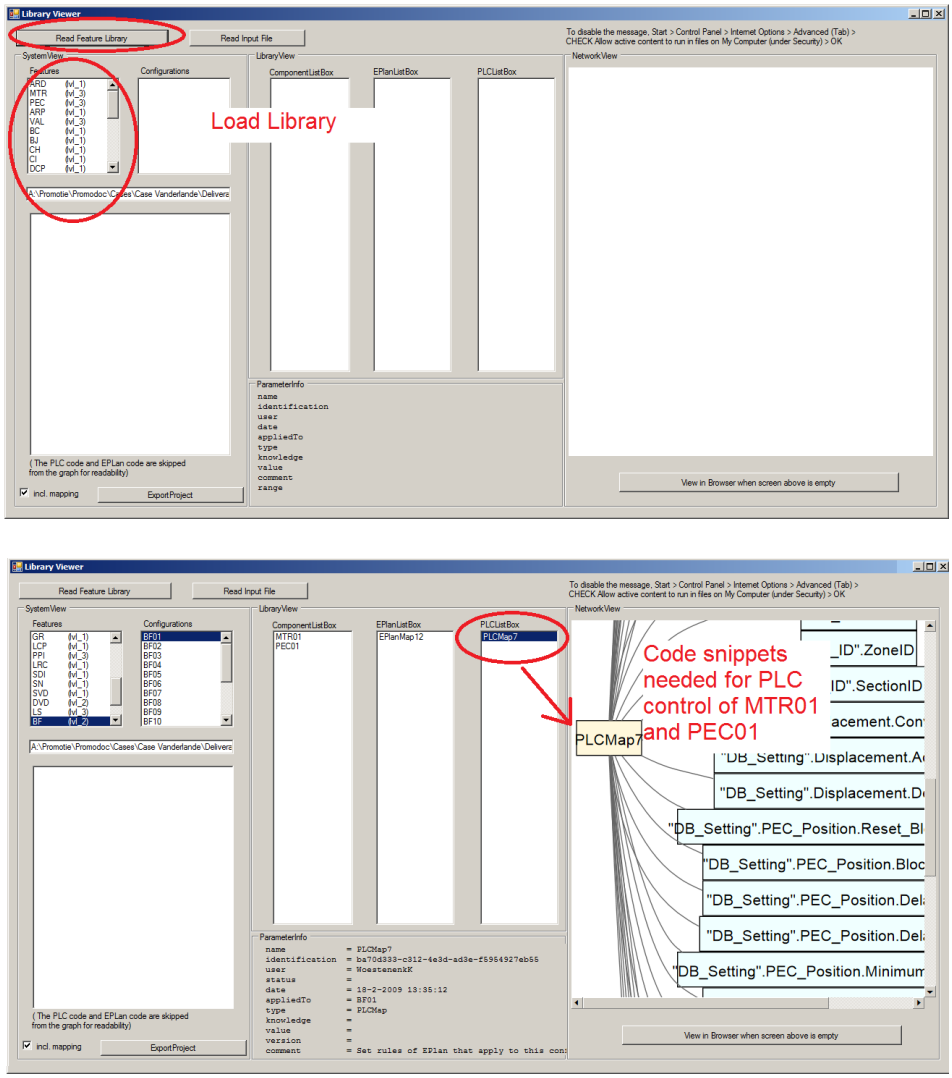


Figure G.1: Demo Vanderlande Tool A: Loading the new multi disciplinary library.

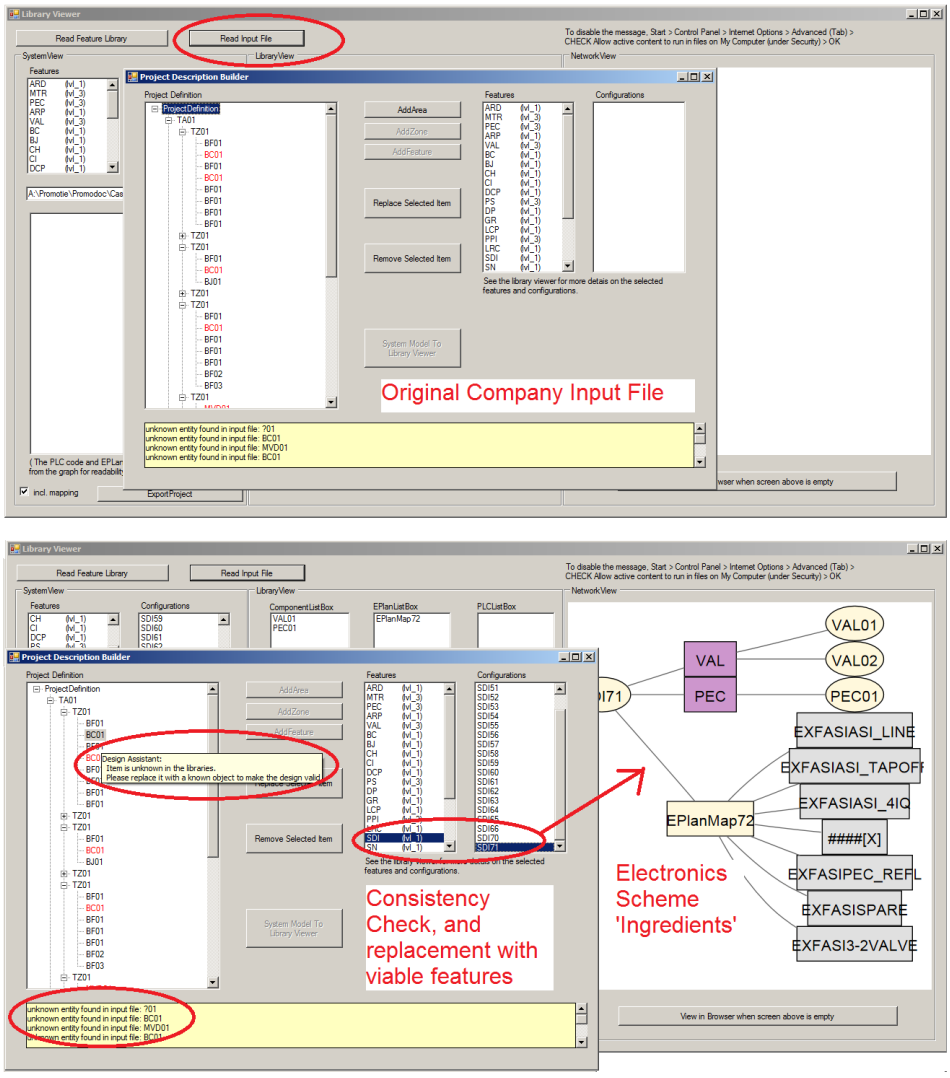


Figure G.2: Demo Vanderlande Tool B: Checking and repairing the input model.

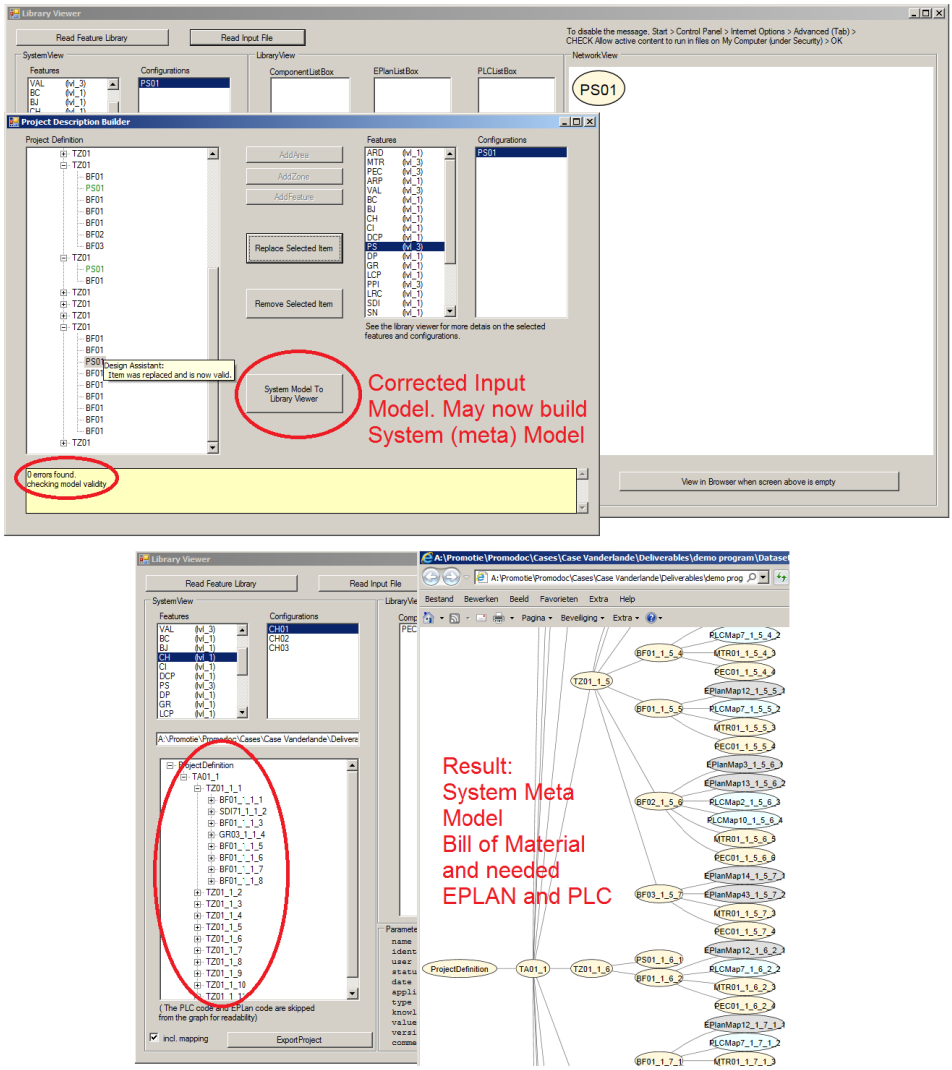


Figure G.3: Demo Vanderlande Tool C: Construction of the system (meta) model of a baggage handler from the input file of figure G.2 and the library components of figure G.1.

H. Multi Platform Extended Architecture Modeling Language

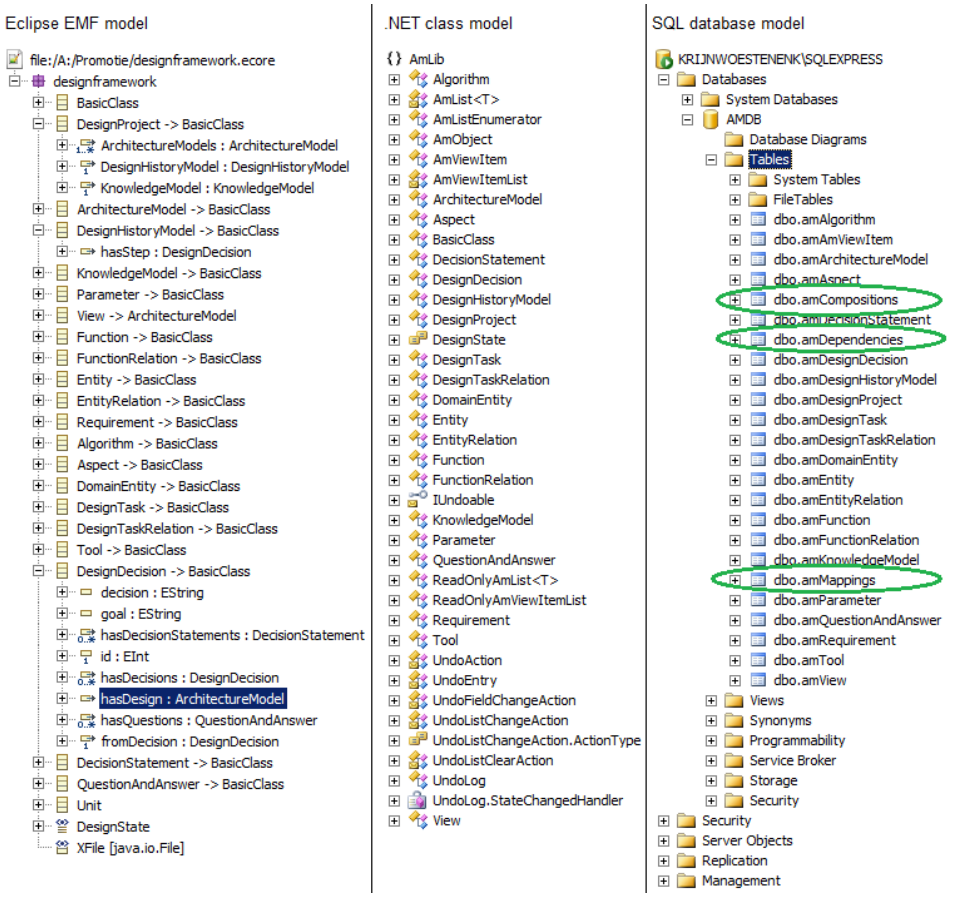


Figure H.1: The architecture modeling language, integrated with the design framework by ESI. The SQL database model is made with a multi client environment in mind. Note that relations between concepts are made explicit in this model.