PolyPublie
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	A Functional Reasoning Framework and Dependency Modeling Scheme for Mechatronics Conceptual Design Support
Auteur: Author:	Elie Haddad
Date:	2015
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Haddad, E. (2015). A Functional Reasoning Framework and Dependency Modeling Scheme for Mechatronics Conceptual Design Support [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2006/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/2006/
Directeurs de recherche: Advisors:	Sofiane Achiche, & Aouni A. Lakis
Programme: Program:	Génie mécanique

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

UNIVERSITÉ DE MONTRÉAL

A FUNCTIONAL REASONING FRAMEWORK AND DEPENDENCY MODELING SCHEME FOR MECHATRONICS CONCEPTUAL DESIGN SUPPORT

ELIE HADDAD

DÉPARTEMENT DE GÉNIE MÉCANIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCE APPLIQUÉES

(GÉNIE MÉCANIQUE)

DÉCEMBRE 2015

© Elie Haddad, 2015.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

A FUNCTIONAL REASONING FRAMEWORK AND DEPENDENCY MODELING SCHEME FOR MECHATRONICS CONCEPTUAL DESIGN SUPPORT

présenté par : HADDAD Elie

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. BARON Luc, Ph. D., président

M. ACHICHE Sofiane, Ph. D., membre et directeur de recherche

M. LAKIS Aouni, Ph. D., membre et codirecteur de recherche

M. RAISON Maxime, Doctorat., membre

ACKNOLEDGMENT

I would like to thank my advisor professor Sofiane Achiche and my co-advisor professor Aouni Lakis for giving me the opportunity to pursue my studies at Ecole Polytechnique de Montreal and supporting me financially during my studies. I would also like to thank them for their kindness, support, valuable time, and patience.

Many people have walked next to me during this thesis, it has been a bumpy road, and without them it would have been hard to reach the end. I would specifically like to thank Abolfazl Mohebbi for his advice, encouragement, and the time he spent motivating me and pushing me to work harder.

Finally, I would like to thank my parents to whom this thesis is dedicated. Thank you for believing in me, and for supporting me financially and emotionally. I hope I have made you proud.

RÉSUMÉ

La conception mécatronique est un processus de design pluridisciplinaire, il repose sur l'intégration synergique des domaines d'ingénierie mécanique, électrique, contrôle et logiciel pour concevoir des produits qui surpassent les autres produits en termes d'efficacité, de précision, de coût et de fiabilité. Toutefois, cela a un coût, la conception de systèmes multidisciplinaire est une tâche ardue qui exige beaucoup de coordination et de coopération entre les ingénieurs concepteurs. Beaucoup de ces difficultés ont été reportées dans les domaines académique et industriel. Il en ressort que la communication technique entre les concepteurs appartenant à diverses disciplines d'ingénierie se fait très difficilement et ce en raison de l'absence d'un langage commun pour représenter les différents concepts. Ceci entraîne des difficultés majeures à transférer les modèles et les informations pertinentes entre les domaines ce qui entrave la possibilité d'appliquer un processus de développement intégré (concurrent). Pourtant, d'une part, un processus de conception intégré et dynamique doit être suivi pour réduire le temps de conception du projet et ainsi réduire les couts et supporter l'innovation. D'autre part, la conception multidisciplinaire se traduit par l'introduction d'un grand nombre de dépendances durant la conception, rendant ainsi les activités de conception difficile à synchroniser entravant le processus intégré.

En raison de l'absence d'outil de support informatique pour le design conceptuel, et l'importance de considérer les dépendances le plus tôt possible dans le processus de conception, un cadre de raisonnement fonctionnel en conjonction avec un système de modélisation des dépendances (liées au produit) ont été développés dans ce mémoire de maîtrise.

Le cadre de raisonnement fonctionnel a été réalisé par la personnalisation du langage SysML (Systems Modeling Language), et par le développement d'un module d'extension (plug-in) dans l'outil de modélisation MagicDraw (No Magic, Inc.). Le plug-in intègre un système expert à base de règles (CLIPS : C Language Integrated Production System - NASA) qui permet d'encapsuler les connaissances d'ingénierie sous la forme de règles pour analyser et effectuer des tâches sur des diagrammes fonctionnels.

Une nouvelle approche d'acquisition et une représentation schématique de dépendances ont été proposées. La notion de "méta-dépendances» a été introduite pour modéliser les dépendances qui sont partagées par un grand nombre d'éléments dans un même système. Cela permet aux

concepteurs de capter efficacement et abstraitement les dépendances tôt dans le processus de conception et ainsi réduire le nombre de relations à construire manuellement entre les éléments dans ce système.

Pour prouver l'efficacité de la méthode de modélisation proposée, les effets indésirables, un type de dépendances qui peut être utilisé avec le schéma de modélisation proposé, ont été utilisés et intégrés dans le cadre de raisonnement fonctionnel. Ce dernier a été programmé pour générer automatiquement une matrice de conception de la structure pour chaque type de dépendance et les présenter aux concepteurs.

Deux études de cas ont été réalisées où des produits mécatroniques ont été modélisés en utilisant le cadre de raisonnement fonctionnel tout en prenant les dépendances d’effets indésirables en compte. Tout d'abord, un dispositif simple qui régule la température de l'eau a été utilisé pour illustrer le principe. Ensuite, le modèle conceptuel d'un drone quadrotor a été conçu. La modularisation fonctionnelle résultante et la disposition géométrique approximative du quadrotor ont été présentées, ainsi qu’un ensemble de problèmes de conception qui ont été évités.

Le cadre de conception de raisonnement fonctionnel en conjonction avec le schéma de modélisation méta-dépendance proposé et développé dans cette thèse de maîtrise ont prouvé d’être un outil de modélisation dynamique et flexible qui permet d’apporter des changements au design durant la conception avec peu d'efforts de la part des concepteurs. Le marquage des fonctions avec les dépendances d’effets indésirables c’est avéré être une méthode efficace et effective pour acquérir et gérer l’information sur ce type de dépendances.

ABSTRACT

Mechatronics is a multidisciplinary design process that relies on the synergic integration of mechanical, electrical, control, and software engineering to deliver products that outperform their competitors in terms of efficiency, precision, cost and reliability. However, this comes at a cost, designing multi-disciplinary systems is a challenging task that requires a lot of coordination and cooperation between designers. Several challenges are reported by both academic and industry-related literature. One of the most important is the tedious communication between engineering designers from various disciplines due to a lack of a common language to represent concepts. This leads to difficulties in transferring models and pertinent information between domains. To succeed in nowadays competitive markets, a concurrent and dynamic design process should be followed to reduce the project lead-time and spark innovation. However, such a process results in many dependencies as a consequence of multi-disciplinary design and it is often difficult to streamline the design activities.

Due to the lack of existing computational support tools for conceptual design of mechatronics and the importance of taking dependencies (product related) into account as early as possible in the design process, a functional reasoning framework as well as a dependency modeling scheme were developed in this master thesis.

The functional reasoning framework was realised by customizing the SysML (Systems Modeling Language) language and developing a plug-in in the modeling tool MagicDraw (No Magic, Inc.). The plug-in integrates the rule-based expert system CLIPS (C Language Integrated Production System - NASA) that allows encapsulating engineering knowledge in the form of rules to analyze and perform tasks on functional diagrams.

A new acquisition method and representation scheme of dependencies was proposed in this master thesis. The concept of “meta-dependency” was introduced to model dependencies shared by a large number of elements in a same mechatronic system or sub-system. It allows engineering designers to efficiently and abstractly capture dependencies early in the deign process and reduces the number of relationships to be built manually between dependent elements in the system. To prove the efficacy of the proposed modeling method, adverse effects, a type of dependency that suits the proposed modeling scheme, were used and integrated into the

functional reasoning framework, which was programmed to automatically generate a Design Structure Matrix for each type of dependency and present them to the engineering designers.

Two case studies were carried-out where mechatronic products were modeled using the functional reasoning framework while taking adverse-effect dependencies into account. First, a simple device that regulates the temperature of water was used to illustrate the principle. Then, the conceptual model of a quadrotor drone was designed. The resulting functional modularization and rough geometric layout of the quadrotor were presented, as well as a set of design problems that were avoided.

The functional reasoning design framework in conjunction with the meta-dependency modeling scheme proposed and developed in this master thesis proved to be a dynamic modeling tool that is flexible and allows changes to be made in the design with little effort from the engineering designer. Tagging functions with adverse effects proved to be an efficient and effective method of acquiring information on this type of dependencies and managing them. It is an intuitive way of handling and capturing abstract dependencies early in the design process without shifting the designer’s focus away from solution finding.

TABLE OF CONTENTS

ACKNOLEDGMENT III

RÉSUMÉ IV

ABSTRACT VI

TABLE OF CONTENTS VIII

LIST OF TABLES X

LIST OF FIGURES XI

CHAPTER 1 INTRODUCTION AND LITERATURE REVIEW 1

1.1 Introduction to mechatronics 1

1.2 Summary of mechatronic design challenges 3

1.2.1 Mechatronic design methods 4

1.2.2 Tools, Support, and Integrated solutions for mechatronic design 5

1.3 Functional modeling 7

1.4 Dependencies in mechatronics (multi-domain) systems 10

1.4.1 Dependency definition, characteristics, and classification 10

1.4.2 Common dependency management tools 12

CHAPTER 2 RESEARCH FRAMEWORK 17

2.1 Challenges addressed and Objective 17

2.2 Proposed framework and dependency modeling 18

2.2.1 Functional Reasoning Framework 18

2.2.1.1 Ontology 18

2.2.1.2 Developing a Representation Scheme 19

2.2.1.3 Implementing a Reasoning method 22

2.2.1.3.1 Linking the system modeling and the reasoning method: MagicDraw openAPI plug-in 23

2.2.1.3.2 Rule-based expert system CLIPS 23

2.2.1.3.3 Assisted modularization using rule-based heuristics 26

2.2.2 Dependency Modeling 30

2.2.2.1 Modeling Using Meta-Dependencies Concept 30

2.2.2.2 Tagging elements with adverse effect dependencies 32

2.2.2.3 Automated Generation of the Dependency Matrix 32

2.2.3 Design Procedure summary 35

CHAPTER 3 CASE STUDY 37

3.1 Illustrating the principle 37

3.2 Quadrotor conceptual design 39

CHAPTER 4 CONCLUSION 53

BIBLIOGRAPHY 55

LIST OF TABLES

Table 1.1 Spatial adjacency (Pimmler & Eppinger, 1994) 13

Table 2.1 Functional basis set of functions and flows (Hirtz et al., 2002).....19

Table 3.1 Relevant Functions, Possible Adverse Effects, Alternative components 41

Table 3.2 Modules, problematic Adverse Effects, Solutions.....47

LIST OF FIGURES

Figure 1.1 Mechatronics Euler diagram (Alciatore, Histand et al. 2007) 1

Figure 1.2 Main stages of product design process 2

Figure 1.3 SysML as a common modeling language and mapping between different domains (Shah, Kerzhner et al. 2010) 6

Figure 1.4 Functional Modeling (Stone, Wood et al. 2000) 8

Figure 1.5 Function behavior state (FBS) (Umeda, Tomiyama et al. 1995) 8

Figure 1.6 Graphical representation of a dependency 10

Figure 1.7 Analysis and synthesis properties and dependencies (Qamar, Paredis et al. 2012) 11

Figure 1.8 Task DSM (Braha 2002) 13

Figure 1.9 Example of a Multiple Domain Matrix (MDM) (http://www.plattformstrategie.de/) 15

Figure 2.1 Material, Energy, and Signal flow stereotypes 21

Figure 2.2 Function stereotype 21

Figure 2.3 Custom functional modeling diagram 21

Figure 2.4 Material flow hierarchical decomposition in SysML 22

Figure 2.5 Function hierarchical decomposition in SysML 22

Figure 2.6 Custom functional modeling diagram example 22

Figure 2.7 Functional modeling diagram representation in CLIPS 25

Figure 2.8 Exported functional diagram using CLIPS templates sample 26

Figure 2.9 Dominant flow module (Stone, Wood et al. 2000) 27

Figure 2.10 Branching flow module (Stone, Wood et al. 2000) 27

Figure 2.11 Conversion-transmission flow module (Stone, Wood et al. 2000) 28

Figure 2.12 Flow template with chain slot CLIPS 29

Figure 2.13 Chain template CLIPS 29

Figure 2.14 Rule that identifies flows that branch CLIPS 30
Figure 2.15 Meta-dependency modeling 31
Figure 2.16 Meta-dependency elements 31
Figure 2.17 Affecter and affected meta-dependency tagging slots 32
Figure 2.18 Adverse effect dependency stereotype bloc 33
Figure 2.19 createDependency CLIPS rule 34
Figure 2.20 Design procedure flowchart 35
Figure 3.1 temperature regulator functional model 37
Figure 3.2 Electric field DSM and heat DSM of temperature regulator 38
Figure 3.3 Partial Quadrotor drone functional model 40
Figure 3.4 Chains of functions resulting from CLIPS 43
Figure 3.5 Chains of flows in SysML functional diagram 44
Figure 3.6 Preliminary modularization 45
Figure 3.7 Vibration DSM 46
Figure 3.8 Heat DSM 46
Figure 3.9 Electric field DSM 47
Figure 3.10 Power supply and Power Control modules 50
Figure 3.11 Quadrotor layout: bottom, top ,and side views 51

CHAPTER 1 INTRODUCTION AND LITERATURE REVIEW

1.1 Introduction to mechatronics

Mechatronics is a multidisciplinary design process that covers a very wide spectrum of products and industries ranging from fully automated manufacturing lines, biomedical devices, agricultural equipment, aerospace, automotive, military defense systems, to various other consumer products. Mechatronics relies on the synergic integration of mechanical, electrical, control and software engineering to deliver simple products that outperform other products in terms of efficiency, precision, cost and reliability. It not only improves the design of products, it also allows to create new functions that were not possible before, such as active suspension and electronic stability in the automotive industry and fly-by-wire in aeronautics (Jürgen Gausemeier & Moehringer, 2003).

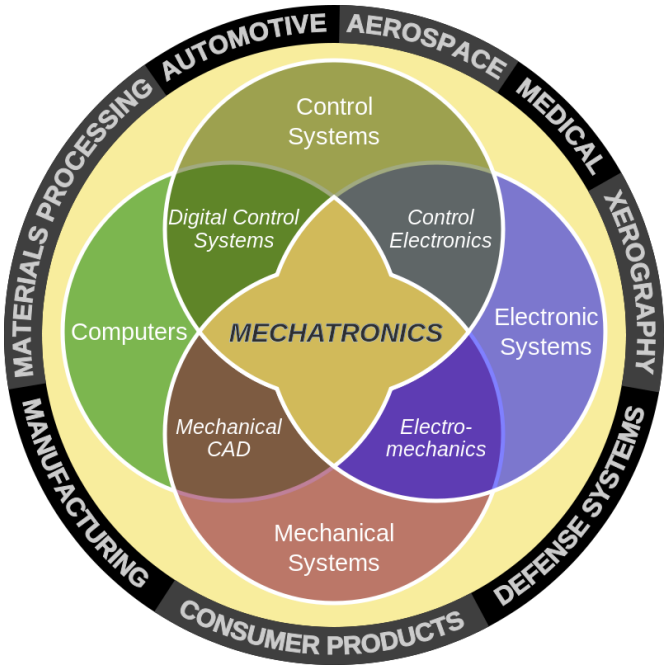

Figure 1.1 Mechatronics Euler diagram (Alciatore, Histand, & Alciatore, 2007)

Figure 1.2. Illustrates general idea of how the design process of a mechatronic product is usually carried out. The nomenclature and subsections can vary depending on the design methodology and the industry in question. The process is divided in two main categories in terms of conceptual

and preliminary design followed by manufacturing and testing. In this master thesis we do not consider the market aspect of the mechatronic products.

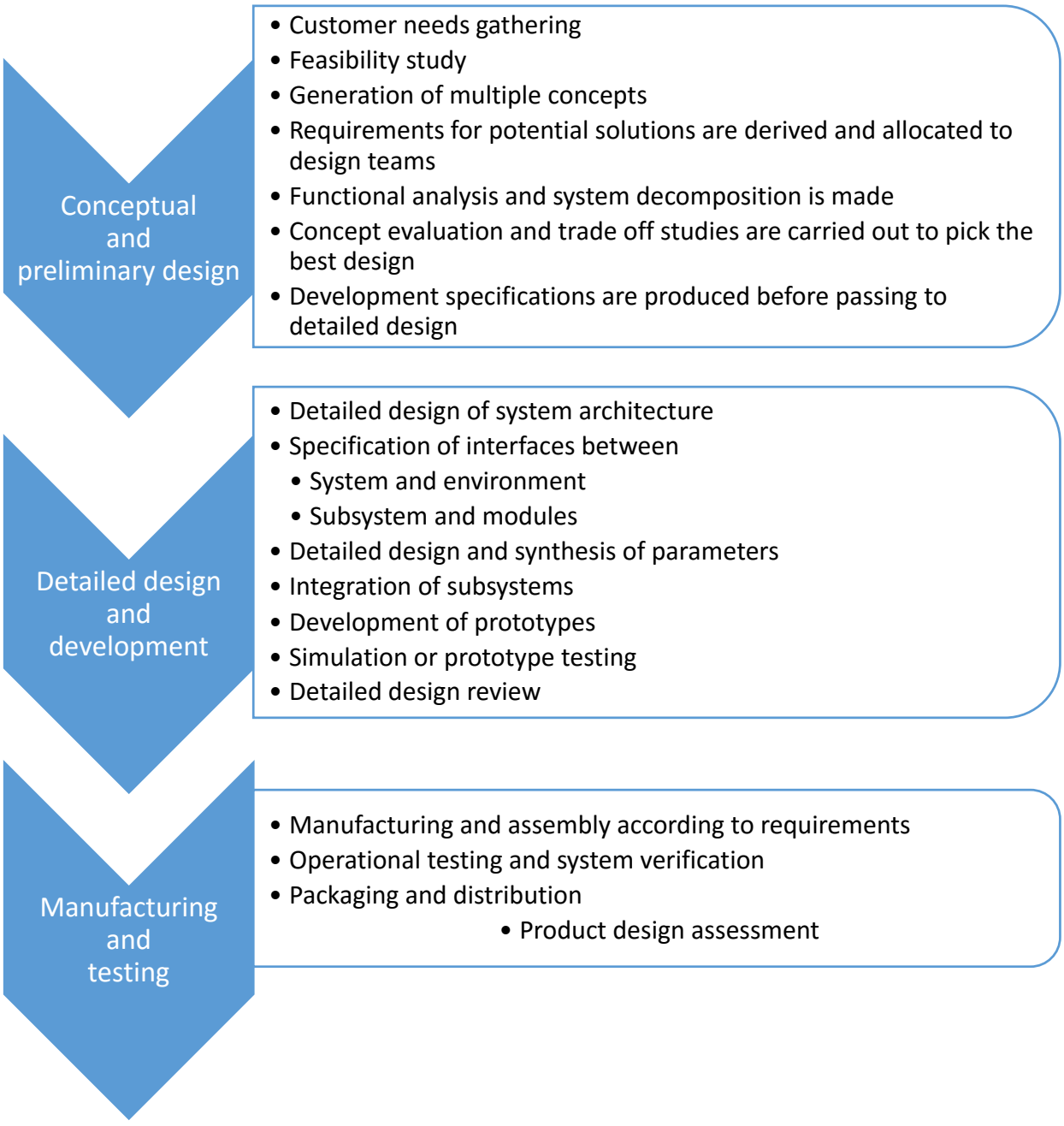

Figure 1.2 Main stages of product design process

Creating development solutions for the design of mechatronic products is not an easy task. In order to contribute to this field of research, it is important to identify and understand the challenges faced in both academia and in the industry throughout the different stages of the design process (Mohebbi, Achiche, Baron, & Birglen, 2014).

1.2 Summary of mechatronic design challenges

The multi-disciplinary nature of mechatronics makes the design process a complex activity where designers from various backgrounds and work proficiencies have to cooperate closely to deliver a functional and optimal product. Unfortunately, designers rarely have sufficient knowledge outside their domain of expertise (Tomiyama, D'Amelio, Urbanic, & ElMaraghy, 2007) and are usually not well trained to integrate their work with other disciplines. During creative meetings they have difficulty sharing models and information, and often hesitate to communicate with one another (Salminen & Verho, 1989) due to a lack of a common language among disciplines to represent concepts (Albers et al., 2011). When looking at the system from a domain specific point of view they interpret realities differently, make various assumptions about the product, and end up forming their own mental model of the system (Danilovic & Browning, 2007). With this mindset, it is difficult to understand the overall design, its goals and purposes, and it makes the design process inefficient. Driven by professional loyalty, designers try to come up with solutions or optimize the subsystems they are working on in ways they think are beneficial to the overall system. However, a good mechatronic design is not always a combination of optimal subsystems but a harmonious interaction of its subsystems. For example, a mechanical engineer might optimize the drive train of a machine to minimize backlash, when this problem can be solved by a control engineer using programmatic compensation. While both alternatives lead to the same result, the mechanical solution increases design and manufacturing costs.

To succeed in an ever evolving market, a fast paced design process with a high degree of innovation is required. During product development designers have to cope with modified customer requirements, changes in regulations, advances in technology, etc. all while trying to reduce the project lead time. Some of the major challenges faced during a design process of this kind are synchronizing the design activities (Jurgen Gausemeier, Frank, Donoth, & Kahl, 2009)

and managing the dependencies that arise as a consequence of the interactions between the different disciplines (J. M. Torry-Smith, Mortensen, & Achiche, 2014). Designers also face difficulty in evaluating design concepts and assessing the consequences of selecting between alternatives (Mohebbi et al., 2014; J. M. r. Torry-Smith et al., 2011).

1.2.1 Mechatronic design methods

A lot of research effort has been put recently in developing a reliable design methodology for mechatronic systems, however, these endeavors have not completely reached their goal yet (Blessing & Chakrabarti, 2009). A Traditional design method, or a sequential method, exhibits a successive flow of activities where design tasks are separated in a manner that some parts use information resulted by the design of previous parts (e.g. control engineers use physical parameters provided by mechanical engineers to design a stabilization system) (Mohebbi et al., 2014). In a concurrent design method, all phases of a product's lifecycle are taken into consideration during conceptual design and products are usually divided into single domain subsystems that are designed simultaneously. To ensure consistency during assembly, special attention is needed when designing the interfaces among the subsystems. Preferably, it is important to focus on the interaction between the different engineering disciplines rather than only on the subsystem interfaces (Wikander, Törngren, & Hanson, 2001). Sequential design does not meet the requirements of today's fast paced multidisciplinary product development. It lacks the flexibility to deal with dynamic markets and is unsuitable for complex system integration, which leads to increases in design cost and development time (Wang, Shen, Xie, Neelamkavil, & Pardasani, 2002). Concurrent design is a means to reduce the project lead time and benefit from the multidisciplinary design synergy (J. M. Torry-Smith et al., 2014). It allows designers, early in the design process, to detect and deal with issues related to later stages of the design (Rzevski, 2003) which can also be used to manage the interactions between designers and their designs (Mohebbi et al., 2014). Even though a concurrent design approach has desirable advantages, it has been reported in the industry that traditional and sequential design methods are still being used (Behbahani & De Silva, 2007) because there is still no formal and systematic approach that promotes it. And that is due to the complexity of multidisciplinary systems, the difficulty in

synchronizing the development activities, and the lack of methods and tools to manage dependencies that arise between subsystems and disciplines during design.

1.2.2 Tools, Support, and Integrated solutions for mechatronic design

To assist engineering designers during the development activity of mechatronics many solutions have been proposed, and many have been implemented and are used in the industry. These solutions range from design guidelines and frameworks to paper based tools and software (J. M. Torry-Smith, Mortensen, Ploug, & Achiche, 2015). Advancements in computer and software technologies to date have extremely facilitated engineering design. Providing computational support early in the design process has a lot of benefits view that most "added value" to a product is contributed during this phase (Starling & Shea, 2005). Contradictorily, most of the available tools, such as CAD drawing tools, thermal simulation tools, computational fluid dynamics tools, or control and instrumentation tools, only support later stages of design. When it comes to conceptual design, designers use a variety of tools for organization activities such as managing requirements, but there is still a lack of tools that support the design synthesis activity (Wölkl & Shea, 2009). Translating customer requirements into functional requirements and developing product architectures are important but difficult parts of conceptual design and there is ongoing research in this field to provide computational support (Albers et al., 2011; Helms & Shea, 2012; Komoto & Tomiyama, 2012).

As previously mentioned, the design of mechatronics greatly benefits from concurrent and integrated design, where all phases of the design are considered starting from the beginning, and all the domains involved cooperate simultaneously to the design activity. The same applies to the software tools used to support design. Interoperable programs can exchange and share models and coupled or linked tools communicate at runtime. A lot of frameworks have been developed to automate model exchange between various tools. For example, SysML, a model based system engineering language, was used to represent a common design model of a system (Shah, Kerzhner, Schaefer, & Paredis, 2010). SysML is a powerful tool that allows designers to capture requirements and to model the structure and behavior of a system. As shown in Figure 1.3, to address issues associated with multi-view modeling, the common SysML model was used to map

models between domain specific tools and to represent the dependencies between them, thus supporting designers throughout the design process.

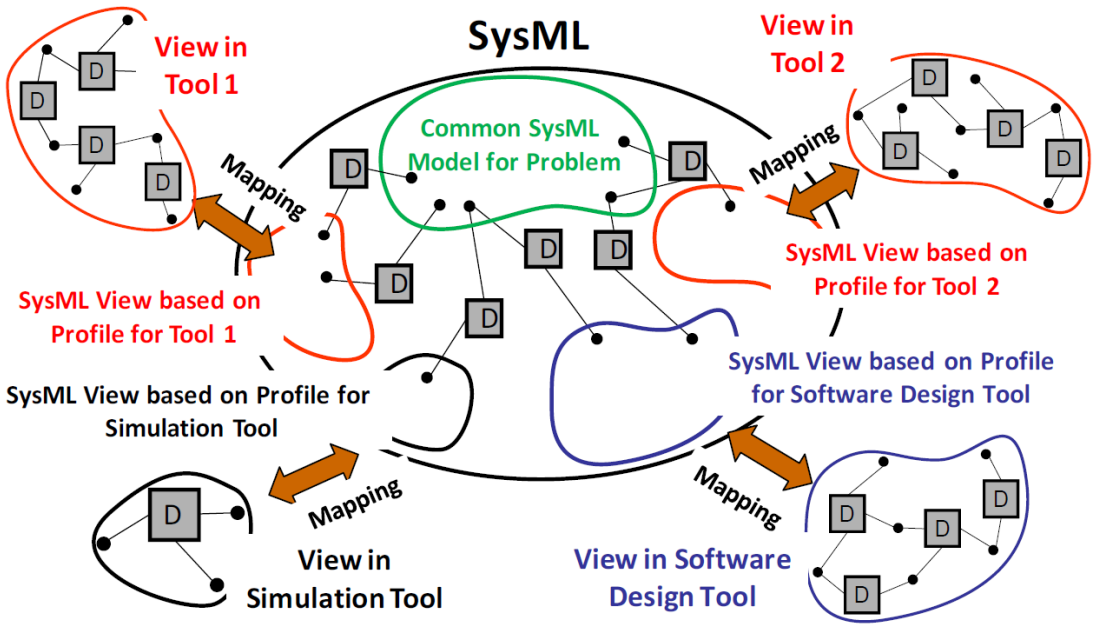

Figure 1.3 SysML as a common modeling language and mapping between different domains (Shah et al., 2010)

Integrated programs on the other hand allow the work in different domains and at various design phases within the same tool (Mohebbi et al., 2014). For example, a lot of mechanical CAD tools incorporate tools from other domains such as electronics and control modeling into their software. While design frameworks and integrated tools are desirable, even with the considerable research being conducted there are sill a number of challenges to be addressed before tools will allow efficient integration and use of multidisciplinary tools and data. Some of the major challenges are developing a single language that can effectively capture all the needed information in multi-disciplinary design, and ensuring consistency between the models of the various domain specific tools (Shah et al., 2010).

1.3 Functional modeling

Designers map ideas they conceive to a semantic domain they understand. They represent concepts by drawing sketches, graphs, and diagrams to facilitate sharing and explanation of thoughts between team members. Functional modeling allows modeling at an abstract level and can be used by designers of various expertise as a common language to communicate ideas during conceptual design. By breaking down the overall system into small easy to solve sub-functions, they can easily concretize requirement specifications into product concepts. (Eisenbart, Blessing, & Gericke, 2012).

The definition of function in the literature varies a lot depending on the field of study (Far & Elamy, 2005). When it comes to engineering design there are multiple definitions that can be put into two categories. Some definitions emphasize that for a function to be fulfilled a transformation has to take place, whereas others state that a goal or a requirement has to be achieved (Crilly, 2010), namely transformation functions and purpose functions.

When modeling a product, different designers tend to generate distinct functional models. To help make the design task more consistent, and get repeatable and meaningful models from such a representation, a formal function representation is needed. A lot of efforts, such as NIST (Szykman, Racz, & Sriram, 1999) and TIPS (Altshuller, 1984), have been made to provide a functional vocabulary to model engineering systems. The functional basis for design reconciles and integrates these two efforts into a more evolved modeling language (Hirtz, Stone, McAdams, Szykman, & Wood, 2002). It consists of a standardized set of functions and flows where transformation functions are characterized by input/output relations in the form of verb-noun pairs (function-flow pairs). Functions can be connected by flows of material, energy, or signal to form a functional diagram that abstractly describes the product. Using a standardized set of functions and flows not only reduces ambiguity and ensures consistency; it forms a formal language that can be analyzed by algorithms and rules, thus it provides a strong foundation for computational support.

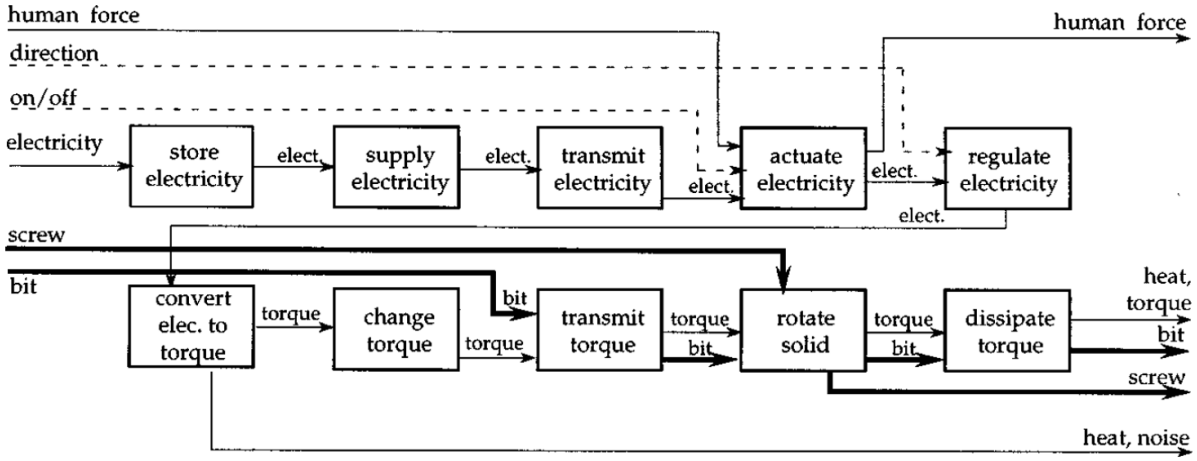

Figure 1.4 Functional Modeling (Stone, Wood, & Crawford, 2000)

Another research effort, the function-behavior-state (FBS) (Umeda, Tomiyama, & Yoshikawa, 1995), a modeling scheme of function for conceptual design, states that transformation functions do not provide enough flexibility for designers to model without reference to behavior and structure. Purpose functions are used instead, and are defined as human intentions. These functions are then associated with behaviors which are a more objective representation of the design based on physical principles.

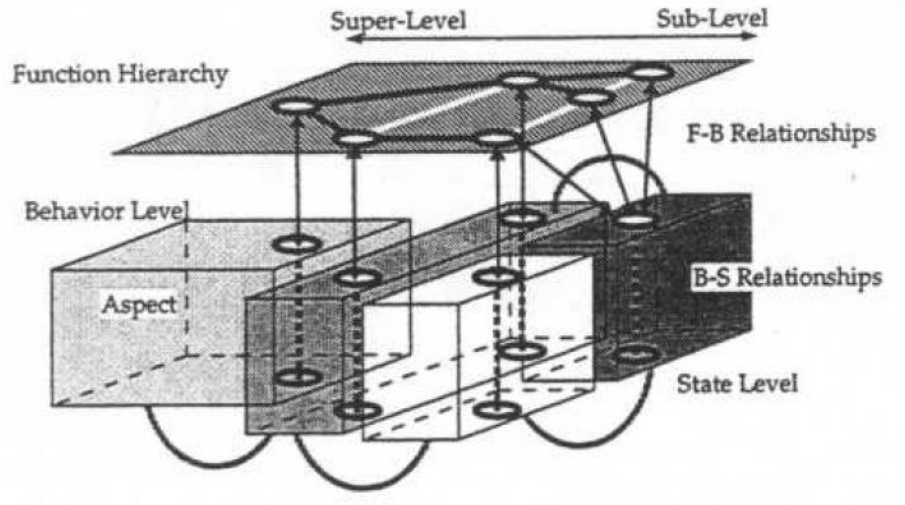

Figure 1.5 Function behavior state (FBS) (Umeda et al., 1995)

Functional modeling also acts indirectly as a design scheme that guides design activities such as decomposing problems, generating concepts, and creating product architectures (Hirtz et al., 2002). To allow modular design to be carried on earlier in the product development, heuristics were developed to search for and identify chains of functions in a functional diagram (Stone et al., 2000). A chain of functions is a group of functions that are aggregated based on the type of transformations that occur to the flows that go through them. Chains, also called modules, can either be replaced by existing components that fulfill their task or labeled as parts of the product that need to be designed. This helps dividing the design activity and assigning tasks to teams starting from conceptual design.

Functional modeling acts as a bridge between human intentions and the physical structure and behavior of a product (Umeda et al., 1995). When used in conjunctions with artificial intelligence techniques, smart modeling tools can be created to assist designers in product development related activities. The research field that relates these two fields is called functional reasoning (Chandrasekaran, 1994). There are three aspects to a functional reasoning framework, an ontology, a representation scheme, and a reasoning method (Far & Elamy, 2005). Ontology is defined as "*a formal, explicit specification of shared conceptualization.*" (Studer, Benjamins, & Fensel, 1998). For example, the functional basis for design (Hirtz et al., 2002), previously presented, is an ontology that describes the function modeling domain and the entities in it. The representation scheme models the entities of the ontology and the relations between them, such as their hierarchical decomposition. The reasoning method infers and explains how the entities function. A functional reasoning system can be used for planning, conceptualization, or explanation purposes, by applying various artificial intelligence techniques such as heuristic search, exploration and exploitation, pattern matching, and clustering (Erden et al., 2008).

A substantial amount of work has been put in this field of research. For example, an automated modularization scheme was developed by using functional reasoning (Van Beek, Erden, & Tomiyama, 2010). The function behavior state was used to model the system and derive relationships between functions, an adapted k-means clustering algorithm was then used to group related functions together to better assign design activities and visualize dependencies in the system.

Another example that applies exploration and exploitation, automates the synthesis of product architectures (Helms & Shea, 2012). A function behavior structure is used to model a system and

then, using object oriented graph grammars, various combinations components that can achieve the desired functionality are generated.

1.4 Dependencies in mechatronics (multi-domain) systems

1.4.1 Dependency definition, characteristics, and classification

In business, a dependency is defined as "Relationship between conditions, events, or tasks such that one cannot begin or be-completed until one or more other conditions, events, or tasks have occurred, begun, or completed." (BusinessDictionary.com). When it comes to multidisciplinary engineering design there is no clear definition of dependency. In most relevant research, the definition is taken for granted and it is often hard to distinguish between dependency and the particular domain of interest. There have been very few attempts to formalize and describe the distinctive natures of dependencies, most of those attempts are applied in computer science. A generalized definition of dependency is a relationship between two entities, where a change of state in one entity leads to a change of state in the other entity (Cox, Delugach, & Skipper, 2001). As shown in Figure 1.6, in a dependency d (A, B) between two entities A and B, where A depends on B, the entity A is referred to as the dependent and the entity B is referred to as the antecedent.

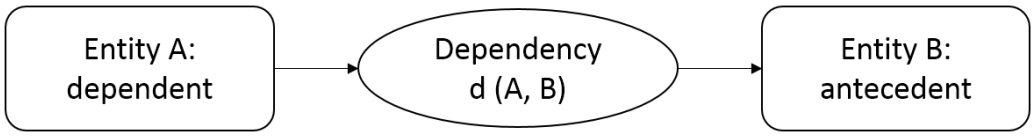

Figure 1.6 Graphical representation of a dependency

Dependencies were assigned the following 6 attributes to facilitate their classification and grouping (Keller, Blumenthal, & Kar, 2000). **Domain, strength, type, activity, formalization, and criticality**. Each attribute can take different discrete values, and dependencies that have similar attributes can be grouped together. Strength for example, expresses how strongly the dependent depends on the antecedent, it can take the 3 following values*, mandatory, optional, and none.*

A different approach (Qamar, Paredis, Wikander, & During, 2012) proposes taking into account the synthesis and analysis nature of properties and dependencies. Synthesis properties (SPs) are used to define system alternatives whereas analysis properties (APs) are used to constitute predictions rather than specifications of the system alternatives. For example, a designer defines the geometry of a part which is a synthesis property and predicts its cost or weight which are analysis properties. A dependency here is defined as a relationship between two properties, where the value of a property depends mathematically on the value of another property. A synthesis dependency reflects choices made by a designers and always results in a synthesis property. For example, a synthesis property can be heuristics for choosing controller gains. Analysis dependencies on the other hand, are relationships between a set of analysis and synthesis properties and are used to derive new analytical properties. An example of an analysis dependency is the prediction of the settling time of a system due to chosen controller gains and other properties such as the mass.

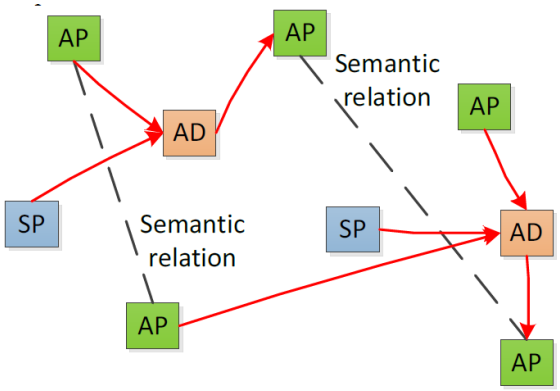

Figure 1.7 Analysis and synthesis properties and dependencies
(Qamar et al., 2012)

The most recent state of the art on dependencies in multi-domain systems is a proposal that classifies product related dependencies during design of mechatronics (J. M. Torry-Smith et al., 2014). By studying and investigating three mechatronic projects, 13 types of dependencies were identified. These dependencies can occur between the following attributes of a product: Functions, means, and properties. For example, a type of dependency between two functions, can be a “sync function” and is defined as a dynamic relation between functions where the timing of the initiation or ending of the concurrently executed functions. Another type of dependency

between a function and a means is an adverse effect, an undesired effect generated by a component and affects another component. For example, electromagnetic waves generated by a motor affect the function of an LCD screen. More information on the following types of dependencies can be found in (J. M. Torry-Smith et al., 2014): *Causal function, state/time function, response function, Fu-M disposition, cumulative Fu-M, property scheme, multidisciplinary means, volume allocation physical, liveliness, physical interface, communication interface.*

1.4.2 Common dependency management tools

Matrix based complexity management tools are the most common way of handling dependencies. Since its introduction in the early 80ies, the Design Structure Matrix (DSM) has been widely taken up in research (Kortler & Lindemann, 2011). The DSM, also referred to as dependency structure matrix, is a square matrix that provides a compact visual representation of dependencies. Elements in a DSM belong to the same domain (components, requirements, tasks…) and are identical on both axes. Dependencies are filled in the matrix cells and should be of the same type, semantically and quantitatively. For example, in Figure 1.8, binary entries in the DSM show tasks that satisfy the same requirements (Braha, 2002), by visually aggregating related tasks and optimally assigning them to working teams the cost and length of product development can be kept at a minimum. In another application, a DSM was used to visualize spatial dependencies between components by mapping them in terms of physical adjacency (Pimmler & Eppinger, 1994). To better express these dependencies, entries in the DSM are numerical instead of binary and range from -2 to 2 as shown in Table 1.1.

		Tasks				
		1	2	3	4	5
Tasks	1	·	0	0	0	0
	2	X	·	0	0	0
	3	X	0	·	0	X
	4	0	0	X	·	0
	5	0	X	0	0	·

Figure 1.8 Task DSM (Braha, 2002)

Table 1.1 Spatial adjacency (Pimmler & Eppinger, 1994)

Required	+2	Physical adjacency is necessary for functionality
Desired	+1	Physical adjacency is beneficial but no necessary for functionality
Indifferent	0	Physical adjacency does not affect functionality
Undesired	-1	Physical adjacency causes negative effects but does not prevent functionality
Detrimental	-2	Physical adjacency must be prevented to achieve functionality

DSMs were extended to Domain Mapping Matrices (DMMs) to include elements from two different domains. DMMs are usually rectangular, and elements from each of the domains are respectively placed either along the vertical axis and the horizontal axis or vice versa. In the work presented by (Danilovic & Browning, 2007), an aircraft design was undergoing a large number of technical upgrades which were based on multiple business deals with various clients; the project

was also divided into functional elements for final system approval and certification (landing gear and breaking system, secondary power system…). Here, a DSM was used to visualize and group related business deals into sub-projects and assign them to different teams. However, with configuration, many designers had to work separately on different business deals that are highly interdependent on the functional level. These dependencies and their degree of interaction (the extent to which the antecedent affects the dependent) were identified and entered in DMM that maps business deals against the elements of the functional organization. By rearranging columns in the DMM, dependencies were clustered to identify sub-projects (group of business deals) that require a high level of coordination and integration.

Managing dependencies in mechatronic systems is a complex procedure that requires modeling multiple domains while viewing various dependency types. These requirements can be met by the Multiple Domain Matrix (MDM), a square matrix that has system elements in exact order on both axes. It is comparable to a DSM except that it can include elements of different types that are grouped into domains on each axis. As shown in Figure 1.9, the resulting MDM is formed out of smaller matrices with DSMs on the diagonal and DMMs elsewhere. Different types of dependencies can be used individually in each sub matrix, therefore MDMs can include all possible combinations of domains and dependency types that can be viewed individually in its DSMs and DMMs. This makes it easier for designers to understand the relationships between elements and apply various analysis methods, such as clustering, separately on each one of the sub-matrices.

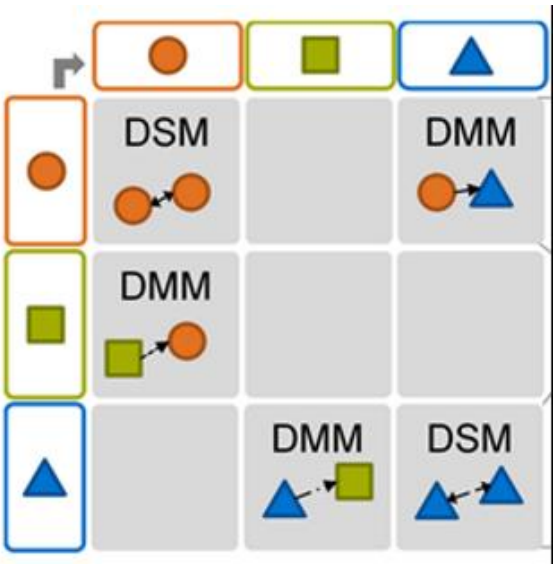

Figure 1.9 Example of a Multiple Domain Matrix (MDM) (http://www.plattformstrategie.de/)

One of many applications where MDMs are used, is the analysis of multiple architectures of a BMW hybrid electric vehicle (Gorbea, Spielmannleitner, Lindemann, & Fricke, 2008). Data were collected through meetings with engineering domain experts and dependencies between and amongst the functional and component domain were used to build the MDM model. ΔMDMS and ΣMDMs, respectively MDM matrix subtraction and matrix addition, where used to compare and highlight the differences between individual architectures of the hybrid electric vehicle. Given that entries in the MDMs are either 0 or 1, a ΔMDM results in a matrix that has 0s where no differences are present and 1s/-1s where dependencies are present in only the first/second MDM. ΣMDM on the other hand provides another kind of information, for example, by looking at the component/function DMM in the ΣMDM of all MDMs (sum of all matrices) one can see the different components that can fulfill a function. This kind of information is very valuable for automation of design, it can be used to compile a repository of components with information on which functions they can fulfil and to which degree (how often).

One of the difficulties that limit the use of matrix based complexity management tools is he complexity and the cost in time of information acquisition. In the previously presented examples, most of the work done using DSMs was applied to designs of existing products, and the

researchers relied on meetings with experts to get information on dependencies between elements in the matrices. When developing new products there is usually a lack of existing documentation, acquiring information to use in DSMs requires time consuming interviews (Lindemann, Maurer, & Braun, 2008). Another difficulty is the manual filling of dependencies in matrices, this activity is also time consuming and highly error prone. These issues are addressed in (Van Beek et al., 2010) and overcame by automatically extracting information on dependencies from the function-behavior-state model and filling them in the matrices. There is also ongoing research to develop clustering algorithms and methods (Hölttä-Otto, Tang, & Otto, 2008; Zakarian, 2008) for complexity management tools, because to get meaningful information from DSMs and understand it, designers rely heavily on rearranging and grouping dependencies in the matrices.

CHAPTER 2 RESEARCH FRAMEWORK

2.1 Challenges addressed and Objective

The main challenges in multi-disciplinary design have been identified in the literature review, the research carried out in this master thesis addresses difficulties faced during preliminary phases of design. By studying other academic efforts, it was revealed that there is a strong need for a design framework for preliminary design complemented by a computational support tool that integrates the conceptual model of mechatronic products with software used at later stages of design, such as CAD and other domain specific tools. As previously mentioned, nowadays' markets are constantly evolving, new competitive products are rolled out, new regulations are put in place, new technologies are developed, and so on. To meet the changes in requirements without much increasing the project lead time, the design framework and support tool should be flexible enough to allow changes to the design with minimum efforts from designers. The major difficulties that arise when modifying a design are the ability to determine the consequences that result because of those changes, assessing the effects of choosing between alternatives, and evaluating the design concept.

Dealing with these difficulties requires strong communication between the various disciplines and management of dependencies. Advancements in matrix based dependency management tools have come a long way, their limitations do not lie in the tools themselves but in the difficulty acquiring information about dependencies and modeling them. In the preliminary phases of design, the product concept is not very well defined yet and most of the underlying dependencies are only known at a coarse grain level in the designers' mind (Qamar et al., 2012). As the number of component in a system increases, the number of dependencies increases exponentially. Modeling them often proves to be quite time consuming, and shifts the designers' attention away from solution finding. Often these dependencies are left unattended, only to reappear at later stages of design and cause integration problems which require costly design iterations to fix.

The work in this research proposes a design methodology using a functional reasoning framework and a new way to model dependencies during conceptual design. The aim is to provide a dynamic modeling tool that allows designers to focus on solution finding, and a

methodical acquisition and representation scheme of dependencies that encourages designers not to ignore pertinent information and boosts the synthesis activity.

This master thesis is framed by the following research questions: Can a functional reasoning framework act as a common language understood among designers and improve collaboration?

Of this research question stems the following sub-questions: Can the project lead time be reduced by capturing and managing abstract dependencies during conceptual design? Will it help designers better integrate work from various domains to avoid design iterations?

2.2 Proposed framework and dependency modeling

2.2.1 Functional Reasoning Framework

In this section, we will introduce the functional reasoning framework that was used in order to describe the product flow and help highlight the dependencies.

2.2.1.1 Ontology

The term ontology has its roots in philosophy where it is defined as a systematic account of existence. In engineering, ontology is an explicit specification of conceptualisation (Gruber, 1993). It is a set of objects and relationships among them that are used to represent knowledge in a domain, this set forms a primitive vocabulary for knowledge based systems (Kitamura, Ikeda, & Mizoguchi, 1997) such as the reasoning method presented in the next sections. Flexibility and formality are two qualities of ontology that often have a negative correlation. For example, representing functions as a finite number of types lacks flexibility and does not allow designers to cover all possible functionalities (Erden et al., 2008). However, such a representation provides a formal vocabulary that can be easily computationally searched and analyzed.

The functional basis for design presented in the literature review will be used as an ontology in this functional framework, it is a comprehensive list of function-flow that can be used to model multi-domain systems. The set of functions and flows are shown in Table 2.1. Such a formal representation allows repeatable and meaningful results from modeling and is meant for use in

design repositories, product architecture, and design synthesis (Hirtz et al., 2002). The functional basis for design has its limitations and advantages, the aim here is to develop a functional reasoning framework that serves as a proof of concept and a structure for the dependency modeling method proposed in the next sections.

Table 2.1 Functional basis set of functions and flows (Hirtz et al., 2002)

Functions		***Flows***	
Primary	***Secondary***	***Primary***	***Secondary***
Channel	Distribute, Import, Export, Transfer, Guide	Material	Human, Gas, Liquid, Solid, Plasma, Mixture
Connect	Couple, Mix	Signal	Status, Control (Analog, discrete)
Control	Actuate, Regulate, Change, Stop		
Convert	--	Energy	Human, Acoustic, Biological, Chemical, Electrical, Electromagnetic, Hydraulic, Magnetic, Mechanical, Pneumatic, Radioactive/Nuclear, Thermal
Provision	Store, Supply		
Signal	Sense, Indicate, Process		
Support	Stabilize, Secure, Position		

2.2.1.2 Developing a Representation Scheme

The scope of this research is to support designers with concept generation and dependency management, however the farfetched ultimate goal is to provide a tool that allows concurrent and integrated design throughout the design process. SysML, defined in italic below, is a very

powerful language that can, with modifications, fulfill this task and act as a representation scheme for the functional basis for design. Also, SysML has already been used as a common modeling language to map between various domain specific tools as shown in Figure 1.3 (Shah et al., 2010).

"*SysML is a general-purpose graphical modeling language that supports the analysis, specification, design, verification, and validation of complex systems. These systems may include hardware and equipment, software, data, personnel, procedures, facilities, and other elements of human made and natural systems. The language is intended to help specify and architect systems and to specify components that can then be designed using other domain-specific languages, such as UML for software design, VHDL for electrical design, and three-dimensional geometric modeling for mechanical design. SysML is intended to facilitate the application of an MBSE approach to create a cohesive and consistent model of the system." (Friedenthal, Moore, & Steiner, 2014).*

SysML is decomposed into meta-classes that are used to illustrate concepts in the modeling domain. It uses *stereotypes* to support domain specific modeling, they are mechanisms that allow customization of the meta-classes to suit the designers' needs. To speed up modeling, SysML also supports the creation of model libraries, collections of reusable elements that can be defined by a system modeller.

There are multiple tools in the market that support the modeling language SysML. MagicDraw, *the multi award-winning UML business process, architecture, software and system modeling tool with teamwork support*, is used here due to the availability of an OpenAPI that facilitates integrating a reasoning method to the functional framework.

To create a representation scheme and provide a computational tool to use for functional modeling, the customization capabilities of SysML were used in this master thesis to create a custom functional modeling diagram and a model library of the set of functions and flows presented in the functional basis for design.

The material, energy, and signal flow stereotypes shown in Figure 2.1 extend the *activity parameter node*, *object flow*, and *class* meta-classes, while the function stereotype in Figure 2.2 extends the meta-class *call behavior action*. This represents one of the original contributions of the work presented in this master thesis and will be further explained in the following sections.

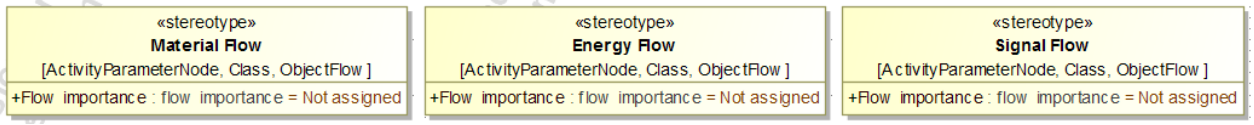

Figure 2.1 Material, Energy, and Signal flow stereotypes

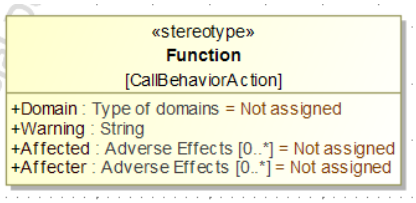

Figure 2.2 Function stereotype

The meta-classes *activity parameter node*, *object flow*, and *call behavior action* in SysML are used in activity diagrams. Extending these stereotypes allows the creation of a custom functional modeling diagram and a user interface in MagicDraw as show in Figure 2.3.

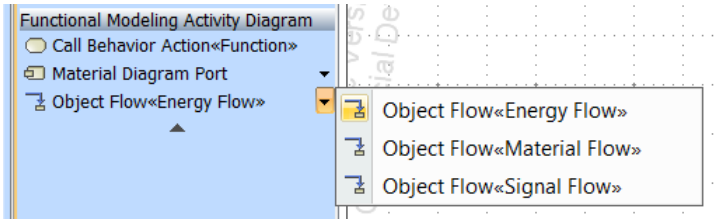

Figure 2.3 Custom functional modeling diagram

Figure 2.4 and Figure 2.5 show the functional basis for design libraries. A hierarchical decomposition was achieved using the *generalization* path symbol which indicates for example that a *particulate* flow is a *solid* flow, which is also a *material* flow. The hierarchy allows flexible modeling, where a more specific flow can be connected to a more abstract port. For example, a gas-liquid flow can be connected to a to mixture port, but not vice versa. It also allows

writing algorithms and rules at different levels of abstraction which will be elaborated in the next section.

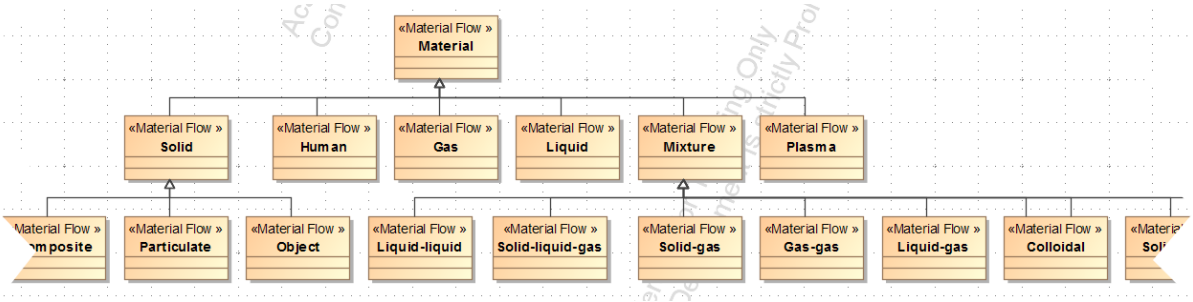

Figure 2.4 Material flow hierarchical decomposition in SysML

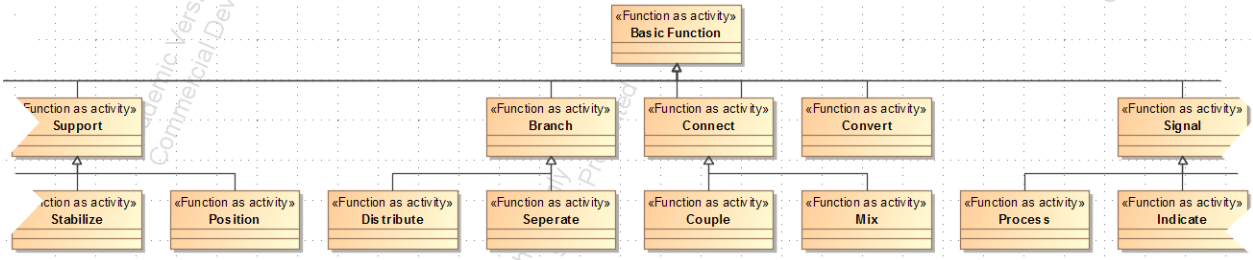

Figure 2.5 Function hierarchical decomposition in SysML

Figure 2.6 shows a part of the functional diagram found in Figure 1.4 modeled in MagicDraw using the custom functional diagram and the functional basis libraries.

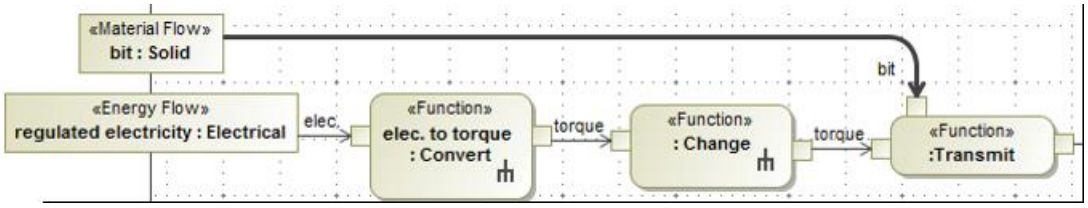

Figure 2.6 Custom functional modeling diagram example

2.2.1.3 Implementing a Reasoning method

This section is divided into three parts. The first two parts explain how the MagicDraw openAPI and the rule based expert system “CLIPS” were used to develop and implement the reasoning method, and how they can infer information from the functional modeling diagram to support the

engineering designers in planning and conceptualization. The third part illustrates an example where possible heuristics followed by designers were integrated into the reasoning method to automatically modularize the functional model.

2.2.1.3.1 Linking the system modeling and the reasoning method: MagicDraw openAPI plug-in

The openAPI supports the modification of Magic Draw's functionality through the creation of plug-ins. After customizing SysML to support functional modeling using the functional basis for design, a plug-in was developed using JAVA to check the functional diagrams for consistency at the representation level and report it back to the designer. For example, the plug-in would check for things like: ports that are not connected to a flow, types of flows that do not match, a "branch" function that does not split the incoming flow into 2 outgoing flows, ... If the functional diagram is proved to be complete/consistent, the plug-in was coded to proceed and extract relevant information from the functional diagram and export it in a more simplified format that will be used by the rule based expert system (CLIPS) and explained in the next section.

This automated approach reduces the burden on the engineers as it helps them carryout a very high-level of consistency check-up in their models in a very effective, systematic, and automated manner. It is worth noting that from now on, in this master thesis, the MagicDraw openAPI plug-in will be referred to as MagicDraw plug-in.

2.2.1.3.2 Rule-based expert system CLIPS

Expert systems are computer programs where the knowledge and the reasoning process of human experts are codified in an attempt to mimic their decision making skills. Rule-based expert systems are usually composed of these main components: an interaction mechanism, a knowledge base, and an inference engine. Rule-based expert systems are used in various engineering domains (Achiche, Appio, McAloone, & Di Minin, 2012; Achiche, Baron, Balazinski, & Benaoudia, 2007; Ren, Balazinski, Jemielniak, Baron, & Achiche, 2013)

The **interaction mechanism** can be either a user interface that the expert system uses to interact with a user or an integration platform that allows it to interact with other computer applications.

The **knowledge base** contains a set of rules that are coded to encapsulate information acquired from human experts. Rules are written in an *If A Then B* format, where A is called the antecedent and B is called the consequent. When the antecedent of a rule is matched to data in the working memory, data usually provided by the interaction mechanism, the rule is said to be satisfied and can be triggered to perform the action stored in its consequent. Actions could be anything from modifying data in the working memory, storing new information for other rules to use, to reporting back to the interaction mechanism.

The **inference engine** is the main processing unit of an expert system, it manages the rules that are satisfied and choses in which order to trigger them based on priorities set in the code and the followed search strategy.

CLIPS, an acronym for **C L**anguage **I**ntegrated **P**roduction **S**ystem, is an expert system tool developed by NASA and released in 1986, and will be used for reasoning in the functional framework. As previously mentioned, the MagicDraw plug-in was used to extract relevant information from the functional diagram and export it to the working memory of CLIPS. The functional modeling diagram in CLIPS is represented using the templates shown in Figure 2.7 and developed in this research work. Input and output templates define the elements that are on the boundary of the functional diagram.

All elements in a SysML diagram have an unique identifier. To have an exchangeable model between MagicDraw and CLIPS, the slot “id” was added to all templates of the CLIPS representation and is used to store the unique identifier.

The slot “name” is not needed for computational purposes and is there only to facilitate debugging and explanation.

The slot “type” is used to indicate the category of the function or flow found in the functional basis for design set.

```
(deftemplate function
        (slot name)
        (slot id)
        (slot type)
)
(deftemplate flow
        (slot name)
        (slot id)
        (slot type)
        (slot source-id)
        (slot destination-id)
)
(deftemplate input
        (slot name)
        (slot id)
        (slot type)
)
(deftemplate output
        (slot name)
        (slot id)
        (slot type)
)
```

Figure 2.7 Functional modeling diagram representation in CLIPS

The slots “source-id” and “destination-id” store the unique identifiers of the functions to which a flow is connected. Figure 2.8 shows a sample of how the functional diagram template representation is written when exported by the plug-in to CLIPS.

Since the ontology and the representation scheme are hierarchically structured, rules can be written to reason at various levels of abstraction. Rules can be used for various goals such as pattern matching and component selection, heuristic search, clustering … The next section illustrates the implementation of heuristic search on the functional diagram in CLIPS to identify modules of functions. An example of a rule is shown in Figure 2.14.

```
(flow (name _) (type Liquid) (id _17_0_4_104e02e2_1415811236860_761427_15901)
(source-id _17_0_4_104e02e2_1415811070265_235859_15829) (destination-id
_17_0_4_104e02e2_1415811179996_395061_15881))

(flow (name _) (type Analog) (id _17_0_4_104e02e2_1415811487496_115363_15972)
(source-id _17_0_4_104e02e2_1415811070265_235859_15829) (destination-id
_17_0_4_104e02e2_1415811034743_711086_15801))

(function (name _get_hot) (type Import) (affecter Heat ) (affected Heat ) (id
_17_0_4_104e02e2_1415808885697_488550_15366))

(function (name _hot_flow) (type Regulate) (affecter Electric_Field Heat
Vibrations ) (affected Heat Vibrations ) (id
_17_0_4_104e02e2_1415810183386_279355_15663))

(input (name _water) (type Liquid) (id
_17_0_4_104e02e2_1415808399436_528481_15173))

(input (name _temperature) (type Analog) (id
_17_0_4_104e02e2_1415809332106_317220_15581))

(output (name _mixed_water) (type Liquid) (id
_17_0_4_104e02e2_1415811179996_395061_15881))
(chains)
```

Figure 2.8 Exported functional diagram using CLIPS templates sample

2.2.1.3.3 Assisted modularization using rule-based heuristics

As previously mentioned in the literature review, functional modeling acts indirectly as a design scheme that guides designers in activities such as decomposing problems, generating concepts and creating product architectures (Hirtz et al., 2002). To allow modular design to be carried on earlier in the product development phase, three heuristic methods for identifying modules in a functional diagram were developed. The efficacy of these methods was confirmed and verified by applying them on a database of 70 consumer products (Stone et al., 2000).

In the work presented here, we aim to support engineering designers in modularising functional models. Before explaining how these heuristics are translated, in this work, to be used as embedded rules in CLIPS, they are presented below, each with a corresponding figure that illustrates it.

The **Dominant flow** heuristic, as shown in Figure 2.9, defines a module as a *set of sub-functions which a flow passes through, from entry or initiation of the flow in the system to exit from the system or conversion of the flow within the system*.

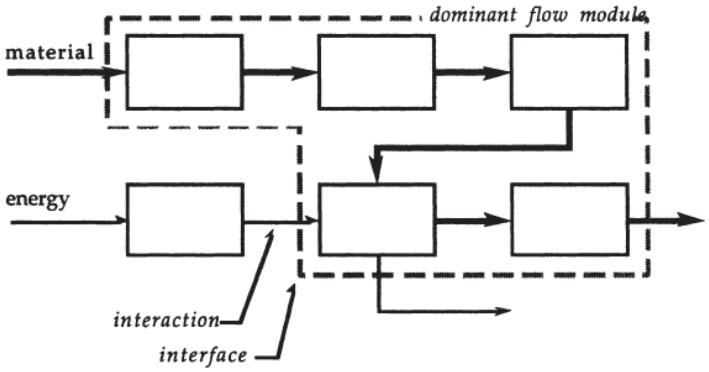

Figure 2.9 Dominant flow module (Stone et al., 2000)

The **Branching flow** heuristic, as shown in Figure 2.10, defines modules as *the limbs of a parallel function chain. Each of the modules interface with the remainder of the product through the flow at the branch point.*

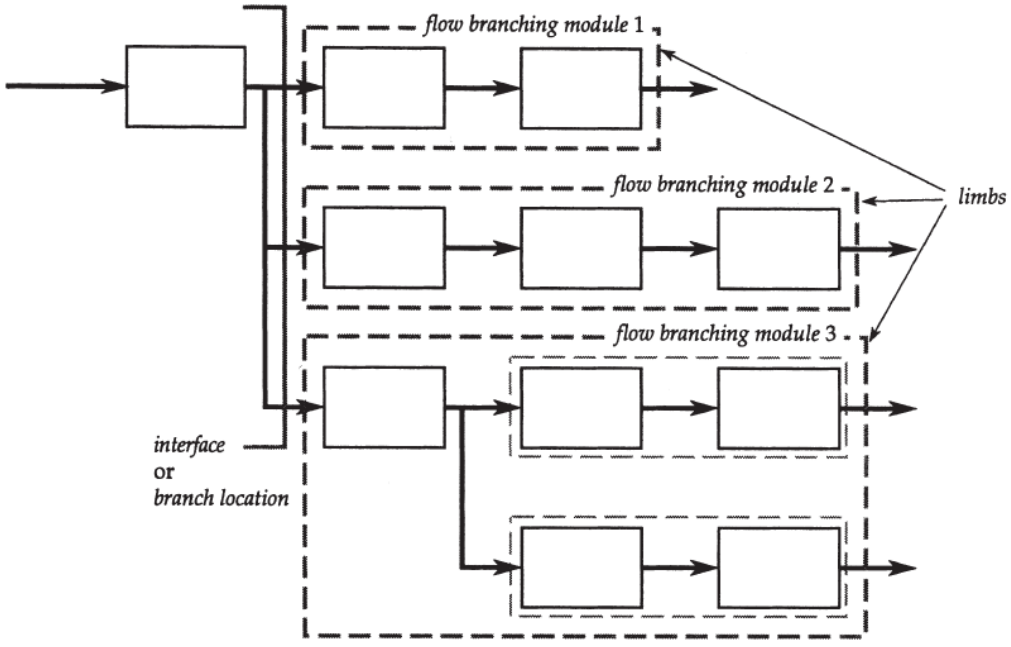

Figure 2.10 Branching flow module (Stone et al., 2000)

The **Conversion-transmission flow** heuristic, as shown in Figure 2.11, defines a module as *a conversion sub-function or a conversion—transmission pair or proper chain of sub-functions.*

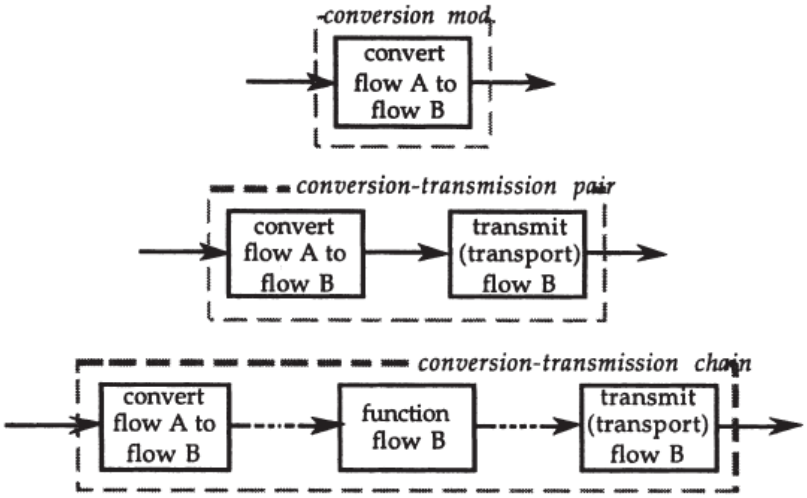

Figure 2.11 Conversion-transmission flow module (Stone et al., 2000)

Creating chains of functions by coding the heuristics at a high level of abstraction is very difficult if not impossible. Getting more information from the designing user about the functional diagram, such as to which flow each of the functions are applied, hinders the dynamic modeling capacity of the modeling tool, one of the main goals of this research master thesis.

Instead of identifying chains of functions, rules that identify chains of flows are coded. The results are then presented to assist the designing user modularize the functional diagram. If the diagram has a simple layout, the chains of flows will always correspond to the expected chains of functions. When the diagram is more complex, such as having multiple flows passing through functions, the user can be given the choice to select the resulting chains of flows that are pertinent. All the flows in a chain that is selected are marked as “primary flows” and the functions that they pass through are grouped to create a module. This method facilitates information acquisition about primary flows in the diagram and assists the designer in identifying modules faster.

The following logic was coded into CLIPS and is centered around the functional modeling representation defined in the templates above. To note, due to time limitation, but not technical, only the flow chain identification was coded and tested, but not integrated into MagicDraw.

First, a new slot “chain” was added to the flow template, and a new “chain” template was created as shown in Figure 2.12 and Figure 2.13. The slot “chain” is an integer, its default value is 0, and it is used as a flag that indicates if a flow was added to a chain or not. The template chain, has a multi-slot “flow-ids” where the unique identifiers of the flows that belong to it can be added.

```
(deftemplate flow
        (slot chain (type INTEGER))
        (slot name)
        (slot id)
        (slot type)
        (slot source-id)
        (slot destination-id)
)
```

Figure 2.12 Flow template with chain slot CLIPS

```
(deftemplate chain
        (slot number)
        (slot type)
        (slot id)
        (multislot flow-ids)
)
```

Figure 2.13 Chain template CLIPS

As previously mentioned, the inference engine in an expert system decides in which order satisfied rules can be executed. CLIPS allows giving priority to some satisfied rules to be executed before other using “salience”. Salience can be added to the definition of a rule and can take values up to 99, rules with higher salience have higher priority.

To identify chains of flows, two types of rules are coded, ones that identify a starting point of a chain (a starting flow) and ones that propagate from the starting points adding flows to each chain until it they can no longer be satisfied. Rules that identify starting points have a higher priority to be triggered and identify these types of flows: input flows, flows that branch, flows that merge.

Figure 2.14 shows an example of a rule that labels a branch of a flow as a starting point of a new chain. Section 4 shows an example where assisted modularization is used.

```
(defrule startingpoint3
        (declare (salience 3))
        (flow (destination-id ?f-id) (type ?t))
        (flow (id ?id1)(source-id ?f-id) (type ?t))
        ?to-mod2<-(flow (id ?id2) (source-id ?f-id)(type ?t)(chain 0))
        ?to-mod1<-(chains(nbr ?n1))
        (test (neq ?id1 ?id2))
        =>
        (modify ?to-mod2 (chain (+ ?n1 1)))
        (modify ?to-mod1 (nbr (+ ?n1 1)))
)
```

Figure 2.14 Rule that identifies flows that branch CLIPS

2.2.2 Dependency Modeling

This section proposes a methodical acquisition and representation scheme of dependencies that encourages designers not to ignore pertinent information and boosts the synthesis activity. During preliminary phases of design there is a lot of non-quantified abstract information that needs to be modeled, the following method aims to help experts concretize their knowledge on dependencies in multi-domain systems.

2.2.2.1 Modeling Using Meta-Dependencies Concept

Modeling dependencies using common graphical methods, such as shown in Figure 1.6, requires building relationships between each and every single element in the system. As the number of elements in a system increases, the number of dependencies increases exponentially, and

managing these dependencies becomes costly and time consuming if not impossible, specially when doing design modifications.

Instead of building relationships between all dependent elements, we propose the concept of meta-dependencies such as shown in Figure 2.15, where a dependency is defined as a relationship between an element (function, module, component, …) and a meta-dependency. Similarly to the graphical dependency shown in Figure 1.6, the elements E1 and E2 in Figure 2.16 are considered dependent and antecedent elements respectively and the potential relationship dependency between them is shown in dashed lines.

Early in the design process, dependencies tend to be too abstract to be easily represented and their existence is often uncertain. Since several elements share the same type of dependency, using a meta-dependency modeling scheme allows grouping these related dependencies under the same model. Therefore, this helps reduce the number of relationships to be created, and encourages implicit acquisition of abstract information during design activities.

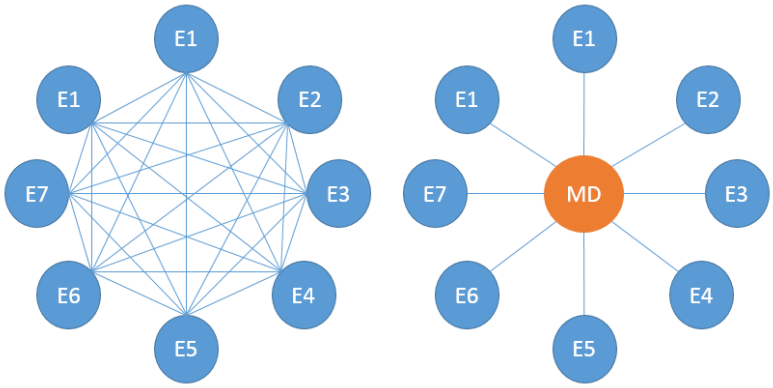

Figure 2.15 Meta-dependency modeling

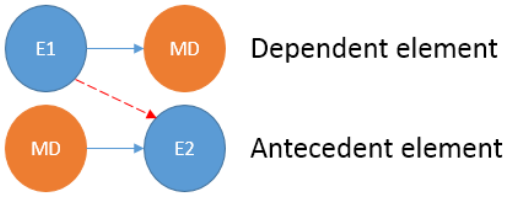

Figure 2.16 Meta-dependency elements

2.2.2.2 Tagging elements with adverse effect dependencies

To use a meta-dependency modeling scheme, the dependency at hand should be shared among several elements in the system. Adverse effects for example, are undesired effects such as heat, vibration, and electromagnetic waves, that are generated by one element and affect the functionality of other elements in the system. Adverse effects are one of many types of dependencies that can be modeled using the meta-dependency modeling scheme and will be used as an illustration in the rest of this research.

To reduce the graphical clutter and make the design process more dynamic, instead of building relationships with a meta-dependency we propose to tag elements with adverse effects. Using SysML's customization mechanism two new property slots were added to the function stereotype, ***affecter*** and ***affected*** as shown in Figure 2.17. These slots can take as values adverse effects that are defined by the designer. When the slot ***affecter*** is tagged by an adverse effect, it implies that the function can generate such an undesired effect and that it has an antecedent relationship with this meta-dependency. When the slot ***affected*** is tagged, it implies that this function's performance can be hindered by this adverse effect and that it has a dependent relationship with it.

Name	Electricity to Torque
Applied Stereotype	«» Function [CallBehaviorAction] [Functional M
Behavior	Convert(in, out) [Functional Modeling Li...
Warning	
Domain	Not assigned
Affecter	Electric Field Vibrations Heat
Affected	

Figure 2.17 Affecter and affected meta-dependency tagging slots

2.2.2.3 Automated Generation of the Dependency Matrix

As cited above, dependency structure matrices (DSMs) offer designers an overview of the system at hand and the relationships between its elements but are quite complex and time consuming to generate manually by engineers. By using the meta-dependency modeling scheme, DSMs can be automatically generated, thus freeing designers from filling them which saves them time and reduces possible human induced errors.

To achieve this, SysML's customization mechanism, the functional reasoning expert system, and the MagicDraw plug-in were used in this master thesis.

As shown in Figure 2.18, the dependency relationship in SysML was extended to create adverse effects using a stereotype.

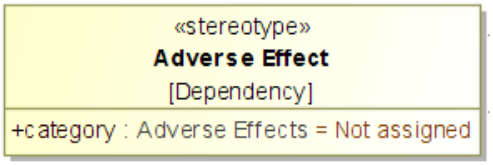

Figure 2.18 Adverse effect dependency stereotype bloc

Two multi-slots for the attributes ***Affecter*** and ***Affected*** were added to the function template in the CLIPS knowledge base. Then new rules were developed and then embedded to create an adverse effect dependency between functions that affect each other. An example of a rule is show in Figure 2.19 and explained in italic below, more complex rules can be written to derive dependencies by taking multiple factors into account and will be further discussed in the future work section.

If *Function1 is affected by AdverseEffectA*

& *Function2 is an affecter of AdverseEffectA*

Then *a Dependency of type AdverseEffectA will be created*

with dependent Function1 and antecedent Function2

```
(defrule createDependency
        (function (name ?n1) (id ?id1) (affecter $? ?aff $?))
        (function (name ?n2) (id ?id2) (affected $? ?aff $?))
        =>
        (assert (dependency (type ?aff)(affecter-id ?id1)(affected-id ?id2) ))
)
```

Figure 2.19 createDependency CLIPS rule

The openAPI was then used to import the dependencies from the CLIPS generated file and create adverse effect dependencies between the two functions in the SysML model; and then create a DSM for each type of adverse effect to present them to the designer user. Section 4.1, Illustrating the principle, shows a design process example where multiple DSMs are created.

2.2.3 Design Procedure summary

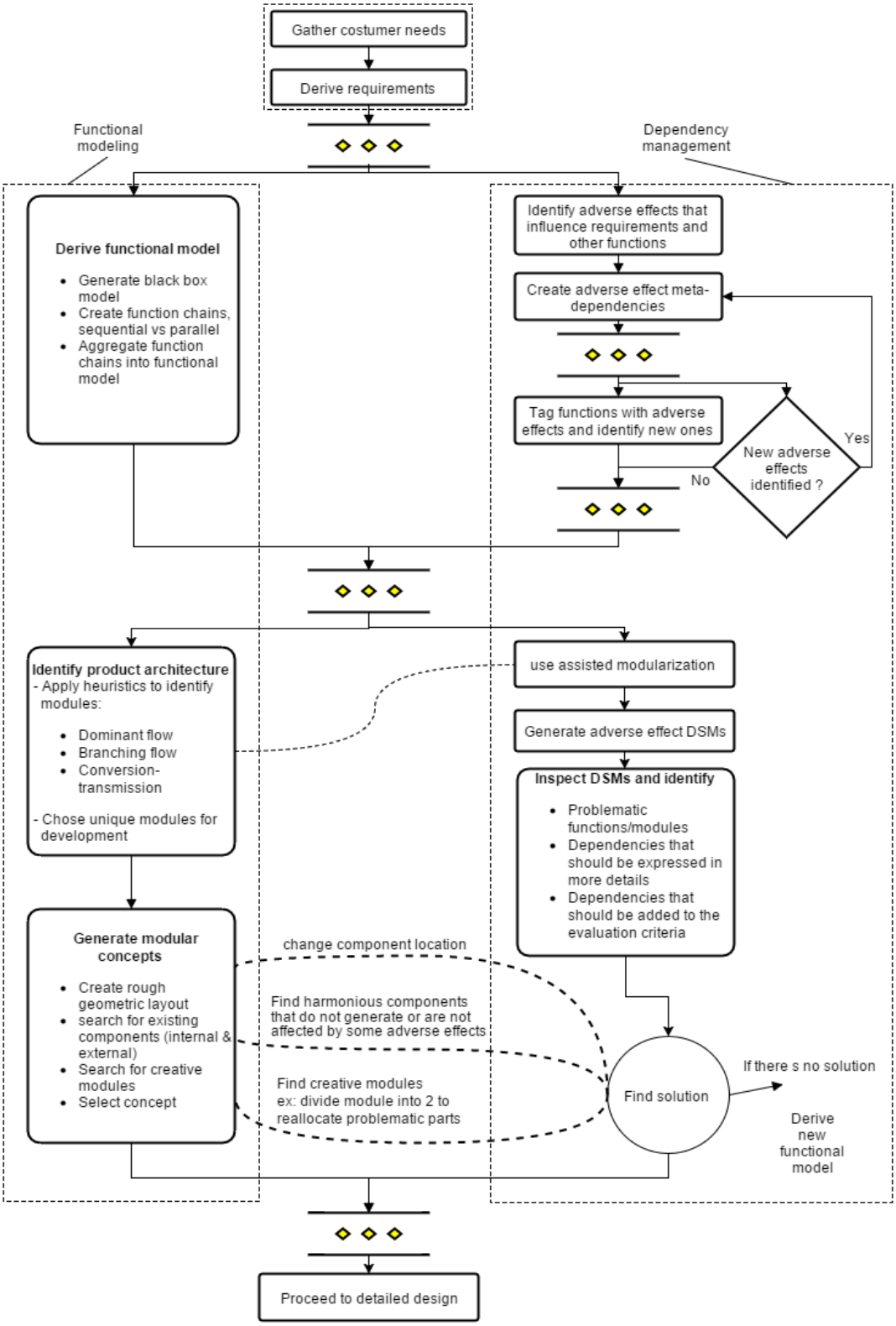

Figure 2.20 Design procedure flowchart

Figure 2.20 sums up the design procedure and methodology that should be followed while using the developed design support tool in this master thesis. Engineering designers start the conceptualization of a mechatronic product by gathering customer needs and deriving requirements and then they proceed by translating these requirements into a functional model of the product. Here designers are required to simultaneously identify adverse effects that can possibly affect or be generated by the various functions and tag them appropriately.

To identify the product architecture, designers are required to use the assisted modularization to help them apply the heuristics and identify modules. They should also inspect the automatically generated dependency matrices to identify problematic functions or modules, identify important dependencies that should be expressed in more detail or be added to the evaluation criteria of the product.

While generating modular concepts, designers should take account of all the problematic dependencies. The easiest way to avoid some adverse effect dependencies is to choose compatible components in a system, however sometimes they can be avoided by creating new creative modules and changing the physical allocation of some components. If no solution can be easily implemented sometimes it is better to derive a new functional model of the product.

The following section will illustrate how this design method and the support tool can be used in two concrete examples.

CHAPTER 3 CASE STUDY

3.1 Illustrating the principle

In order to illustrate the developed design support tool, a small example of a design of a mechatronic device that regulates the temperature of water by mixing a cold flow with a hot flow will be used. Figure 3.1 shows a simplified functional diagram of the device, it is an abstract model of the concept and can be transformed into various working physical models.

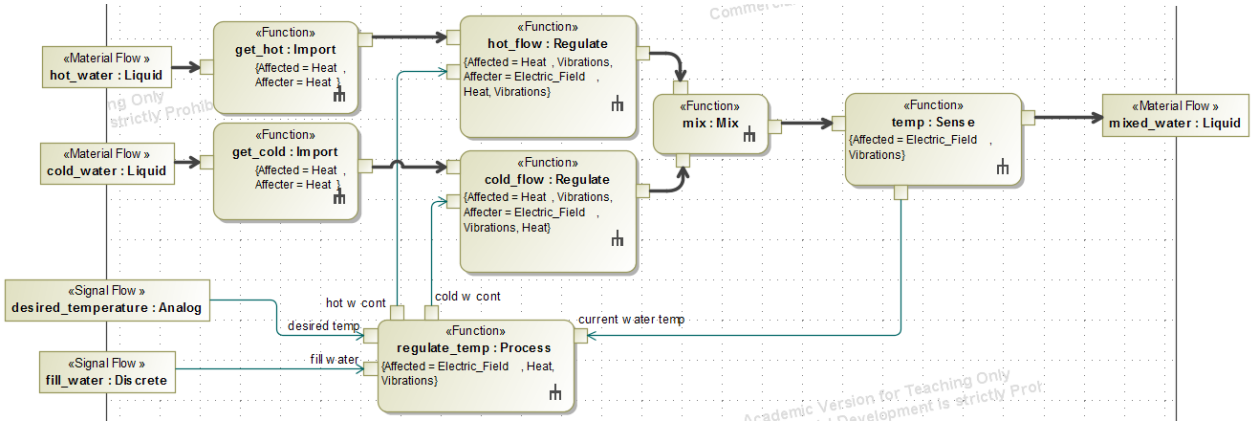

Figure 3.1 Temperature regulator functional model

Many components, such as a proportional solenoid valve or a motorized valve, can fulfill the function "regulate hot flow", It is up to the designer to decide which one suits the application better. However, while one component might perform better than another, it might hinder the overall performance of the device.

Functions are tagged by effects capable of impairing their intended behavior, as well as effects they might generate. Dependencies between functions are then automatically generated by the MagicDraw plug-in and presented in design structure matrices as shown in Figure 3.2.

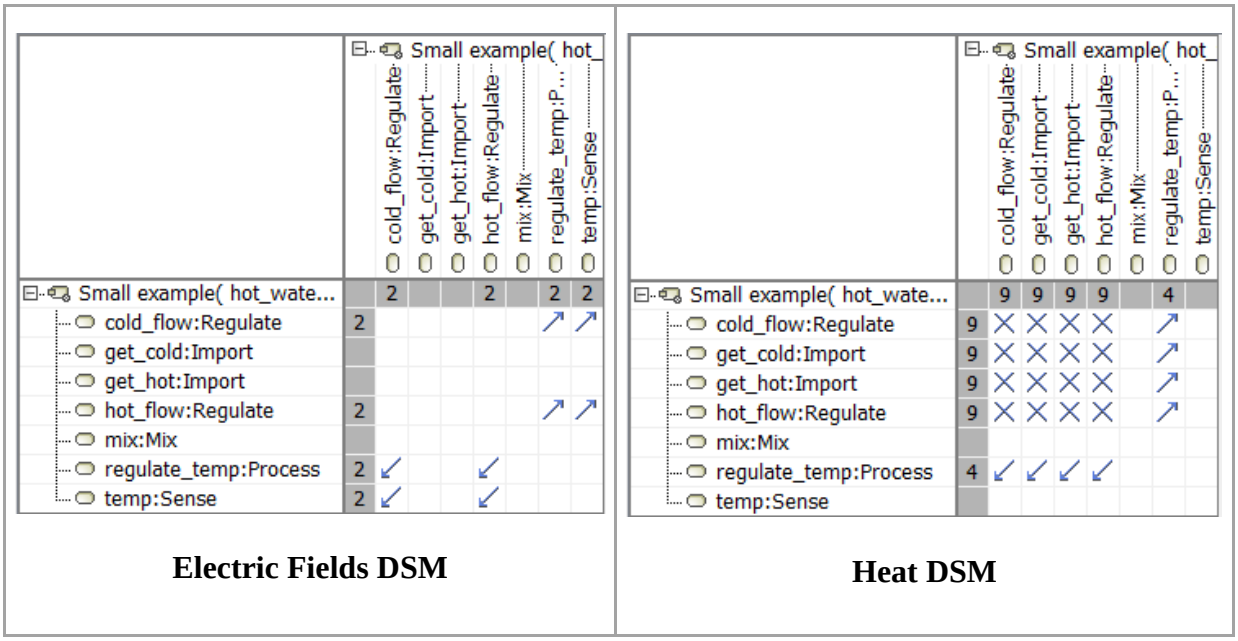

Figure 3.2 Electric field DSM and heat DSM of temperature regulator

Solenoid valves generate electric/magnetic fields when activated, and a lot of heat when operating continuously, these two undesired adverse effects can cause other components such as the temperature sensor and the control circuit to malfunction. This creates dependencies between them. Requirements come into play when trying to find the best solution and the designer needs to answer several questions such as:

- Is there a device size limit?
- How far from the valves can the temperature sensor be positioned?
- What's the required temperature accuracy?
- Can a less accurate sensor that is not affected by electric fields be used?
- For how long will the solenoids operate?
- Is heat management required?
- Is it enough to insulate the electric circuit?
- Or is a heat dissipation device required?
- Which one of these possible solutions is cheaper to produce?
- Etc.

All these questions were derived out of two adverse effect dependencies between three components. If left unaddressed, they might reappear at some point during the detailed design

phase or cause functional problems after production, which will require costly design feedback iterations in order to fix them.

However, one can argue that, for a simple system as the one presented above, one does not necessarily need a design support since the adverse effects dependencies can be easily found by the engineering designers. Therefore, in order to better illustrate the developed design support tool, it will be applied to a more complex mechatronic system such as a quadrotor drone.

3.2 Quadrotor conceptual design

A quadrotor is a highly complex system to design, it involves various engineering domains that affect each other such as aerodynamics, mechanics, control, and intelligence (Mohebbi, Achiche, & Baron, 2015). Integrating various sensors, actuators, a power supply, and other components into a lightweight flying system that can operate for a relatively long period is not an easy task and requires a well-defined methodology.

In order to validate the conceptual design framework developed in chapter 3, it was applied on a conceptual design task of a quadrotor while taking into account adverse effect of complex dependencies. Nowadays, quadrotor drones are considered as mature products; their design went through much iteration and has been largely optimised. The experience acquired over the time by designers is very well documented in the literature, on online websites, and hobbyist forums [1][2]. This well documented information was surveyed in this research to find the various possible component alternatives that can fulfill the desired functionality of a quadrotor drone, and to identify the typical design problems faced by engineers during development that are related to adverse effect dependencies.

Figure 3.3 shows the developed functional model of the quadrotor, the structure is considered rigid and symmetric, thus only one of the four symmetric ends of the quadrotor was modeled.

---

[1] http://copter.ardupilot.com/wiki/common-vibration-damping/

[2] http://dronespeak.com/article-topics/uas-potential-topics/electromagnetic-radio-frequency-emission-interference-reduction/

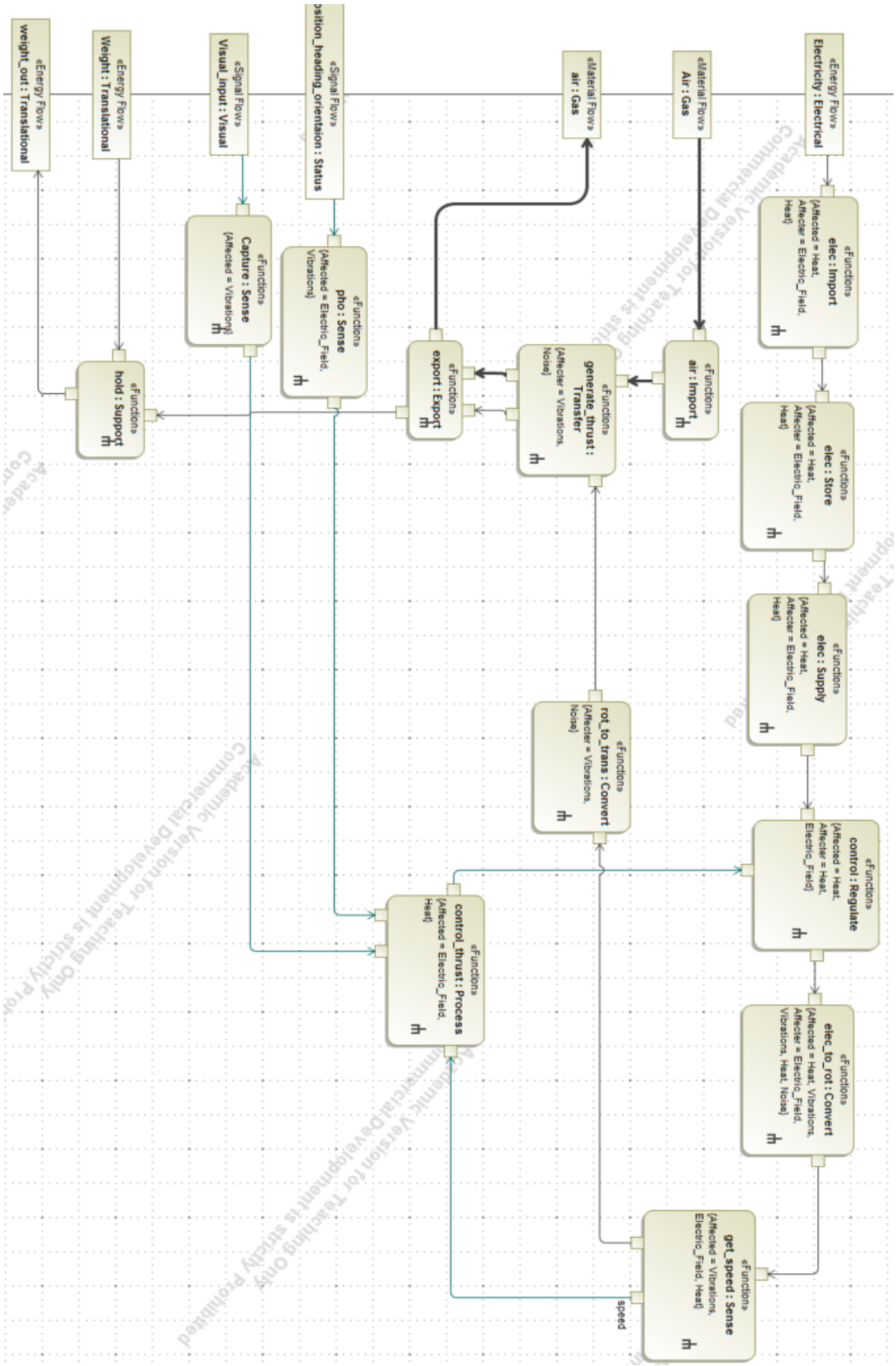

Figure 3.3 Partial Quadrotor drone functional model

The main flows that go in and out of the black box system representing the drone are:

- Energy in: Electrical, Weight
- Energy out: Weight
- Material in/out: Air
- Signals in: Position, Heading, Orientation, and Visual input

While developing the functional model, each function was tagged with possible adverse effects that it can generate or that can hinder its functionality, regardless of the alternative chosen. At this stage of design, these dependencies are still not very clear in the designer's mind, and they should only be implicitly captured in the form of tags to be examined closely later on.

The adverse effects related to the relevant functions in the system were documented in Table 3.1, as well as the alternative components that can possibly fulfill them.

Table 3.1 Relevant Functions, Possible Adverse Effects, Alternative components

Relevant Function	**Possible adverse effects**	**Alternative components**
Store Electricity	Affected: Heat Affecter: Electric field, Heat	Lithium ion Lithium polymer Alkaline
Supply Electricity	Affected: Heat Affecter: Electric field, Heat	Power regulator Battery eliminator circuit

Table 3.1 (continued) Relevant Functions, Possible Adverse Effects, Alternative components

Regulate Electricity	Affected: Heat Affecter: Electric field, Heat	Power transistors (some DC motors) Electronic Speed Controller (AC motors)
Convert Electricity To Rotational energy	Affected: Heat, Vibrations Affecter: Electric field, Vibrations, Heat, Noise	AC motors • Brushed • Brushless DC motors
Sense Rotational speed	Affecter: Affected: Vibration, Electric field, Heat	Encoder Tachometer Current sense None (open loop)
Sense Position, Heading, Orientation	Affecter: Affected: Electric field, Vibrations	Accelerometer, Gyroscope, Magnetometer, GPS IMU GPS + Compass
Sense Visual input	Affecter: Affected: Vibrations	Camera

The MagicDraw plug-in was then used to assist in the modularization of the functional diagram. Figure 3.4 shows the results displayed by the CLIPS rule-based expert system and Figure 3.5 shows these results displayed on the SysML functional diagram.

```
(flow (chain 1) (name _) (id _17_0_4_104e02e2_1433352288919_193259_12975) (type Electrical) (sourc
(flow (chain 1) (name _) (id _17_0_4_104e02e2_1433352717762_122541_13016) (type Electrical) (sourc
(flow (chain 1) (name _) (id _17_0_4_104e02e2_1433786690123_579388_13123) (type Electrical) (sourc
(flow (chain 1) (name _) (id _17_0_4_104e02e2_1433786713823_108631_13135) (type Electrical) (sourc
(flow (chain 1) (name _) (id _17_0_4_104e02e2_1434549877416_766474_13183) (type Electrical) (sourc
(flow (chain 2) (name _) (id _17_0_4_104e02e2_1433355964752_887815_13429) (type Gas) (source-id _1
(flow (chain 2) (name _) (id _17_0_4_104e02e2_1433355988074_499910_13441) (type Gas) (source-id _1
(flow (chain 2) (name _) (id _17_0_4_104e02e2_1433356465417_483092_13533) (type Gas) (source-id _1
(flow (chain 2) (name _) (id _17_0_4_104e02e2_1433356595811_701093_13581) (type Gas) (source-id _1
(flow (chain 3) (name _) (id _17_0_4_104e02e2_1433357325072_334792_13673) (type Status) (source-id
(flow (chain 4) (name _) (id _17_0_4_104e02e2_1433357614215_503149_13780) (type Visual) (source-id
(flow (chain 5) (name _) (id _17_0_4_104e02e2_1433358188389_847693_13855) (type Translational) (so
(flow (chain 6) (name _) (id _17_0_4_104e02e2_1433359804729_504766_14112) (type Translational) (so
(flow (chain 7) (name _) (id _17_0_4_104e02e2_1433353569491_531912_13134) (type Rotational) (sourc
(flow (chain 7) (name _) (id _17_0_4_104e02e2_1433353940221_778028_13245) (type Rotational) (sourc
(flow (chain 8) (name _) (id _17_0_4_104e02e2_1433353666902_834906_13185) (type Signal) (source-id
(flow (chain 9) (name _) (id _17_0_4_104e02e2_1433353683299_225339_13197) (type Control) (source-i
(flow (chain 10) (name _) (id _17_0_4_104e02e2_1433355890350_62940_13382) (type Translational) (so
(flow (chain 10) (name _) (id _17_0_4_104e02e2_1433356478971_626375_13550) (type Translational) (s
(flow (chain 10) (name _) (id _17_0_4_104e02e2_1433359181903_141112_13993) (type Translational) (s
(flow (chain 11) (name _) (id _17_0_4_104e02e2_1433367063415_182099_14521) (type Discrete) (source
(flow (chain 12) (name _) (id _17_0_4_104e02e2_1433357646893_327149_13795) (type Discrete) (source
```

Figure 3.4 Chains of functions resulting from CLIPS

.

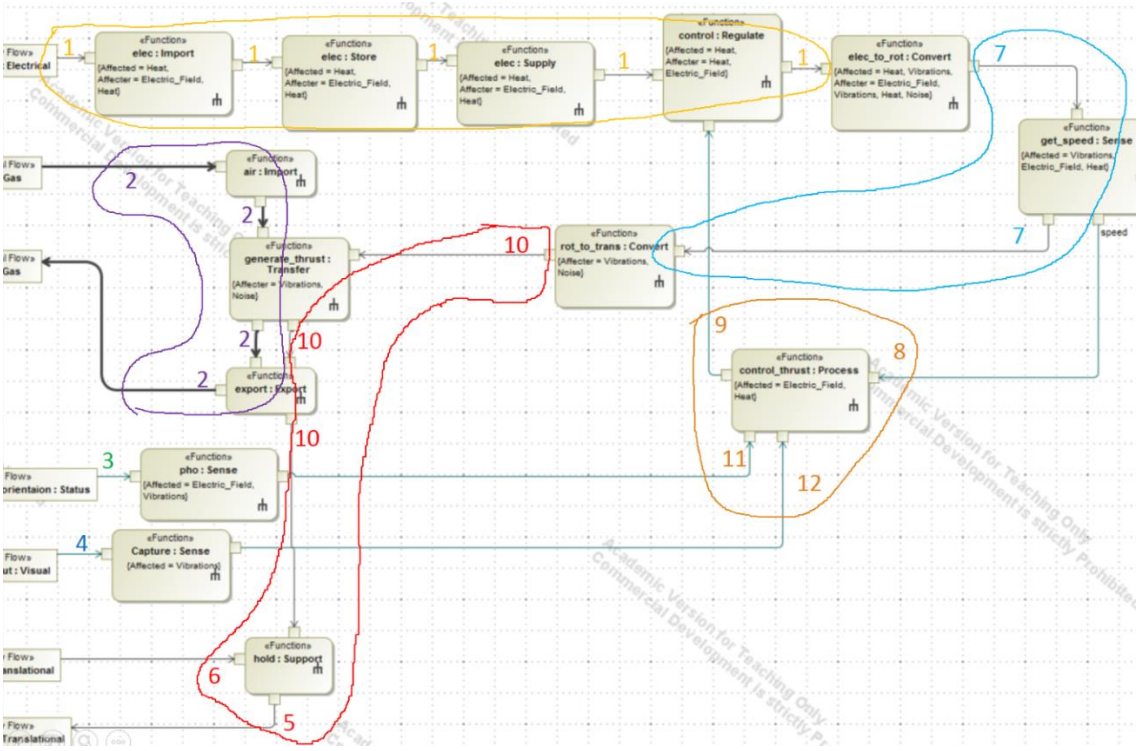

Figure 3.5 Chains of flows in SysML functional diagram

At this point in the design stage, by applying the heuristics presented in 2.2.1.3.3 and based on the work in (Stone et al., 2000), a preliminary modularization is obtained and is illustrated in Figure 3.6. Before starting to generate various modular concepts, the developed MagicDraw plug-in is used to automatically create and display the DSMs of the various adverse effects present in the system. These DSMs will be closely inspected to look for problematic functions/modules, identify the dependencies that should be expressed in more detail and they can even be added to the evaluation criteria of the product.

This information will be used to select components that are compatible, find solutions to eliminate adverse effects, and to create a better modularization of the system. This would help us achieve a modularization where not only aspects such as maintenance and ease of assembly are taken into account, but also efficiency and functionality of the overall system.

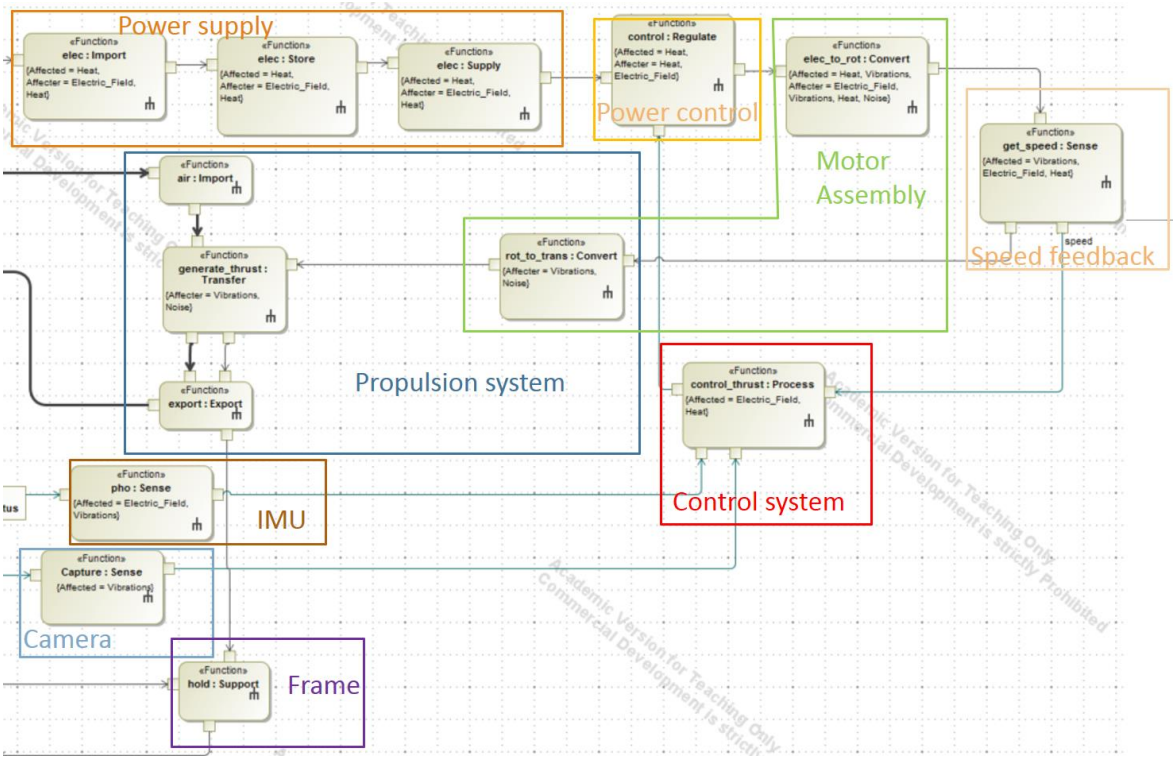

Figure 3.6 Preliminary modularization

Figure 3.7, Figure 3.8, and Figure 3.9 show the automatically generated DSMs for vibration, heat, and electric fields adverse effects, respectively. An arrow in the DSM means that a function affects the function to which the arrow is pointing, an “X” in the DSM means both functions affect each other. Problematic modules/functions were highlighted in each of the DSMs and some possible solutions to avoid them were reported in Table 3.2.

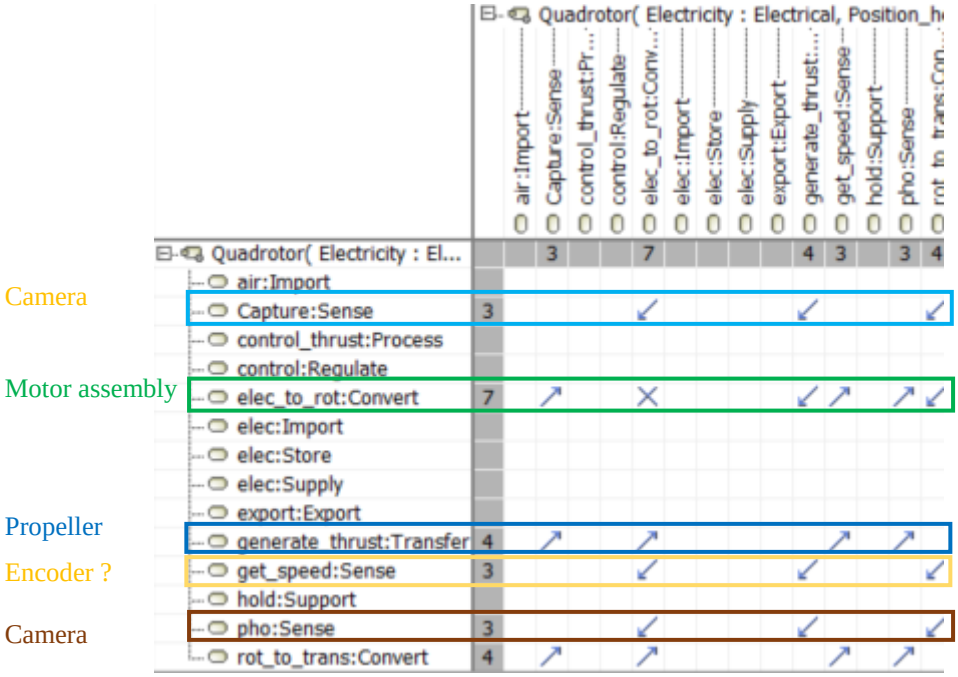

Figure 3.7 Vibration DSM

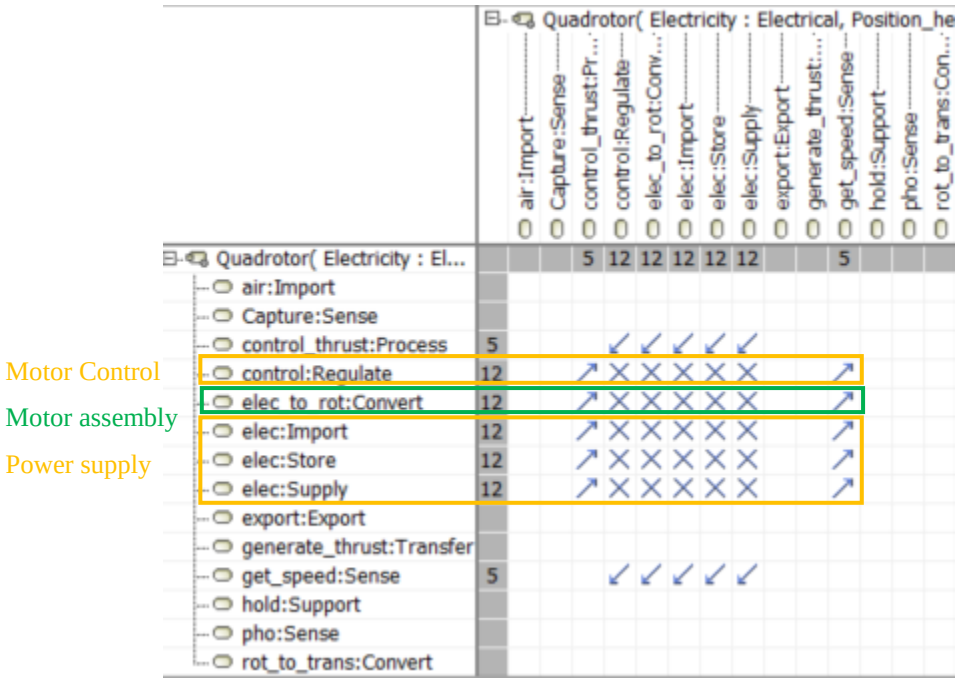

Figure 3.8 Heat DSM

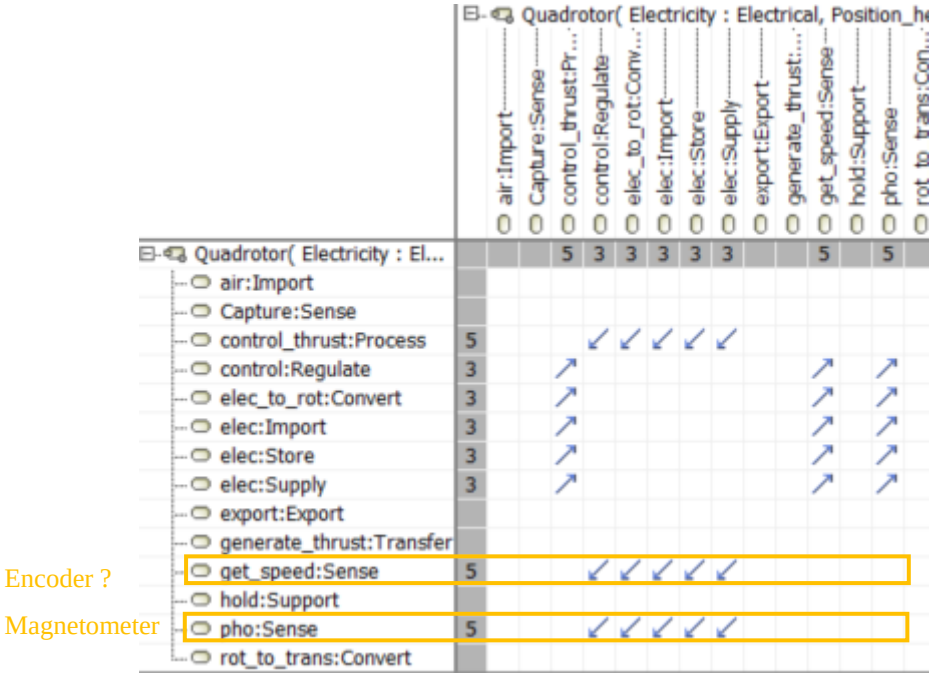

Figure 3.9 Electric field DSM

Table 3.2 Modules, problematic Adverse Effects, Solutions

Component	**Problem**	**Solution**
Camera (Capture: Sense)	**Vibrations** cause noise (Jello effect)	**Dampers** on camera supports Expensive optical stabilization Software stabilization (not always effective)
Accelerometer (pho: Sense)	**Vibrations**	**Dampers** on circuit board support
Propellers (generate_thrust: transfer…)	**Vibrations**	Take vibration into consideration during selection (operating speed…) (unlike helicopters, efficiency is often not taken into account in quadrotors)
Magnetometer + GPS (pho: Sense)	**Electric Fields**	**Physical allocation** (keep away from sources) **Component selection** (select components that generate less electromagnetic noise) **Shield sources** (if possible)

Table 3.2 (continued) Modules, problematic Adverse Effects, Solutions

Battery (Elec: store)	**Heat**	**Heat dissipation** should be taken into account specially if Lithium Polymer batteries are chosen. Should system be waterproof? …
Power transistor or **Electronic speed controller (ESC)** (Elec: control)	**Heat** **Electric Fields**	**Heat dissipation** **Physical allocation**, position on lower side of frame (shield if possible), away from center (control circuits)
Voltage regulator or **Battery eliminator circuit (BEC)** (Elec: supply)	**Heat** **Electric fields**	**Heat dissipation** **Physical allocation**, position on lower side of frame (shield if possible), away from center (control circuits)
Motor (convert: elec to rot)	**Heat** **Electric fields** **Vibrations**	**Component selection** AC motors (more efficient, less heat, AC electromagnetic fields can be shielded if problematic)
Encoder? **Tachometer? Current sense?**	**Electric fields** **Vibrations** **Heat**	**Component selection** Since AC motors are favorable, Effects of vibrations and electromagnetic fields on Encoders and Tachometers can be avoided by choosing Electronic speed controllers that have built in current sense and speed feedback.

After taking adverse effects into consideration and inspecting the DSMs the following design decisions were taken:

Motor selection: Brushless AC motors: They are usually more expensive than other alternatives but were chosen for the following reasons.

- High power to weight ratio
- Low power consumption
- AC vs DC electromagnetic noise can be easily shielded
- Availability of electronic speed controllers with feedback eliminates the need for Encoders or tachometers.
- Some of these motors are available with high quality bearings to withstand forces and vibrations. This allows direct assembly to propellers.

Wiring: Keep high power DC wires as short as possible during physical allocation. Long AC wires can be twisted and need to be shielded.

Battery: Weight, Heat, and Electromagnetic noise should be taken into account when allocating space for batteries and choosing between Lithium ion and Lithium Polymer.
Lithium ion batteries have a better power to weight ratio (or power density)
Lithium polymer batteries have a higher discharge rate (available power when needed), however this generates a lot of electromagnetic noise and heat (they can be hazardous)

Propellers: Take into account the vibrations at the operating motor speeds and sudden load changes. Do ducted propellers reduce vibrations?

Creative modules:

- Split the inertia measurement unit (IMU +GPS) into two modules:

- Accelerometer + Gyroscope Module, placed on circuit board with dampers to reduce vibrations (filtered frequencies to be determined in detailed design)
- GPS + Compass (magnetometer) Module, will be placed on case cover as far away from electromagnetic noise (Battery, ESC, BEC, Motors)

- Remove the “Supply” function from the “power supply module” to “Power Control” module as shown in Figure 3.10. These two functions fulfilled by the Electronic Speed Control (ESC) and the Battery Eliminator Circuit which is needed to increase the voltage provided by Lithium batteries. Each motor will have its own BEC which reduces heat and electromagnetic noise concentration. BEC and ESC should be close to battery to shorten DC wires and away from control circuit board and GPS + Compass.

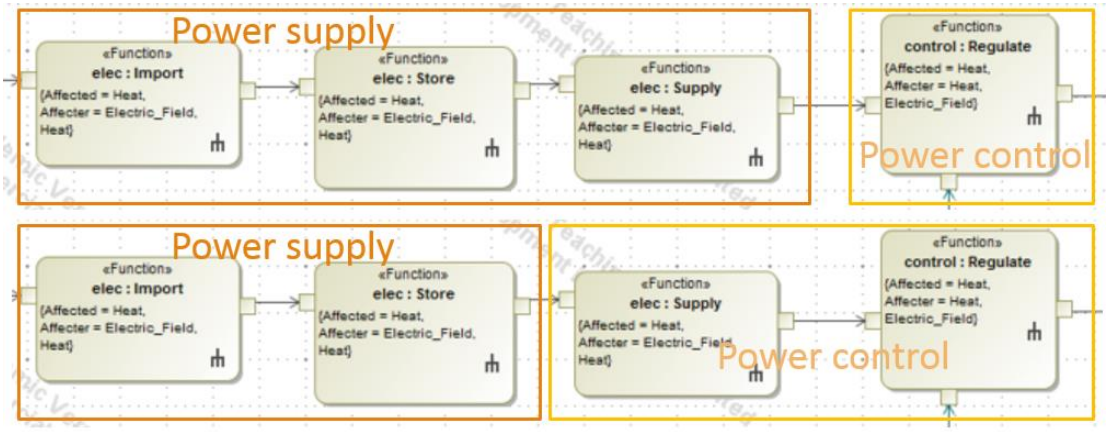

Figure 3.10 Power supply and Power Control modules

The resulting rough geometric layout sketch is shown in Figure 3.11. To reduce effects of Heat and Electromagnetic noise, the Battery, charger input, BEC, and ESC were positioned on the lower side of the frame towards the back of the device. The frame should be made of light material and act as a heat sink and insulator, and as a magnetic shield to protect the rest of the electronic circuit which is placed on the upper part of the device. The frame should also be designed to reduce vibrations with anti-resonant frequencies.

The center of gravity of a quadcopter should always be in the middle, the camera was positioned towards the front to balance the weight of the power supply and power control modules. This also allows to position the GPS + Compass module as far away from sources of electromagnetic noise on the casing towards the front of the device.

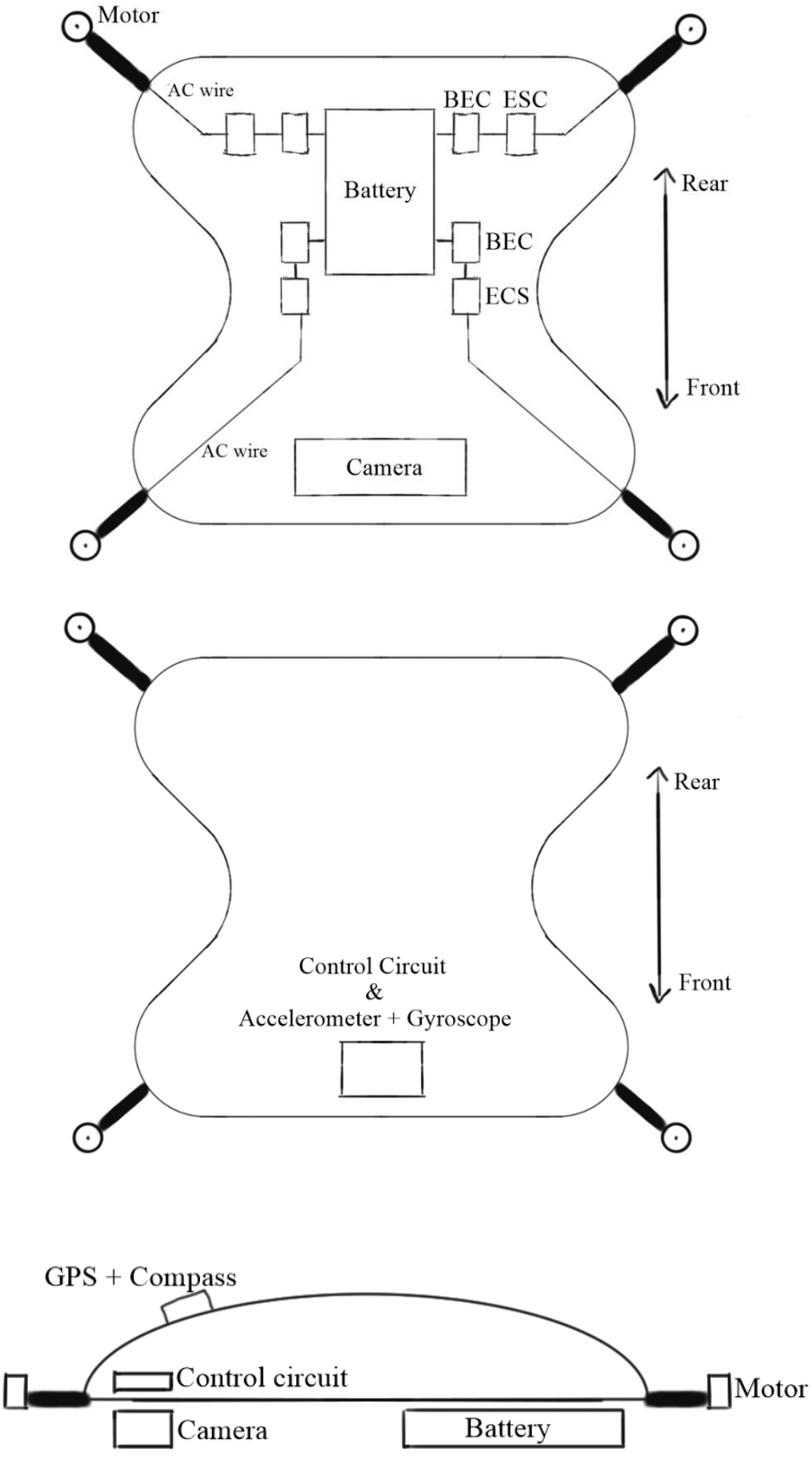

Figure 3.11 Quadrotor layout: bottom, top, and side views

This case study shows that the functional reasoning framework allows designers to develop a functional model of a product in a more dynamic way and makes the resulting diagrams more consistent. Designers can easily apply changes to the diagram and they have to select predefined functions and flows from the developed library and cannot do syntactic errors such as connecting a flow to a port of a different type, etc... The functional reasoning framework also supports designers during functional diagram modularisation, and as seen in the quadcopter example in figure xx, it provides them with preliminary modules which speeds up the modularization process. More importantly, the case study shows that if adverse effect dependencies abstractly captured and taken into consideration during conceptual design, a lot of design iterations could have been avoided by identifying incompatible components and finding solutions to product related problems before passing into detailed design which reduces the project lead time. If the design computational tool and dependency modeling scheme developed in this master thesis were used, the final generated concept of the quadcopter would've been similar if not better than the mature designs available nowadays in the market.

CHAPTER 4 CONCLUSION

In this master thesis, a functional reasoning framework was developed to allow engineering designers to abstractly model a mechatronic product during conceptual stage using a formal functional modeling language, the functional basis for design, which allows the use of computational power to support the design activity. The functional reasoning framework was achieved by customizing the SysML language and developing a plug-in in the modeling tool MagicDraw, the plug-in integrates the rule-based expert system CLIPS that allows encapsulating engineering knowledge in the form of rules to analyze and perform tasks on the functional diagrams. For example, heuristics were coded as rules in CLIPS to assist engineering designers and speed up the modularization of functional diagrams.

A new methodical acquisition method and representation scheme of dependencies was proposed. The concept of “meta-dependency” was introduced to model dependencies that are shared by a large amount of elements in a system. Compared to available methods of managing dependencies, where designers have to model each dependency as a single relationship between two elements, the proposed method allows engineering designers to efficiently and abstractly capture dependencies early in the deign process by modeling them as a relationship between an element and a meta-dependency. To prove the efficacy of the proposed modeling method, adverse effects, a type of dependency, were used and integrated into the functional reasoning framework, which was programmed to automatically generate a Design Structure Matrix for each type of dependency and present them to the engineering designers.

Two case studies were carried-out where mechatronic products were modeled using the functional reasoning framework while taking adverse-effect dependencies into account. First, a simple device that regulates the temperature of water was used to illustrate the principle. Then, the functional model of a quadrotor drone was designed. The resulting modularization and rough geometric layout of the quadrotor were presented, as well as a list of design problems that were avoided.

The functional reasoning design framework in conjunction with the meta-dependency modeling method proposed and developed in this research thesis proved to be a dynamic modeling tool that is flexible and allows changes to be made in the design with little effort from the engineering designer. Tagging functions with adverse effects proved to be an efficient and effective method

of acquiring information on this type of dependencies and managing them. It is an intuitive way of handling and capturing abstract dependencies early in the design process without shifting the designer's focus away from solution finding. By using assisted modularization and by inspecting the automatically generated Design Structure Matrices, collaboration between various disciplines is increased. Additionally, important questions that need to be asked by engineering designers to experts are pinpointed and narrowed down. This leads to the modularization of the product to be allowed early its development stages in a more efficient, effective way. Also, most importantly, the development time is reduced, and the activity streamlined through better integration of multidisciplinary design and by avoiding costly design loop backs.

BIBLIOGRAPHY

Achiche, S., Appio, F. P., McAloone, T. C., & Di Minin, A. (2012). Fuzzy decision support for tools selection in the core front end activities of new product development. *Research in Engineering Design*, 1-18.

Achiche, S., Baron, L., Balazinski, M., & Benaoudia, M. (2007). Online prediction of pulp brightness using fuzzy logic models. *Engineering Applications of Artificial Intelligence, 20*(1), 25-36.

Albers, A., Braun, A., Sadowski, E., Wynn, D. C., Wyatt, D. F., & Clarkson, P. J. (2011). System architecture modeling in a software tool based on the contact and channel approach (C&C-A). *Journal of Mechanical Design, 133*(10), 101006.

Alciatore, D. G., Histand, M. B., & Alciatore, D. G. (2007). *Introduction to mechatronics and measurement systems*: Tata McGraw-Hill Education.

Altshuller, G. S. (1984). *Creativity as an exact science*: Gordon and Breach.

Behbahani, S., & De Silva, C. W. (2007). Mechatronic design quotient as the basis of a new multicriteria mechatronic design methodology. *Mechatronics, IEEE/ASME Transactions on, 12*(2), 227-232.

Blessing, L. T., & Chakrabarti, A. (2009). *DRM: A Design Reseach Methodology*: Springer.

Braha, D. (2002). *Partitioning tasks to product development teams.* Paper presented at the ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

BusinessDictionary.com. What is dependency? definition and meaning. Retrieved from http://www.businessdictionary.com/definition/dependency.html

Chandrasekaran, B. (1994). Functional representation and causal processes. *Advances in computers, 38*(73-143).

Cox, L., Delugach, H. S., & Skipper, D. (2001). *Dependency analysis using conceptual graphs.* Paper presented at the Proceedings of the 9th international conference on conceptual structures, ICCS.

Crilly, N. (2010). The roles that artefacts play: technical, social and aesthetic functions. *Design Studies, 31*(4), 311-344.

Danilovic, M., & Browning, T. R. (2007). Managing complex product development projects with design structure matrices and domain mapping matrices. *International journal of project management, 25*(3), 300-314.

Eisenbart, B., Blessing, L., & Gericke, K. (2012). *Functional modelling perspectives across disciplines: a literature review.* Paper presented at the Proceedings of 12th international design conference, design.

Erden, M. S., Komoto, H., van Beek, T. J., D'Amelio, V., Echavarria, E., & Tomiyama, T. (2008). A review of function modeling: approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 22*(02), 147-169.

Far, B. H., & Elamy, A. H. (2005). Functional reasoning theories: problems and perspectives. *AIE EDAM, 19*(02), 75-88.

Friedenthal, S., Moore, A., & Steiner, R. (2014). *A practical guide to SysML: the systems modeling language*: Morgan Kaufmann.

Gausemeier, J., Frank, U., Donoth, J., & Kahl, S. (2009). Specification technique for the description of self-optimizing mechatronic systems. *Research in Engineering Design, 20*(4), 201-223.

Gausemeier, J., & Moehringer, S. (2003). *NEW GUIDELINE VDI 2206-A FLEXIBLE PROCEDURE MODEL FOR THE DESIGN OF MECHATRONIC SYSTEMS.* Paper presented at the DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm.

Gorbea, C., Spielmannleitner, T., Lindemann, U., & Fricke, E. (2008). *Analysis of hybrid vehicle architectures using multiple domain matrices.* Paper presented at the DSM 2008: Proceedings of the 10th International DSM Conference, Stockholm, Sweden, 11.-12.11. 2008.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition, 5*(2), 199-220.

Helms, B., & Shea, K. (2012). Computational Synthesis of Product Concepts Based on Generalized Graph Grammars. *ASME J. Mech. Des, 134*(2), 021008.

Hirtz, J., Stone, R. B., McAdams, D. A., Szykman, S., & Wood, K. L. (2002). A functional basis for engineering design: reconciling and evolving previous efforts. *Research in Engineering Design, 13*(2), 65-82.

Hölttä-Otto, K., Tang, V., & Otto, K. (2008). Analyzing module commonality for platform design using dendrograms. *Research in Engineering Design, 19*(2-3), 127-141.

http://www.plattformstrategie.de/. ÄnderungAuswirkungsanalyse. Retrieved from http://www.plattformstrategie.de/methoden.php

Keller, A., Blumenthal, U., & Kar, G. (2000). *Classification and computation of dependencies for distributed management.* Paper presented at the Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on.

Kitamura, Y., Ikeda, M., & Mizoguchi, R. (1997). *A causal time ontology for qualitative reasoning.* Paper presented at the IJCAI (1).

Komoto, H., & Tomiyama, T. (2012). A framework for computer-aided conceptual design and its application to system architecting of mechatronics products. *Computer-Aided Design, 44*(10), 931-946.

Kortler, S., & Lindemann, U. (2011). *A Meta Model of the Innovation Process to Support the Decision Making Process Using Structural Complexity Management.* Paper presented at the DS 68-4: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 4: Product and Systems Design, Lyngby/Copenhagen, Denmark, 15.-19.08. 2011.

Lindemann, U., Maurer, M., & Braun, T. (2008). *Structural complexity management: an approach for the field of product design*: Springer Science & Business Media.

Mohebbi, A., Achiche, S., & Baron, L. (2015). *Integrated Design of A Vision-Guided Quadrotor UAV: A Mechatronics Approach.* Paper presented at the 2015 CCToMM Symposium on Mechanisms, Machines, and Mechatronics.

Mohebbi, A., Achiche, S., Baron, L., & Birglen, L. (2014). *Trends in concurrent, multi-criteria and optimal design of mechatronic systems: A review.* Paper presented at the Innovative Design and Manufacturing (ICIDM), Proceedings of the 2014 International Conference on.

Pimmler, T. U., & Eppinger, S. D. (1994). Integration analysis of product decompositions.

Qamar, A., Paredis, C. J., Wikander, J., & During, C. (2012). Dependency modeling and model management in mechatronic design. *Journal of Computing and Information Science in Engineering, 12*(4), 041009.

Ren, Q., Balazinski, M., Jemielniak, K., Baron, L., & Achiche, S. (2013). Experimental and fuzzy modelling analysis on dynamic cutting force in micro milling. *Soft Computing*, 1-11.

Rzevski, G. (2003). On conceptual design of intelligent mechatronic systems. *Mechatronics, 13*(10), 1029-1044.

Salminen, V., & Verho, A. (1989). *Multi-disciplinary design problems in mechatronics and some suggestions to its methodical solution in conceptual design phase.* Paper presented at the International Conference on Engineering Design (ICED89).

Shah, A. A., Kerzhner, A. A., Schaefer, D., & Paredis, C. J. (2010). Multi-view modeling to support embedded systems engineering in SysML *Graph transformations and model-driven engineering* (pp. 580-601): Springer.

Starling, A. C., & Shea, K. (2005). Virtual synthesisers for mechanical gear systems.

Stone, R. B., Wood, K. L., & Crawford, R. H. (2000). A heuristic method for identifying modules for product architectures. *Design Studies, 21*(1), 5-31.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: principles and methods. *Data & knowledge engineering, 25*(1), 161-197.

Szykman, S., Racz, J. W., & Sriram, R. D. (1999). *The representation of function in computer-based design.* Paper presented at the Proceedings of the 1999 ASME design engineering technical conferences (11th international conference on design theory and methodology).

Tomiyama, T., D'Amelio, V., Urbanic, J., & ElMaraghy, W. (2007). Complexity of multi-disciplinary design. *CIRP Annals-Manufacturing Technology, 56*(1), 185-188.

Torry-Smith, J. M., Mortensen, N. H., & Achiche, S. (2014). A proposal for a classification of product-related dependencies in development of mechatronic products. *Research in Engineering Design, 25*(1), 53-74.

Torry-Smith, J. M., Mortensen, N. H., Ploug, O., & Achiche, S. (2015). *INDUSTRIAL APPLICATION OF A MECHATRONIC FRAMEWORK.* Paper presented at the DS 80-7 Proceedings of the 20th International Conference on Engineering Design (ICED 15) Vol 7: Product Modularisation, Product Architecture, systems Engineering, Product Service Systems, Milan, Italy, 27-30.07. 15.

Torry-Smith, J. M. r., Achiche, S., Mortensen, N. H., Qamar, A., Wikander, J., & During, C. (2011). *Mechatronic Design-Still a Considerable Challenge.* Paper presented at the ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

Umeda, Y., Tomiyama, T., & Yoshikawa, H. (1995). *FBS modeling: modeling scheme of function for conceptual design.* Paper presented at the Proc. of the 9th Int. Workshop on Qualitative Reasoning.

Van Beek, T. J., Erden, M. S., & Tomiyama, T. (2010). Modular design of mechatronic systems with function modeling. *Mechatronics, 20*(8), 850-863.

Wang, L., Shen, W., Xie, H., Neelamkavil, J., & Pardasani, A. (2002). Collaborative conceptual design—state of the art and future trends. *Computer-Aided Design, 34*(13), 981-996.

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

Wölkl, S., & Shea, K. (2009). *A computational product model for conceptual design using SysML.* Paper presented at the ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

Zakarian, A. (2008). A new nonbinary matrix clustering algorithm for development of system architectures. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 38*(1), 135-141.