



DIGITAL REPRESENTATION OF PRODUCT FUNCTIONS IN MULTICOPTER DESIGN

Ramsaier, Manuel (1); Holder, Kevin (1); Zech, Andreas (1); Stetter, Ralf (1); Rudolph, Stephan (2); Till, Markus (1)

1: University of Applied Sciences Ravensburg-Weingarten, Germany; 2: University of Stuttgart, Germany

Abstract

In recent years the research concerning the representation of product functions has intensified again. Current studies exhibit a growing interest of design engineers to apply such representations in their daily practice. Simultaneously, a growing interest concerning graph-based methods for the digital representation of product models and the product logic in general can be observed. This paper seeks to combine these two research directions and to present a promising attempt to allow a sensible representation of product functions within a digital engineering framework based on graph based design languages. The main motivation for this research is to look for powerful methods and tools to overcome the limitations which result from the multitude of software tools along the product life cycle and heterogeneous data formats which are present in industrial companies and hinder integrated product development. This research seeks to expand the applicability of graph-based design languages into the domain of product functions. A rather simple product - a multicopter - is used as a basis for explanation and discussion.

Keywords: Digital / Digitised engineering value chains, Functional modelling, Multi- / Cross- / Trans-disciplinary processes

Contact:

Manuel Ramsaier
University of Applied Sciences Ravensburg-Weingarten
IAF
Germany
ramsaier@hs-weingarten.de

Please cite this paper as:

Surnames, Initials: *Title of paper*. In: Proceedings of the 21st International Conference on Engineering Design (ICED17), Vol. 1: Resource-Sensitive Design | Design Research Applications and Case Studies, Vancouver, Canada, 21.-25.08.2017.

1 INTRODUCTION

In recent years the research concerning the representation of product functions has intensified again. Current studies exhibit a growing interest of design engineers to apply such representations in their daily practice. Simultaneously, a growing interest concerning graph-based methods for the digital representation of product models and the product logic in general can be observed. This paper seeks to combine these two research directions and to present a promising attempt to allow a sensible representation of product functions within a digital engineering framework based on graph based design languages. The main motivation for this research is to look for powerful methods and tools to overcome the limitations which result from the multitude of software tools along the product life cycle and heterogeneous data formats which are present in industrial companies and hinder integrated product development. The product development departments of industrial companies use dedicated systems for computer aided design (CAD), finite element analysis (FEA), multi-body simulation (MBS), thermal simulation, control processes and many other activities and processes. These systems are usually relying on proprietary data formats and the transfer of data frequently leads to losses of important information and data quality. Numerous efforts seek to integrate the data by means of product data management (PDM) or product life-cycle management (PLM) systems. However, the enormous investments of large companies into the further development of such systems show that the situation in industrial practice is everything else than satisfactory. Graph-based design languages offer one promising approach to address and resolve these issues. However, the applicability for the early phases of design when the information is abstract is currently still limited. This research seeks to expand the applicability of graph-based design languages into the domain of product functions. A rather simple product - a multicopter - is used as a basis for explanation and discussion. This sample product is introduced in section 2. Section 3 reviews the state of the art in graph based design languages and their application in product development processes. The initial part of section 4 deals with the state of the art of the representation of product functions; in the remaining part of section 4 the focus is on certain aspect of this representation and their merit in a digital representation. Section 5 presents a framework and first results concerning the digital representation of product functions in a graph-based design language.

2 SAMPLE PRODUCT: DEVELOPMENT OF A MULTICOPTER

The example product “multicopter” is a mechanically relatively simple system which receives its stable flying ability mainly through complex control engineering. The basic geometry can be generally easily described by the relative position of its functional components. The simplest structure is a circular arrangement of 4 or more rotors (see Figure 1 left side).



Figure 1. Left: Basic structure of a multicopter; right: realized multicopter

The motors are mounted on a frame that includes a landing gear and appliances for mounting electronics and the power supply. The specific challenge in this light-weight development is to find geometries which lead to a very rigid frame structure that suppresses oscillations and lead to an easier flight control. Furthermore materials needed to be selected which are characterized by high strength and low weight. This multicopter was chosen as one of the example products in the research project due to the importance of the design matching of all parts and the required pay-load and operation time (compare Ramsaier et

al. (2016)). One light-weight multicopter design was realized in the laboratories of the university (see Figure 1 right side).

3 DIGITAL REPRESENTATION OF THE PRODUCT LIFE CYCLE

The aim of the underlying research is the holistic integrated digital description of the product life cycle. The product life cycle includes many technical disciplines (e.g. requirement management, design, engineering, simulation, production, costing, maintenance, ...) and therefore requires cooperation of many people and the interaction with many different software tools. This integration process is supported on the human resources side by complex matrix organization forms which take into account both the technical and project related organizational alignment. On the technical side this integration process is currently focusing on complex PLM software systems which aim to achieve the integration. Immense efforts by research and industry result in numerous publications addressing the product modelling issues. Also graph-based approaches were analysed since the early nineties of the last century (compare Finger and Rinderle (1990)). Several research publications focus on product life-cycles such as Le Duigou and Bernard (2011), on integrated product and process models (for an excellent overview see Eckert et al. (2015)). The distinctive approach in this project is the application of an engineering framework of graph-based design languages, which allow the reuse of pre-existing engineering design models and know-how (Till et al.2016). These design graphs generated from design languages are based on UML (Unified Modelling Language). UML is the primary modelling language in computer science - computer science is known for its very fast development cycles which require very powerful abstract description forms (i. e. modern graphical modelling languages) for the software development process and highly automated development tools to achieve these extremely short innovation cycles. It is one main hypothesis in this project that the product development processes for conventional project can greatly profit from the basic ideas and theories behind these automated development tools and from graph-based design languages.

The concept of graph-based design languages (see e.g. Rudolph 2002) has evolved over the last ten years into a generic framework for the definition of computerized design processes. Generally, a graph is a representation of a set of objects (i. e. the graph nodes) where some pairs of objects are connected (i. e. the graph edges). The interconnected objects are represented by mathematical abstractions which can be called “nodes” and the links are as well represented by abstractions which can be called “edges”. Figure 2 shows a design graph for the multicopter.

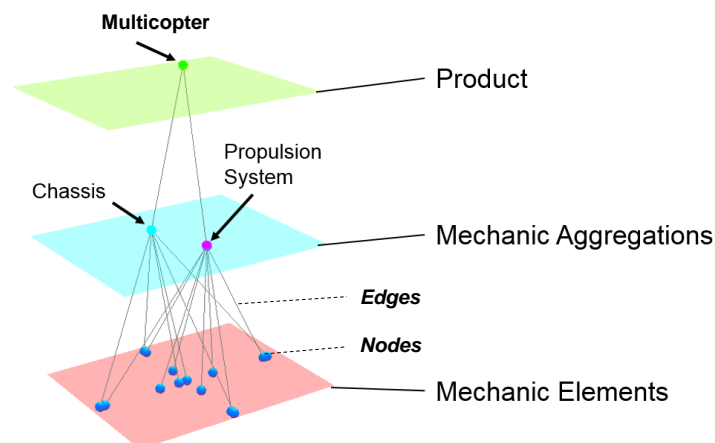


Figure 2. Structure of a design graph

Rudolph (2003) and co-workers have developed a consistent design system to represent geometry and other forms of information using graphs and have built a graph-based design language which can be described by means of the underlying design language process (Figure 3).

The product can be abstracted to its parts and their abstract geometry. The parts can be organized into the class diagram which represents the vocabulary of the design language. This vocabulary can be used together with rules in a production system in order to generate the geometrical model of the product.

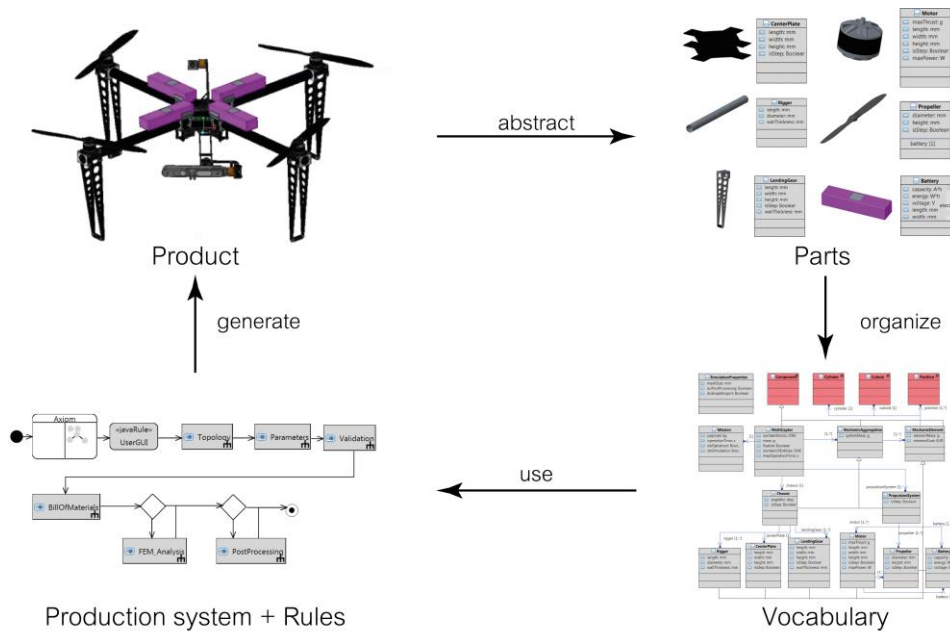


Figure 3. Design Language Process

In principle, graph-based design languages consist of a meta-model that stores all relevant parametrical and topological design information and acts as a centralized model repository (Arnold and Rudolph 2012). It uses a set of design rules and a design compiler that executes the design rules and automatically derives the analysis models of the respective design domains (CAD, CFD, FEM, . . .). The formalisms of design languages are discussed in depth in Alber and Rudolph (2003). Applications are for instance shown in Kormeier and Rudolph (2006) and Groß and Rudolph (2011).

The design language aspect of being ‘graph-based’ allows modifying topology (i.e. the question whether a certain object, property or behaviour exists) which can be changed as easy as the design parameters contained in a node of the graph (Arnold and Rudolph 2012). Design languages offer therefore a combined manipulation scheme for design topology and parameters. In the graph based design language (Figure 4), the engineering objects are represented by the vocabulary and the model transformations are represented by the rules, the grammar of the design language Groß and Rudolph (2012).

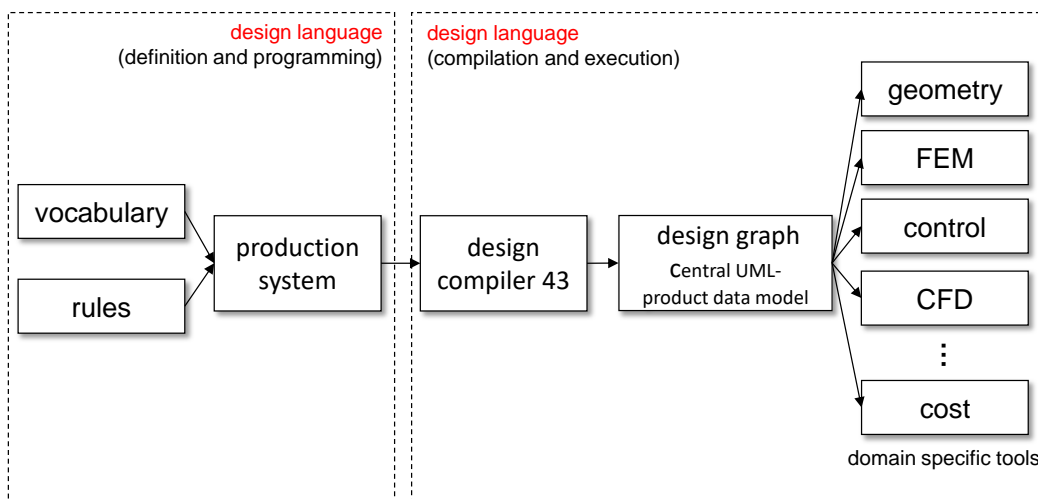


Figure 4. Process chain with graph-based design language (compare Groß and Rudolph 2012)

In a so-called “production system”, the rules are executed in order to instantiate the vocabulary classes. This compilation process builds up the central data model. In this project, the definition of the vocabulary is expressed in the Unified Modelling Language (UML), an international standardized modelling language (Groß and Rudolph 2011). The UML has been developed by software engineers and provides many features to model data to describe object-oriented software (Groß and Rudolph

2012). The latest version of UML 2.5 was released in June 2015. One decisive advantage is the possibility to compile UML models; this characteristic lead to the decision to employ UML as a basis for the graph-based design language. From this high-level central data model, different interfaces generate domain specific, generic models such as geometry models or simulation models.

A machine capable of compiling the graph-based design languages is the "Design Compiler 43" (IILS GmbH 2016). This compiler was developed by the IILS mbH (<http://www.iils.de>) in cooperation with the University of Stuttgart, which is one of the partners in this project. The design compiler is an eclipse-based software tool for the compilation of design languages formulated in UML (Arnold and Rudolph 2012).

4 REPRESENTATION OF PRODUCT FUNCTIONS

4.1 State of the art

One may speculate that the fundamental concepts of function models originate from Rodenacker (1976). He refers to function models as "logical effect relation" (in German "logischer Wirkzusammenhang"). Numerous research activities have dealt with the development of function models - an excellent review was collected by Eisenbart et al. (2016). They were able to identify two main concepts of a function:

- A function describes the ability of a system to achieve a goal or fulfil a given task by showing certain behaviour.
- A function describes the transformation, conversion or change of states of distinct operands (i.e. typically specifications of material, energy or signals).

Consequently, a function model can be any kind of representation of one or more functions and respective logical relationships. A standard convention in mechanical engineering literature is the function structure after Pahl et al. (2007). Figure 5 shows a function structure representing on main function of the multicopter - the generation of propulsion.

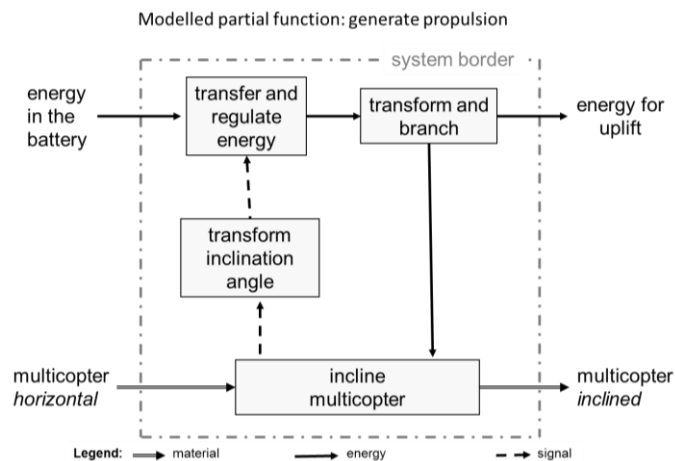


Figure 5. Function structure of the multicopter acc.to the notation of Pahl et al. (2007)

Here only the function propulsion in forward direction (not in upward direction) is shown as it exhibits an appropriate complexity in order to explain different perspectives of function modelling in later parts of this paper. In the lower left corner on Figure 5 a possible initial state of the multicopter is shown - a horizontal orientation. In order to generate propulsion such multicopters need to be inclined so that a movement in direction of the lower rotors is generated. A control system uses sensors at the multicopter in order to get the current inclination angle. According to the current wish of propulsion, a desired inclination angle is generated and the current from the batteries to the motors is regulated. The motors in combination with the propellers transform the electrical energy into mechanical energy for the uplift, but depending on the current inclination also for the propulsion. The regulated difference between front and back motors will lead to a desired inclination of the multicopter.

It is important to note that a function structure is usually not leading to one definite result. Other engineers might come up with a quite different functional description of the same partial function; there might be differences in size and resolution but also in the ways to express certain functionality. The

function structure in Figure 5 was mainly created in order to be able to discuss additional possible perspectives.

It is also important to note that the current study of Eisenbart et al. (2016) showed a large interest of engineers in industrial practice towards function models. They had developed a framework for integrated function modelling. In this study, 15 out of 17 engineers from industrial practice stated that they are willing to apply parts of the framework (at least if certain adaptations are implemented. One might come up with the hypothesis that maybe in earlier years the demand for an abstract language was not strong enough in industry but has increased in the last years because of the necessity of cross-domain problem solving and synthesis. The framework proposed by Eisenbart et al. (2016) includes six views, allowing a comprehensive look on the functional logic of a product and enables representing several aspects of functional modelling. However, for the integration into a graph-based design language a conscious selection of these aspects was necessary in order to achieve product models which can still be handled with today's technology. The selection of these aspects is explained in the next subsections.

4.2 Representation of flows

The central perspective are transformation processes which are in most cases modelled in relation to a flow in time, especially in the models found in industrial practice (Eisenbart 2014). The function structure by Pahl et al. (2007) as shown in Figure 5 is strongly focused on this perspective.

4.3 Representation of states

One central function modelling perspective can be referred to as "state view" and can either represent a certain situation that the system resides in (for instance in real-time state-charts compare e.g. Heinzemann et al. (2013) or the states of operands (operands are sometimes also described as flows) before and after a transformation process (Eisenbart et al. 2016). In the study of Eisenbart et al. (2016) 12 out of 14 mechanical designers from industry considered the state perspective as useful. In the function structure after Ehrlenspiel and Meerkamm (2013) a state perspective is included. This inclusion was amongst other identified as helpful for designing products for recycling (compare Phleps (1999)) and for designing diagnosis functionalities (compare Stetter and Phleps 2011). A very similar notation which is easier to understand for persons from other domains by depicting the transformation processes by means of block arrows was proposed by Stetter and Simundsson (2016). Figure 6 shows a function structure including states according to this notation (the modelled partial function is identical to Figure 5) - here also actors and hierarchies are shown (compare Section 4.4).

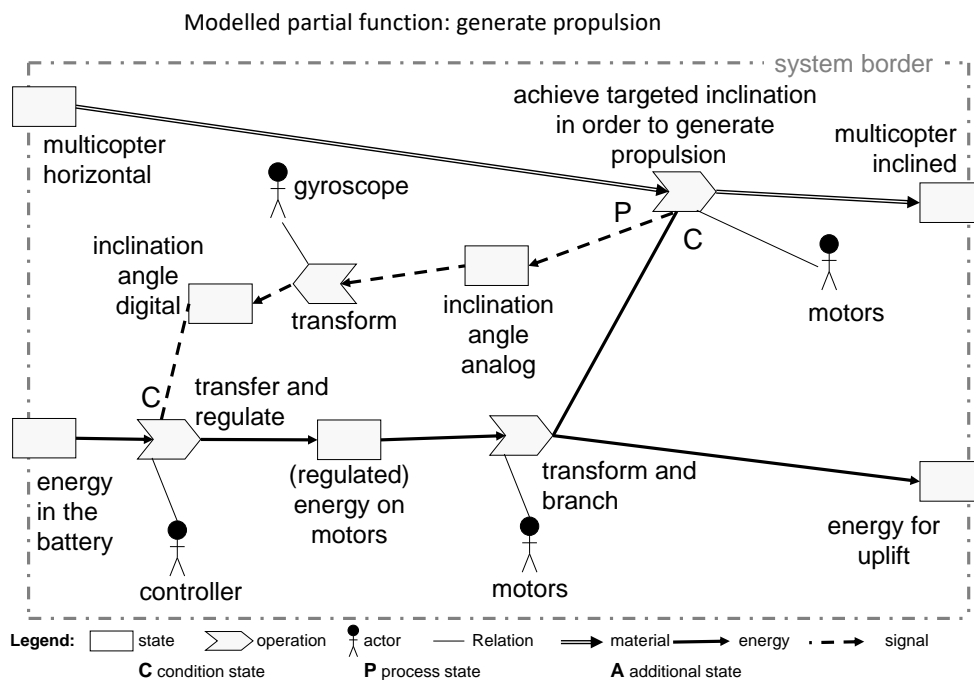


Figure 6. Function structure of the multicopter including states, actors and hierarchies

In this special case it is rather obvious that a function structure including states is easier to understand and may foster a deeper understanding. However, there can be no denying of the fact that this advantage might be rather small for products which mainly rely on transformation processes. Also the inclusion of states will automatically lead to larger depictions of the same product functions thus endangering the clarity of a function model. Modern technology offers many possibilities to enhance clarity (for instance a view without states even if they were modelled) therefore the authors conclude that the advantages of an inclusion of states will outweigh the possible disadvantages.

4.4 Representation of actors and hierarchies

A more controversial aspect may be the question if actors (e.g. the elements of a system which carry out a transformation process) should be part of a function model. One of the central advantages of a function model can be that it is an abstraction of a concrete technical system and is therefore less solution specific. It is often hypothesized that less solution specific models can lead to a larger solution space and may foster the development of more innovative solutions. However, it is important to note that function models are very often used to analyse and explain existing systems. The experience of the authors shows that a majority of function models is created with at least a virtual model of a rather concrete technical system. In such cases the transformation from the system to the abstract functional model is much easier if function carriers - i. e. actors - are mentioned, especially for persons not regularly concerned with functional modelling. Also the function modelling framework of Eisenbart et al. (2016) includes an actor view and a majority of the engineers in industry in their study considers this view as useful. Still, for innovative solutions design engineers should always be aware that the represented actors only show one solution possibility and can always be replaced for a better product. Such representation can foster a beneficial (feedback-)cycle between abstract and concrete. Figure 6 shows a function structure including actors. These actors allow a sensible connection of product functions with the sub-systems of this product thus leading to a more complete depiction of the product logic.

The function structure in Figure 6 additionally includes a visible hierarchy of flows which is also an important part of the function structure of Ehrlenspiel and Meerkamm (2013). They describe three different types of linking possibilities of secondary flows to primary flows (types: condition state, process state, additional state). In Figure 6 the primary flow is the material transition of the multicopter. One secondary flow of energy is connected as a condition (C) to the transformation "achieve targeted inclination in order to generate propulsion". Here the (controlled) energy in the secondary flow is a prerequisite - a condition - for the transformation in the primary flow. Another secondary flow is a signal flow starting with a process state (P) - the current inclination angle of the multicopter. Its digital transformation is again a condition (C) for the regulation processes in the motor controller. With a small adaption it is also possible to include this kind of hierarchy into the function modelling framework of Eisenbart et al. (2016). The consideration described in this section were the logical basis for an integration of functions into a graph based design language which is described in the next section.

5 DIGITAL REPRESENTATION OF PRODUCT FUNCTIONS

This section explains the integration of function models into a graph-based design language. The logical basis for the integration is the class diagram which is explained in the next sub-section. The actual integration is realised by means of the production system and is explained on the basis of the product example multicopter in section 5.2.

5.1 Class Diagram for function modelling

In order to maintain the general nature of the graphical design language, a stand-alone class diagram for the function modelling framework has been established which is shown in Figure 7.

In addition to the classes several types of connections (links) are visible. Links in blue colour indicate associations, which can be understood as a "has a"-relationship whereas links in black colour are inheritance, which can be understood as a "is a"- relationship. Basically, the modelling process starts with defining the requirements of the product. Some of these (functional) requirements are mapped to product functions. This mapping is realised by assigning unidirectional associations. A requirement has a function with whom it is fulfilled. The function can have several self-associations.

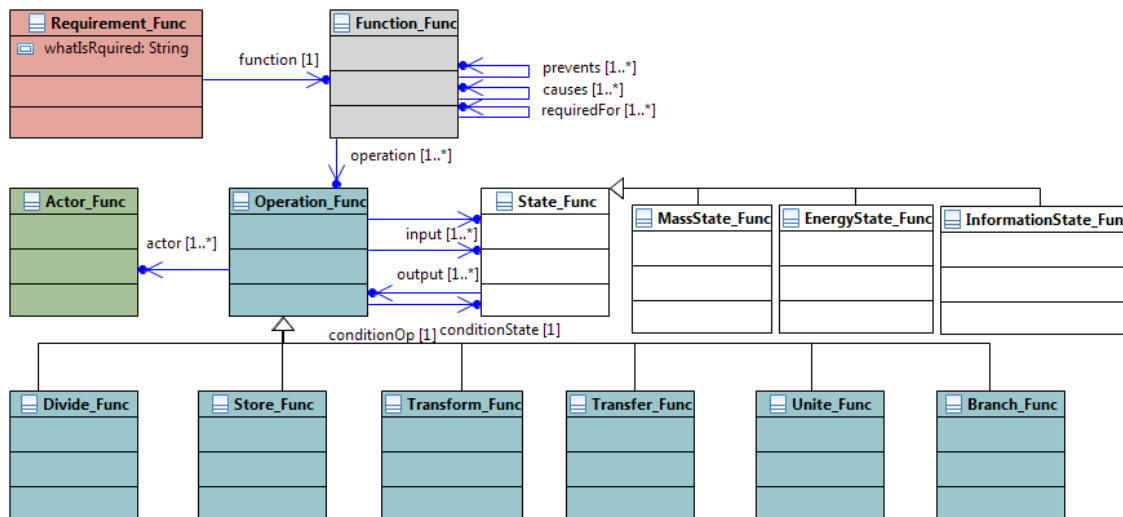


Figure 7. Class diagram of the function modelling framework

One function may be introduced to prevent or cause another function or maybe one function is required for one another. These associations can be used to create relation oriented function models such as the function models used in TRIZ (compare e. g. Herb (2000)). Every function is split up into one to several operations. The classes Change, Divide, Store, Transform, Transfer, Unite and Branch inherit from the class operations, which means they are an optional layer to specify the type of operation more precisely. Operations can have one or several Actors, through which they are realised as well as two or more States. States, which are Mass, Energy and Information can be changed by operations. Therefore one state can be linked as an input state whereas the other one can be linked as an output state.

The classes from Figure 7 are used in the design language multicopter to implement product functions. Therefore the class Actor is integrated into the multicopter class diagram. Every Part of the multicopter can be seen as an actor.

5.2 Function modelling within the design language multicopter: First Results

For the multicopter design language the following three requirements were stated. First, the multicopter shall be able to fly. Second, the multicopter shall be able to fly in a specified direction and third, the multicopter shall be able to yaw while hovering. For every requirement a function is introduced with which the requirement is fulfilled. For the second requirement, the function "generate propulsion" is introduced and described in this section.

In graph-based design languages the above mentioned rules (which accompany the vocabulary) are formulated with model-to-model transformations and with two sides for each rule. On the left hand side (LHS) of the rule the instances are formulated which the engine shall search within the design graph. The right hand side (RHS) of the rule shows what transformations are to be done by the engine. This distinguishes the right hand side (result of the rule) from the left hand side (input of the rule).

For function modelling the rules were split up into two different subprograms. In a first subprogram, the functions are modelled as described above. Only the actors are left blank in this stage. After designing the multicopter, the second subprogram maps the parts of the multicopter to the functions in their role as actors (Figure 8).

The top of Figure 8 shows the production system. The second rule after "requirements" is the subProgram "FunctionDiagram". Its rules have on the left hand side the functional requirements and on the right hand side the functions consisting of operations and states. Then three rules are carried out which generate the topology of the multicopter and abstract mechatronic relationships. The subProgram at the end named "FunctionMapping" describes the mapping of functions to actors and also disposes of a left hand side (operations and actors are present but independent) and a right hand side (operations and actors are sensibly linked). The "contents" of the rules are shown in the middle of Figure 8.

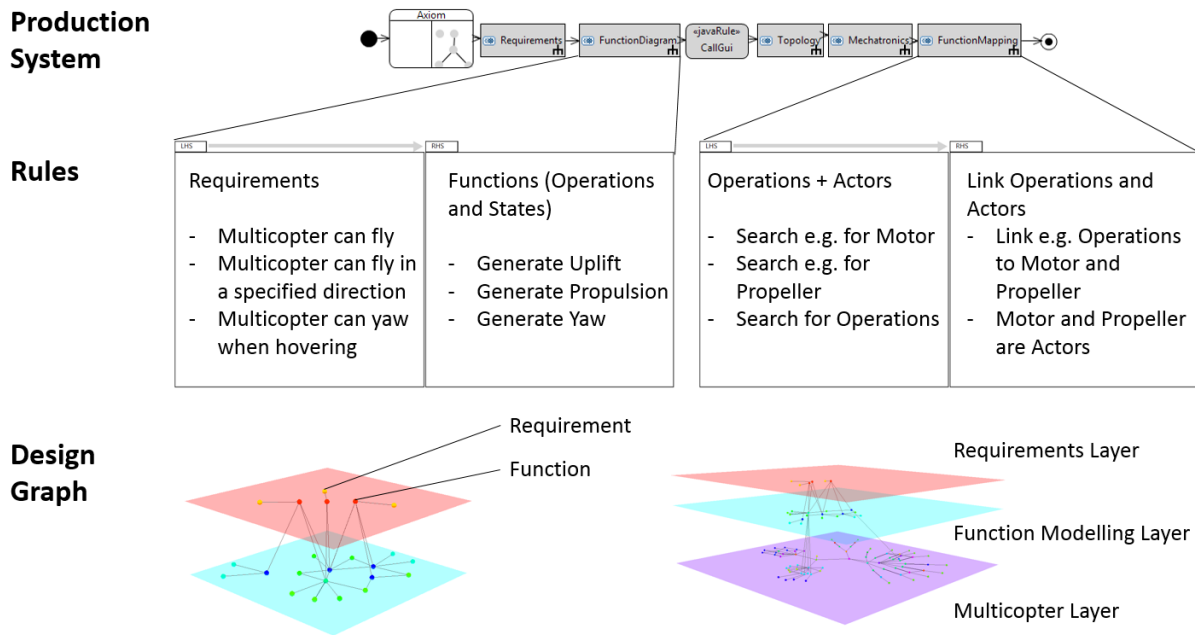


Figure 8. Model-to-model transformations for functional modelling

The lower part of Figure 8 represents concerned layers of the design graph. The first subProgram "FunctionDiagram" concerns the two layers "Requirements Layer" and "Function Modelling Layer". The second subProgram "FunctionMapping" additionally concerns the "Multicopter Layer" which contains the aggregations and components of the multicopter. This digital representation of product functions which was developed and optimised with several product examples and was also established for the sample product - the multicopter. It is important to note that it was not a goal to achieve a more understandable representation of product functions but a fully logical digital model which can be an integral part of a holistic digital product model.

During the modelling several insights could be gathered:

- It is possible to integrate a function model sensibly in a digital model which is expressed in a graph-based design language. It is possible to represented logical links which are not shown in other holistic product models.
- Not all requirements can be linked to a function (only functional requirements). Some requirements directly concern product modules or characteristics of product modules such as corrosion protection.
- It is not sensible to try to find functions for all modules and components of a product because even for a rather simple product the function model will become very complex and will also cover numerous trivial relationships.

6 SUMMARY AND OUTLOOK

In recent years the research interest concerning sensible representations of product functions has intensified again. It is probable that ever increasing product complexity, a general tendency towards domain-spanning products and sub-systems and the trend towards cyber-physical systems are driving the wish for an abstract holistic and digital product model. Several current publications show that the research interest concerning graph-based design languages has intensified as well. The research described in this paper intends to combine functional product modelling with recent advances in graph-based design languages. The major advantage can be a mapping of the (abstract) functional structure with an abstract representation of the product geometry. This mapping can offer an extended product logic which may foster more sensible simulations processes and sensitivity studies. In this paper the mapping was explained on the basis of a challenging sample product - a multicopter. Currently more complex products are being modelled and the advantages and limitations are investigated.

REFERENCES

- Alber, R.; Rudolph, S. (2003), "A Generic Approach for Engineering Design Grammars". *Proceedings of the AAAI Spring Symposium Computational Synthesis*, 2003.
- Arnold, P.; Rudolph, S. (2012), "Bridging the Gap between Product Design and Product Manufacturing by Means of Graph-Based Design Languages". In: *Proceedings of TMCE 2012*.
- Eckert, C.; Albers, A.; Bursac, N.; Chen, H. X.; Clarkson, P. J.; Gericke, K.; Gladysz, B.; Maier, J. F.; Rachenkova, G.; Shapiro D.; Wynn D. (2015), "Integrated Product and Process Models: Towards an Integrated Framework and Review". In: *Proceedings of the 20th International Conference on Engineering Design (ICED15)*, Milan, Italy, 27.-30.07.2015.
- Ehrlenspiel, K.; Meerkamm, H. (2013), *Integrierte Produktentwicklung. Denkabläufe, Methodeneinsatz, Zusammenarbeit*. Hanser: 2013.
- Eisenbart, B., Gericke, K., Blessing, T. M., McAloone, T.C. (2016), "A DSM-based framework for integrated function modelling: concept, application and evaluation". In: *Research in Engineering Design* (2016).
- Eisenbart, B. (2014), *Supporting interdisciplinary system development through integrated function modelling*. Dissertation, University of Luxembourg, Luxembourg, 2014.
- Finger, S.; Rinderle, J. (1990), *A transformational approach to mechanical design using a bond graph grammar*. Technical Report. Carnegie Mellon University. Engineering Design Center. 1990.
- Groß, J.; Rudolph, S. (2012), "Generating simulation models from UML – a FireSat example". In: *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation – DEVS Integrative M&S Symposium*. San Diego: Society for Computer Simulation International, 2012.
- Groß, J.; Rudolph, S. (2011), *Hierarchie von Entwurfsentscheidungen beim modellbasierten Entwurf komplexer Systeme*. Tag des System Engineerings. 2011.
- Heinzemann, C.; Rieke, J.; Bröggelwirth, J.; Pines, A.; Volk, A. (2013), *Translating Mechatronic UML Models to MATLAB/Simulink and Stateflow*. University of Paderborn: 2013.
- Herb, R. (Hrsg.) (2000), *TRIZ – Der systematische Weg zur Innovation*. Landsberg: Verlag Moderne Industrie, 2000.
- Kormeier, T.; Rudolph, S. (2006), "Topological Design of Shell Structures by Design Languages". *Proceedings of Design Engineering Technical Conferences DETC 2006*.
- Le Duigou, J.; Bernard, A. (2011), "Product Lifecycle Management Model for Design Information Management in Mechanical Field". In: *Proceedings of the 21st CIRP Design Conference*, Mar 2011, Daejeon, South Korea. pp. 207-213, 2011.
- Pahl, G.; Beitz, W.; Feldhusen, J.; Grote K.H. (2007), *Engineering design: a systematic approach*. Springer, 2007.
- Phleps, U. (1999), *Recyclinggerechte Produktdefinition – Methodische Unterstützung für Upgrading und Verwertung*, Dissertation, Technische Universität München, 1999.
- Ramsaier, M.; Spindler, C.; Stetter, R.; Rudolph, S.; Till, M. (2016), "Digital representation in multicopter design along the product life-cycle". In: Teti, R. (Ed.): *Proceedings of the 10th Conference on Intelligent Computation Manufacturing Engineering (CIRP ICME '16)*. 20- 22 July 2016, Ischia (Naples), Italy.
- Rodenacker, W. G. (1976), *Methodisches Konstruieren*. Berlin: Springer 1976.
- Rudolph, S. (2002), *Übertragung von Ähnlichkeitsbegriffen*. Habilitationsschrift, Universität Stuttgart, 2002.
- Rudolph, S. (2003), *Aufbau und Einsatz von Entwurfssprachen für den Ingenieurentwurf*. Forum Knowledge Based Engineering, CAT-PRO: 2003.
- Rudolph, S. (2012), "On the problem of multi-disciplinary system design - and a solution approach using graph-based design languages". *1st ACCM Workshop on Mechatronic Design*, Linz, November 30, 2012.
- Rudolph, S.; Beichter, J.; Eheim, A.; Hess, S.; Motzer, M.; Weil, R. (2013), "On Multi-Disciplinary Architectural Synthesis and Analysis of Complex Systems with Graph-based Design Languages", in 62. *Deutscher Luft- und Raumfahrtkongress (DLRG 2013)*, Stuttgart, 2013.
- Stetter, R., Phleps, U. (2011), "Design for Diagnosis". In: *Proceedings of the 18th International Conference on Engineering Design (ICED 11)* Vol. 5, 2011, pp. 91-102.
- Till, M.; Stetter, R.; Rudolph, S. (2016), "Multi-disziplinäre digitale Repräsentation des Produktlebenszyklus auf der Basis graphenbasierter Entwurfssprachen". In: *Forschungsreport Baden-Württemberg 2016*. Bingen: Public, 2016. S. 3 – 6. ISSN 2196-8659.

ACKNOWLEDGMENTS

The project „digital product life-cycle (ZaFH) (information under: <https://dip.reutlingen-university.de/>) is supported by a grant from the European Regional Development Fund and the Ministry of Science, Research and the Arts of Baden-Württemberg, Germany (information under: www.rwb-efre.baden-wuerttemberg.de).