HOLISTIC DIGITAL FUNCTION MODELLING WITH GRAPH-BASED DESIGN LANGUAGES

Elwert, Michael (1); Ramsaier, Manuel (1); Eisenbart, Boris (2); Stetter, Ralf (1)

1: University of Applied Sciences Ravensburg-Weingarten; 2: Swinburne University of Technology

ABSTRACT

Graph-based design languages offer a promising approach to address several major issues in engineering, e. g. the laborious manual transfer between CAD and CAE. Such languages generate a digital meta- or system model storing all relevant information about a design and feed this into any relevant CAE tool as needed to simulate and test the impact of any design variation on the resulting product performance. As this can be automated in digital compilers to perform systematic design variation for an almost infinite amount of parameters, such graph-based languages are a powerful means to generate viable design alternatives and thus permit fast evaluations.
To leverage the full potential of graph-based design languages, possibilities are presented to expand their applicability into the domain of product functions. This possibilities allow to cohesively link integrative function modelling to product structures. This intends to close the gap between the early, abstract stages and the systematic, concrete design generation and validation with relevant CAE tools. In this paper, the IFM Framework was selected as integrated function model to be linked with the graph-based design languages.

Keywords: Functional modelling, Systems Engineering (SE), Digital / Digitised engineering value chains

Contact:
Stetter, Ralf
University of Applied Sciences Ravensburg-Weingarten
Mechanical Engineering
Germany
ralf.stetter@hs-weingarten.de

Cite this article: Elwert, M., Ramsaier, M., Eisenbart, B., Stetter, R. (2019) 'Holistic Digital Function Modelling with Graph-Based Design Languages', in *Proceedings of the 22nd International Conference on Engineering Design (ICED19),* Delft, The Netherlands, 5-8 August 2019. DOI:10.1017/dsi.2019.158

1 INTRODUCTION

Determining a suitable solution to a given design problem is an interplay between reasoning about what is required and what particular combination of solution elements will provide for it and how (Gero & Kannengiesser 2014; Chakrabarti & Bligh 2001). It is an iterative process as in analysing and gradually synthesising the potential solution, some of its features or constraints may necessitate redefinition of the problem (Braha & Reich 2003). Such iterations are not limited to individual design stages, but carry through the entire design process. This has to go hand-in-hand with continuously updating of models representing related information (Eckert *et al*., 2015). To support this, several recently proposed models specifically aim to coherently model functions and solutions, thus linking the conceptual and subsequent design stages (i.e. bridging between abstract/qualitative and increasingly quantitative information). These include System Modeling Language (SysML), Object-Process Methodology (OPM, Dori 1995) and the Integrated Function Modelling Framework (IFM, Eisenbart *et al*., 2016). Ultimately, all of these still remain at an abstract level in describing the resulting engineering system. This is to say, they mainly focus on the product structure and abstractly describe its compounding sub-systems, parts, etc. and their functional or operational links. The transition of this information into simulation tools for system design and testing, however, currently still relies on laborious manual transfer into individual calculation or Computer-Aided Engineering (CAE) tools like 3D modelling environments, Finite Element Analysis (FEA) for structural safety, thermodynamics, electromagnetic performance or the likes. Then, results from these CAE tools are not efficiently linked due to data formatting and exchange problems (Sztipanovits *et al*., 2012). Graph-based design languages offer a promising approach to address these issues. Such languages generate a digital meta- or system model storing all relevant information about a design and feed this into any relevant CAE tool as needed to simulate and test the impact of any design variation on the resulting product performance. As this can be automated in digital compilers to perform systematic design variation for an almost infinite amount of parameters, such graph-based languages are a powerful means to generate viable design alternatives – and thus permit fast comparison and selection – much faster than any design team manually could.

Parameter variation and computing, by nature, mainly pertains to later stages, when product information is more quantitative. To leverage the full potential of graph-based design languages, we seek to expand its applicability into the domain of product functions. As such, we want to cohesively link integrative function modelling, spanning from the functional level to product structures with graph-based design models, to close the gap between the early, abstract stages of product development and systematic, concrete design generation and validation with relevant CAE tools. In this paper, the IFM Framework was selected as integrated function model to be linked with the graph-based design languages. Both are independent of a particular software platform, yet equally comprehensive representation tools in their respective domain. Consequently, the main research question can be formulated as: How can the IFM Framework and graph-based design languages based on UML be combined to achieve holistic digital processes which include rich presentations on the functional level. In the following sections, the IFM Framework and graph-based design languages are subsequently introduced. Chapter 4 describes the conceptual, operational linking of both approaches, while Chapter 5 showcases their combined application based on a suitable, interdisciplinary engineering product, i.e. a multicopter. Chapter 6 discusses the results and proposes direction for further research.

2 THE IFM FRAMEWORK

The IFM framework is intended to provide designers with an integrated, cross-disciplinary approach for modelling system functionality, coupled with system structural modelling. Here, function is defined as an intended or already perceivable behaviour of a technical system to fulfil a specific task. In the framework, integration is facilitated through linking different types of contents prominently addressed in discipline-specific and cross-disciplinary function models (see Eisenbart *et al*., 2016). The respective information is presented in associated views, which are briefly described in Table 1. Their adjacent placement (see Figure 1) supports their parallel development and allows verification of their mutual consistency across the entire framework. Views are modular as they can be seamlessly added or omitted to ease flexible, demand-specific adaptation.

The central process flow view (Figure 1) represents the flow of transformation and interaction processes using standard symbols established for process/activity flow modelling similar to SysML, for instance. The

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

remaining views link to it and comprise of matrices equivalent to Design-Structure-Matrices (DSM) encompassing different entities related to system functionality and their interdependencies. Entities comprise use cases, transformation and interaction processes, effects, states, operands and actors. Use cases represent different scenarios of applying the technical system for a specific purpose (e.g. fulfilling a goal, changing the state of the system or user, etc.).

Table 1: Associated views in the IFM framework

View	Description
Process flow view	…qualitatively visualises the flow of sequential or parallel (interaction or transformation) processes related to a specific use case. For each use case an associated set of views is created. In the vertical direction, the process flow is visualised related to time which matches the gradual devolution of states in the associated state view. Horizontally, process blocks are spread from left to right to enable a direct link to the actor view.
State view	…represents the states from initial to final of operands and actors as well as their changes associated with related processes.
Actor view	…indicates the involvement of one or more actors in the realisation of individual processes. Involvement may be active or passive. Actors can part of the system or external.
Use case view	…indicates the involvement of individual processes in the different use cases.
Effect view	…represents the effects, which enable individual processes and are provided by actors. For each process block in the process flow view, a separate effect view may be created.
Interaction view	…maps the bilateral impacts between actors and operands as well as their complementary contributions (or any other kind of dependency between them) in the realisation of use cases, associated processes, etc., and thus represents the system's structure.

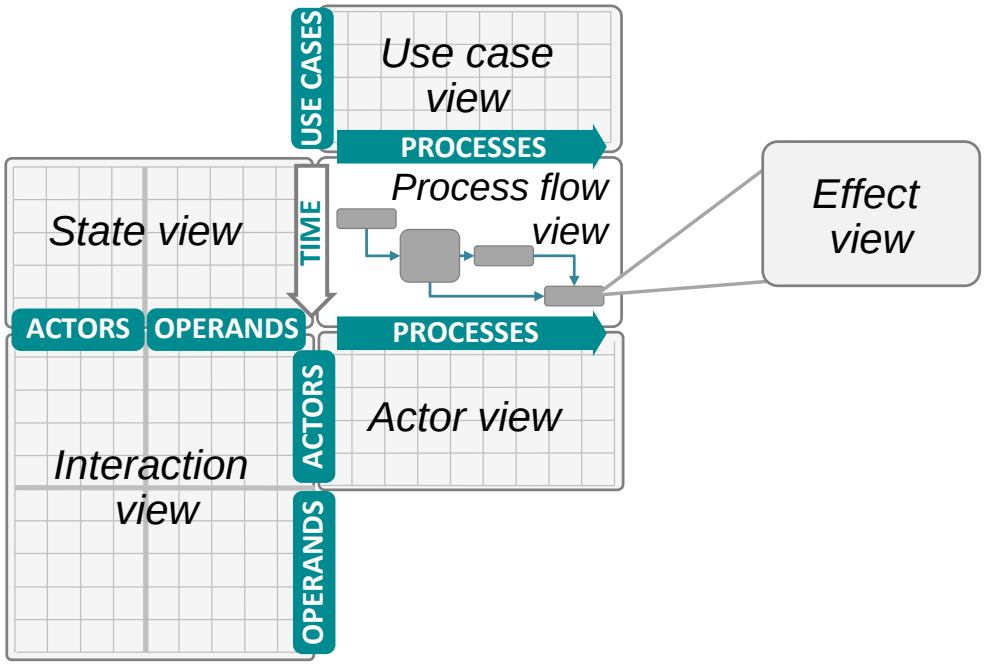

Figure 1: The IFM framework

Transformation processes describe technical and/or human processes - realised by basic physiochemical effects - that result in a change of state of operands or actors per every use case. Operands are specifications of energy, material and signals. Actors can be any human or other animate being and technical systems (hard- and/or software) that are actively or passively contributing to function fulfilment. Furthermore, actors include relevant parts of the environment that provide influences onto the system. Finally, interaction processes describe "cross-boundary" interactions between different actors jointly contributing to function fulfilment (similar to Eder & Hosnedl 2008). Figure 2 illustrates the particular entity-relations in the framework using a UML-based domain diagram.

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

In the application of the framework, designers start by deriving the central use cases the system will have to fulfil from a requirements specification. Subsequently, main processes for the use cases are determined and gradually detailed also taking into account any related operands and already known actors. Then, the views are gradually filled with more and more information until, finally, the system is known well enough to determine the mutual interactions among all involved actors and/or operands to fill the interaction view, thus resulting in a comprehensive system description on functional level. Examples of the application can be found in (Eisenbart *et al.*, 2016).

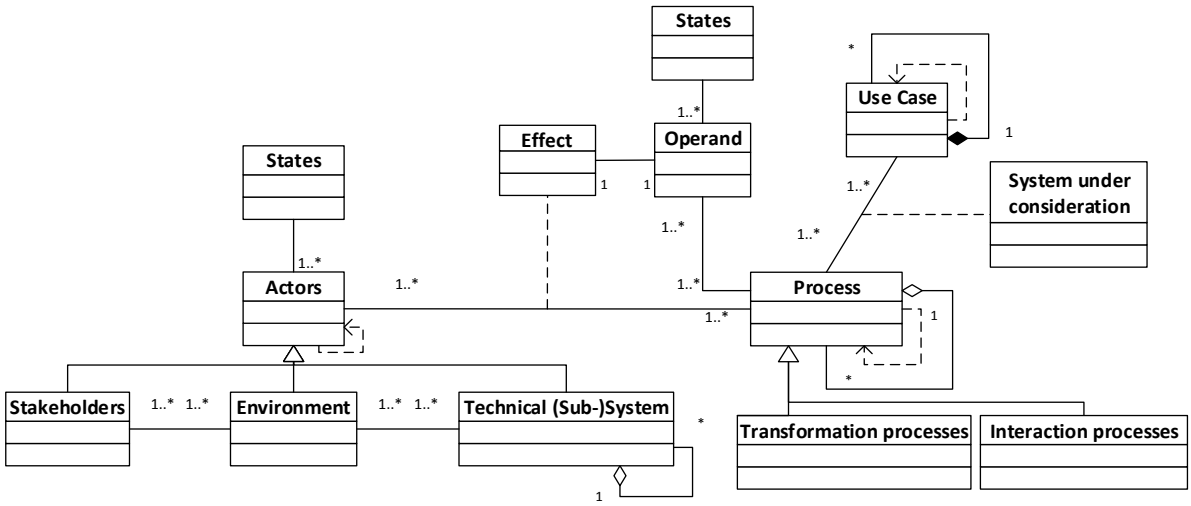

Figure 2: Domain diagram of the entity relations in the IFM framework

3 GRAPH-BASED DESIGN LANGUAGES

Graph-based design languages map product information into vocabulary and rules typically based on the Unified Modeling Language (UML) language. They provide a powerful engineering framework through their automates, systematic variation and calculation features, feeding into any relevant CAE tool, as per need. They permit machine execution and the reuse of design and manufacturing knowledge, through automated model generation, calculation and simulation. As such, it allows generation of a large number of design alternatives based on parameter variation across multiple simulation tools, in a fast and automated manner (Rudolph 2002, Kröplin & Rudolph 2005, Holder *et al.*, 2017, Ramsaier *et al.*, 2017, Zech *et al.*, 2019). In this context, graph-based means that the nodes of a graph serve as abstract placeholders for real objects and that a graph representation is machined on behalf of the design object. The advantages of graph-based languages can be used to their full extent if these methods are transferred to all phases of the lifecycle of products. One important objective of transferring these methods is to accelerate innovation. A second central goal of this transfer is to develop knowledge representations and, through this, to enable a processing of the product lifecycle that allows for more human-independent technological process integration, thereby improving knowledge transfer.

The basic concept of a graph-based design language is illustrated in Figure 3. Individual components are UML classes, which represent the objects of the product (vocabulary, bottom right). These components can be arranged in ontologies, i.e. organised, annotated, related and/or hierarchical arrangements of engineering entities which may represent aspects of engineering knowledge. These classes are connected in the UML class diagram by different associations and are instantiated via UML model transformations (rules, bottom left). A sequence of such rules is modelled in an activity diagram. Here is also where we identified a suitable way to link to the IFM Framework, as will be discussed below: actors and their states and activities/processes overlap directly between both modelling approaches. In this paper, we specifically focus on the Design Compiler 43™ (DC43) by IILS mbH. DC43 uses this information to create the design graph. From this central meta- or system model, further models (for example CAD geometry models, simulation models like FEM or multi-body-system, MBS, reports, etc.) can be generated in a fully automatic manner.

The main difference between graph-based design languages and conventional product development is that not a single product is designed but a whole family of products is designed automatically. It is one of the main advantages of graph-based design languages that the “members” of this product family cannot only be parametrically different but also topologically different. However, it also has to be

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

mentioned that the design automation of product families requires a novel mind-set of the designer and thus can be understood as paradigm shift. A central advantage of using graph-based design languages in connection with a design compiler is the possibility of a fully automated process that includes calculation and simulation steps. This characteristic allows the generation of a multitude of possible product configurations and through this the realization of optimization cycles which can lead to product configuration which represent an optimum solution in a multi-dimensional solution space. This is especially advantageous for systems with many configuration possibilities such as wire harnesses and pipe systems such as exhaust systems (Vogel and Rudolph 2016). The third notable advantage is the possibility to include multiple physical domains, e. g. it is possible to address simultaneously the mass and moment balance and thermal validation of a satellite (Vogel and Rudolph 2018).

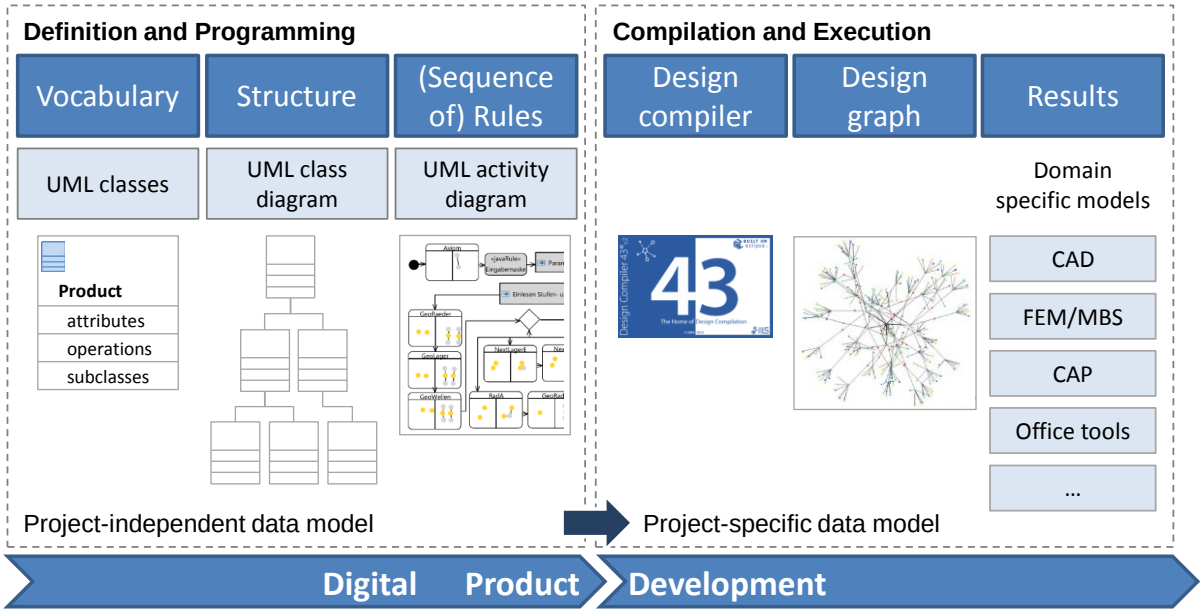

Figure 3: Basic concept of graph-based design languages

4 LINKING BOTH APPROACHES

For linking both approaches, the class diagram shown in Figure 2 has been re-modelled in UML in order to be able to use it in a design language. The respective class diagram can be seen in Figure 4.
The entities as they are represented in the IFM Framework are integrated with the classes and instantiations represented/modelled in DC43. Different from the IFM, for instance, DC43 can be used for requirements engineering as well. Therefore, the class "Requirement" has been added in Figure 4. The requirements are directly linked to use cases. This means, it is possible to identify, which use case fulfils which requirement and which requirements are not covered yet by any use case (c.f. Holder *et al.*, 2017). Furthermore some links and hand-over interfaces had to be expanded and updated to match the DC43 UML environment, these changes are highlighted in red in Figure 4.
The aim of the design language is to convert an existing IFM framework into a UML model fully automatically. A UML model enables the automatic further processing of the function model up to the creation of automatic verification scenarios. In this paper we concentrate on the creation of the UML-model and not on its use in the further course of the modelling process. Based on the current state it is possible to identify actors, which are not connected to any functions. This means, the actors have no active role in the function model and their role needs to be clarified. Additionally, functions, which have no actors fulfilling them, can be identified.

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

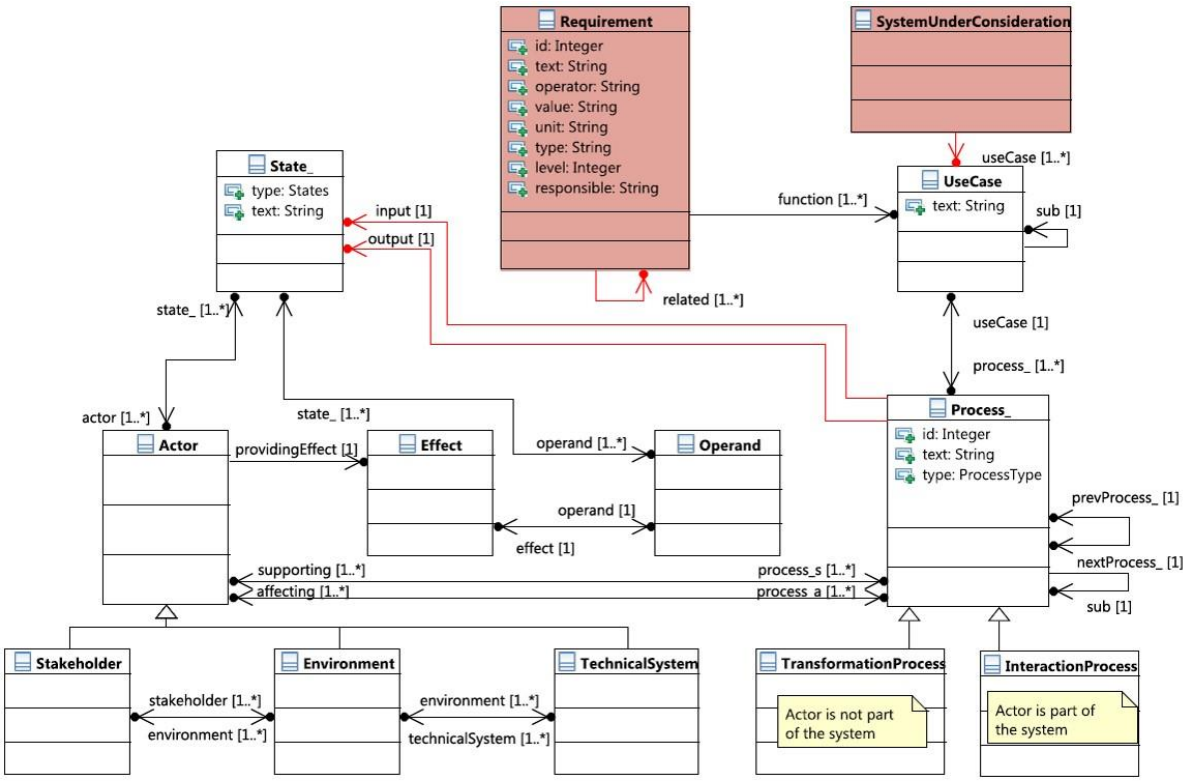

Figure 4: UML Class Diagram for Function Modelling in DC43

In the following, the workability of the expanded class diagrams for the integration of the IFM Framework into DC43 is tested through application to the example of a multicopter (compare Ramsaier *et al*., 2016), as shown in Figure 5.

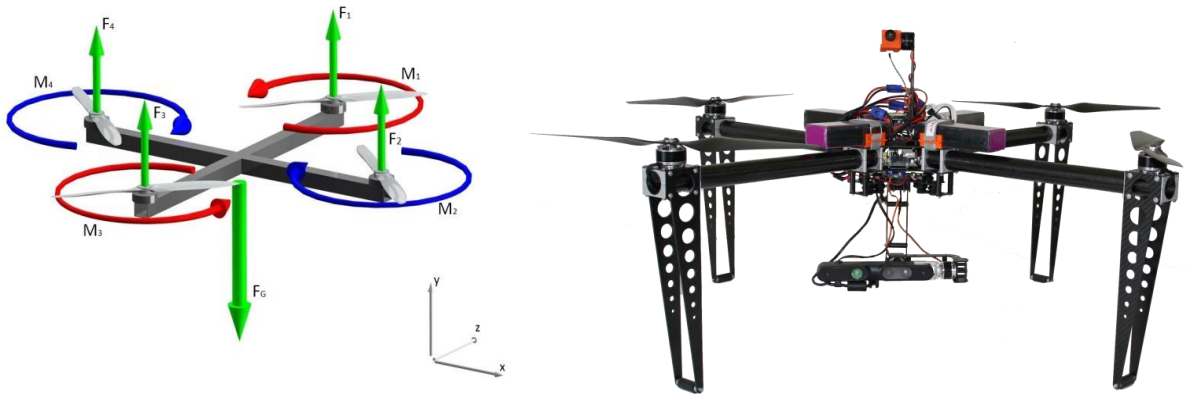

Figure 5: Basic structure of a multicopter and its concrete realisation

Mechanically, a multicopter is a rather simple system, which receives its stable flying ability mainly through complex control engineering. The simplest structure for a multicopter is a circular arrangement of four rotors placed in an angle of 90 degrees to each other (see Figure 5). The motors are mounted on a frame that includes a landing gear and appliances for mounting electronics and the power supply. This multicopter is one of the use cases in the underlying research project and serves for an investigation of promising approaches to combine mechanical design and control engineering in a holistic digital model. Additionally, it serves as an example for investigating lightweight design. The specific challenge in this lightweight development was to find geometries which lead to a very rigid frame structure that suppresses oscillations and lead to an easier flight control. Thus, the multicopter is modelled using graph-based design languages (compare Ramsaier *et al*., 2017) in order to be able to analyse multiple configurations and carry out multi-domain simulations. For a functional representation of the multicopter, it is necessary to take the fact into account that several use-cases (hovering, descending,

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

etc.) need to be considered. This leads initially to the formulation of a list of processes. One example of a process could be Process 2 "distribute energy according to need" which describes the controlling of the motor currents of the electrical motor, which drive the rotors on an abstract, solution-neutral level. The processes formulated in the multicopter function model are: (1) start system, (2) distribute energy according to need, (3) sending steering command, (4) send steering command via RC, (5) calculate flightpath and motor engagement, (6) increase / decrease rotation speed of motors, (7) track position, (8) track rate of ascent/descent, (9) track roll rate and angle, (10) hold position, (11) landing quadrocopter, (12) switch system off and (13) distribute forces as illustrated in the process flow view in Figure 7.

The developed IFM model of the multicopter is illustrated in the subsequent figures. As stated above, the IFM consists of up to five views on the system under development; Figure 6 shows the state view.

time

Resting on ground	Inactive	Securing rigs	Inactive, securing sensors	Providing mechanical support to ground	Securing camera
P1, P2, P13	P1, P2, P13	P1, P2, P13	P1, P2, P13	P1, P2, P13	P13
in flight, distributing stresses	distributing forces	distributing forces, securing rigs	distributing forces	inactive	
P11	P11	P11	P11	P11	
Resting on ground, distributing forces from rotors	distributing forces from rotors	distributing forces from rotors, securing rigs	distributing forces from rotors	Providing mechanical support to ground	
P12	P12	P12	P12		
Resting on ground	Inactive	Securing rigs	Inactive	Providing mechanical support to ground	Securing camera

Chassis Frame	Rigger	Rigger clamp	Centre Plate	Landing Gear	Camera Mounting

Figure 6: IFM Model: State view

This state flow starts with the state "resting on ground" of the chassis/frame and shows possible processes which may lead to a change in state. For instance, the processes P1, P2 and P13 may result in a state change to "in flight, distributing forces" again for the chassis/frame. A close look on these process is fostered through the process flow view (compare Figure 7). In this view, the processes also visible in the state diagram are linked and the interplay of these processes becomes transparent. It is important to note that certain decision points can be included in the process flow view.

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

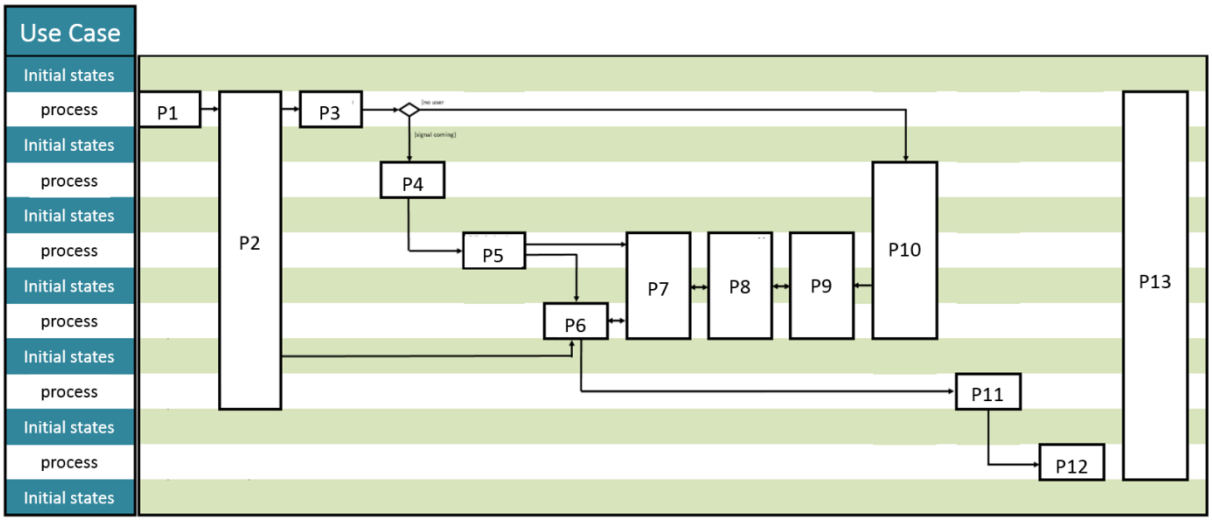

Figure 7: IFM Model: Process flow view

All processes in technical systems such as the multicopter under consideration are realised by means of actors; their connection to the processes are illustrated in the actor view (compare Figure 8).

			P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
Chassis	Chassis Frame	Actors											x		x
	Rigger												x		x
	Rigger clamp												x		x
	Centre Plate												x		x
	Landing Gear														
	Camera Mounting						o		o	o	o	o	x		x
	Sonar Mounting						o		o	o	o	o	x		x
Cuboid	Cuboid														
Cylinder	Cylinder														
Propulsion System	Motor		o	o			o	o	o	o	o	o	x	o	x
	Propeller		o	o			o	o	o	o	o	o	x	o	x
	Battery		o	x			o	o	o	o	o	o		o	
	Electronic Speed control		o	o			o	x	o	o	o	o		o	
Sensors	Gyroscope		o				o		x	x	x	x		o	
	Laser IR Camera		o				o		x	x	x	x		o	
	Camera Pitch		o				o		x	x	x	x		o	
	Cameral Roll		o				o		x	x	x	x		o	
	Sonar		o				o		x	x	x	x		o	
	Sonar Pitch		o				o		x	x	x	x		o	
	Gimbal Controller														
	Raspberry PI														
	Flightcontrol		o	o		o	x	x	x	x	x	x		o	
Energy Supply	Voltage Converter		o	x										o	
External	Remote Control		o		x	x	x"	x"					x"		
	User		x		x	x							x"	x	

Figure 8: IFM Model: Actor view

In the actor view (Figure 8), a distinction between active actors indicated by an “x” (such as the actor “Battery” in the process “distribute energy according to need”) and passive actors indicated by an “o” (such as the “Battery” in the process “start system”) is possible, thus allowing the identification of especially active actors.

Here, the discussed link between the IFM Framework and the DC43 UML model becomes nicely pertinent: through the actor view and interaction view that represent a concise system model which links directly into the UML model in DC43 and the instantiation of objects/actors therein. This model can be a basis for the further development steps of the technical system (i.e. the multicopter).

Based on the IFM model presented above, the design language automatically generates a design graph represented in UML (see Figure 9). Based on the UML model, the above mentioned automated reasoning tests can be applied. From the high-level central data model represented in the design graph, different interfaces generate domain specific, generic models such as geometry models or simulation models. In the multicopter project, several generic models can be generated (Figure 9).

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

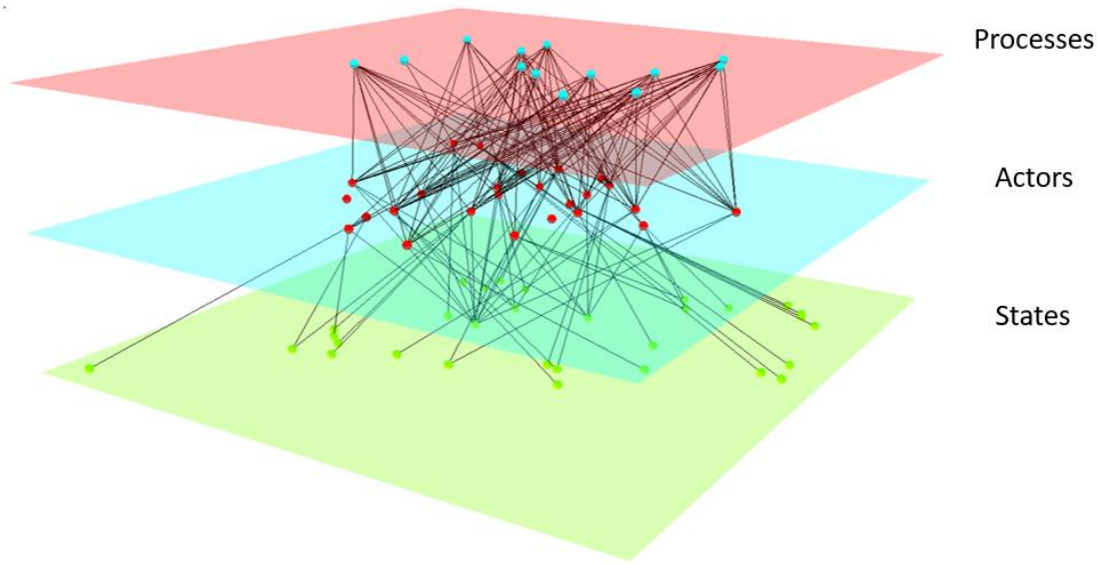

Figure 9. Design graph illustration of the multicopter function model

Thus, the IFM model presents a framework which allows modelling the abstract structure of a systems in a way which fosters the automated generation of cross-domain models including geometry and allowing the automated exploration of numerous product configurations; i.e. a large product portfolio. As it is shown in Figure 9, different domain-specific models are generated based on the design graph. These models are often used in order to verify the products required functions. For the multicopter example, a desired rate of climbing and turning around each axis is required. This is verified by simulation with a control model. Further, the frame has to withstand the loads from the lifting force of the rotors. This can also be simulated with an FEM model. Additionally, a cost target can be set at the beginning of the design phase. After all parts are defined and the manufacturing methods/process are chosen, the bill of materials and the cost calculation will tell if the goal can be achieved - in turn, design variations that do not fulfil the cost requirements can be flagged and automatically sifted out. With the IFM model it is possible to check, which actor has achieved its functions and which actor has not - and also if the function fulfilment and behaviour is within the set design targets based on simulation. This enables the engineer to focus quickly on what has to be changed in order to comply with all requirements. Additionally it is possible, due to the fully automated execution, to investigate a large number of possible multicopter configurations and to generate an optimum configuration.

5 CONCLUSIONS

In recent years, several models were proposed which share the objective to coherently model functions and solutions. However, up to now these models still remain at an abstract level in describing the resulting engineering system. The research described in this paper intends to expand the possibilities to apply function modelling approaches for the automated generation of product configurations, i. e. parametrical and topological variations of the developed product family. In order to do so, graph-based languages are used in combination with the well-known IFM Framework. It was shown, how function modelling can be done with the IFM model approach within an integrated framework. Further, this framework can be automatically processed by a graph-based design language, which translates the IFM function model into an UML model. This UML model can be used for automated reasoning tests of the function model. A unique strength of the proposed application of graph-based languages is the possibility to generate viable design alternatives and to simulate their behaviour, thus permitting fast comparison and selection. Another strength of the developed approach, compared with other powerful approaches based on SysML, is represented in the possibility to create detailed geometrical models directly from a central data model formulated in UML.The language generates a digital meta- or system model which contains all relevant information about a design. This information can be fed into many relevant CAE tool such as finite element analysis systems, thus allowing an in-depth evaluation of the impact of a design variation. In this paper, a multicopter design served as a basis for validation. Obviously, future research work is necessary in order to achieve general applicability. Additional aspects of further research are to increase the automation level and expanding the cross-domain

https://doi.org/10.1017/dsi.2019.158 Published online by Cambridge University Press

modelling capabilities. Future application possibilities include automated reasoning test on the functional level, which will range from plausibility checks to simulations of the parametrical and topological relations between the different elements of the function model.

REFERENCES

Braha, D. and Reich, Y. (2003), "Topological Structures for Modeling Engineering Design Processes", *Research in Engineering Design*, Vol. 14 No. 4, pp. 185–199.

Chakrabarti, A. and Bligh, T.P. (2001), "A Scheme for Functional Reasoning in Conceptual Design", *Design Studies*, Vol. 22 No. 6, pp. 493–517.

Dori, D. (1995), "Object-process Analysis: Maintaining the Balance Between System Structure and Behavior", *J Log Comput*, Vol. 5 No. 2, pp. 227–249.

Eckert, C., Albers, A., Bursac, N., Chen, H., Clarkson, P., Gericke, K., Gladysz, B., Maier, J., Rachenkova, G., Shapiro, D. and Wynn, D. (2015), "Integrated Product nad Process Models: Towards an Integrated Framework and Review", *Proceedings of the 20th International Conference on Engineering Design - ICED2015*.

Eder, W. and Hosnedl, S. (2008), *Design Engineering: a Manual for Enhanced Creativity*. CRC Press, Boca Raton, London, New York.

Eisenbart, B., Gericke, K., Blessing, L. and McAloone, T. (2016a), "A DSM-based Framework for Integrated Function Modeling: Concept, Application and Evaluation", *Research in Engineering Design*, Vol. 28 No. 1, pp. 25–51. https://doi.org/10.1007/s00163-016-0228-1

Gero, J.S. and Kannengiesser, U. (2014), "The Function-Behaviour-Structure Ontology of Design", in Chakrabarti, A. and Blessing, L.T.M. (Eds.), *An Anthology of Theories and Models of Design: Philosophy, Approaches and Empirical Explorations*, Springer-Verlag, Berlin.

Holder, K.; Zech, A.; Ramsaier, M.; Stetter, R.; Niedermeier, H.-P.; Rudolph, S. and Till, M. (2017), "Model-Based Requirements Management in Gear Systems Design Based On Graph-Based Design Languages", *Applied Sciences* Vol. 7 No. 11-12, https://doi.org/10.3390.

IILS mbH (2016), *Design Compiler 43™ is a trademark of IILS mbH*. www.iils.de.

Kröplin, B. and Rudolph, S. (2005), "Entwurfsgrammatiken – Ein Paradigmenwechsel?", *Der Prüfingenieur*, Vol. 26, pp. 34–43.

Ramsaier, M.; Spindler, C.; Stetter, R.; Rudolph, S. and Till, M. (2016), "Digital representation in quadrocopter design along the product life-cycle", *CIRP ICME, Ischia, Italy.*

Ramsaier, M.; Holder, K.; Zech, A.; Stetter, R., Rudolph, S. and Till, M. (2017), "Digital Representation of Product Functions in Multicopter design", *Proceedings of the 21st International Conference on Engineering Design - ICED17*.

Rudolph, S. (2002), "Übertragung von Ähnlichkeitsbegriffen", *Habilitationsschrift, Fakultät Luft- und Raumfahrttechnik und Geodäsie*, Universität Stuttgart, Stuttgart.

Sztipanovits, J., Koutsoukos, X., Karsai, G., Kottenstette, N., Antsaklis, P., Gupta, V., Goodwine, B., Baras, J. and Wang, S. (2012), "Toward a Science of Cyber-Physical System Integration", *Proceedings of the IEEE*, Vol. 100, pp. 29–44. https://doi.org/10.1109/JPROC.2011.2161529.

Vogel, S. and Rudolph, S. (2016), "Automated Piping with Standardized Bends in Complex Systems Design", *Complex Systems Design and Management: Proceedings of the Seventh International Conference on Complex Systems Design and Management, CSD&M Paris 2016*. https://doi.org/10.1007/978-3-319-49103-5_9.

Vogel, S. and Rudolph, S. (2018), "Complex System Design with Design Languages: Method, Applications and Design Principles", *Proceedings of the CSCMP 2018, XXth International Conference on Complex Systems: Control and Modeling Problems*.

Zech, A.; Stetter, R.; Holder, K.; Rudolph, S. and Till, M. (2019), "Novel approach for a holistic and completely digital represented product development process by using graph-based design languages",. *Procedia CIRP*, Vol. 79, pp. 568–573.

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).